cadence®

Virtuoso Module Generator User Guide

Product Version ICADVM20.1
October 2020

Printed in the United States of America.

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

Contents

Preface

Virtuoso® Module Generators (Modgens) let you generate multiple Pcell instances into a complex, highly matched, structured array. Modgens provide an intuitive way to quickly generate Pcell instances into a complex, highly matched, and structured array. You can specify the devices to be arrayed and the interdigitating pattern to be applied. You can also insert dummy devices, body contacts, and guard rings.

This user guide is aimed at developers and designers of integrated circuits and assumes that you are familiar with:

- The Virtuoso design environment and application infrastructure mechanisms designed to support consistent operations between all Cadence® tools.
- The applications used to design and develop integrated circuits in the Virtuoso design environment, notably, the Virtuoso Layout Suite, and Virtuoso Schematic Editor.
- The Virtuoso design environment technology file.
- Component description format (CDF), which lets you create and describe your own components for use with Layout XL.

This preface contains the following topics:

Scope

Unless otherwise noted, the functionality described in this guide can be used in both mature node (for example, IC6.1.8) and advanced node (for example, ICADVM20.1) releases.

Label	Meaning
(ICADVM20.1 Only)	Features supported only in the ICADVM20.1 advanced nodes and advanced methodologies releases.
(IC6.1.8 Only)	Features supported only in mature node releases.

Licensing Requirements

The licenses that are checked out for running the various Modgen editing and routing topology features depend on the licenses that are available. Therefore, irrespective of the active product cockpit, when a Modgen feature is launched, the available licenses are checked out based on the following precedence:

In IC6.1.8:

1. Virtuoso_Layout_Suite_EAD + 2 GXL tokens OR
2. Virtuoso_Layout_Suite_GXL + 2 GXL tokens OR
3. Virtuoso_Layout_Suite_XL + 2 GXL tokens

In ICADVM20.1

1. Virtuoso_Layout_Suite_EXL OR
2. Virtuoso_Layout_Suite_XL + 2 GXL tokens

Note: Modgen features are available through scripting in the Command Interpreter Window without a cockpit, with the same licenses as listed above.

For information on licensing in the Virtuoso design environment, see *Virtuoso Software Licensing and Configuration Guide*.

Related Documents

What's New and KPNS

- *Virtuoso Module Generator What's New*
- *Virtuoso Module Generator Known Problems and Solutions*

Installation, Environment, and Infrastructure

- *Cadence Installation Guide*
- *Virtuoso Design Environment User Guide*
- *Virtuoso Design Environment SKILL Reference*
- *Cadence Application Infrastructure User Guide*

Technology Information

- *Virtuoso Technology Data User Guide*
- *Virtuoso Technology Data ASCII Files Reference*
- *Virtuoso Technology Data SKILL Reference*
- *Virtuoso Technology Data Constraint Reference*

Virtuoso Tools

IC6.1.8 Only

- *Virtuoso Layout Suite L User Guide*
- *Virtuoso Layout Suite XL User Guide*
- *Virtuoso Layout Suite GXL Reference*

ICADVM20.1 Only

- *Virtuoso Layout Viewer User Guide*
- *Virtuoso Layout Suite XL: Basic Editing User Guide*

- *Virtuoso Layout Suite XL: Connectivity Driven Editing Guide*
- *Virtuoso Layout Suite EXL Reference*
- *Virtuoso Design Planner User Guide*
- *Virtuoso Multi-Patterning Technology User Guide*
- *Virtuoso Placer User Guide*
- *Virtuoso Width Spacing Patterns User Guide*
- *Virtuoso RF Flow Guide*

IC6.1.8 and ICADVM20.1

- *Virtuoso Custom Digital Placer User Guide*
- *Virtuoso Floorplanner User Guide*
- *Virtuoso Fluid Guard Ring User Guide*
- *Virtuoso Layout Suite SKILL Reference*
- *Virtuoso Parameterized Cell Reference*
- *Virtuoso Symbolic Placement of Devices User Guide*

Additional Learning Resources

Video Library

The Video Library on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

To view a list of videos related to a specific product, you can use the *Filter Results* feature available in the pane on the left. For example, click the *Virtuoso Layout Suite* product link to view a list of videos available for the product.

You can also save your product preferences in the Product Selection form, which opens when you click the *Edit* icon located next to *My Products*.

Virtuoso Videos Book

You can access certain videos directly from Cadence Help. To learn more about this feature and to access the list of available videos, see Virtuoso Videos.

Rapid Adoption Kits

Cadence provides a number of Rapid Adoption Kits that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

In addition, Cadence offers the following training courses on Modgens:

- MODGEN
- Modgen on Canvas
- Row Based Placement

The above courses are available in North America. For specific information about the courses available in your region, visit Cadence Training or write to training_enroll@cadence.com.

Note: The links in this section open in a separate web browser window when clicked in Cadence Help.

Help and Support Facilities

Virtuoso offers several built-in features to let you access help and support directly from the software.

- The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.
- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

  The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (provided you have not set a custom home page).

For more information, see Getting Help in *Virtuoso Design Environment User Guide.*

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

  Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit https://www.cadence.com/support.

- Log on to Cadence Online Support

  Customers with a maintenance contract with Cadence can obtain the latest information about various tools at https://support.cadence.com.

Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Provide Feedback* box.

Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

text	Indicates names of manuals, menu commands, buttons, and fields.

`text`	Indicates text that you must type exactly as presented. Typically used to denote command, function, routine, or argument names that must be typed literally.
`z_argument`	Indicates text that you must replace with an appropriate argument value. The prefix (in this example, *`z_`*) indicates the data type the argument can accept and must not be typed.
`|`	Separates a choice of options.
`{ }`	Encloses a list of choices, separated by vertical bars, from which you **must** choose one.
`[ ]`	Encloses an optional argument or a list of choices separated by vertical bars, from which you **may** choose one.
`[ ?argName` *`t_arg`* `]`	Denotes a *key argument*. The question mark and argument name must be typed as they appear in the syntax and must be followed by the required value for that argument.
`...`	Indicates that you can repeat the previous argument. Used with brackets to indicate that you can specify zero or more arguments. Used without brackets to indicate that you must specify at least one argument.
`,...`	Indicates that multiple arguments must be separated by commas.
`=>`	Indicates the values returned by a Cadence® SKILL® language function.
`/`	Separates the values that can be returned by a Cadence SKILL language function.

If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash (\ ) indicates that the current line continues on to the next line.

1

Getting Started

This chapter covers the following topics:

Understanding Module Generators

Module generators are designed to provide a way to generate multiple Pcell instances into a complex, highly matched, structured array. With the Modgen tool, you specify the devices to be arrayed, then specify an interdigitation pattern, insert dummy devices, body contacts, and guard rings. Finally, you control the routing style and generate internal routing geometry.

Module generators can be created from either the schematic or the layout. When you create a Modgen from the schematic, the tool creates a Modgen constraint on the schematic that will then be used when the layout is generated using *Generate All From Source* or *Generate Selected From Source*. When run from the schematic, by default, the Modgen creates a temporary physConfig. Use the modgenPhysConfigs environment variable to specify a different physconfig that already exists to be used by the Modgen.

The Modgen router cannot route devices with horizontal terminals. If you want all devices in the Modgen to be horizontal, first route the Modgen with the terminals in a vertical orientation and then rotate the entire Modgen to accomplish the desired effect.

To know more about the Modgen Flow in Virtuoso, view or download the Modgen Rapid Adoption Kit. This kit provides information about using various Modgen functionalists to increase layout productivity.

Technology File Requirements

The following Mogen features require specific technology file information.

Routing

Routing in the module generators requires a complete Layout Function section for metal and poly layers and basic width and spacing rules on the metal and poly layers.

Body Contacts

In order to use body contacts in the module generators, you must complete either the Standard Via definition or a custom via definition, which define a connection to well/active layers. In addition, the well and active layers should be defined in the Layer Function section.

Guard Rings

To utilize guard rings in module generators, define your Multiple Part Paths (MPPs).

Technology Group and Constraint Group Support

Modgens read rules from user-defined Constraint Groups as well as the Foundry technology group. The default lookup path for these tools is in the following order:

1. Specified tool Constraint Group, which is stored either in the design or in the technology library
2. Design-level default Constraint Group
3. Foundry Constraint Group, which is stored in the technology library

The tool Constraint Group is stored outside the Foundry Constraint Group. So before running the tool, you need to specify the tool Constraint Group to be used. Use the following environment variables to specify the tool Constraint Group.

Modgen: modgenPlacementConstraintGroup

While specifying the environment variable, if the string is left empty, then the Foundry Constraint Group is used. If the string has an invalid value, a warning is displayed and then the Foundry Constraint Group is used.

Note: If the default Wire Editing Constraint Group is set, then Modgen reads from it as well.

Placement Grid Support

Placement grid support is a method that keeps all shapes on specific grids per layer. The placement grid rule is obeyed at a higher precedence than custom spacing or detailed spacing rules. So, even if, according to the custom spacing rule, an instance is placed off the instance grid, it is automatically snapped to the next nearest placement grid. In addition, the origin of the Modgen FigGroup is placed on the placement grid. This ensures correct placement of instances on their respective placement grids when the entire Modgen is placed by the placer.

Use environment variable chainPermutePins to controls the application of the placement grid rule.

If the aapUsePlacementGrid is set to *t*, you need to set the following placement grid rules either in the techfile or in a custom Constraint Group that the Modgen refers to.

- **horizontalPlacementOffset** – Specifies the horizontal offset from the origin for the vertical placement grid
- **horizontalPlacementPitch** – Specifies the horizontal spacing between each vertical placement grid line
- **verticalPlacementOffset** – Specifies the vertical offset from the origin for the horizontal placement grid.
- **verticalPlacementPitch** – Specifies the vertical space between each horizontal placement grid line.

Example:

```
(placementGrids
    (verticalPitch 1.0)
    (verticalOffset 0.0)
    (horizontalPitch 1.5)
    (horizontalOffset 0.5)
)
```

Situation 1: The placement grid values exist and are on the grid with respect to the manufacturing grid

- During placement, the origin of each instance is snapped to the nearest placement grid horizontally and vertically. For the Modgen, the instances are snapped up.

Situation 2: The placement grid values exist, but are not on the grid with respect to the manufacturing grid

- A warning is displayed and placement continues using only the manufacturing grid.

Setting the Default Workspace for Modgen

Use the modgenWindowConfigFile environment variable to specify the default workspace to be loaded when Modgen is invoked. The specified workspace, if available either in the local .cadence directory or in any directory in the CSF hierarchical search, is applied to the current Modgen session. If the workspace is invalid, then a warning message is displayed and the default Modgen Edition workspace is loaded.

You can also create a workspace for the current Modgen session. For detailed information on workspaces and how you define them, see Getting Started with Workspaces and Working with Workspaces in the *Virtuoso Design Environment User Guide* and Workspaces in the *Virtuoso Layout Suite XL User Guide*.

Creating a Modgen

You can use the Modgen tool to create one placeable object from multiple instances.

Video

For more information on creating Modgens, see the Creating Modgens video.

Modgens can be created from both, schematic and layout designs.

Important

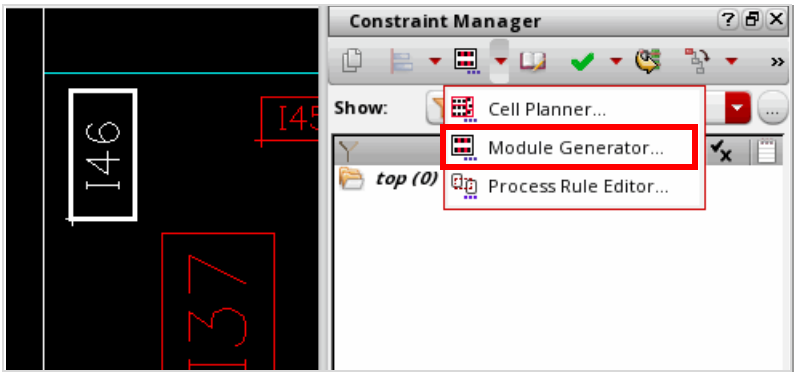
When creating a Modgen from a layout design, the device spacing values are compacted to the minimum DRC and constraint correct spacing. To retain and apply the layout spacing values as the custom spacing values, set the modgenCreatePreserveSpacing to `t`.

The modgenReferencePoint environment variable specifies the reference point for aligning Modgens that are either created or repositioned. The default value is `center`, where the center of the new or edited Modgen is aligned with the center of an existing Modgen. Use modgenCreateReferenceLPP to specify the layer geometry to be used to determine the Modgen's pattern.

There are several ways to create a Modgen.

Method 1 - Using the Constraint Manager Toolbar

1. Select the devices you want to include as one placeable object from the schematic or layout design.
2. Choose *Module Generator* from the Constraint Manager toolbar.



A new Modgen constraint is created and the Modgen Editor is invoked.

Method 2 - Using the Place Menu

1. Select the instances, Modgens, figGroups, and mosaics you want to include as one placeable object from the schematic or layout design.
2. Select *Place—Modgen—Create/Edit Modgen*.

A new Modgen constraint is created. However, the Modgen Editor is not invoked. Use the Modgen on-canvas commands to edit the Modgen directly in the layout canvas.To know more about these commands, see Using the Modgen On-Canvas Commands.

Method 3 - Using Convert to Modgen

A mosaic is a compact array of instances that belong to the same mfactored instance, and therefore have the same master and connectivity. You can convert a mosaic in the layout design to a Modgen. To do this:

1. Select the mosaic to be converted into a Modgen.
2. Select *Edit—Convert—To Modgen*. Alternatively, right-click the mosaic in the canvas and choose *Convert to Modgen* from the shortcut menu.

A new Modgen constraint is created. However, the Modgen Editor is not invoked. Use the Modgen on-canvas commands to edit the Modgen directly in the layout canvas. To know more about these commands, see Using the Modgen On-Canvas Commands.

Method 4 - Using SKILL Functions

Use one of the following SKILL Functions:

- **ciConCreate**: Creates the Modgen constraint.

  Example 1: Creates a Modgen constraint that includes the specified members. The parameters define the number of rows and number of columns that the Modgen constraint should have:

```
modgen = ciConCreate(cache 'modgen
    ?members list( list("M0" 'inst) list("M1" 'inst)   list("M2" 'inst)
list("M3" 'inst) )
    ?params list( list( "numRows" 1) list( "numCols"  4)
     ); list
); ciConCreate
```

  Example 2: Includes the pattern parameter, which maps the symbols in the Modgen to the required alphabets.

```
Modgen = ciConCreate(cache `modgen
```

```
    ?members list(
    list("M0" 'inst  list( list("row" 0) list("col" 0)))
    list("M0" 'inst  list( list("row" 0) list("col" 1)))
    list("M1" 'inst  list( list("row" 1) list("col" 0)))
    list("M1" 'inst  list( list("row" 1) list("col" 1)))
    ); list
    ?params   list(
    list( "numRows" 2)
    list( "numCols" 2)
    list( "pattern" "mapping (  (M0  X) (M1 Y)  ) ")
    ); list
); ciConCreate
```

- **gpeCreateSandbox**: Creates a Modgen sandbox object that includes the specified instances.

  Example: Creates a Modgen similar to Example 2 above.

  ```
  sbox = gpeCreateSandbox( ?sync t);
  gpeSetMap(sbox list( list("M0" "X") list("M1" "Y")));
  gpeSetGrid(sbox list( "X" "X") list("Y" "Y")));
  ```

- **mgCreateModgenAsLayout:** Creates a Modgen from the list of selected instances and keeps the instance locations inside the Modgen, as close as possible to their original locations in the layout.

  Example: Creates a Modgen in the current editing cellview (`geGetEditCellView`) that includes all instances current selected (`geGetSelSet`).

  ```
  mgCreateModgenAsLayout(geGetEditCellView() geGetSelSet())
  ```

- **mgCreateModgenConstraintAsLayout:** Creates a Modgen constraint that uses the current layout to drive the initial Modgen constraint and row/column assignments.

  Example: The following example works only if the current cellview is of the layout type. `geGetEditCellView` returns a valid layout cellview ID. The Modgen includes the instances that are selected in the cellview.

  ```
  mgCid = mgCreateModgenConstraintAsLayout(geGetEditCellView()
  geGetSelSet())
  ```

Editing a Modgen

Use one of the following methods to open an existing Modgen in the Modgen editor:

- Double-click the Modgen in the design.
- Double-click the Modgen constraint in the Constraint Manager toolbar.
- Select the Modgen and choose *Place—Modgen—Create/Edit Modgen*.
- Select the Modgen and choose the Module Generator icon in the Constraint Manager assistant toolbar.
- Select the Modgen figGroups in the layout canvas, right click, and select *Edit Modgen*.

Corresponding SKILL function: `mgCreateOrEdit`

Tip

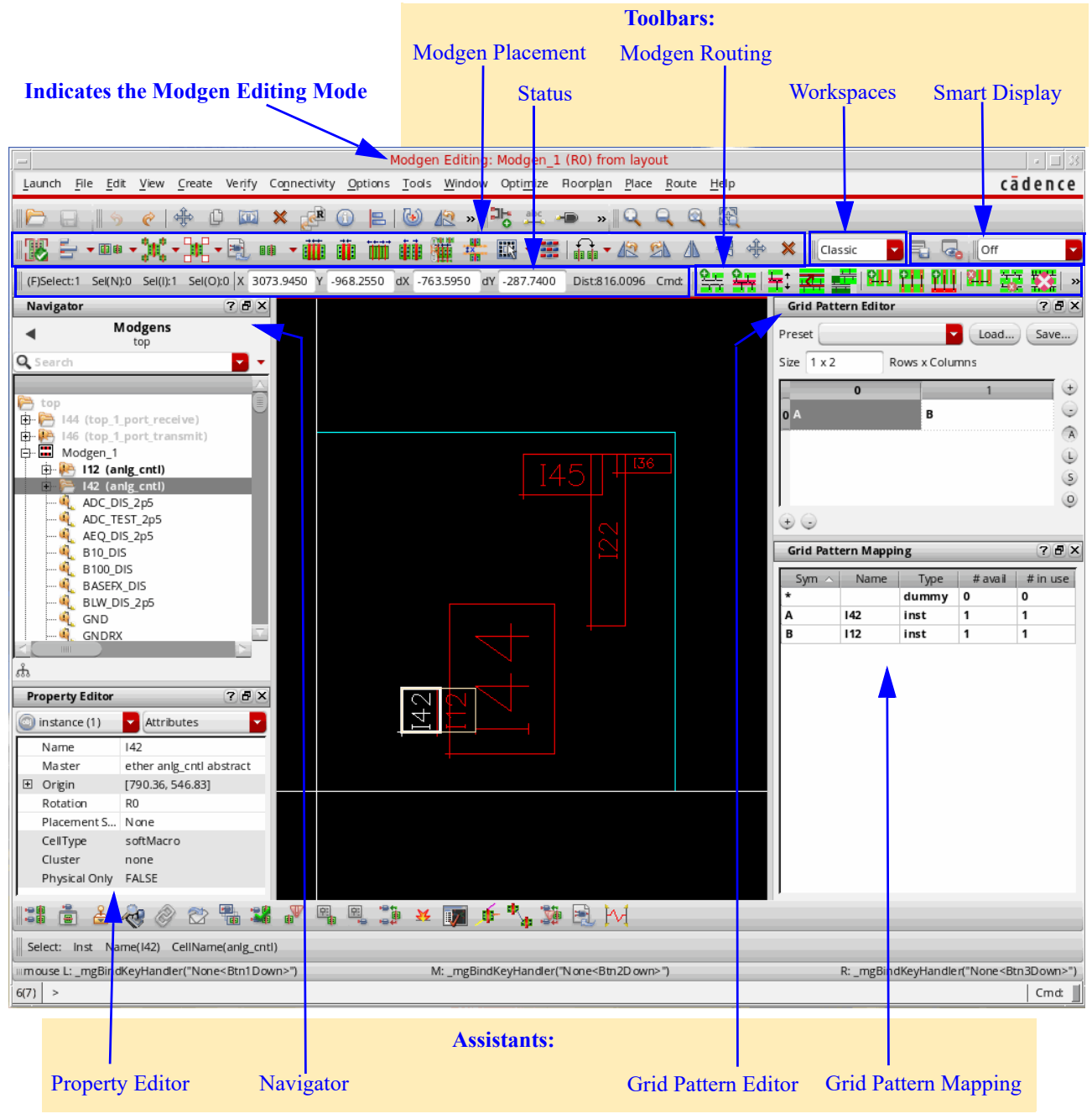
If the Constraint Manager isn't visible, choose *Windows – Assistants – Constraint Manager*. If the *Modgen* icon isn't visible, click the *>>* button.

(Layout only) If you launched Modgen from the layout, place the module in the layout.

The module appears in the layout window, and the layout window moves into Module Generator Edit mode. The default workspace is displayed. You can either use the existing default workspace or define a different default workspace and set it as the default workspace. For more information about setting a different default workspace, see Modifying a Modgen.

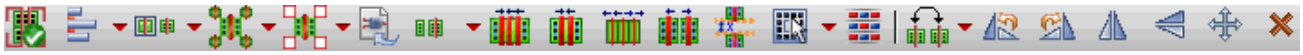
Note: When a Modgen is invoked from a schematic cellView, it is displayed in the Modgen Previewer window. Changes made to the Modgen are saved in the constraint view for that schematic cellView. Later, you can transfer these changes from the schematic to the layout cellView.

Here is a quick look at the default workspace:



The following toolbars are specific to Modgen:

- Modgen Placement Toolbar:



- Modgen Routing Toolbar:

As with all other toolbars, you can enable or disable the Modgen toolbars by right-clicking anywhere in the main toolbar area and selecting the required toolbar name. You can use the handle on the left hand side of the toolbar to reposition it anywhere within the Modgen editor.

You can use the Toolbar Manager to make incremental and local changes to the Modgen toolbars. For example you can add, edit, or remove toolbar items. For more information about Toolbar Manager, see Using Toolbar Manager in *Virtuoso Design Environment User Guide*.

Important

You can also prevent Modgen from unauthorized edits. If a user edits a Modgen by moving instances/body contacts or stretching wires, then the edit is essentially ignored. Therefore, when the Modgen is updated as a result of such edits, any moved/stretched objects will be reset to where they should be according to the constraint.

Note: This is not done by invoking an undo command under the covers. However, it is done by simply re-doing the Modgen layout according to the information in the constraint.

- If geometry is added or deleted from the Modgen, then the Modgen constraint will become disabled. This means that it is no longer a Modgen. However, it will remain a figGroup of type `none`.
- If the geometry is accidentally added or deleted, it can be removed or re-added by undoing the operations (pressing the 'u' key). Invoking undo will actually remove (or re-add) the geometry and re-enable the Modgen constraint.
- If the geometry was purposely added, then the user can exit the Modgen Editor and the result will be a standard figGroup.

  Note: At this point, if users want to turn this figGroup back into a Modgen, then they will have to first edit the figGroup to remove (add) the added (deleted) geometry before attempting to re-enable the Modgen constraint.

Specifying Modgen Parameters

You can now set up the module generator parameters, including the number of rows, number of columns, the type of router, the interdigitation pattern, etc. These parameters are specified in the Module Generator panel at the bottom right.

1. In the Module Generator panel, specify the number of rows for the array in the *Nb of Rows* field.
2. In the *Nb of Cols* field, specify the number of columns you want in the array.

While other parameters are listed in the Module Generator panel, there are easier, graphical interfaces to these parameters available from the Module Generator toolbar at the top.

You can now do any of the following:

- specify an interdigitation pattern for this module and save the pattern for future use (see Setting Up the Custom Interdigitation Pattern File and Using the Modgen Pattern Editor)
- specify dummy device properties and add dummy devices (see Creating Dummy Devices)
- specify properties for body contacts and add body contacts to the module (see Adding Body Contacts)
- specify guard rings on the module (see Specifying Guard Rings)
- define topology patterns and creating pin to trunk routes (see Creating Topology Patterns and pin-to-trunk Routing)
- abut devices (see Abutting Devices)

The Module Generator commands are all invoked from the Module Generator form, pop-up menus, and the Modgen toolbar. Typical Virtuoso commands such as Edit—Delete are not supported in the Modgen Editor mode.

If you want to perform some interactive customizations, such as moving devices inside a Modgen, then you need to first disable the Modgen constraint by selecting it in the Constraint Manager and setting the Enabled parameter to False. This will turn the Modgen into a standard

database figGroup. Then you need to do an Edit in Place to customize the Modgen. Any operation, such as UCN, should not affect any custom edits in this state.

This operation should only be done if you are satisfied with the final layout and configuration of the Modgen. After making these modifications, if you re-enable the Modgen constraint by setting the Enabled parameter to True, then all modifications performed outside the Modgen environment will be lost.1

Modifying a Modgen

Once you have created a Modgen, you can modify any of its properties.

Video

For more information on modifying modgens, see the Modifying Modgens video.

To modify a module from the schematic or layout:

1. Select the **Modgen** constraint in the Constraint Manager.
2. Click the *Modgen*  icon.

   The Modgen Editor appears.

Customizing a Modgen

To customize the Modgen geometry after the Modgen has been created and edited using the Modgen editor in the Layout Editor, disable the Modgen constraint. That will change the Modgen FigGroup type to *none*, which will then allow you to utilize *Hierarchy — Edit In Place* to customize any of the Modgen geometry, or add any geometry to the FigGroup.

Any geometry that was created or customized inside the FigGroup will be discarded and replaced with the regenerated Modgen geometry. Other edits or customizations in the cellview, such as constraint addition/deletion/editing or instance parameter changes will be retained.

Quitting the Module Generator Tool

When you are finished specifying your module, you can quit Module Generator mode.

Use one of the following methods to quit the Modgen mode:

- Click the Close button in the upper-right corner of the window.
- Click the *Exit*  icon on the toolbar.
- Use the `Shift` + `B` key combination.
- Right-click to display the shortcut menu and select *Exit Modgen*.

The Module Generator now creates a new Modgen constraint. When you start Virtuoso Layout Suite and Generate from Source, this module generator constraint is transferred to the layout and respected during analog auto placement.

Use the modgenSaveOnClosePreviewWindow environment variable to control the behavior when the Modgen Previewer window is closed using the Close button.

Note: This environment variable is not applicable when the Modgen Previewer window is closed using any other method.

You can set this environment variable to one of the following values:

- `prompt`: A pop-up message is displayed requesting for confirmation whether changes to the Modgen need to be saved. Choose *Yes* to save changes, *No* to discard changes, or *Cancel* to reject the closing of the window.
- `yes`: No pop-up is displayed. Instead, all modifications are saved and the window is closed.
- `no`: No pop-up is displayed. Instead, all modifications are discarded and the window is closed.

Regenerating a Modgen

After performing certain tasks, such as moving a Modgen from one placement area to another, you may want to regenerate the Modgen. Here are the steps to regenerate a Modgen:

1. Select one or more Modgen-type figGroups in the layout canvas.
2. Right-click to display the shortcut menu.
3. Select *Regenerate Modgen*.

Corresponding SKILL API: `mgRegenerateModgen`

2

Working with Patterns

This chapter covers the following topics:

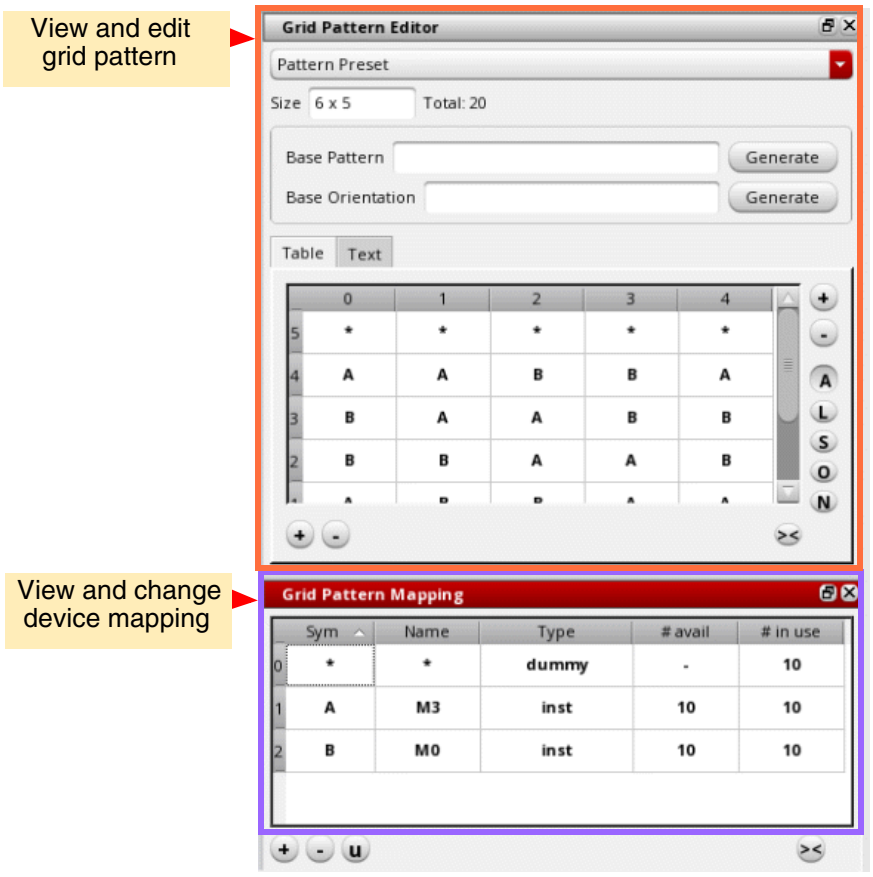
- Using the Modgen Editor Assistants
  - Grid Pattern Editor
  - Grid Pattern Mapping
  - Working with Presets
- Support for Stacks in Modgens
- Specifying Interdigitation Patterns
  - Using the Modgen Pattern Editor
  - Specifying Custom Interdigitation
  - Setting Up the Custom Interdigitation Pattern File

Using the Modgen Editor Assistants

The Modgen Editor provides a grid-based interface that can be used to define custom patterns for placing devices. The following Modgen Editor assistants are available to assist you with the placement of devices in the Modgen Layout Editing window:

- Grid Pattern Editor: View and manipulate the pattern.
- Grid Pattern Mapping: View and change the mapping of devices.

These Modgen Editor Assistants are available as dockable assistants in the Virtuoso® Layout Suite XL (VLS XL) layout editor. You can directly access these assistants from the layout canvas, without opening the Modgen Editor.



Video

For more information on how to use these assistants, see the Using the Modgen Editor Assistants video.

Grid Pattern Editor

When a Modgen is created, a table-based (grid) pattern of the selected devices is generated according to the layout. Subsequent changes made to the grid in the Pattern Editor Assistants are synchronized with the layout canvas and vice versa.

Note: (ICADVM20.1 Layout EAD Only) Modgens are automatically aligned according to the rows, created using the row-based placement utilities, that exist in the layout canvas and are applicable to the Modgen. For more information about the row infrastructure, see Creating Rows in *Virtuoso Placer User Guide*.

The Grid Pattern Editor assistant provides options to quickly specify a pattern for a Modgen layout.

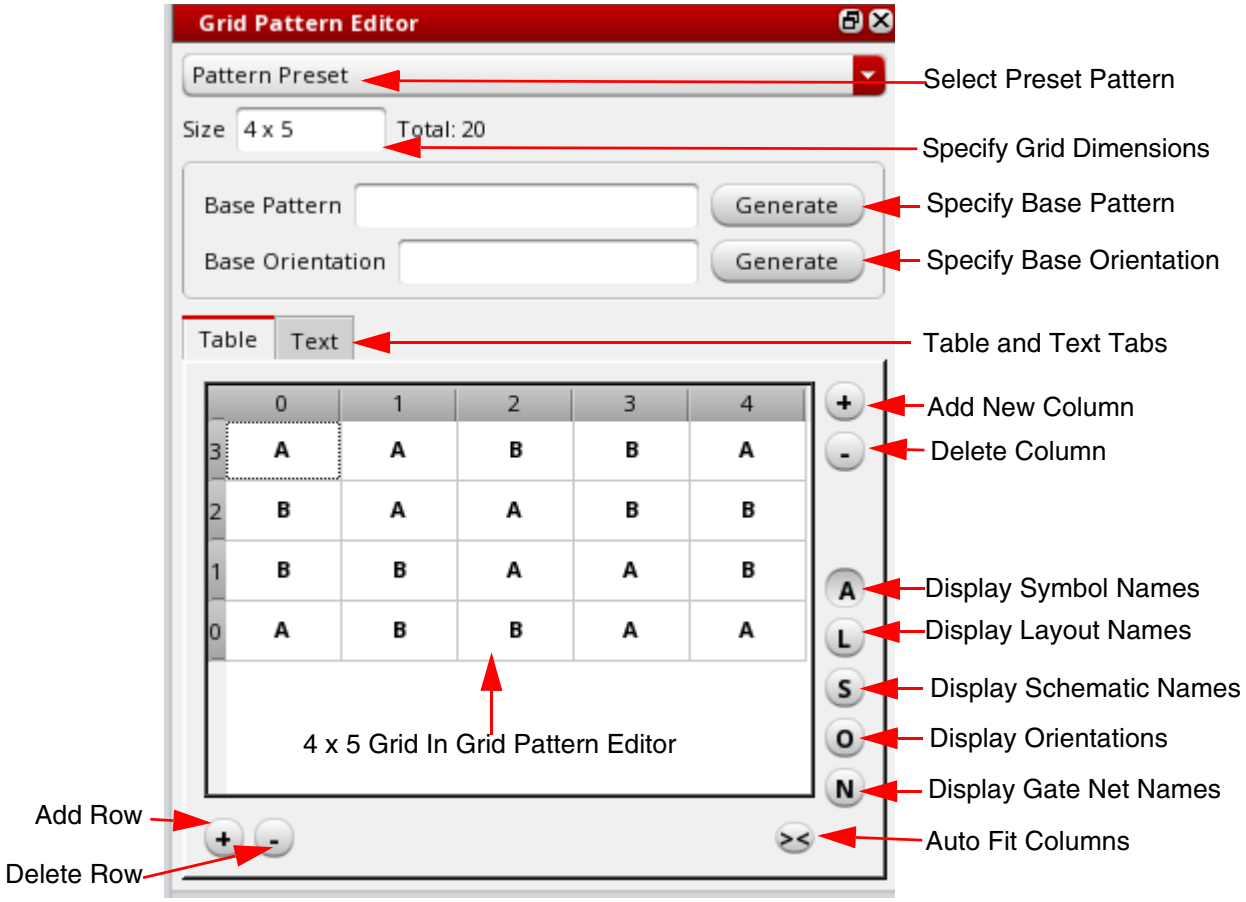
Use one of the following methods to specify a grid pattern:

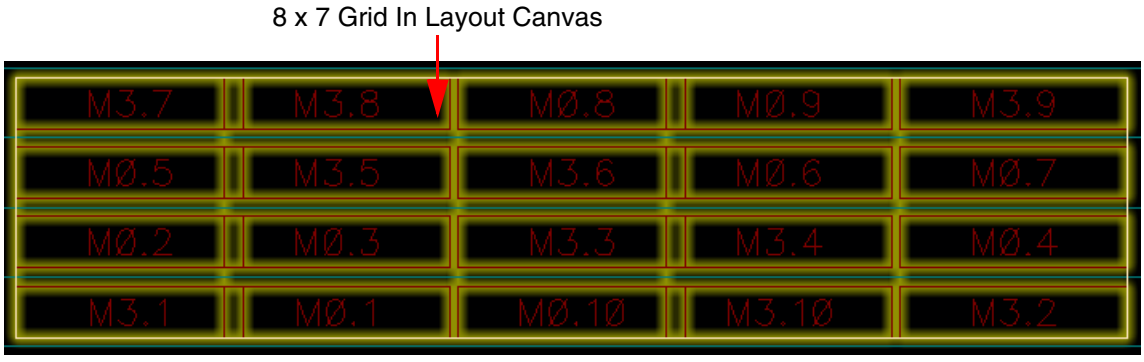
- Choose a *Preset* pattern from the drop-down list.

  For more information about preset functions, see Working with Presets.

- Specify the grid dimensions in the *Size* box and press `Enter`.

- Use the interactive functions to manually create a grid pattern. Here is a quick view of the various options in the Grid Pattern Editor:





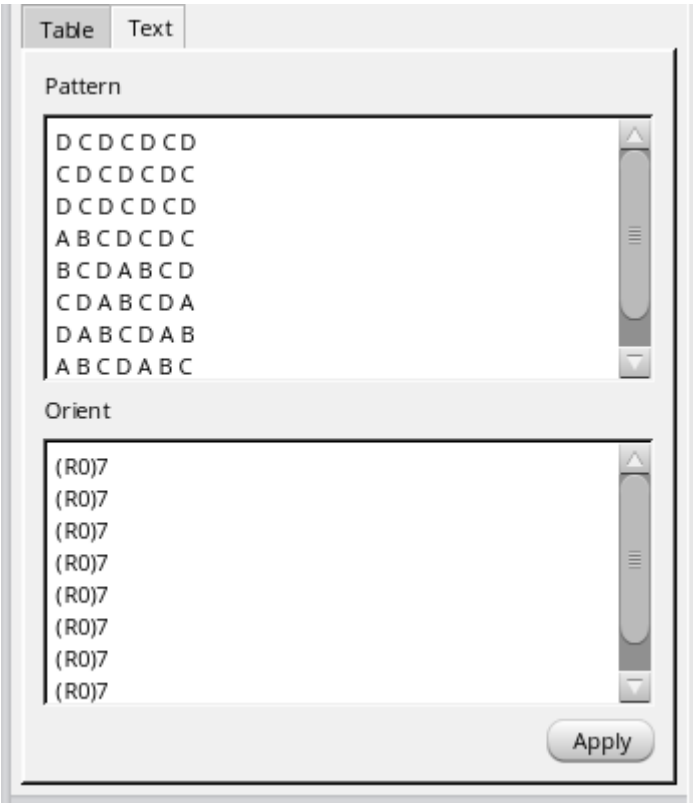
If you have altered the size of columns in the GPE assistant, you can click the >< button to auto-fit the columns.

When you select a device instance in the GPE, it is highlighted in the layout canvas.

Tip

To enable colored representation of iterated instances in the GPE, choose *Abstracted* from the *Display Mode* drop-down list in the Smart Display toolbar. This setting is applicable only to iterated instances.

- The *Text* tab lets you enter a textual pattern for the pattern. Click *Apply* to generate the pattern in the layout canvas.



Each row in the *Pattern* box corresponds to the corresponding row in the Modgen. You can use shortcuts to specify the pattern, for example AA, A2 or (A)2 — all indicate two instances of device A. *Orient* indicates the orientation of devices in each row.

Note: Multi-character symbols such as `P1` and `MN` are not supported. Each symbol must be represented by a single character, such as `A`, `a`, `B`, and `b`.

Use the Grid Pattern Editor to perform the following tasks:

- **Resize the Grid:** Enter the required grid dimensions in the *Size* field and press `Enter`.

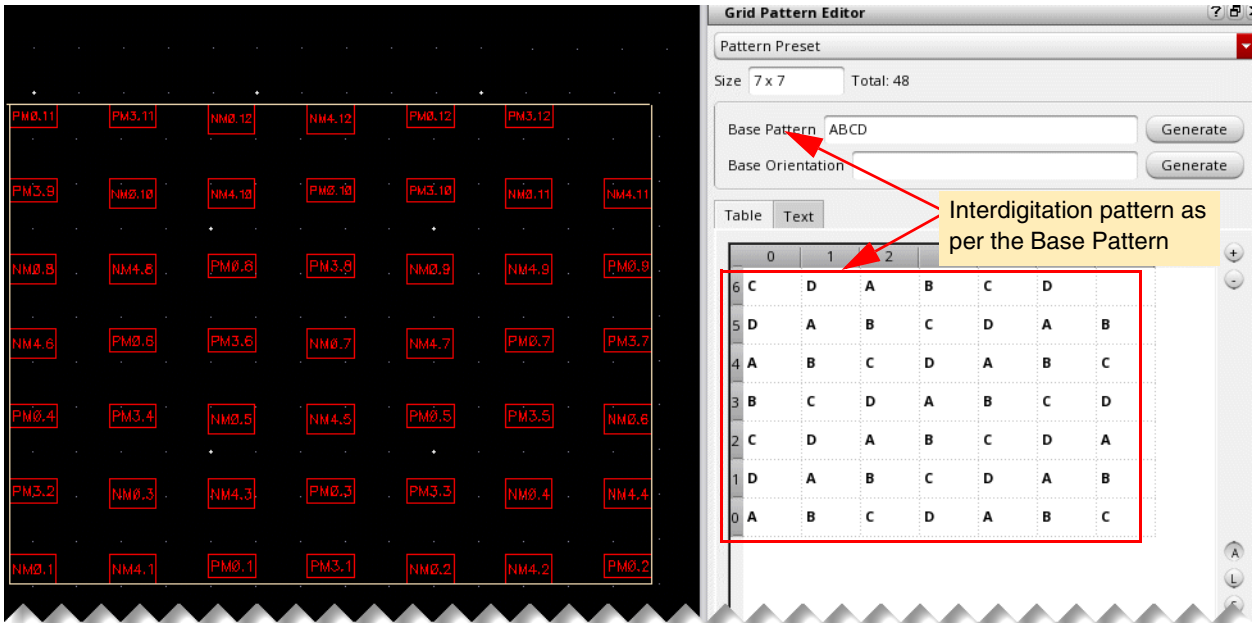
  Scenarios:

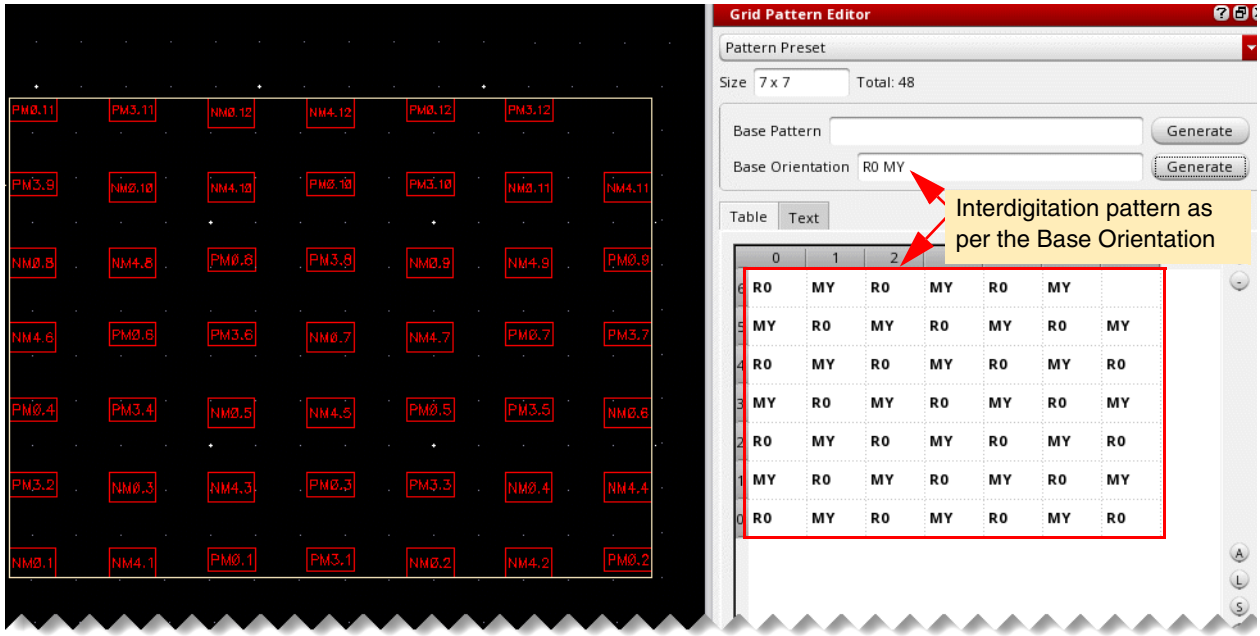
  - If you type 5 x 6 in the *Size* field, then a grid of 5 rows and 6 columns is generated.

- If you type 5 in the *Size* field, then a grid of 5 rows is generated. The engine calculates the number of columns required to accommodate the instances and creates a suitable grid.
- If you type 4r in the *Size* field, then the engine calculates the number of columns required to accommodate the instances and creates a suitable grid.
- If you type 4c in the *Size* field, then the engine calculates the number of rows required to accommodate the instances and creates a suitable grid.

Devices that are placed on the grid are automatically interdigitated.

- **Defining Base Patterns and Base Orientations:** Use the *Base Pattern* and *Base Orientation* fields to specify the interdigitation pattern for instances in the Modgen.





You can then *Generate* the resulting pattern and modify, if needed. For more information about generating base patterns, see Generating a Base Pattern.

- **Add New Rows/Columns:** Use one of the following methods:

- **Using the Add ( ) button:** There are two instances of the button:
  - Top-right corner: Adds a new column to the bottom of the grid.
  - Bottom-left corner: Adds a new row to the right of the grid.
- **Using the shortcut (context-sensitive) menu:** Use this method to add a new row above a selected row or to add a new column to the right of a selected column.

  1. Select the required row/column.

  2. Right-click and choose *Insert Row(s)* / *Insert Column(s)*.

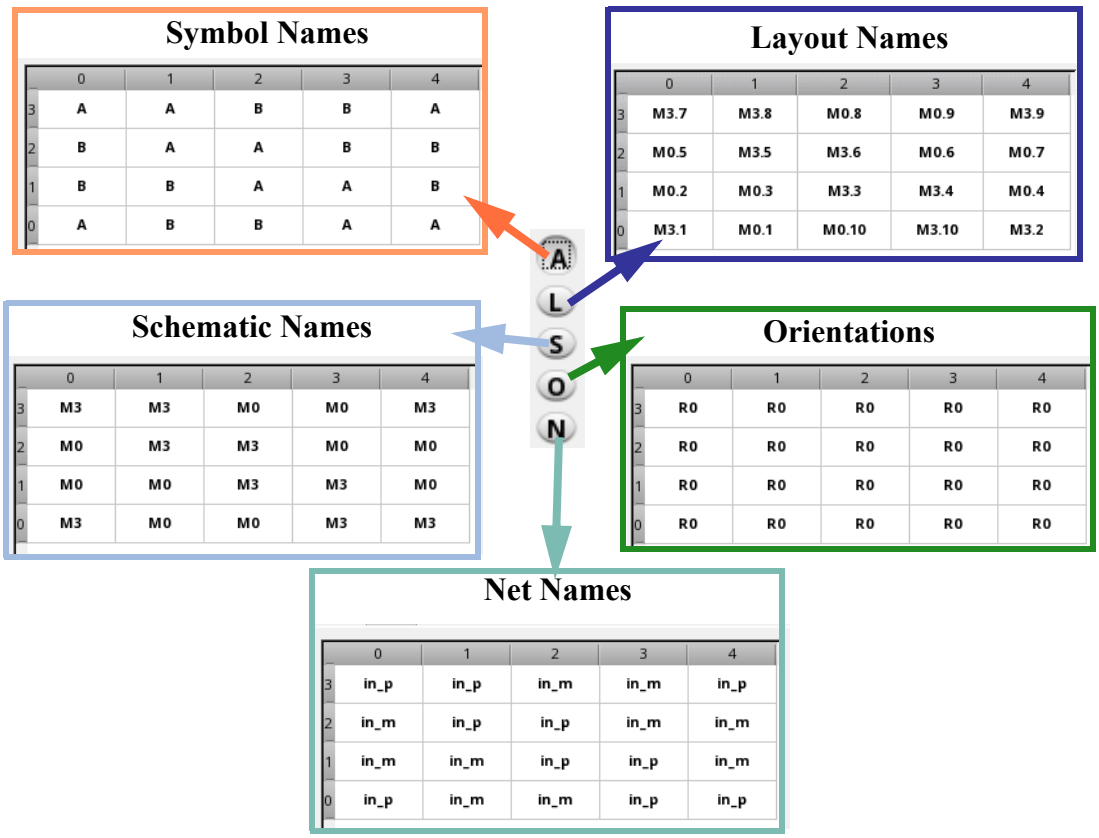
- **Delete Existing Rows/Columns:** Use one of the following methods:
  - **Using the Delete ( ) button:** There are two instances of the button:
    - Top-right corner: The bottom most column is deleted.
    - Bottom-left corner: The right most row is deleted.
  - **Using the shortcut (context-sensitive) menu:** Use this method to delete a selected row/column.

    1. Select the required row/column.

    2. Right-click and choose *Delete Row(s)* / *Delete Column(s)*.

- **Toggling between the Display Names of Instances:** By default, the cells in the Grid Pattern Editor display the symbols that are mapped to instances in the GPM assistant.

Use the following buttons to toggle between symbol names, layout names, schematic names, and orientations:



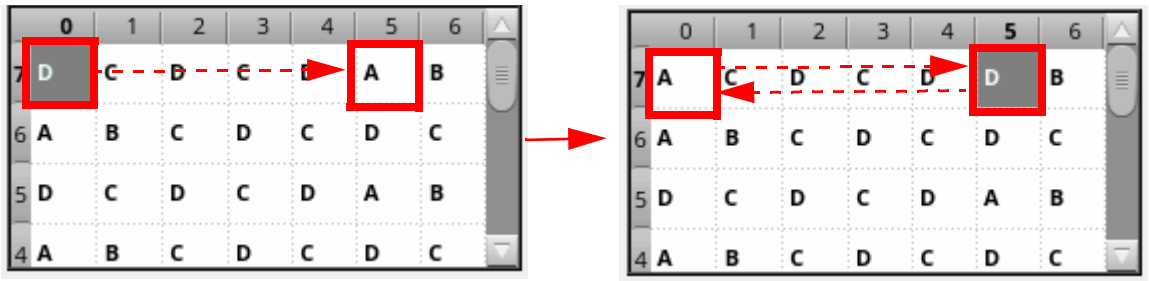
- **Moving Instances:** You can move one or more instances by dragging and dropping them to their new location either in the Grid Pattern Editor or on the layout canvas. The behavior of the move command depends on the selected mode. Use the moveAsSwap environment variable to select one of the following modes:

- **Move as Swap (Default):** This mode is switched on when moveAsSwap is set to `t`. When instances are moved in this mode, the instances in the source and target cells are swapped.

  Instances that are selected for movement in the GPE are highlighted in the layout canvas.

  Example:

  The source and target instances are swapped as shown below.



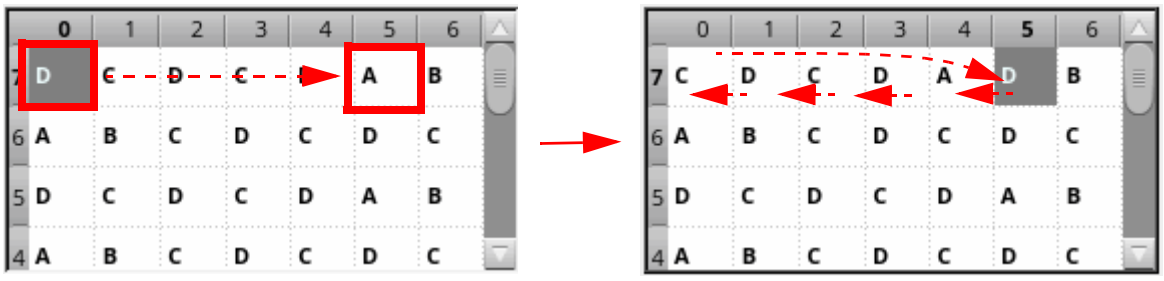
  Similar behavior is observed while moving rows and columns in this mode.

- **Move as Insert:** This mode is switched on when moveAsSwap is set to `nil`. When an instance is moved in this mode, the instance first shifts horizontally, and then vertically, until the source location is back-filled.

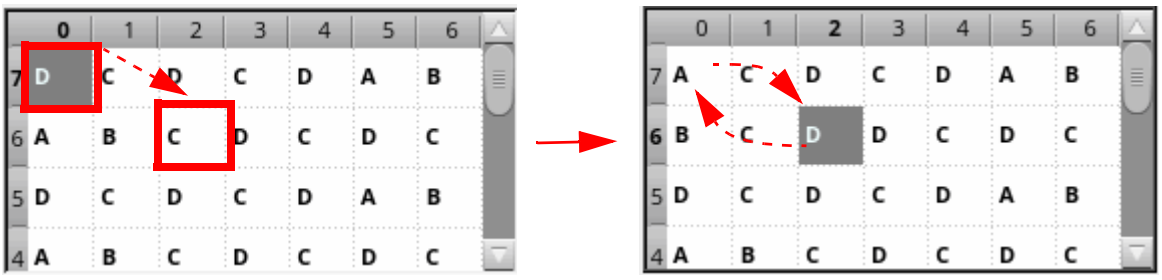
  Instances that are selected for movement in the GPE are highlighted in the layout canvas.

  Examples:

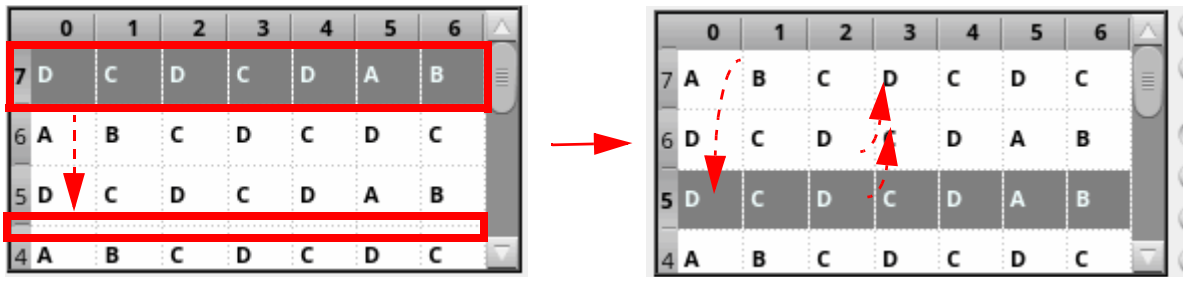
  Moving an instance horizontally: The source instance is moved to the target location. The other instances are shifted horizontally to fill the empty cell as shown below.



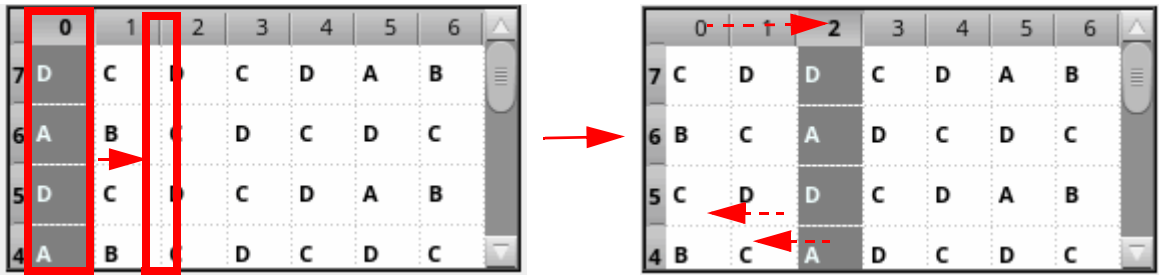
Moving an instance diagonally: The source instance is moved to the target location. The other instances are shifted horizontally and then vertically to fill the empty cell as shown below.



Moving an entire row: The instances in the source row are moved to the target location. The remaining rows are moved up one step at a time to fill the empty row as shown below.



Moving an entire column: The instances in the source column are moved to the target location. The remaining columns are moved horizontally one step at a time to fill the empty column as shown below.



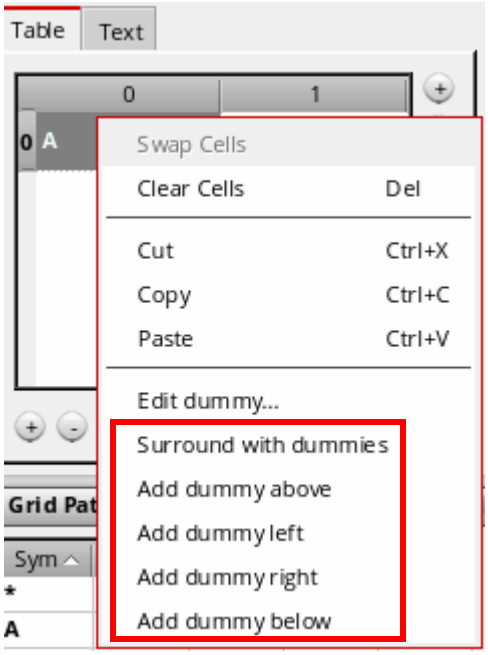
- **Copy-Pasting Instances:** You can use the `Ctrl + C` and the `Ctrl + V` key combinations to copy and paste instances inside the GPE.
- **Clear Cells:** Select the required cells (individual cells or entire rows/columns). The corresponding instances are highlighted in the layout canvas. Either press `Delete` or right-click and choose *Clear Cell(s)*. Deleted instances are highlighted in the Grid Pattern Mapping assistant.

- **Swap Instances:** You can perform the following tasks:
  - To swap two cells:
    1. Select the cells in the Grid Pattern Editor.
    2. Right-click and choose *Swap Cells*.
  - To swap two rows:
    1. Select the rows in the Grid Pattern Editor.
    2. Right-click and choose *Swap Rows*.
  - To swap two columns:
    1. Select the columns in the Grid Pattern Editor.
    2. Right-click and choose *Swap Columns*.

Tip

Use `Ctrl+`click to select two non-adjacent rows/columns.

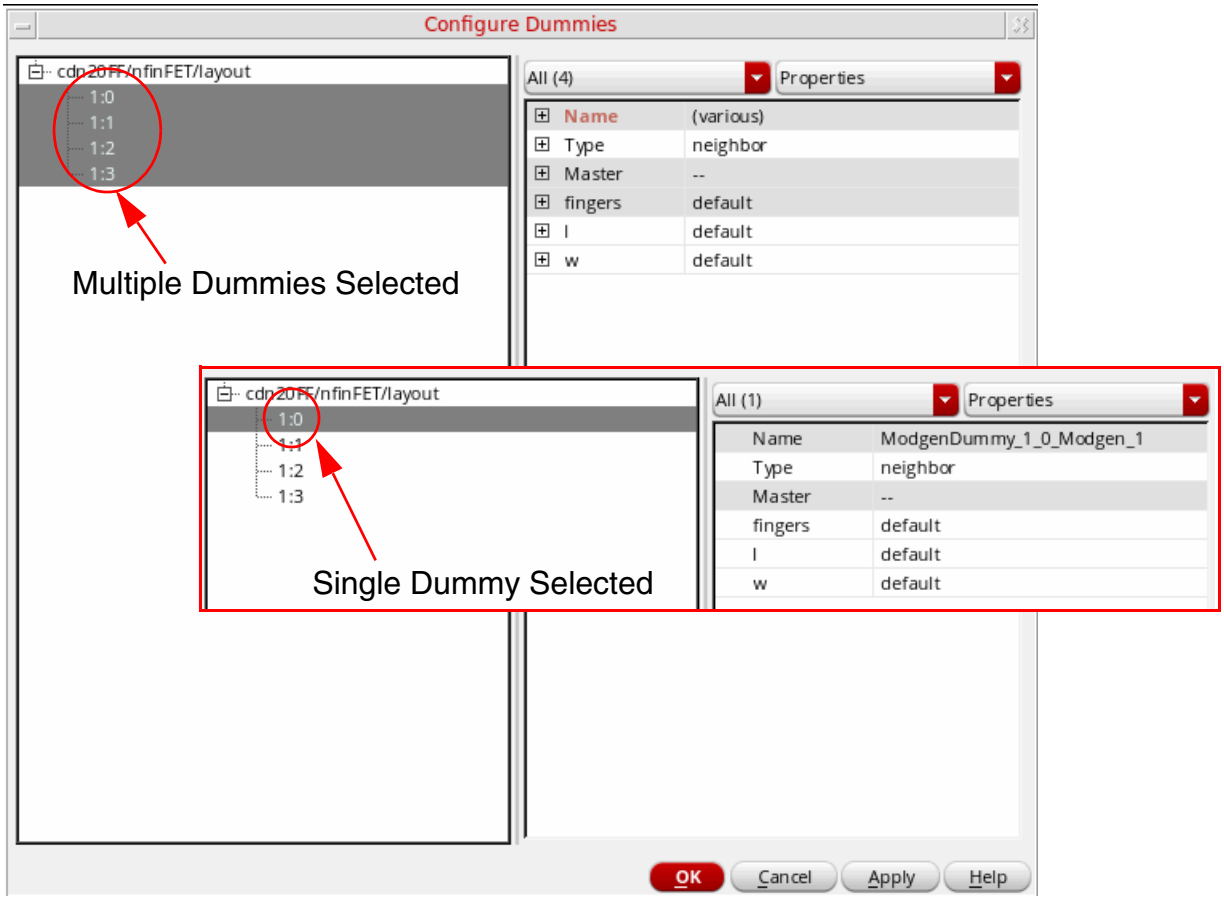
- **Add Dummies:** To add dummies around instances, select the required instances from the Grid Pattern Editor, right-click, and select a direction.



The dummies created using this method are backward-compatible with the Modgen dummies creation options. For more information about these options, see Adding Dummy Device Rows or Columns.

- **Edit Dummies:** You can use either the Property Editor assistant or the Configure Dummies form to edit the properties of the existing dummies.

  To display the Configure Dummies form, select the dummies in the Grid Pattern Editor, right-click, and select *Edit Dummy*. The Configure Dummies form is displayed.



The Configure Dummies form comprises two panes:

- The left pane—lists the dummies that are selected in the layout canvas.
- The right pane—displays the *Properties* and *Connectivity* information of the dummies selected in the left pane. Use the drop-down list at the top-left side to choose whether the *Properties*, *Connectivity*, or both need to be displayed.

You can either select individual dummies or multiple dummies in the left pane and edit their following *Properties* in the right pane:

Name: Name of the dummy. If multiple dummies are selected, then *(various)* is displayed.

Type: Type of dummy. Valid values are:

copy: Uses the dummy parameters and default values of the source instances.

lcv: You can specify the *Master* `lib:cell:view` of the dummies.

neighbor: Considers the `lib:cell:view` of the neighboring device as the master.

Master: Defines the master `lib:cell:view` when the *Type* is set to *lcv*.

Fingers: Number of fingers. Valid values are *default* from the Pcell, *match_src* from the reference dummy, and the numeric value found from *match_src*.

l: Length of the dummies. Valid values are *default* from the Pcell, *match_src* from the reference dummy, and the numeric value found from *match_src*.

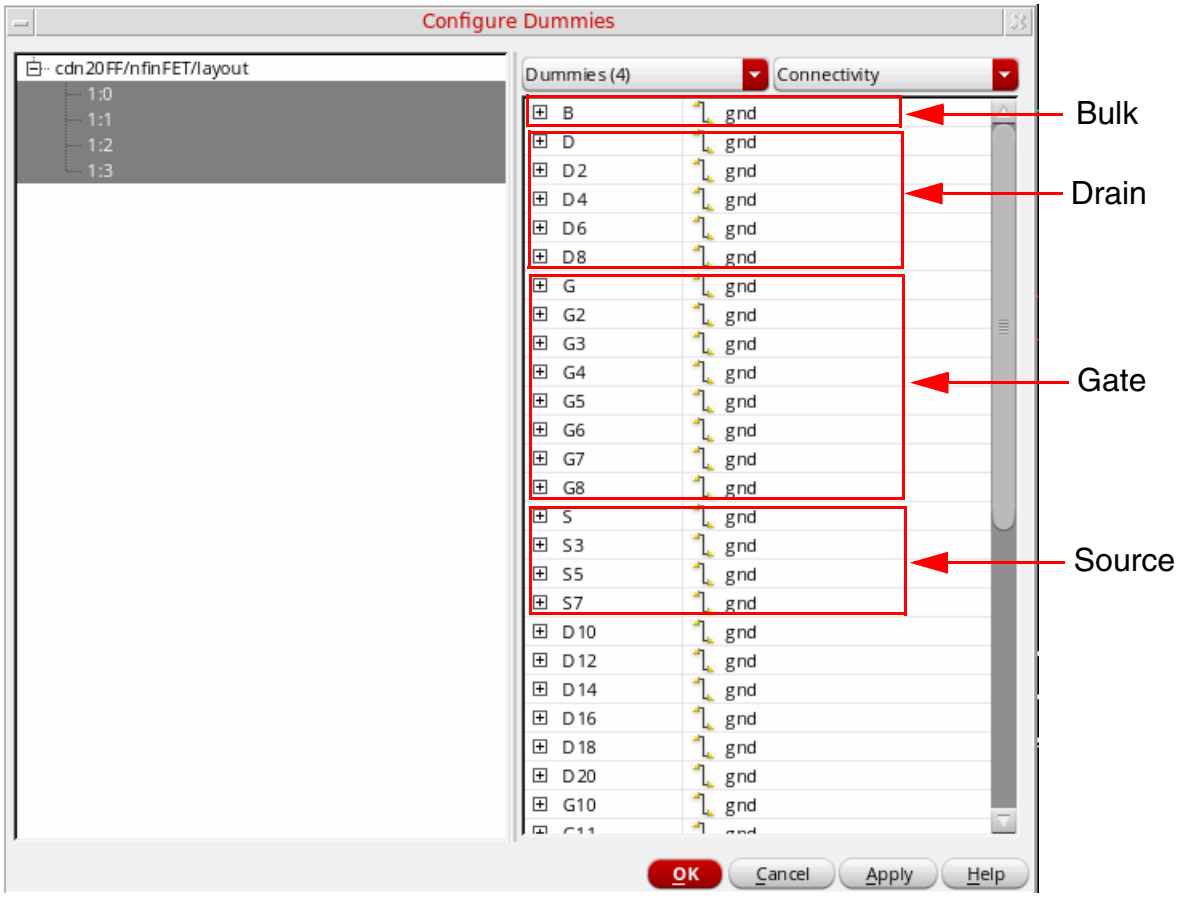
w: Width of the dummies. Valid values are *default* from the Pcell, *match_src* from the reference dummy, and the numeric value found from *match_src*.

Note: For all above options, when multiple dummies of different values are selected, then value *(various)* is displayed.

The default *Type* is *neighbor*. Therefore the dummy device parameters length, width, and number of fingers are set as per their neighboring instances or dummies. However, if there are no neighboring instances or dummies, the following environment variables are used to determine the default values:

- modgenDummyWidthOptions
- modgenDummyLengthValue
- modgenDummyWidthOptions
- modgenDummyWidthValue
- modgenDummyNumFingersOptions
- modgenDummyNumFingersValue

Select *Connectivity* to display the list of connections for Bulk, Drain, Gate, and Source as shown in the figure below. You can edit the net names associated with a terminal by typing the new net name or by choosing another net from the drop-down list.



Working with Presets

The Grid Pattern Editor assistant provides a set of preset formats, or predefined patters, that can be edited, loaded, and saved. The *Preset* drop-down list in the Grid Pattern Editor lists the following preset formats:

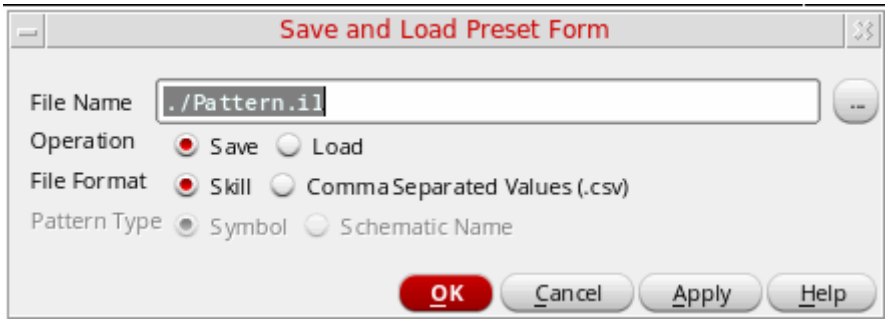
- **Pattern Preset**
  - **Save and Load:** Lets you save a grid pattern to a file and load a preset pattern from a file. For more information, see Saving and Loading Presets.
  - **Common Centroid:** Places devices in a grid, while following a common centroid pattern. For more information, see Creating Patterns with a Common Centroid.
  - **Flip Horizontal** and **Flip Vertical:** Flips the selected instances horizontally or vertically, as per the selection. For more information, see Flipping Devices.
  - **Interdigitate:** Specifies an interdigitation pattern for devices in a grid pattern. For more information, see Interdigitating Presets.
- **Resistor Preset**
  - **Interdigitate Resistor by Instance:** Specifies the interdigitation pattern for devices that are placed in the grid. For more information, see Interdigitating Resistors By Instance Preset.
  - **Interdigitate Resistor by Row/Col:** Places similar instances in the same row or column. For more information, see Interdigitating Resistors By Rows and Columns.
  - **Create Resistor Pattern:** Define the resistor pattern that must be followed while placing devices in the grid. To edit the resistor preset, select Edit Resistor Pattern. The Edit Resistor Topology form is displayed. For more information, see Creating and Editing Resistor Pattern Presets.
  - **Edit Resistor Pattern:** Lets you edit resistor patterns for presets. For more information, see Creating and Editing Resistor Pattern Presets.
  - **Quick Reorient Resistor Pattern:** Flips the orientations of the members of a preselected Modgen, either in the design or in the Constraint Manager, to reduce the overall connectivity wire length. The placement of the members is not altered. Corresponding SKILL function: gpePresetReorientResistor
  - **Reorient Resistor Pattern:** Changes the orientation of the first instance in the design. For more information, see Reorienting Resistors.

Note: The default preset values are derived from the active Modgen.

You can also define the preset functions that must be displayed in the *Preset* drop-down list. For more information, see Using the Preset Generator Functions.

Saving and Loading Presets

Select *Save and Load* from the *Pattern Preset* drop-down list to display the Save and Load Preset Form.



Choose to either *Save* the current grid pattern to the specified *File Name,* or *Load* a preset pattern from the specified *File Name*. Supported file formats are *SKILL* and *Comma Separated Values (.csv)*.

If you choose to *Save* in the comma-separated values (CSV) format, you can specify a *Pattern Type*:

- *Symbol:* Prints the letter that is mapped to that instance in the Modgen Pattern Map.
- *Schematic Name*: Prints instance names from schematic.

Understanding the CSV Format

A CSV file allows data to be saved in a tabular format. As the name suggests, a CSV file comprises character strings that are separated by commas.

Important

Do not use white spaces to separate symbols or between commas because white spaces within lines are ignored in the CSV file format. However, you can use white space between lines can to delimit sections.

There are certain rules to be followed while defining grid patterns in CSV format. In a CSV file:

- The first section represents the placement of devices, followed by one or more blank lines. The placement of devices can be represented either by a list of schematic names or by a list of symbols that are mapped in the third section of the file.

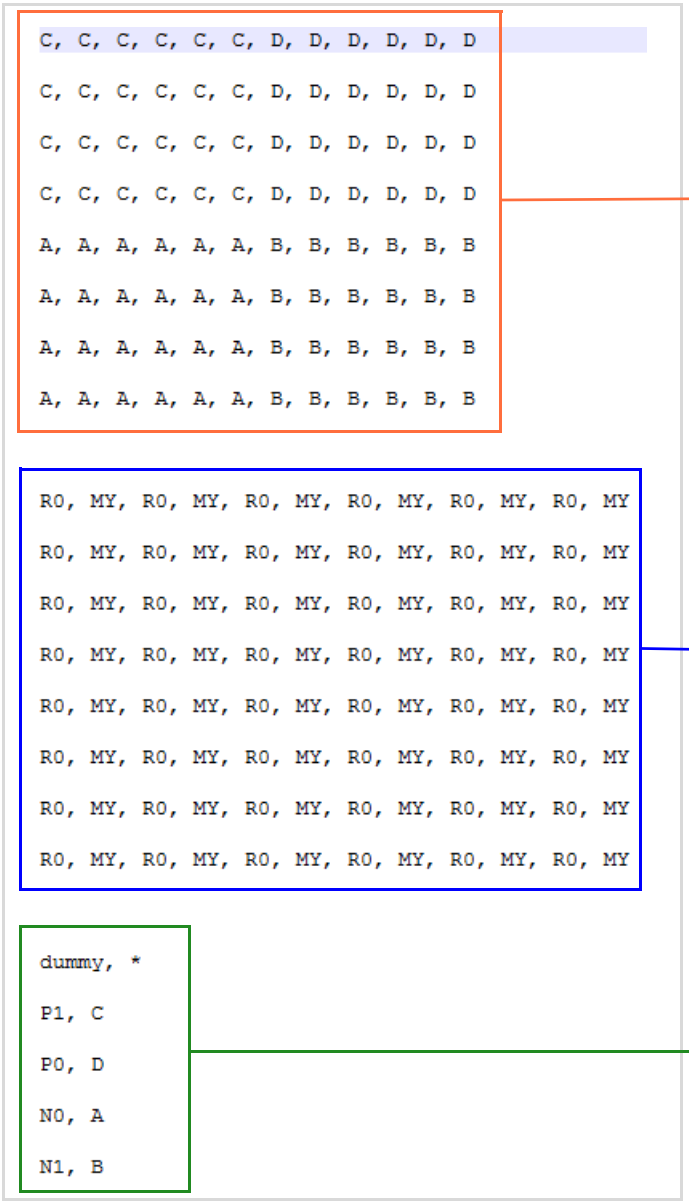
  Note: An unoccupied grid entry is represented by the `"-"` character.

- The second section represents the device orientations, followed by one or more blank lines. If orientations are not specified, then the default orientation is `"R0"`.

- The third section represents mapping. The mapping format is `"schematic name, symbol"`. For dummy devices, use the keyword `"dummy"` as the schematic name. If the mapping section is not included, then the default is the current GPE mapping.

Here is a sample CSV file that contains the three sections:

The CSV File Sample



First Section:

- Represents the placement of devices
- Each row is followed by a blank line
- Lists symbol names, as mapped in the third section

Second Section

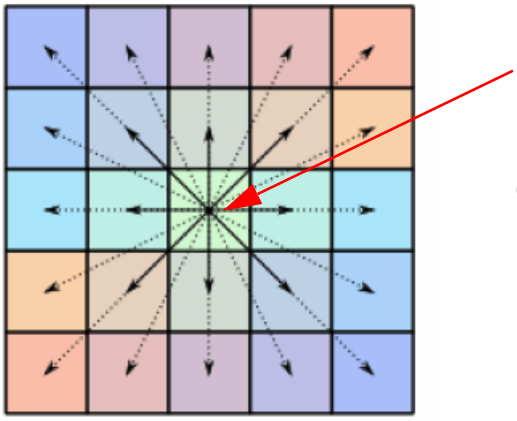
- The keyword "`dummy`" represents the device orientations
- Each row is followed by a blank line

Third Section

- Represents a dummy device
- Specifies the mapping of devices

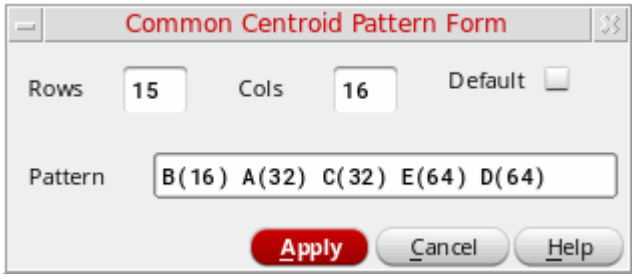
Creating Patterns with a Common Centroid

The common centroid preset lets you place devices in a grid, while following a common centroid pattern. The following diagram depicts how devices are placed:



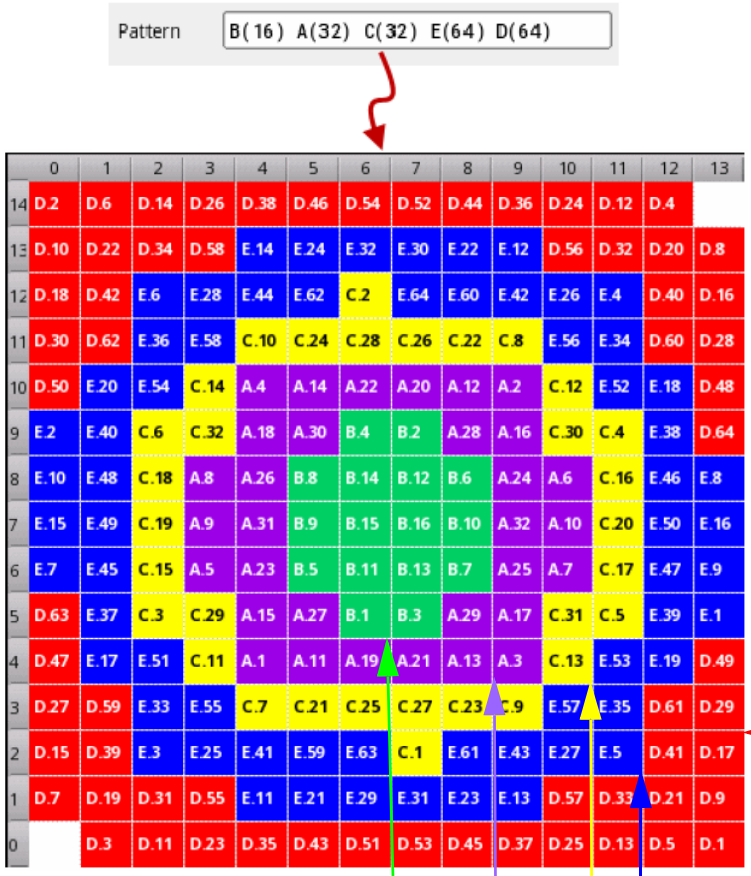
Use the options in the Common Centroid Pattern Form to define a centroid pattern that must be followed while placing devices on the grid.

1. Select the devices to which the common centroid pattern needs to be applied. If no devices are selected, then the pattern is applied to all the devices in the Grid Pattern Editor.
2. Select *Common Centroid Pattern* from the *Preset* drop-down list to display the Common Centroid Pattern Form.



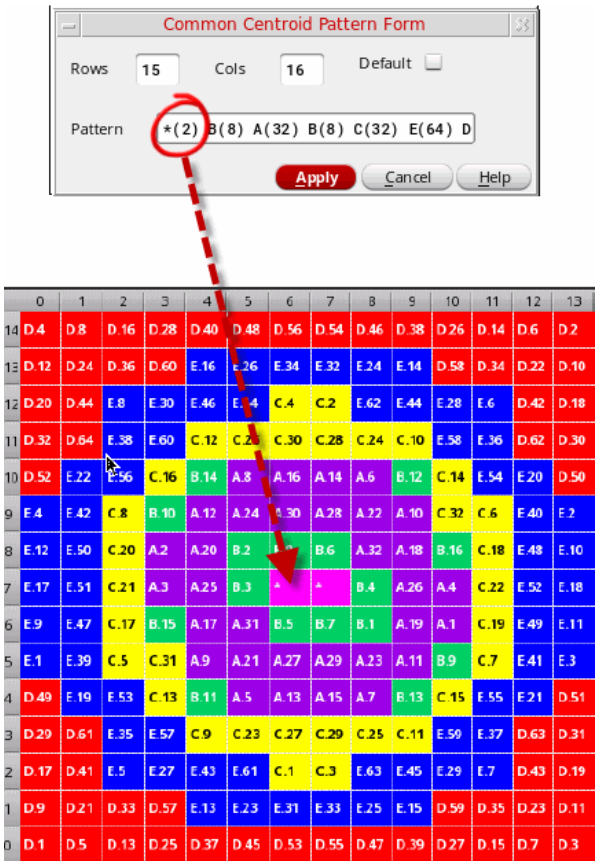
3. You can either specify the number of *Rows* and Columns (*Cols*) to be generated, or select *Default* to automatically calculate the optimum value.
4. Specify the common centroid pattern in the *Pattern* field, which determines the order in which instances are placed in the common centroid pattern.
5. Click *Apply*.

The following diagram depicts the common centroid pattern for the above specification, which is the default pattern:

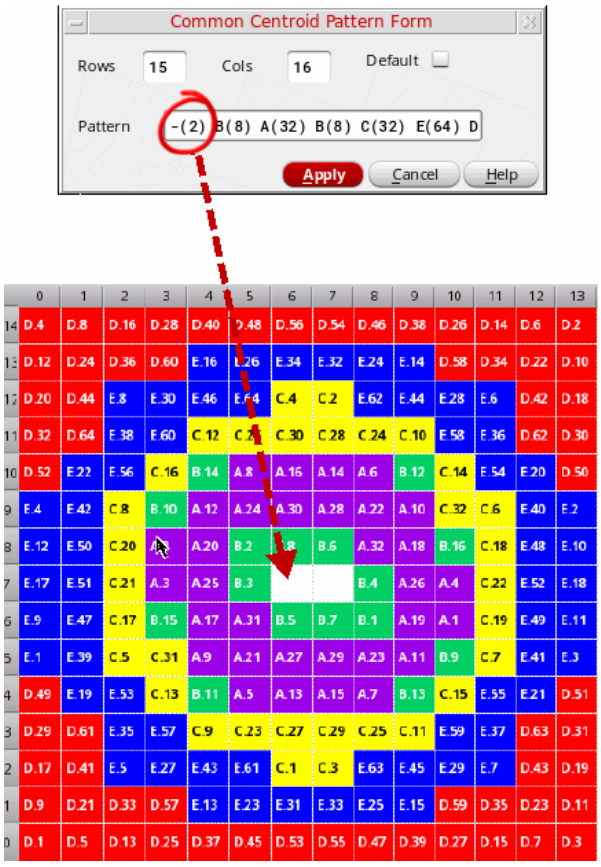


1. 16 instances of device B are placed at the center.
2. 32 instances of device A are placed around the above.
3. 32 instances of device C are then placed around instances of device A.
4. 64 instances of device E are placed around instances of device C.
5. 64 instances of device D are finally placed around the above instances.

Dummy devices and blank spaces can also be included in the common centroid pattern. Here are examples of how dummy devices and blank spaces can be inserted:



Dummy devices are indicated by **'*'**



Blank spaces are indicated by **'-'**

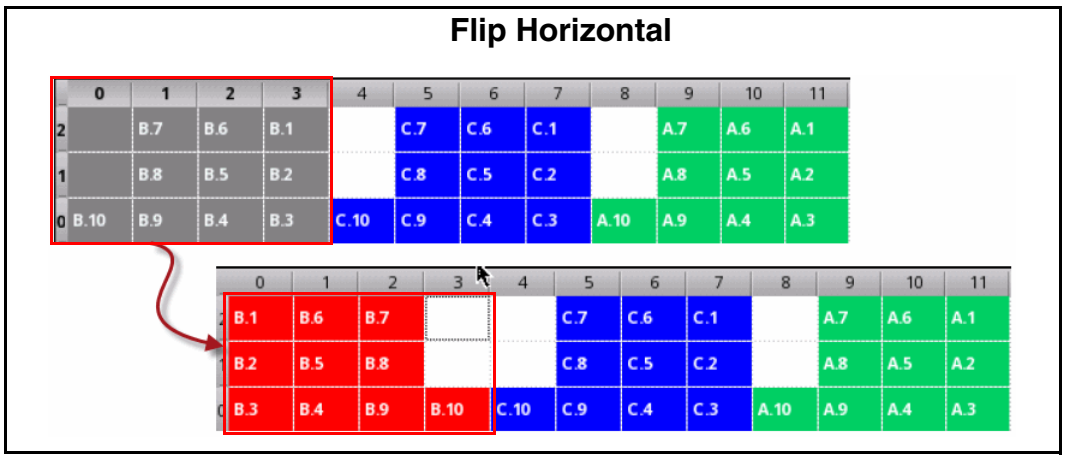
Note: In the above pattern, a slightly different pattern is followed - All 16 instances of device B are not placed together. Instead, 8 instances of device B are first placed around the dummy and blank devices, followed by instances of device A. The remaining 8 instances of device B are then placed.

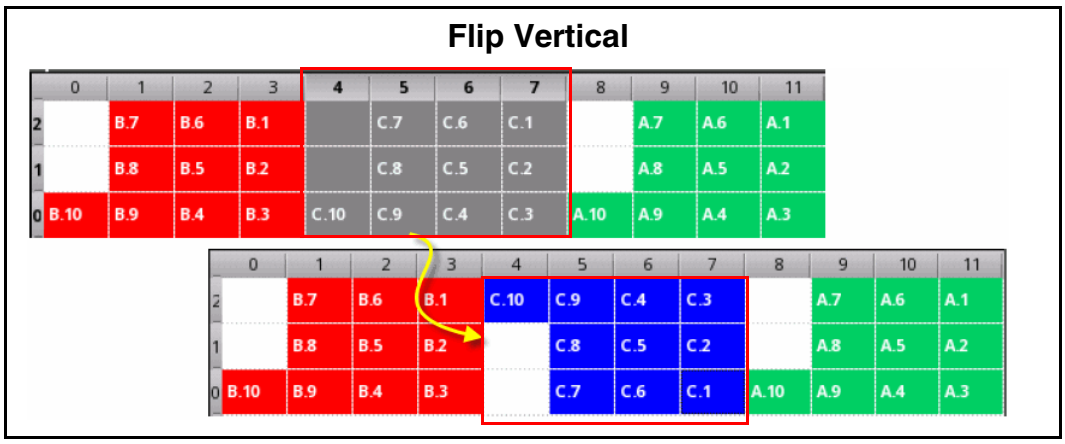
Flipping Devices

You can flip the selected devices either horizontally or vertically. To flip devices:

1. Select the required devices in the Grid Pattern Editor.
2. Select *Flip Horizontal* or *Flip Vertical* from the *Presets* drop-down list. The selected instances are flipped accordingly.

The following diagrams depict how devices are flipped horizontally and vertically:



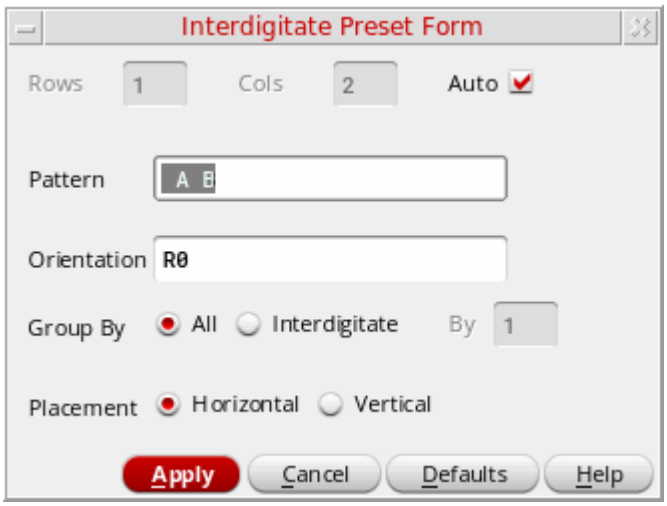


Interdigitating Presets

The Interdigitate Preset form lets you specify an interdigitation pattern for devices in a grid pattern.

1. Select the required instances. If no instances are selected, then the settings are applied to all the instances on the grid.

2. Click the *Preset* drop-down list and choose *Interdigitate Preset* to display the Interdigitate Preset Form.



Note: You can either directly *Apply* the defaults or modify the values described below before generating the interdigitation pattern.

3. Specify the number of *Rows* and *Columns* in the grid, or choose *Auto* to automatically determine these values.
4. Specify the required interdigitation *Pattern*.
5. Specify the default *Orientation* of devices in the grid pattern.
6. Indicate the grouping behavior - all devices or only the interdigitated devices. Also specify the number to be used for interdigitation in the *By* field.
7. Specify the direction of *Placement* of devices - *Horizontal* or *Vertical*.
8. Click *Apply*.

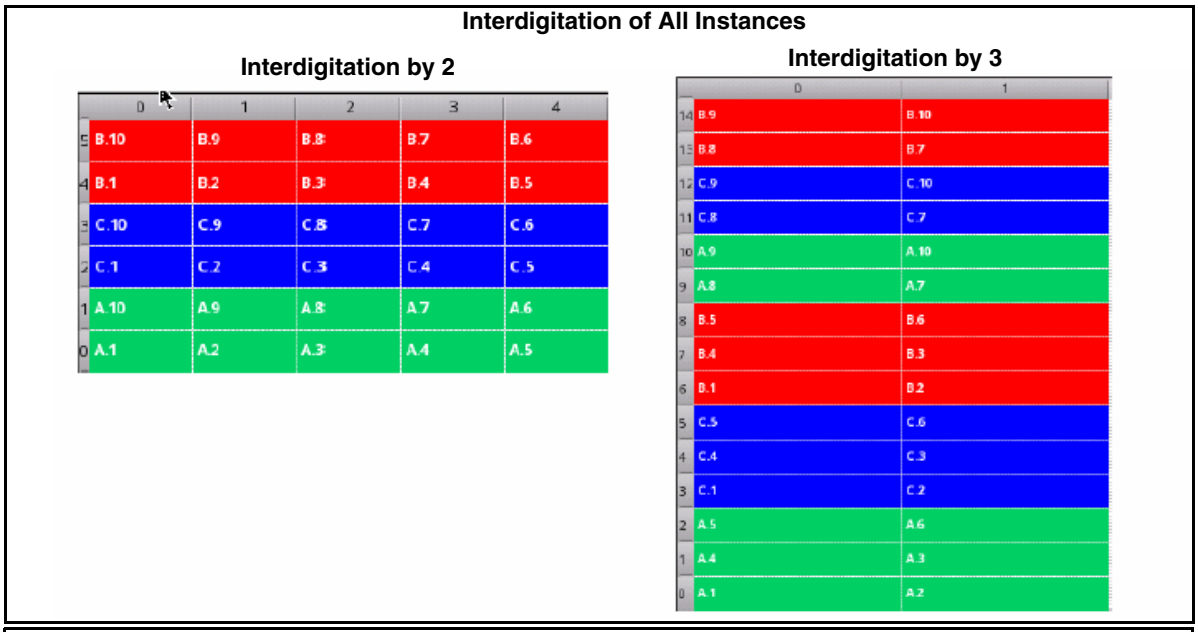
Interdigitating Resistors By Instance Preset

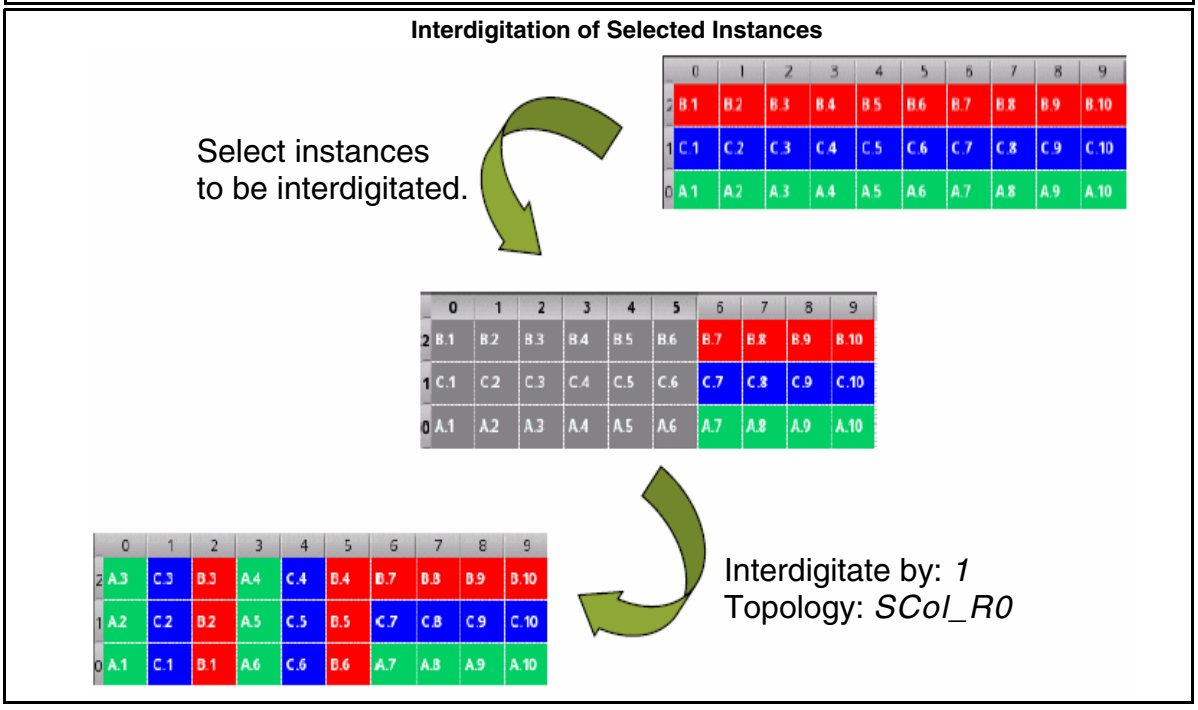
Use the Interdigitate Resistor by Instance Preset form to specify the interdigitation pattern for devices that are placed in the grid. By default, instances are interdigitated by `1`. To interdigitate instances in a grid:

1. Select the required instances. If no instances are selected, then the settings are applied to all the instances on the grid.
2. Select *Preset* - *Interdigitate* to display the Interdigitate Resistor By Instance Preset Form.

3. In the *Interdigitate by* field, specify the integer value based on which instances need to be interdigitated. The default is `1`. If you set the value to `0`, then instances are not interdigitated. You can set the value to a higher number.
4. Edit the custom interdigitation *Pattern*. Click *Update Pattern* to apply the interdigitation settings to the active design.
5. Select the required pattern *Topology*, which specifies the direction (rows or columns) and orientation of devices.
6. Click *Apply*.

Here are examples of interdigitation in some designs:

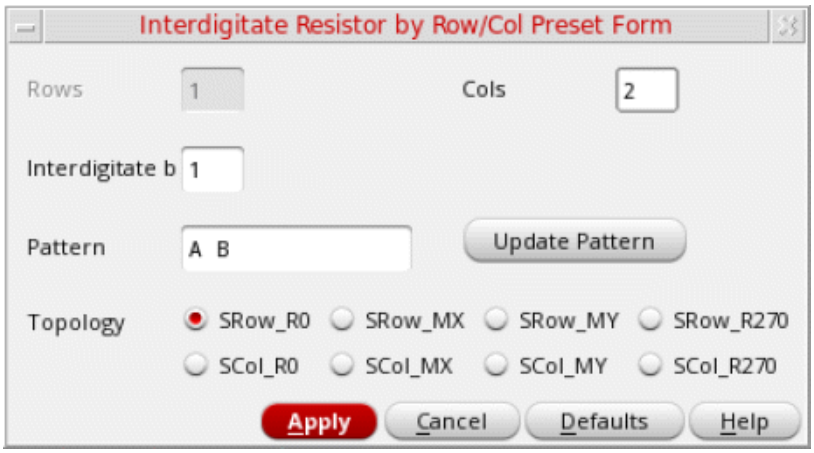




Interdigitating Resistors By Rows and Columns

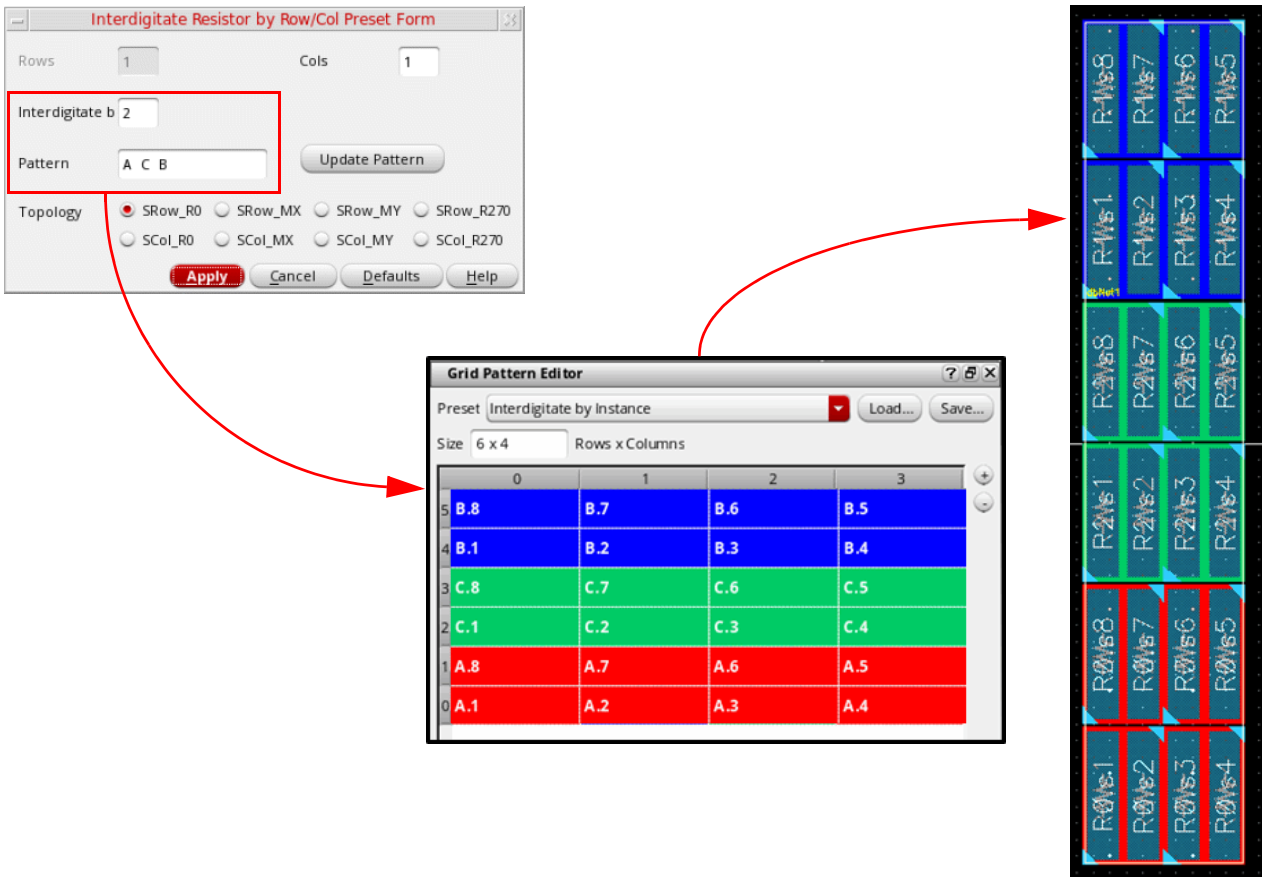
Use the Interdigitate Resistor by Row/Col Preset form to place like instances in the same row or column. To do this:

1. Select the required instances. If no instance is selected, the settings are applied to all the instances in the grid.
2. Select *Preset* - *Interdigitate Resistor by Row/Col* to display the Interdigitate Resistor by Row/Col Preset form.



   Depending on the selected *Topology*, either *Rows* or *Cols* (columns) can be specified. If any of the *SRow_** values are selected, the *Rows* value is fixed and *Cols* can be specified, and vice versa.
3. The default *Interdigitate by* value is 1. You can specify a different value, if required.
4. Specify the required *Pattern* and click *Update Pattern*.
5. Select the required pattern *Topology* to specify the direction and orientation of devices.

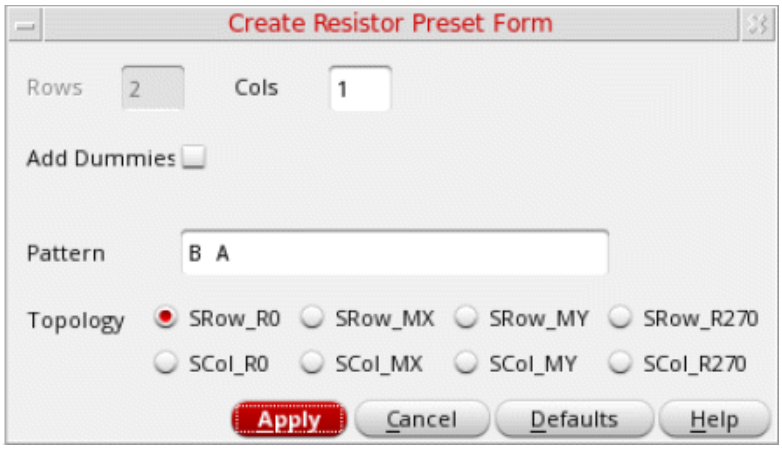
6. Click *Apply* to view results in the Grid Pattern Editor and in the layout canvas.



Creating and Editing Resistor Pattern Presets

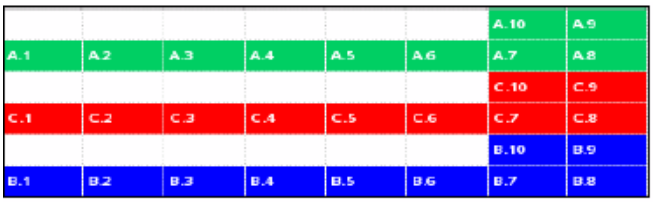
Use the options in the Create Resistor Preset Form to define the resistor pattern that needs to be followed while placing devices on the grid. To define a resistor pattern:

1. Select *Create Resistor Pattern* from the *Presets* drop-down list to display the Create Resistor Preset Form.

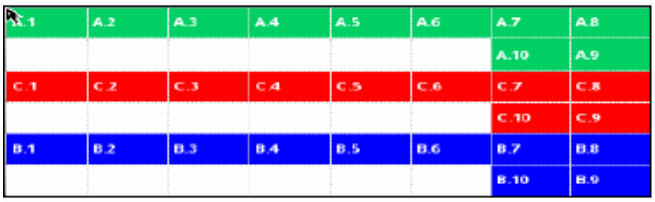


2. Specify the number of *Rows* and Columns (*Cols*) to be generated.
3. (Optional) Use *Add Dummies* to specify whether blank spaces in the grid must be left blank (default) or filled with identical dummy devices.
4. Specify the resistor pattern in the *Pattern* field.

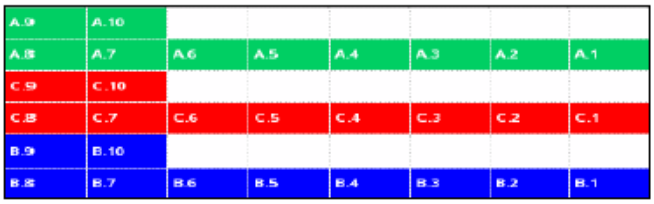
5. Select the required pattern *Topology*. When an *SRow_** value (default) is selected, mfactor and s-factor instances are placed in rows, and instances are placed as per their numbering along the columns, as shown below:



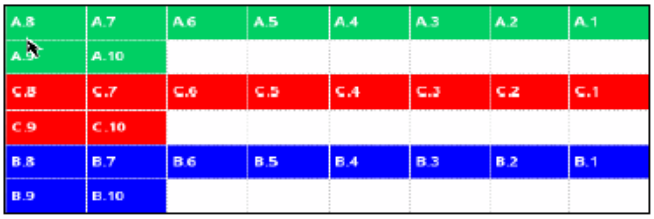
SRow_R0



SRow_MX



SRow_MY



SRow_270

When an *SCol_** value is selected, instances are placed as per their numbering along the rows, and the mfactor and s-factor instances are placed in columns, as shown below:

SCol_R0

SCol_MX

SCol_MY

SCol_R2

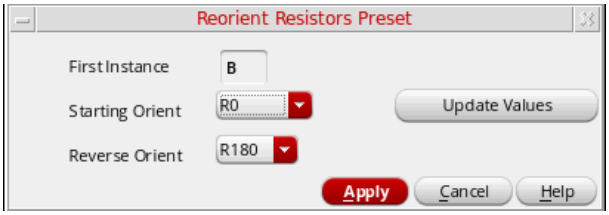
6. Click *Apply*.

Select *Edit Resistor Preset* from the *Preset* drop-down list in the Grid Pattern Editor to display the Edit Resistor Topology Form. Select a different topology to be applied to the pattern (as discussed above).

Reorienting Resistors

Use the options in the Reorient Resistor Preset form to change the orientation of the first instance in the design. The orientations of the other instances are reset accordingly.

1. Select the required instances. If no instances are selected, then all the instances in the grid are considered.
2. Select *Reorient Resistors* from the *Preset* drop-down list. The Reorient Resistors Preset form is displayed.



3. The *First Instance* is selected alphanumerically from the list of available instances.
4. *Starting Orient* indicates the orientation of the first instance. You can change the value as per your requirement. The *Reverse Orient* value is changed accordingly.
5. *Reverse Orient* can be set to either the default reverse orientation or the mirrored orientation of the instance.

Note: At any point, click *Update Values* to reset all the options to their default values.

6. Click *Apply*.

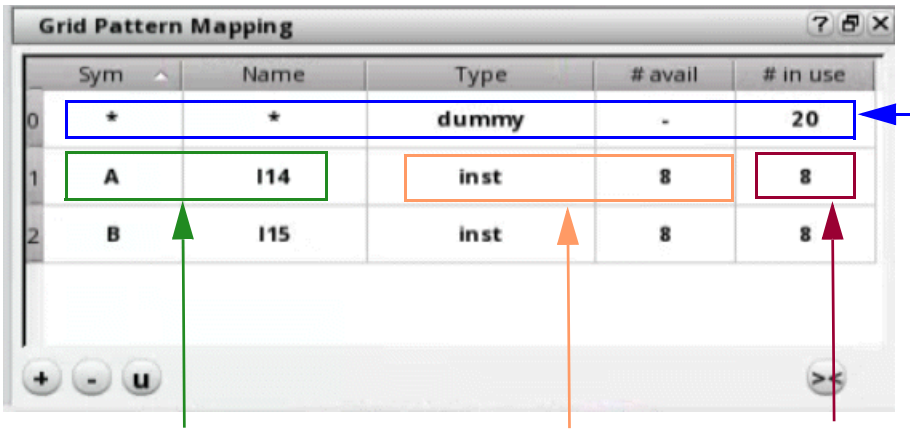
Using the Preset Generator Functions

Use the following SKILL functions to manage presets:

- gpeRegisterPresetGen: Registers preset generator functions. Once registered, these functions are displayed in the *Preset* drop-down list.
- gpeIsRegisteredPresetGen: Checks whether a preset generator function is registered.
- gpeRunPresetGen: Invokes a preset generator function on the active figGroup. The current figGroup is updated according to the preset generator logic.
- gpeIsPresetGenDisplayable: Checks whether a preset generator should be displayed in the *Preset* drop-down list of the Grid Pattern Editor.
- gpeClearPresetGenerators: Deletes all the registered preset generator functions from the system.
- gpeUnregisterPresetGen: Unregisters the given preset generator function from the Grid Pattern Editor.

Grid Pattern Mapping

Use the Grid Pattern Mapping assistant to view and change the mapping of instances and dummies. The following figure explains the various columns in the Grid Pattern Mapping assistant.



Selecting Instances of a Device: When you select a device in the Grid Pattern Mapping assistant, all instances of the device are highlighted in the layout canvas.

Sorting Devices: To sort the devices alphabetically, click the header of the column that you want to sort. Alternate clicks sort in ascending and descending order.

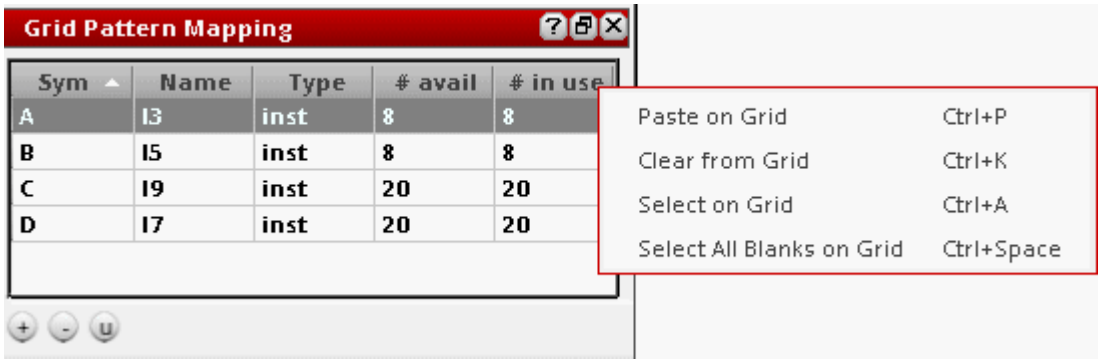
Remapping Instances: When you change the symbol of a device in the Grid Pattern Mapping assistant, all instances of the device in the Grid Pattern Editor are updated and remapped to the new symbol name.

Adding and Removing Instances from Modgens: Use the  and  buttons to add or remove selected instances from the active Modgen. You can add instances from the layout or schematic cellviews. Corresponding SKILL functions: gpeAddInstance, gpeRemoveInstance

Updating Modgen Instances: Select the  button to update the connectivity and nets on the instances in the Modgen. The device connectivity, parameters, and multipliers are refreshed from the corresponding schematic cellview.

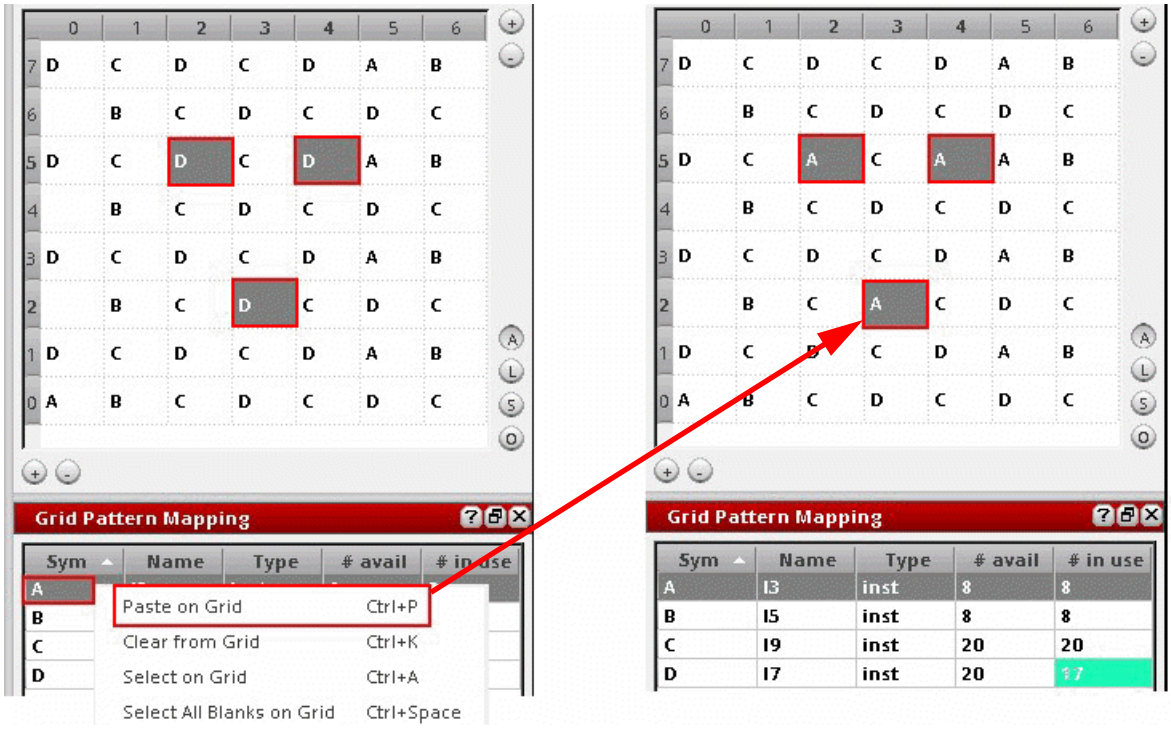
Resetting Column Width: If you have altered the size of columns in the GPM assistant, you can click the  button to auto-fit the columns.

Using the Shortcut Menu: Select a device and use the shortcut menu to perform the following tasks:



- **Paste on Grid:** Select a few devices in the Grid Pattern Editor before running this command. The selected devices are replaced by the device highlighted in the Grid Pattern Mapping assistant.

  Example: In the following example, *Paste on Grid* has replaced all the selected instances of device D in the Grid Pattern Editor with device A.



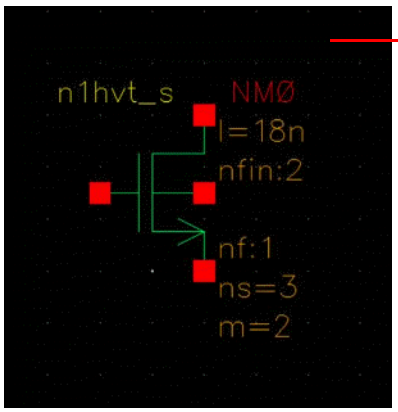
- **Clear from Grid:** All instances of the selected device in the Grid Pattern Mapping assistant are deleted from the Grid Pattern Editor.

- **Select on Grid:** All instances of the selected device in the Grid Pattern Mapping assistant are selected in the Grid Pattern Editor.
- **Select All Blanks on Grid:** All blank instances are selected in the Grid Pattern Editor. Following this command, you can select another device in the Grid Pattern Mapping assistant and choose *Paste on Grid* from the shortcut menu.

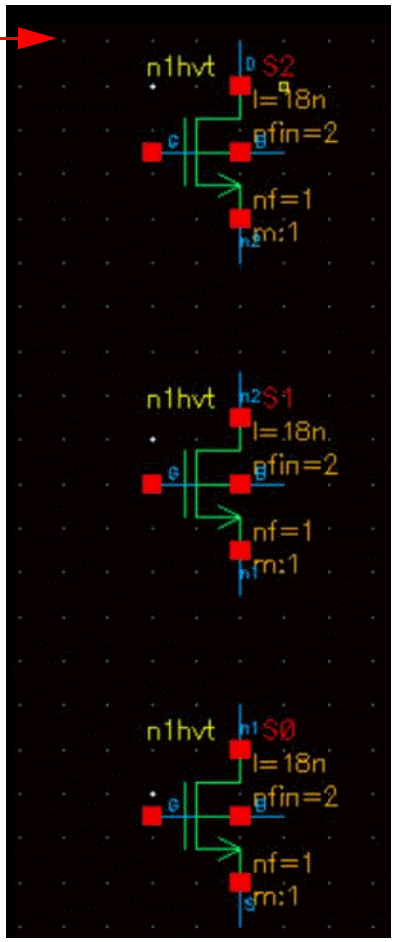
Support for Stacks in Modgens

(ICADVM20.1 Only) A stack of series-connected devices (stack) is a set of devices that are connected in sequence from the source to drain. The devices in a stack share a common gate connection, bulk connection (if the devices have a bulk terminal), and super-master. In addition, stacks have no external connections to any internal source or drain terminal. The following image depicts a stack in the schematic view.

A Stack Series Cell



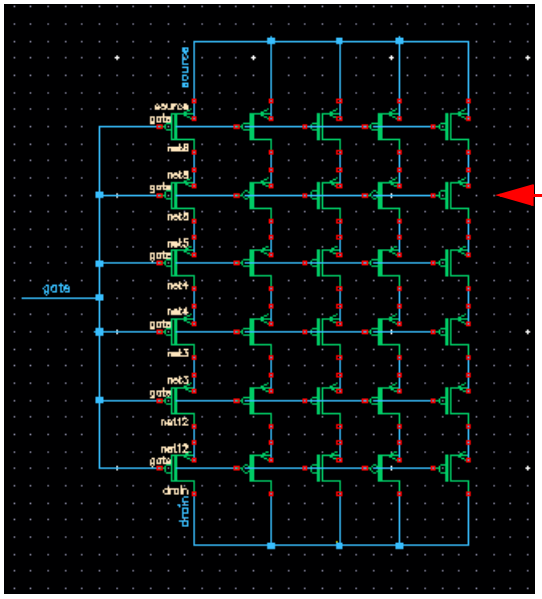
Stack of Cells Displayed when Descending the Stack



Stacking devices helps achieve circuit performance goals in advanced node PDKs, where the gate lengths for analog devices are more uniform and discrete. A properly arranged stack can be internally abutted to reduce the area of the layout.

When a new Modgen is created with the GPE or GPM assistant visible, all stacks within the Modgen are detected, rearranged according to connectivity (if necessary), and their members are abutted.

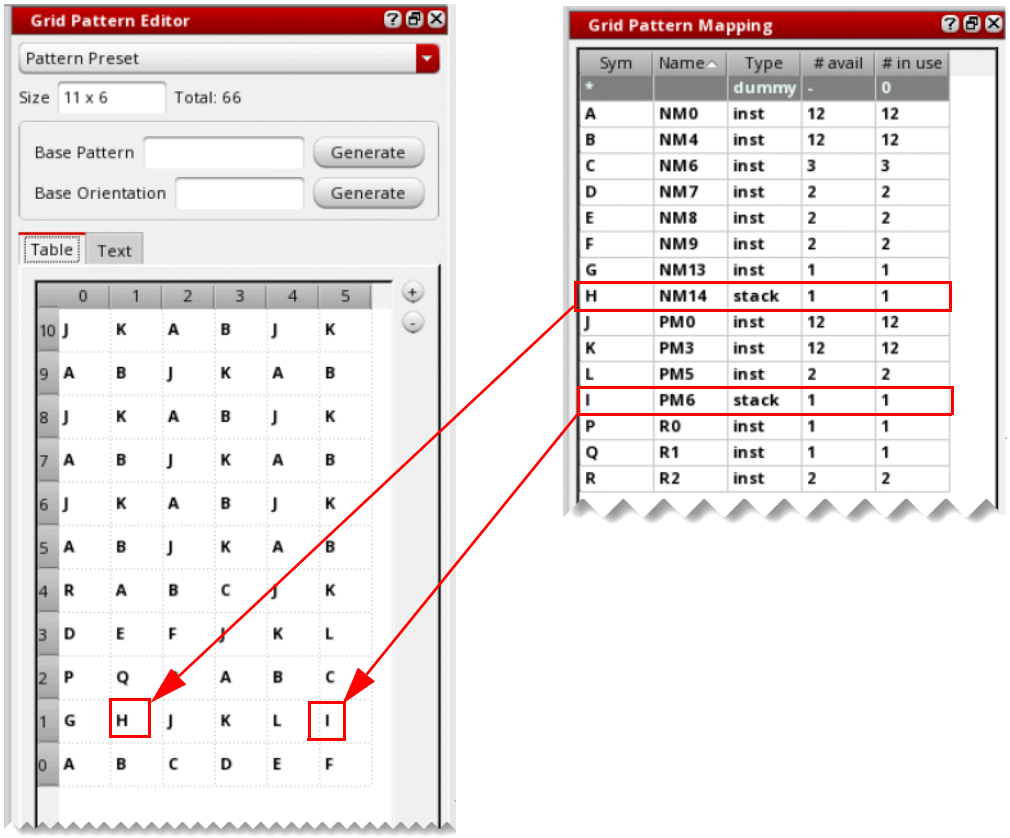
For existing Modgens, stacks are detected only if their devices are arranged and abutted properly.



A fully expanded and stacked m-factored device

Each abutted stack is represented as a single symbol, as if it were a single device, in the GPE and GPM assistants. The tooltips for these symbols in the GPE and GPM assistants provide information about the constituent stacked devices.

Stacks sets that are connected in parallel also share a common symbol.



Note: Series-connected, unabutted devices with shared gate connectivity are displayed in the GPE and GPM as individual symbols.

Dummies that are adjacent to abutted stacks are also stacked and abutted.

Related SKILL Functions

- gpeStacksCompress
- gpeStacksAreCompressed
- gpeStacksUncompress

Features of Stacks

- **Changing the Orientations of Stacks:** A stack may have valid orientations of `R0`, `MX`, `MY`, or `R180`. The orientation applies to the entire stack; the orientations of the stack members may (or may not) be identical to the orientation of the stack. This means that

the absolute orientation of a stack is arbitrary. Therefore, the displayed stack orientation may not always be preserved when the stack is rotated. When a stack is rotated, the orientation is applied to each stacked device individually. `MY` and `R180` rotations also result in a left-to-right reversal of the ordering of the stack members.

- **Naming Stacks:** Stacks are named in one of the following ways:
  - If a naming pattern is recognized, then its base pattern is used to identify the stack.

    Examples: A stacked named `M0` is created in the following situations:
    - A stack that represents the layout devices `M0.0`, `M0.1`, `M0.2`, and `M0.3` that are bound to schematic device `M0` with an s-factor.
    - A stack that represents iterated layout devices `M0<0>`, `M0<1>`, `M0<2>`, and `M0<3>`.
  - If a naming pattern is not recognized, then the stack name is based on the name of the first stack member, alphabetically.
- **Adding Dummies:** Dummies can be added around stacks.
  - Dummies added above or below a stack result in a stack of dummies with the same number of members as the reference stack.
  - Dummies added to the left or right of a stack result in a single dummy being inserted.
  - Surround dummies inserts dummy stacks above and below the stack and a single dummy on each side.
- **Using Pattern Presets:** Built-in presets, other than the resistor presets, support the use of Modgen sandbox objects in both, the stacked and unstacked formats. Resistor presets support only the unstacked format.

Specifying Interdigitation Patterns

You have two options for setting up interdigitation. You can specify an interdigitation pattern specific to this module using the Modgen Pattern Editor. The module will, by default, be interdigitated by `1`, and you do not have to set up interdigitation if you are satisfied with the default. However, you can change the default value by changing the cdsenv variable, modgenInterdigitationFactor. In case no interdigitation in the Modgen is required, you can specify `0` in the *Interdigitate By* text box. Therefore, when you make a Modgen, it will not be interdigitated.

The environment variable, modgenUseIteratedAsMfactor has been introduced that allows Modgen to consider iterated instances, with same or different connectivity, equivalent to an m-factor (multiplier). As a result, the pattern mapping shows the number of instances equivalent to the iterations for each different master.

You can also specify custom interdigitation patterns in a file. These patterns will be available on a user, site, or design basis, depending on where it is stored; see the *Cadence Application Infrastructure User Guide* for more information on where to store this file. Each pattern defined in the file is then available from the *Pattern Option* cyclic field in the Module Generator form. For more information, see Setting Up the Custom Interdigitation Pattern File.

Using the Modgen Pattern Editor

In the Modgen Pattern Editor, you can specify an interdigitation pattern for this module. You can then save this pattern for future use.

You have three options for interdigitation. You can specify that the Module Generator automatically interdigitate by a specified integer (see below), you can specify a custom interdigitation pattern (see Specifying Standard Interdigitation).

Tip

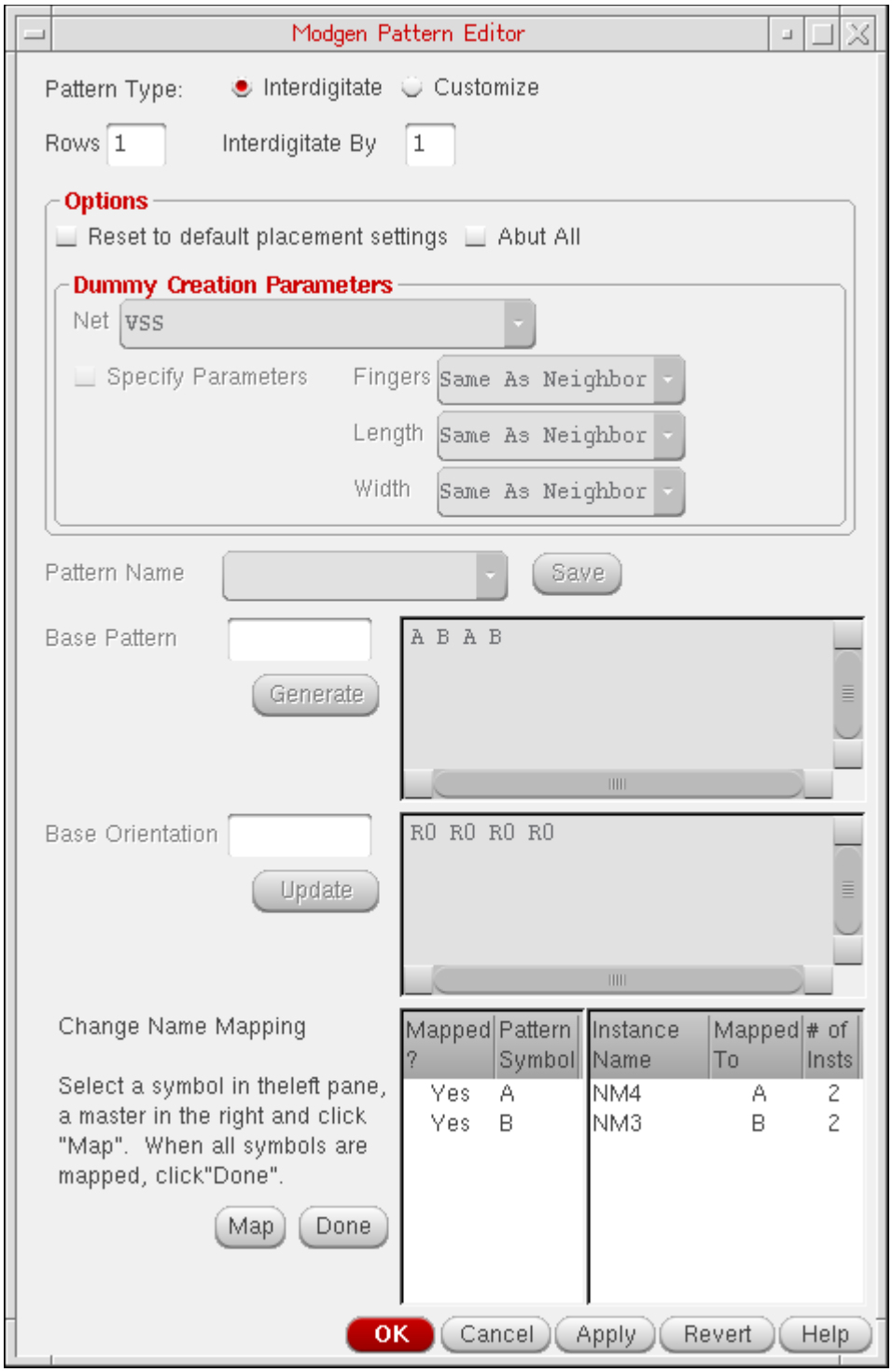
The module will, by default, be interdigitated by 1, and you do not have to set up interdigitation if you are satisfied with the default.

Note: The names in the *Pattern Name* fields are filtered by the number of schematic and layout instances and the number of rows.

Specifying Standard Interdigitation

1. Click the *Pattern*  icon on the toolbar at the top.

The Modgen Pattern Editor appears.



2. For Pattern Option, select the *Interdigitate* radio button.
3. In the *Interdigitate By* field, enter the integer to use for interdigitation.
4. Click *OK* to save your settings and dismiss the Modgen Pattern Editor. The Modgen Pattern Editor is dismissed and the module layout is modified based on the specified pattern. Optionally, click *Revert* to undo all changes in the Modgen Pattern Editor. All fields are reset to the previous successful applied values.

Important

If you select the *Abut All* check box, auto abutment is triggered automatically for all devices in a Modgen once the pattern editor has been applied.

Specifying Custom Interdigitation

You can also specify a custom interdigitation pattern. When you specify a custom interdigitation pattern, you can either begin with a base pattern or specify a custom interdigitation pattern from scratch.

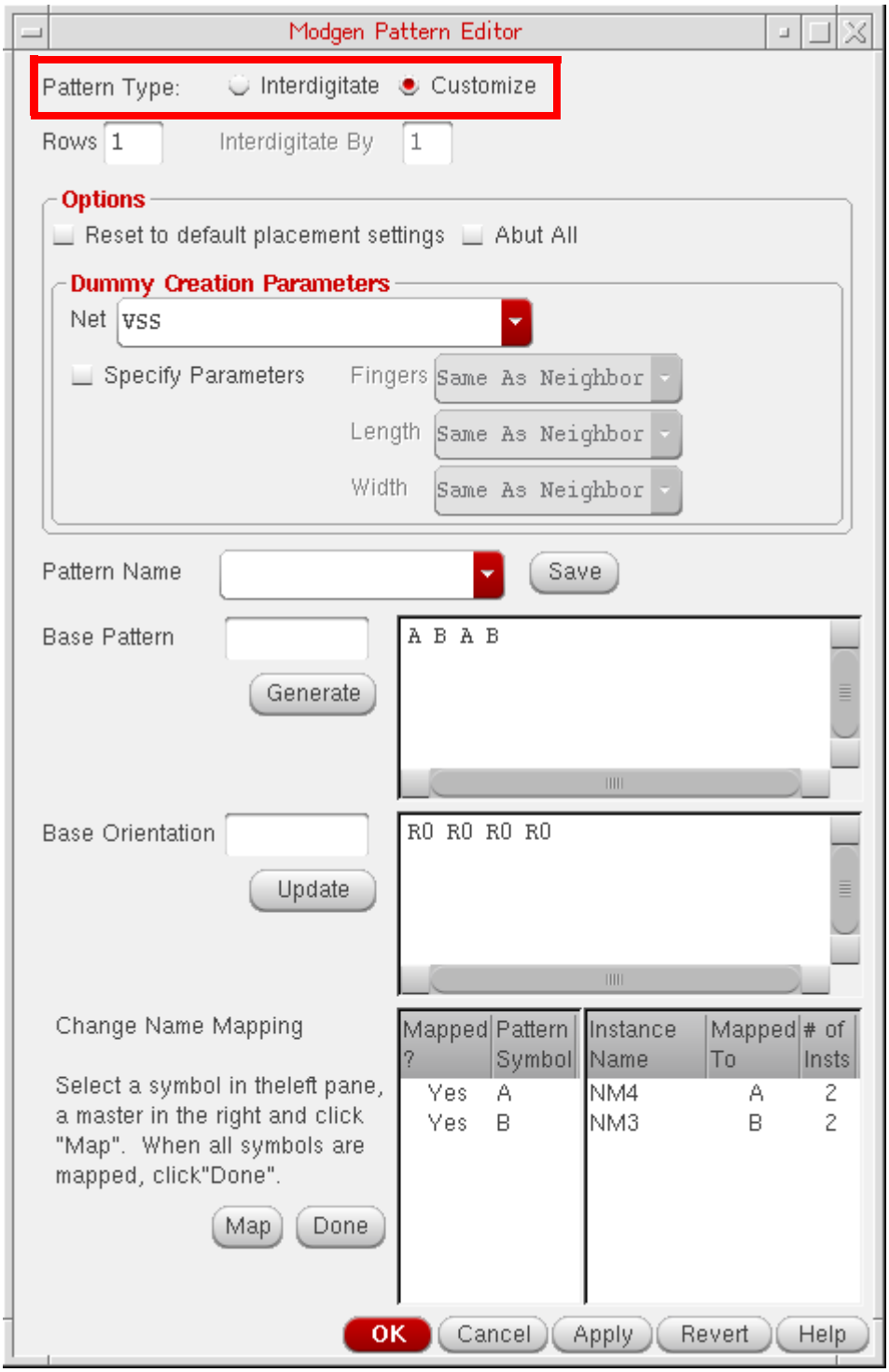
To begin custom interdigitation:

1. Click the *Pattern*  icon on the toolbar at the top.

   The Modgen Pattern Editor appears.

   The Modgen Pattern form lists all the device instances currently specified in this module generator and maps each one to a letter. You can also specify a dummy device using an asterisk (*) or an empty cell using a dash character (-).

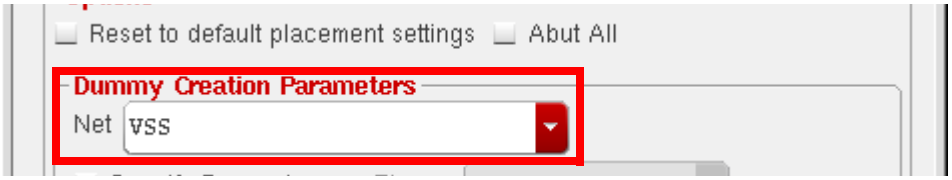
2. For Pattern Option, select the *Customize* radio button.



You can now either modify the default pattern (see Modifying a Custom Interdigitation) or you can generate a base pattern and modify it if necessary (see Generating a Base Pattern).

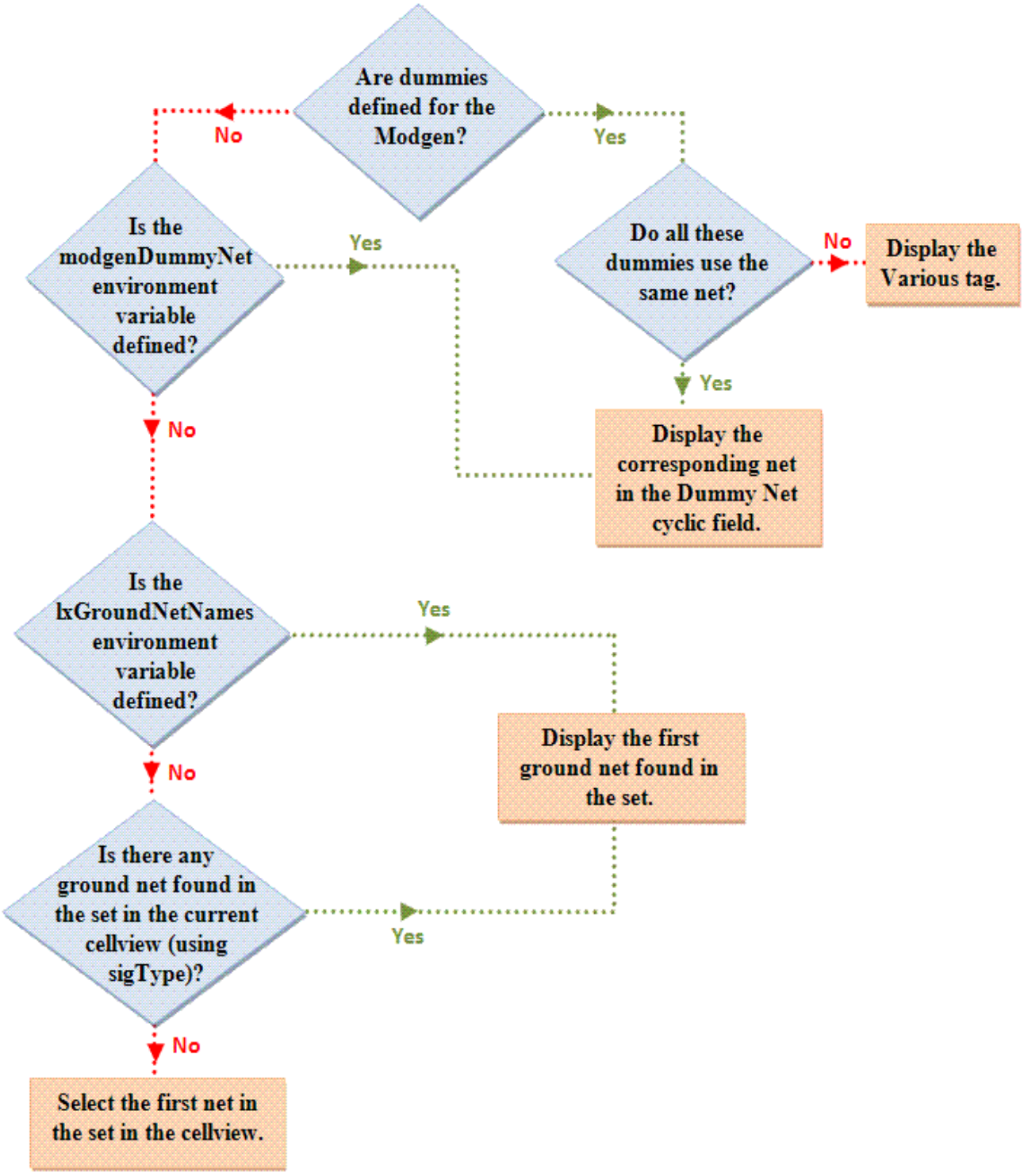
Specifying Connectivity for Dummies

You can specify the connectivity for dummies that are placed in the pattern specification.



The *Net* combo box lists the nets that are connected to objects in the active Modgen. Choose the net to which you want all the dummies in the pattern for the Modgen to be connected. To choose a net that is present in the cellView, but not in the current Modgen, type the net name in the *Net* combo box. The entry is validated against all existing nets in the current cellView. If a matching net is not available in the current cellView, then a warning message is issued and the net is created.

Based on the following algorithm, a default value is selected and displayed in the *Net* combo box.



Select the *Specify Properties* check box to enable the options for specifying dummy device properties.

1. From the *Fingers* cyclic field, choose one of the following:
   - *CDF Default* – The default number of fingers as specified in the CDF is created
   - *Same As Neighbor* – The same number of fingers as the neighboring device is created
   - *Specify* – You can specify the number of fingers to be created in the box beside the *Fingers* cyclic field
2. From the *Length* cyclic field, choose one of the following:
   - *CDF Default* – The default number of fingers as specified in the CDF is created
   - *Same As Neighbor* – The length of fingers of the neighboring device is considered
   - *Specify* – You can specify the length of fingers in the box beside the *Length* cyclic field

   Note: Scale factors can be used to specify the length; for example .1u.
3. From the *Width* cyclic field, choose one of the following:
   - *CDF Default* – The default number of fingers as specified in the CDF is created
   - *Same As Neighbor* – The width of fingers of the neighboring device is considered
   - *Specify* – You can specify the width of fingers in the box beside the *Width* cyclic field

Scale factors can be used to specify the width; for example 200n.

The above parameters apply to all dummies in the current pattern. The number of fingers for specific dummies can be overridden specifying a pattern in the Base Pattern field. For example, to set a specific dummy's number of fingers to 2, type '*:2' in the Base Pattern field.

For more information about specifying patterns, see Generating a Base Pattern.

For more information about specifying dummy device properties, see .

Generating a Base Pattern

You can begin your interdigitation by specifying a base pattern and the number of rows, then generate the resulting pattern and modify it as desired. Base patterns begin at the lower left corner of the grid and proceed left to right until all rows are filled.

For example, if you have two devices with an m-factor of four, and you specify two rows with the base pattern of ABBA, you would create the following interdigitation pattern:

```
A B B A
A B B A
```

The base pattern does not have to cover an entire row. If, instead, you specified a base pattern of ABB, the resulting pattern would be as follows:

```
BBAB
ABBA
```

To generate a base pattern:

1. Specify the number of rows to create in the *Rows* field.
2. Type the base pattern using the letters mapped to each instance as displayed in the *Name Mapping* list box in the *Base Pattern* text box.

   You can also specify dummy devices and empty rows using the values displayed in the Name Mapping list box.

This is a space-separated list; each value must be followed by a space.

3. Click *Generate*.

   The resulting pattern is displayed in the *Base Pattern* list box.

Repetition Factor for Base Pattern

You can also specify a repetition factor to repeat a pattern symbol multiple times in the *Base Pattern* text using the following syntax:

```
<pattern symbol>:<repetition count>
```

Example:

If you type `A B:8 C` in the *Base Pattern* text box, then the resultant value will be displayed in the following pattern:

```
A B B B B B B B B C
```

In addition, the repetition factor can also be used in the Base Orientation field. For example, entering `R0 MY:8 R0` would result in:

```
R0 MY MY MY MY MY MY MY MY R0
```

Repetition Sequences of Pattern Symbols

The pattern notation specifies repeating sequences of pattern symbols. This is useful for more complicated patterns required to generate the device configurations, such as common centroid.

Note: A pattern symbol is a single letter, '`*`' or '`-`', which is optionally followed by a number. A pattern sequence contains one or more pattern symbols or pattern expressions concatenated together.

In addition, a shorthand for filling out the rows of a pattern has been introduced. In this, if the pattern text has less lines than what is specified in the *Rows* field of the form, then the lines entered so far gets repeated to fill out the remaining rows.

Important

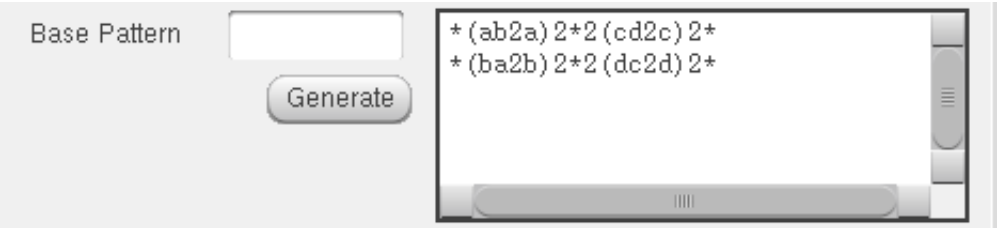
The shorthand notation can only be used if the number of schematic devices in the Modgen are less than or equal to 25. This is because if there are more than 25 schematic devices, Modgens generate symbols with numbers concatenated to them, such as `A1` or `B1` to make them unique. Therefore, the syntax for the shorthand repetition factor and the syntax to make unique symbols are ambiguous.

Though, you can still specify a repetition factor if there are more than 25 schematic devices, but it must be delimited with a ':'. For example, `a2` is interpreted as the pattern symbol `A2` whereas, `a:2` is interpreted as the pattern symbol `A` repeated `2` times.

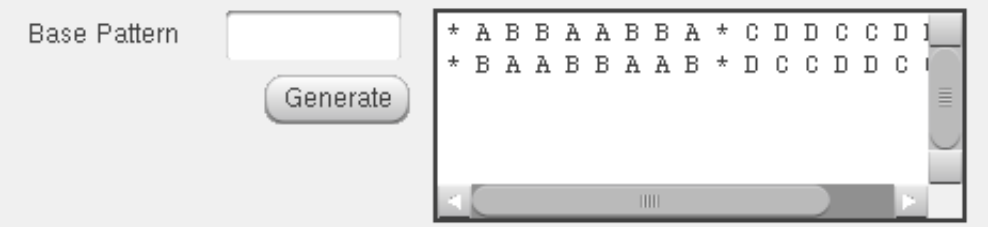
Example 1:

To generate a pattern in which each dummy in the middle has two fingers:

1. Type the shorthand notation of the pattern in the expanded *Base Pattern* text box of the Modgen Pattern Editor.



2. Click the *Apply* button.



   Notice that the shorthand notation in the expanded *Base Pattern* text box is automatically converted to the corresponding base pattern.

3. Click *OK* to save the setting and dismiss the Modgen Pattern Editor.

Example 2:

If you want to generate the pattern for `(b2a2)2` and `(a2b2)2`, where a Modgen has `4` rows, then the following pattern is generated.

```
B B A A B B A A

A A B B A A B B

B B A A B B A A

A A B B A A B B
```

Modifying a Custom Interdigitation

1. In the *Pattern* list box, modify the default pattern, or the pattern generated using a base pattern (see Generating a Base Pattern), using the letters mapped to each instance as displayed in the *Name Mapping* list box.

   Each row in the Pattern list box maps to a row in the module.

Caution

This is a space-separated list; each value must be followed by a space.

2. In the *Orientation* list box, specify an orientation for each instance in the order in which they are listed in the *Pattern* list box.

This is a space-separated list; each value must be followed by a space.

For example, if the pattern is A B C D, and you want to specify the following orientations:

A	R0
B	R0
C	R90
D	R90

Then you would specify the following in the *Base Orientation* list box: `R0 R0 R90 R90`

You can also specify the orientation for dummy devices. Dummy devices are represented by an asterisk (*) in the Custom Orientation list box. Consider pattern `*AB*` with the following orientation:

*	R180
A	R0
B	R90
*	R180

Before specifying the base orientation for dummy devices, set the `MG_BASE_ORIENT_OVERWRITE_DUMMY` shell environment variable to `t`. Then, specify the following in the *Base Orientation* list box: `R180 R0 R90 R180`.

If you do not set the shell environment variable, then the dummy devices will have the same orientation as their neighboring non-dummy devices. For example, for pattern `*AB*`, if you specify the orientation as `R0 R90`, then the orientation of the first dummy device will be `R0`, and for the last dummy device will be `R90`.

3. If there are any errors in the pattern or orientation, use the *Revert* button to undo the changes. All fields are reset to the previous successful applied values. Now you can made the required edits to the pattern and orientation.
4. If you want to save this pattern, type the pattern name from the *Pattern Name* combo box and click the *Save* button.

   Note: You can also select applicable pattern names for the current Modgen configuration

from the *Pattern Name* combo box.

Virtuoso saves the pattern to the first writable Modgen.patterns file it can find, based on the Cadence search rules defined in the *Cadence Application Infrastructure User Guide*. If it cannot locate a Modgen.patterns file, it will create one in the current working directory.

5. Click *OK*.

   The Modgen Pattern Editor is dismissed and the module layout is modified based on the specified pattern.

   To save the pattern, specify a *Pattern Name* and click *Save*. In case you choose to save the pattern, then the changes will not apply across different Modgens, and therefore the pattern file will not be saved.

Specifying pattern Symbols for Devices Connected in Series

To differentiate among the individual elements of an s-factored device, you need to specify pattern symbols in the *Base Orientation* text box using the following syntax:

```
<pattern symbol>.<integer>
```

Example:

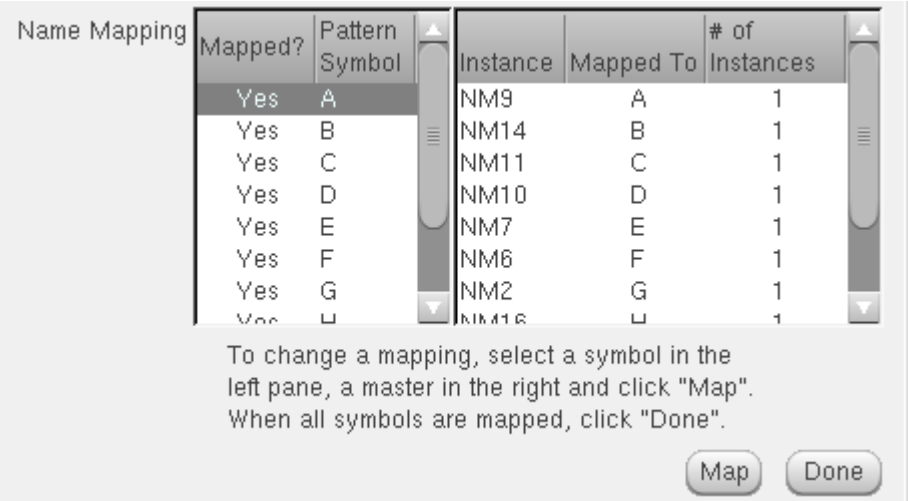
If two resistors `R0` and `R1` each have two segments, `s0` and `s1`, then you can use `A.1` to represent `R0.s0`, `A.2` to represent `R0.s1`. Similarly, `B.1` to represent `R1.s0` and `B.2` to represent `R1.s1`. The resultant value will be displayed in the following pattern:

```
A.1 A.2 B.1 B.2
```

Changing the Pattern Name Mappings

To change the mapping of pattern names to instance master names, you need to perform the following steps:

1. Select the desired pattern symbol from the left pane of the *Name Mapping* list box.



2. In the right pane of the *Name Mapping* list box, select the desired instance master.
3. Click the *Map* button to complete the mapping. The entries in both panes of the form will update to show the new mapping.
4. Continue steps 2 to 4 until all pattern symbols are mapped.
5. Click *Done*.
6. Click the *OK* button for new mapping to take effect.

Important

If all the iterated instances have the same connectivity, then there is only one pattern symbol to represent them.

For example, if there is an iterated instance, `M1<0:2>` and all the instances have the same connectivity, then its entry in the name mapping pane is:

```
A      M1<0:2>
```

However, if the instances have different connectivity, then entries in the name mapping pane are:

```
A      M1<0>
B      M1<1>
C      M1<2>
```

Note: If the user selects the *Customize* option and enters a pattern and/or an orientation string in the Pattern text or Orientation text fields, then those strings will be retained such that if the user switches to the Interdigitate mode and back to the Customize mode, then the Pattern and Orientation text fields will reflect the last entered strings. However, if the user clicks the *Apply* button or changes the number of rows while the Interdigitate option is selected, then these retained strings are discarded and the user must enter new pattern and orientation strings.

Setting Up the Custom Interdigitation Pattern File

You can specify custom interdigitation patterns in a *Modgen.patterns* file, which can be stored on a design, user, or site basis (for more information, see the *Cadence Application Infrastructure User Guide*).

Syntax

```
( name <Schematic_inst_num> <layout_inst_num> <row_num>
( (<row1_pattern_list>)
    (<row2_pattern_list>)
    ...
    (<rowN_pattern_list>)
  )
( (<row1_orient_list>)
    (<row2_orient_list>)
    ...
    (<rowN_orient_list>)
  )
)
```

Parameters

The following table defines the parameters for the custom interdigitation pattern file.

Table 2-1 Custom Interdigitation Pattern Parameters

Parameter Name	Parameter Type	Description
name	string	The name of the pattern.
Schematic_inst_num	integer	The number of different Pcell masters in the Modgen module.
layout_inst_num	integer	The number of instances in the Modgen array.
row_num	integer	The number of rows in the Modgen array.
row_pattern_list	list	A list composed by the letters that represent the interdigitation of instances; each list represents one row in the array.

Table 2-1 Custom Interdigitation Pattern Parameters

Parameter Name	Parameter Type	Description
row_orient_list	list	A list composed by an orient string representing the orientation of instances; each list represents one row in the array. Orientations are defined using one of the following strings: ■ R0 ■ R90 ■ R180 ■ R270 ■ MX ■ MY

Example

Here are several examples of interdigitation pattern definitions.

```
( Pattern2
  2 4 1
  (
   (A B B A)
  )
  (
   (MX MX MY MY)
  )
)

( AB_BA
  2 4 2
  (
   (A B)
   (B A)
  )
  (
   (R0 MX)
```

```
    (R0 MX)
   )

)
( Quad1
  2 8 2
  (
   (A B B A)
   (A B B A)
  )
)
```

3

Editing Modgens

This chapter covers:

- Using the Modgen On-Canvas Commands
- Support for Row Regions
- Generating Reusable Modgen Templates
- Adding Dummy Devices
- Adding Body Contacts
- Defining Grid Placement
- Specifying Guard Rings
- Abutting Devices
- Specifying Device Alignment and Spacing
- Merging Layers

Using the Modgen On-Canvas Commands

Virtuoso provides the following Modgen on-canvas commands that can be used to perform certain tasks without opening the Modgen Editor. To access the Modgen on-canvas commands, either select the Modgen constraint in the layout canvas and choose *Modgen* from the shortcut menu or select *Place—Modgen*.

Command Name	Description
Create/Edit Modgen 	If the selected device(s) are not part of an existing Modgen constraint, creates a new Modgen constraint. The Modgen Editor is not displayed. If the selected device(s) are part of an existing Modgen constraint, opens it in the Modgen Editor. For more information, see Creating a Modgen.
Extract Template 	Generates a reusable template for each device group available in the given Modgen.
Create Modgen From Template 	Applies existing Modgen reuse templates to the specified instances to generate matching Modgens.
Grid Pattern Editor	Displays the Grid Pattern Editor and Grid Pattern Mapping assistants. For more information, see Grid Pattern Editor.
Split Rows 	Splits each Modgen row into two rows.
UnSplit Rows 	Combines every two rows of Modgen into a single row.
Abut Instances 	Abuts the Modgen devices.
UnAbut Instances 	Unabuts the Modgen devices. For more information about the abutment and unabutment options, see Abutting Devices.

Dummies	Displays a submenu with the following commands that can be used to add or remove dummies: ■ Add Dummy Column Left  ■ Add Dummy Column Right  ■ Add Dummy Row Top  ■ Add Dummy Row Bottom  ■ Add Surround Dummies  ■ Remove Surround Dummies  ■ Delete All Dummies  For more information, see Adding Dummy Devices.
Set Member Alignment/ Spacing 	Displays the Set Member Alignment and Spacing form to specify the alignment and spacing values for the Modgen devices. For more information, see Specifying Device Alignment and Spacing.
Merge Layers	Displays the Select merge layers form, in which you can specify the layers that need to be merged. For more information, see Merging Layers.
Guard Ring	Displays a submenu with the following commands that can be used to create different types of guard rings: ■ Add Fluid Guard Ring  ■ Add MPP Guard Ring  ■ Add Identical Guard Ring  ■ Remove Guard Ring  For more information, see Specifying Guard Rings.

Assistants:

- Grid Pattern Editor
- Grid Pattern Mapping

As in the case of other layout instances, you can use the Property Editor assistant to interactively update the properties of a Modgen.

You can use the Property Editor assistant to edit attributes of the Modgen currently selected in the layout design. Any changes you make to the Modgen properties are immediately applied to the Modgen in the layout canvas, as displayed in the figure below.

Support for Row Regions

(ICADVM20.1 EXL Only) All Modgen creation, modification, and regeneration commands, including the Modgen on-canvas commands, recognize row regions. Therefore, a Modgen, when created or regenerated, automatically fits into an existing row region that is valid for the members in the Modgen configuration. If invalid, the row region is ignored during Modgen generation.

When a Modgen is created interactively from the layout design, the orientations of the Modgen members may be altered to ensure compact placement of the Modgen members within the placement rows.

For more information about the row infrastructure, see Creating Rows in *Virtuoso Placer User Guide*.

For more information about the various row-based SKILL functions, see Row-based Functions in *Virtuoso Design Environment SKILL Reference*.

For more information about the SKILL functions to define placement area, see Placement Area Definition Functions in *Virtuoso Design Environment SKILL Reference*.

Generating Reusable Modgen Templates

(ICADVM20.1 Only) Virtuoso supports a template-driven Modgen reuse solution to help improve layout productivity. A Modgen template comprises a set of Modgen parameters, such as the interdigitation pattern, abutment, dummy definitions, and spacing values, that can be reused to create a gridded layout of matching structures.

Note: Mosaics are not supported for extraction and reuse.

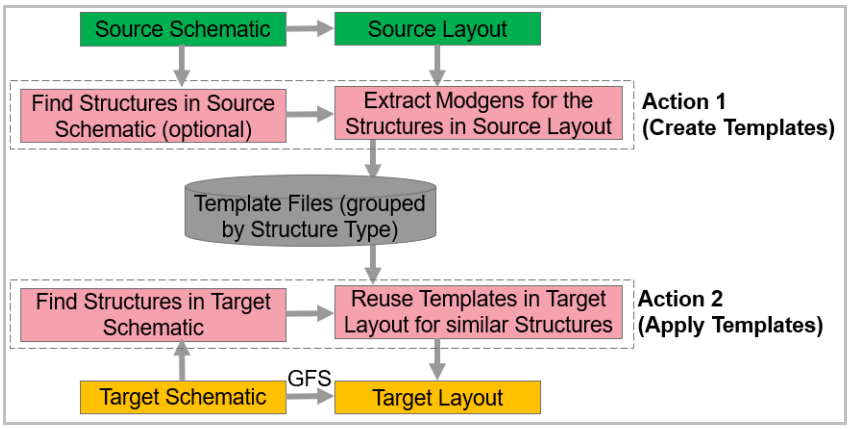
This section covers the following topics:

- Template-Driven Modgen Reuse Flow
- Generating a Modgen Template File
- Reusing the Modgen Template File

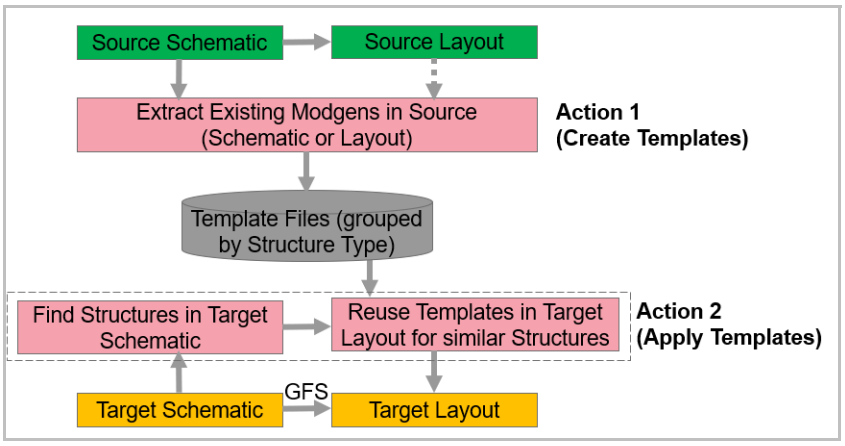
Template-Driven Modgen Reuse Flow

The following diagrams depict the template-driven Modgen reuse flow.

Flow 1 - There are no Modgen Structures in the Source



Flow 2 - There are Modgen Structures in the Source

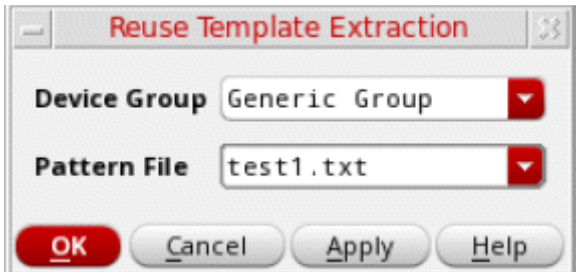


1. The tool extracts Modgens from the source layout for important structures in the source schematic (Flow 1) and for existing Modgens, if available (Flow 2).
2. The information is saved in the Modgen template files.
3. The template files are then grouped by their structure type. The Modgen templates are reused in the target layout to generate Modgens that include similar structures in the target schematic.

Generating a Modgen Template File

To generate a Modgen template file:

1. Select the required Modgen constraint either in the layout canvas or in the Constraint Manager assistant.
2. Open the Reuse Template Exaction form using one of the following methods:
   - ❑ Choose *Place—Modgen—Extract Template*.
   - ❑ Choose *Layout Reuse—Create Reuse Templates from Modgens* in Constraint Manager.
   - ❑ Choose *Layout Reuse—Create Reuse Templates from Layout* in Constraint Manager.
   - ❑ Use the gpeTemplateExtractLaunchForm SKILL function.



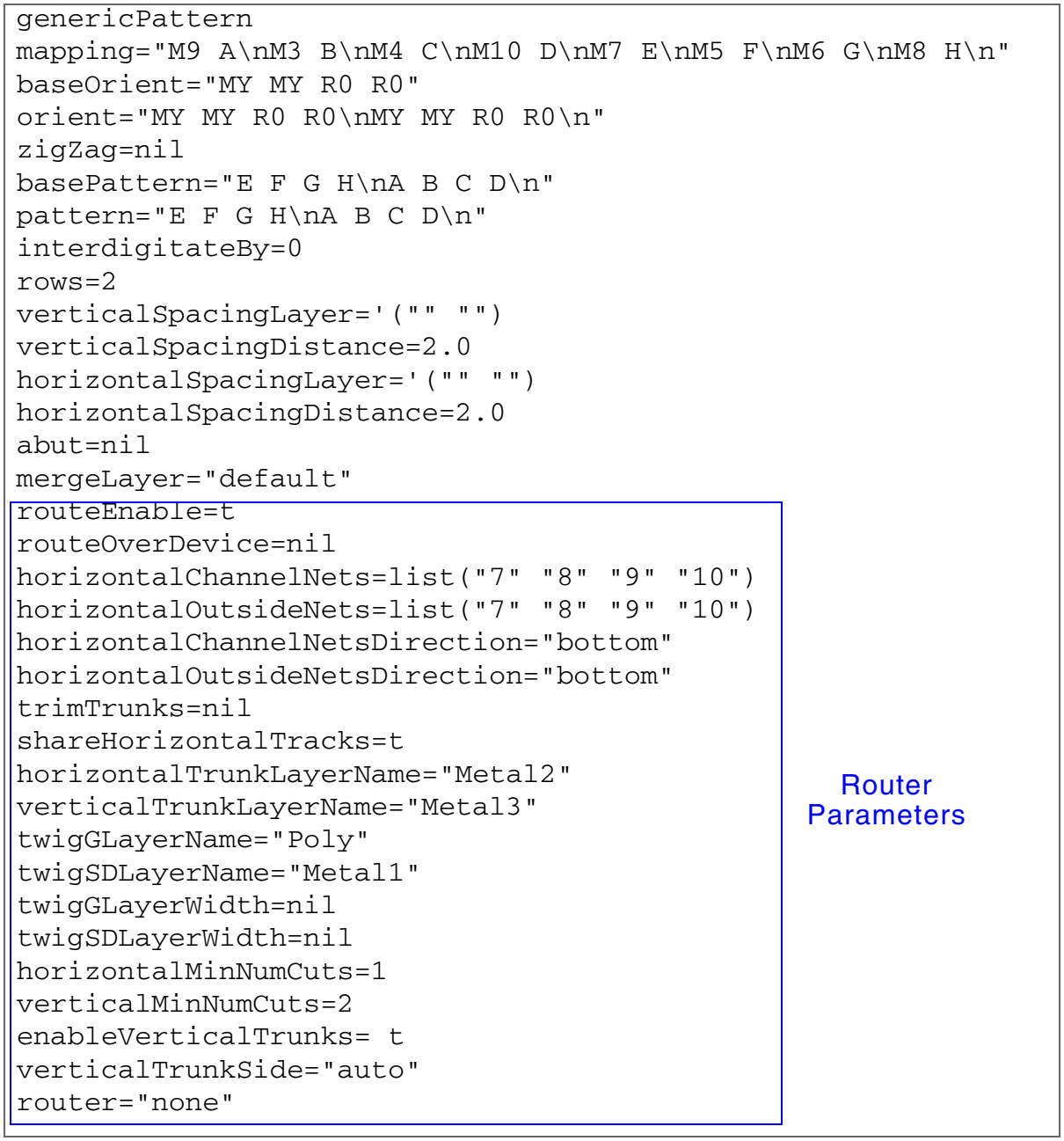
3. Specify a *Device Group*. Template files are created under matching or specified group names. Default is *Generic Group*.
4. Specify a *Pattern File* name with the `.txt` extension. This is the Modgen template file.
5. Click *Apply* to save the template file.
6. Click *OK*. Constraint parameters from the source Modgen are stored in the template file.

Corresponding SKILL Functions:

- gpeTemplateExtractLaunchForm
- gpeExtractTemplateFromMG

Parameters of a Modgen Reuse Template File

A Modgen template comprises a set of Modgen parameters that can be reused to create a grid-based layout of matching structures. The following image shows a sample Modgen reuse template file:



The following table describes the parameters of a Modgen template file:

`routeEnable = t | nil`	**Description:** Specifies whether the pin-to-trunk router is to be used to route the Modgen devices. **Valid Values:** `t` and `nil` When set to `nil`, all route-related parameters are ignored and a topology pattern is not created. When `routeEnable` is set to `t` and `router` is set to `nil`, a topology pattern is created but the pin-to-trunk router is not called.
`routeOverDevice = t | nil`	**Description:** Allows horizontal routes to be generated over devices. **Valid Values:** `t` and `nil`
`horizontalChannelNets = l_netNames | nil`	**Description:** (Optional) Lists channel nets for the horizontal trunks. If not specified (which is recommended), channel nets in the new Modgen are created based on the net connections. **Note:** Channel nets are the nets between Modgen rows. **Valid Values:** *l_netNames* and `nil` **Default Value:** `nil`, which indicates no channel nets are to be created. **Example:** `list("net1" "net2" "net3")`
`horizontalOutsideNets = l_netNames | nil`	

Description: Lists the nets that are above or below the Modgen. When this parameter is not specified, outside nets are created based on connections.

Valid Values: *l_netNames*, nil

Default Value: nil, which indicates that no trunks are to be created above or below the Modgen.

Example:
```
horizontalOutsideNets=list("net1 net2 net3")
```

`horizontalTrunkWidths = l_netWidths | nil`

Description: Specifies a space-separated list of horizontal trunk widths.

Valid Values: *l_netWidths*, nil

If a single value is specified, it is applied to all channel nets.

Default Value: nil, which indicates that the default values from either the technology file or a predefined WSP are used.

Example:
```
list(list("D1" 0.22) list("D2" 0.22))
```

`horizontalNetOrder = l_netOrder | nil`

Description: Specifies the net order of channel nets.

Valid Values: *l_netOrder* and nil

Default Value: nil

horizontalNetOrder values depends on the value of horizontalChannelNets.

`trimTrunks = t | nil`

Description: Specifies whether the ends of the horizontal trunks are to be trimmed while routing.

Valid Values: t and nil

`shareHorizontalTracks = t | nil`

Description: Specifies whether two nets can share the same horizontal trunk.

Valid Values: `t` and `nil`

`horizontalTrunkLayerName = t_layerName`

Specifies the layer on which horizontal trunks are to be generated. This is a mandatory parameter.

Valid Values: *`t_layerName`*

`twigGLayerName = t_layerName`

Description: Specifies the layer in which the twigs that are connected to the gate terminal are to be generated. This is a mandatory parameter.

Valid Values: *`t_layerName`*

`twigGLayerWidth = f_twigGLayerWidth`

Description: Specifies the width of the gate net twigs.

Valid Values: *`f_twigGLayerWidth`*

`twigSDLayerName = t_layerName`

Description: Specifies the layer in which the twigs connected to the source and drain terminals are to be generated.

Valid Values: *`t_layerName`*

`twigSDLayerWidth = t_twigSDLayerWidth`

Description: Specifies the width of the source and drain gate twigs. This is a mandatory parameter.

Valid Values: *`t_twigSDLayerWidth`*

`horizontalMinNumCuts = nil | n_numCuts`

Description: Specifies the minimum number of cuts for the vias connecting the twigs to other objects.

Valid Values: `nil` and *`n_numCuts`* }

Default Value: `1`

`anchorReference = nil | t_refLayer`

Description: Specifies the reference layer or the anchor from which the trunk chain must start.

Valid Values: `nil` and *`t_refLayer`*

`firstHorizontalChannelTrackOffset = nil | f_offset`

Description: Specifies the offset of the first horizontal channel track from the specified `anchorReference` value.

Valid Values: `nil` and *`f_offset`*

`firstHorizontalOutsideTrackOffset = nil | f_offset`

Description: Specifies the offset of the first horizontal track outside the channel from the specified `anchorReference` value.

Valid Values: `nil` and *`f_offset`*

`firstHorizontalDeviceTrackOffset = nil | f_offset`

Description: Specifies the offset of the first horizontal device track of the `overDevice` trunk chain.

Valid Values: `nil` and *`f_offset`*

`enableVerticalTrunks = t | nil`

Description: Specifies whether vertical trunks can be created. The default value is `nil`.

Valid Values: `t` and `nil`

`verticalTrunkLayerName = t_layerName`

Description: Specifies the layer on which vertical trunks are to be generated. The option can be set only if `enableVerticalTrunks` is set to `t`.

Valid Values: *`t_layerName`*

`verticalTrunkWidths = l_netWidths | nil`

Description: Specifies a space-separated list of vertical trunk widths. If a single value is specified, it is applied to all channel nets. The option can be set only if `enableVerticalTrunks` is set to `t`.

Valid Values: *`l_netWidths`* and `nil`

Default Value: `nil`, which indicates that the default values from either the technology file or a predefined WSP are to be used.

Example:

```
list(list("D1" 0.22) list("D2" 0.22))
```

`verticalNetOrder = `*`l_netOrder`* `| nil`

Description: Specifies the net order of vertical channel nets. The default value is `nil` because the default `horizontalChannelNets` is `nil`. The option can be set only if `enableVerticalTrunks` is set to `t`.

Valid Values: *`l_netOrder`* and `nil`

`verticalMinNumCuts = nil |` *`n_numCuts`*

Description: Specifies the minimum number of cuts for the vias connecting the twigs to other objects. The default value is `1`. The option can be set only if `enableVerticalTrunks` is set to `t`.

Valid Values: `nil` and *`n_numCuts`*

`firstVerticalTrackOffset = nil |` *`f_offset`*

Description: Specifies the offset of the first vertical track from the specified `anchorReference` value.

Valid Values: `nil` and *`f_offset`*

`verticalTrunkSide ="both" | "auto" | "left" | "right"`

Description: Specifies the side along which vertical trunks are to be generated.

The option can be set only if `enableVerticalTrunks` is set to `t`.

Valid Values: `left`, `right`, `both`, and `auto`.

`router {"none" | "pinToTrunk" }`

Description: Specifies whether the pin-to-trunk router is to be used for routing.

`horizontalChannelNetsDirection ="top" | "bottom" | "both"`

Description: Specifies the direction of the horizontal channel nets, which is the direction from the trunk to the instance terminal.

Valid Values: `top, bottom, both`

Default Value: `both`

`horizontalOutsideNetsDirection ="top" | "bottom" | "both"`

Description: Specifies the direction of the horizontal nets that are outside channels.

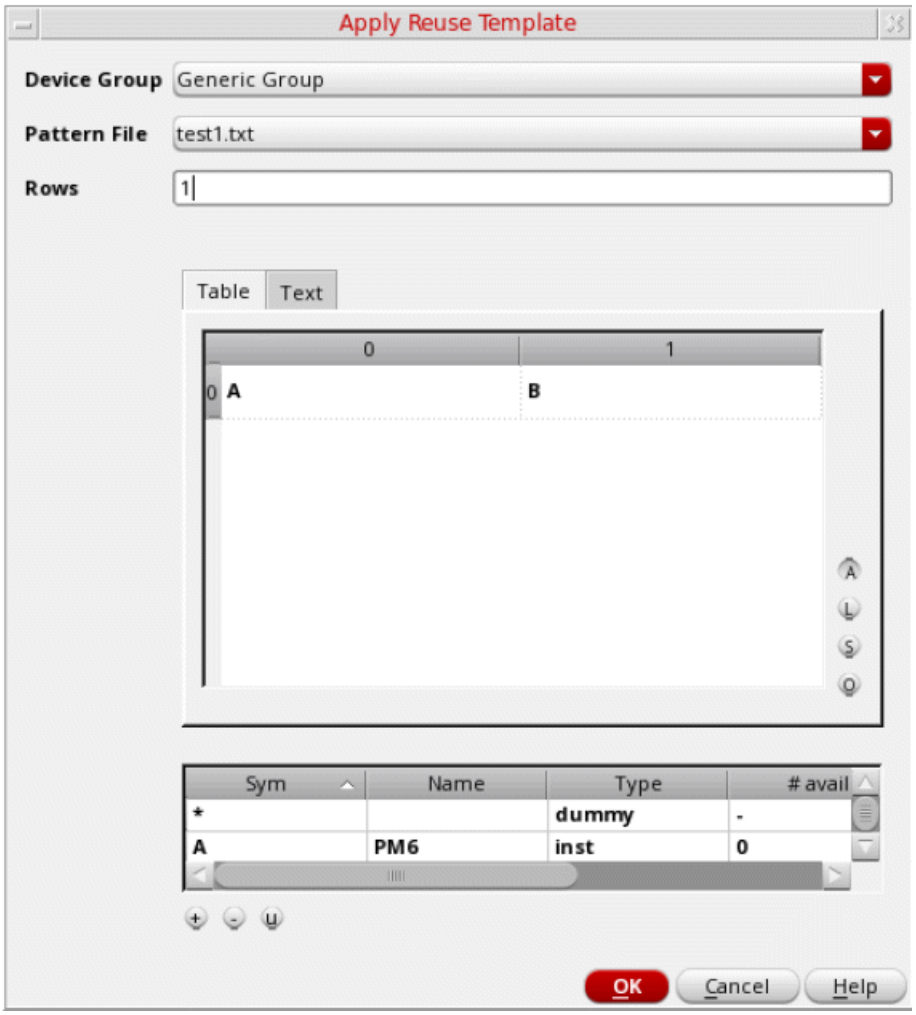
Valid Values: `top, bottom, both`

Default Value: `both`

Reusing the Modgen Template File

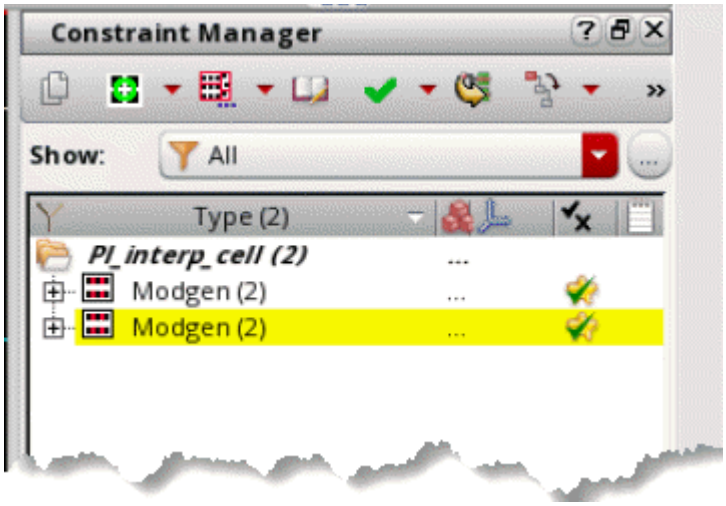
The Modgen template file created above can be reused to generate a new Modgen constraint. To create a Modgen using a template file:

1. Navigate to the schematic design.
2. Select the target devices in the Circuit Prospector assistant. These are the devices to be included in the new Modgen.
3. In Layout EXL, choose *Place—Modgen—Create Modgen From Template*. The Apply Reuse Template form is displayed.



4. Select the *Device Group* that contains the required Modgen reuse template.
5. Select the *Pattern File* corresponding to the Modgen template to be applied to the new Modgen. Parameters from the pattern file are loaded in the form.

6. Specify the number of *Rows* to be generated.
7. Update the pattern as per your requirements.
8. Click *OK* to generate the Modgen. The new Modgen is listed in the Constraints Manager assistant.



Corresponding SKILL function: gpeLoadTemplateLaunchForm

Adding Dummy Devices

Dummy devices are created to counter electrical effects that are observed at small geometries. You can create dummies around devices in Modgens.

A device instance is considered a dummy if one or more of the following conditions are met:

- The instance has one of the following properties set to `t`:
  - `lxDummy`
  - `ignore`
  - `lvsIgnore`
- The instance has the `lxDummyOwner` property.
- The instance is not bound to the schematic. This behavior is controlled by a the `modgenCreateUnboundAsDummies` environment variable. The default value is `t`. In this state, the instance is considered a dummy. When set to `nil`, the instance is not considered a dummy.
- The instance gate net is the same as that assigned to the environment variable `modgenDummyNet`.

This section covers the following topics about creating and editing dummy devices:

- Creating Dummy Devices
- Adding Dummy Device Rows or Columns
- Adding Dummy Devices to the Array
- Backannotating Dummy Devices
- Removing Dummy Devices
- Deleting Dummies

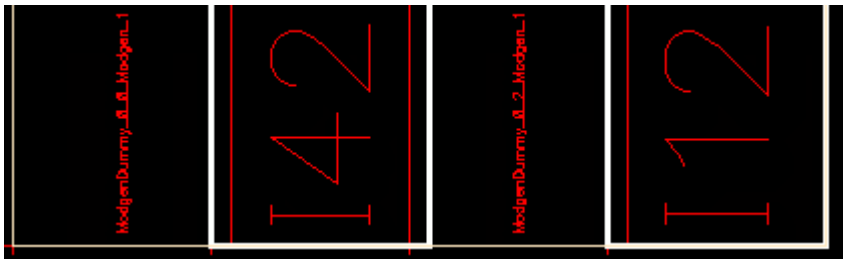
Creating Dummy Devices

To create dummy devices, with the Modgen editor open:

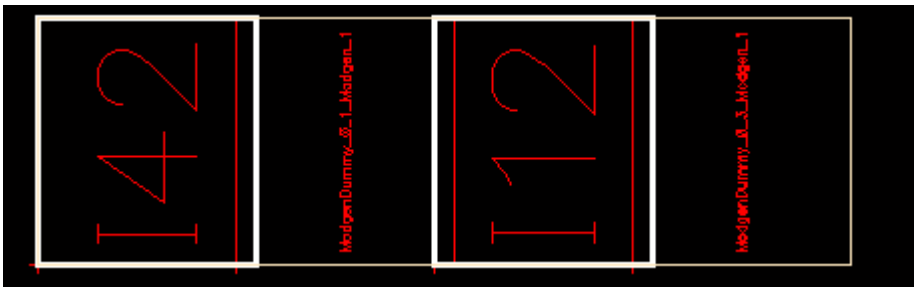
1. Select one or more Modgen instances.
2. Click the arrow next to the *Add Dummy*  button on the toolbar.

   Alternative method: If the Modgen editor is not open, use the Modgen on-canvas command *Place—Modgen* and select the required location option.
3. Choose one of the following location options:

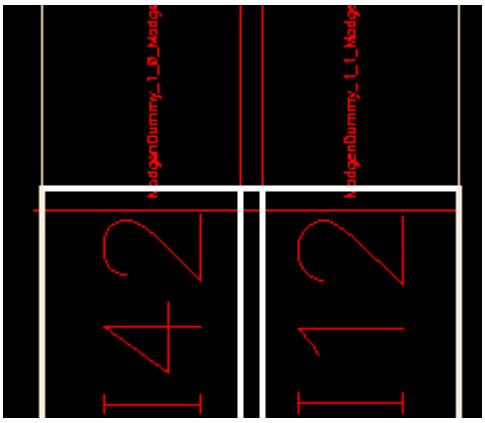
   - **Add Dummy Left:** Adds dummy devices to the left of the selected Modgen instances



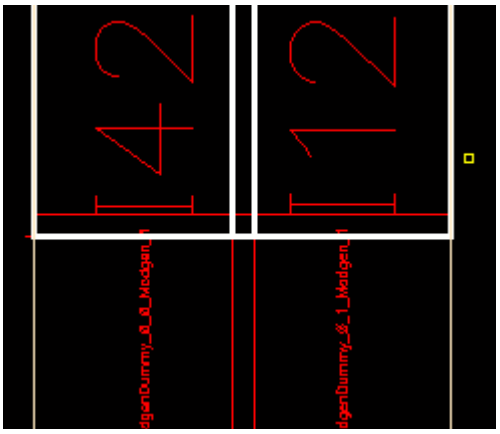
   - **Add Dummy Right:** Adds dummy devices to the right of the selected Modgen instances



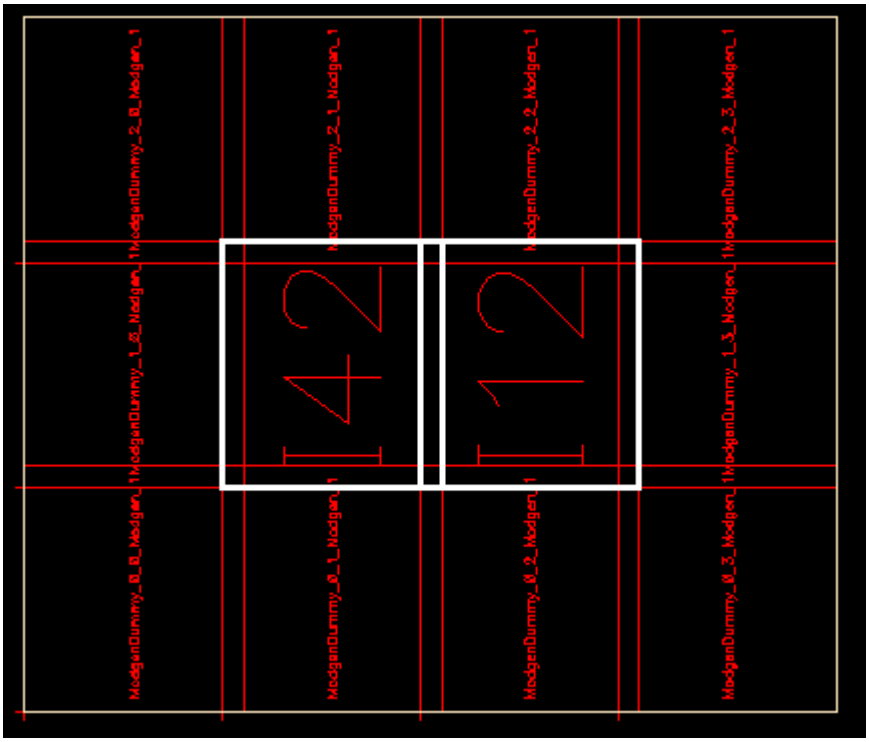
- **Add Dummy Top:** Adds dummy devices above the selected Modgen instances



- **Add Dummy Bottom:** Adds dummy devices below the selected Modgen instances



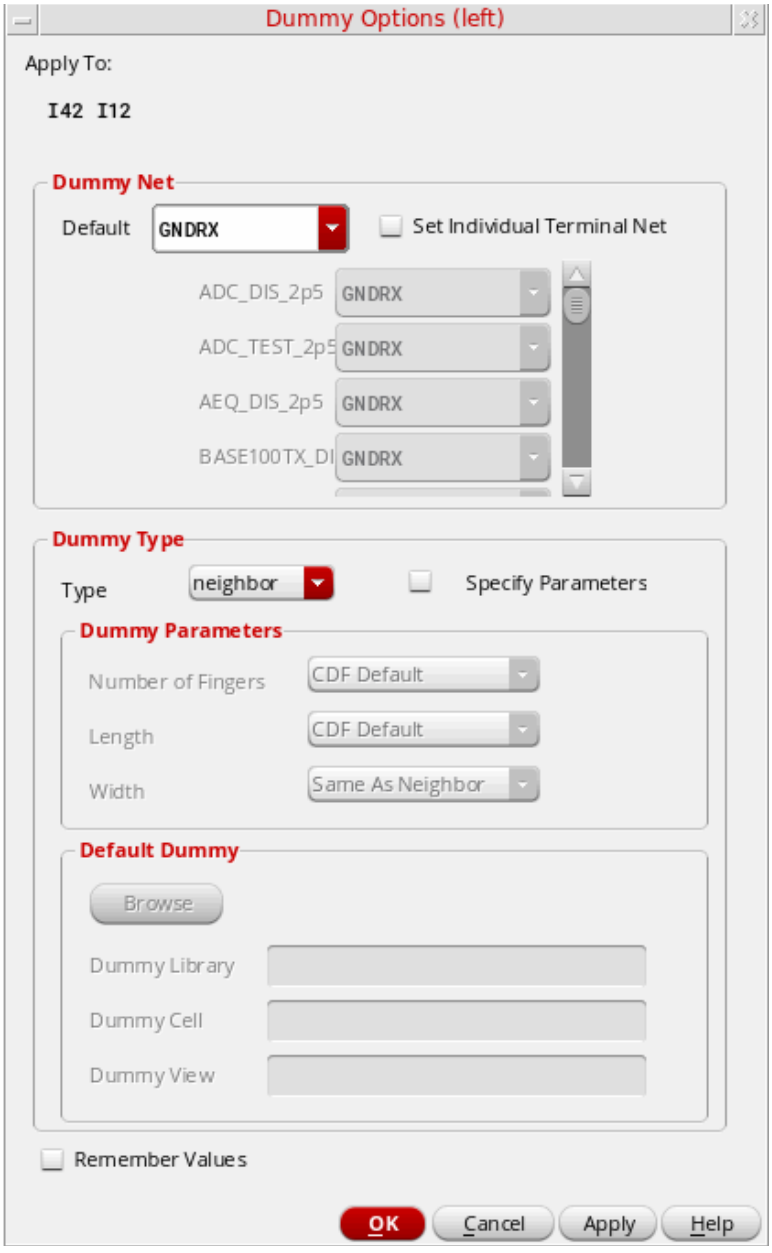
- **Surround Dummies:** A ring of dummies is added around the selected Modgen instances. For more information, see Adding Surround Dummies.



Note: If no Modgen instance is selected, depending on the specified location option, either a dummy row or a dummy column is added to the array of instances. For example, selecting *Add Dummy Left* adds a column to the left of the array, and selecting *Add Dummy Bottom* adds a row of dummies at the bottom.

Adding Dummies to Sides

On selecting a dummy side, the Dummy Options form is displayed.



1. *Apply To* lists the selected Modgen instances.
2. In the *Dummy Net* section, specify the nets to which you want the dummy terminals attached.

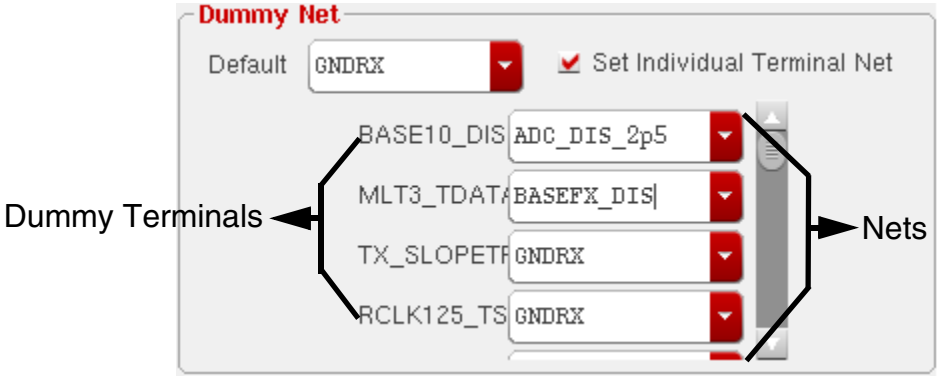
- To attach all selected dummy terminals to the same net, choose a net name from the *Default* combo box, which lists all nets that are connected to objects in the active Modgen.

  To choose a net that is present in the cellView, but not in the current Modgen, type the net name in the *Default* combo box. The entry is validated against all existing nets in the current cellView. If a matching net is not available in the current cellView, then a warning message is issued and the net is created.

  Corresponding Environment Variable: modgenDummyNet

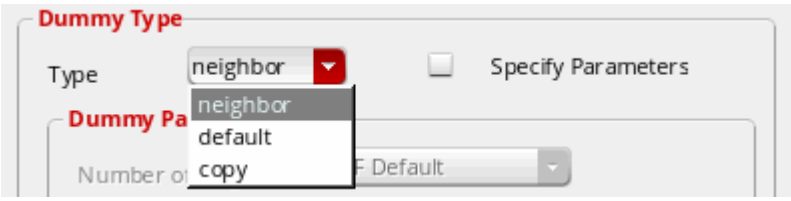
- To connect individual dummy terminals to different nets, enable the *Set Individual Terminal Net* check box.

  A list of dummy terminals that are available in the current design is displayed. Use the combo box beside each dummy terminal name to specify the net to which it needs to be connected.



  When *Set Individual Terminal Net* is enabled, the *Default* combo box is also still available. If you choose a different net in the *Default* combo box, then the selected net is applied only to the dummy terminals for which you have not specified individual terminal nets.

3. In the *Dummy Type* section, from the *Type* cyclic field, choose one of the following options:



- neighbor: The master `lib:cell:view` of the dummy will be the same as the neighboring device.
- default: you can specify a different master `lib:cell:view` for the dummy devices.
- copy: Creates identical dummies of the selected instances. In this mode, the dummy parameters and default values of the source instances are used. Different values cannot be specified.

Corresponding Environment Variable: modgenPhysConfigs

4. Select *Specify Parameters* to edit the default dummy parameters.
5. In the *Dummy Parameters* section, from the *Number of Fingers* cyclic field, choose one of the following:
   - *CDF Default* – The default number of fingers, as specified in the CDF, is created
   - *Same As Neighbor* – The same number of fingers as the neighboring device is created
   - *Specify* – You can specify the number of fingers to be created in the box beside the *Number of Fingers* cyclic field
6. From the *Length* cyclic field, choose one of the following:
   - *CDF Default* – The default finger length, as specified in the CDF, is considered
   - *Same As Neighbor* – The length of fingers of the neighboring device is considered
   - *Specify* – You can specify the length of fingers in the box beside the *Length* cyclic field

   Note: Scale factors can be used to specify the length; for example .1u.
7. (IC6.1.8 only)

   From the *Width* cyclic field, choose one of the following:
   - *CDF Default* – The default number of fins, as specified in the CDF is considered
   - *Same As Neighbor* – The width of fingers of the neighboring device is considered
   - *Specify* – You can specify the width of fingers in the box beside the *Width* cyclic field

   Note: Scale factors can be used to specify the width; for example 200n.

   (ICADVM20.1 Only)

In ICADVM20.1, the *Width* cyclic field is replaced by the *Number of Fins* cyclic field. From the *Number of Fins* cyclic field, choose one of the following:

- *CDF Default* – The default number of fins, as specified in the CDF is considered
- *Same As Neighbor* – The number of fins of the neighboring device is considered
- *Specify* – You can specify the number of fins in the box beside the *Number of Fins* cyclic field

Important

The default values for the *Number of Fingers*, *Length*, and *Width / Number of Fins* cyclic fields are determined by the modgenMakeMinDummies environment variable. For more information, see Understanding the Neighbor Dummy Type.

When *Width* represents the total width for the device (number of fingers * finger width), set the modgenWidthParamProportionalToFingers environment variable to `t` to indicate that the finger width of Modgen dummy instances must be proportional to the number of fingers. For FinFET devices, the default value is `no`; for non-FinFET devices, the default value is `yes`.

8. If you chose the Dummy *Type* as *default*, click *Browse* under *Default Dummy* to browse for the library, cell, and view you want for the dummy devices.

   Corresponding Environment Variable: modgenDummyLib, modgenDummyCell, modgenDummyView

9. Select the *Remember Values* check box to save these values for all dummy devices.

   These values are saved on a per-user basis. So the Module Generator will always load these values until you overwrite them with new saved values.

   Corresponding Environment Variable: modgenRememberDummyVals

10. Click *OK* or *Apply*.
    - Clicking *OK* applies the current settings and closes the form.
    - Clicking *Apply* commits the changes, but keeps the form open for further modifications.

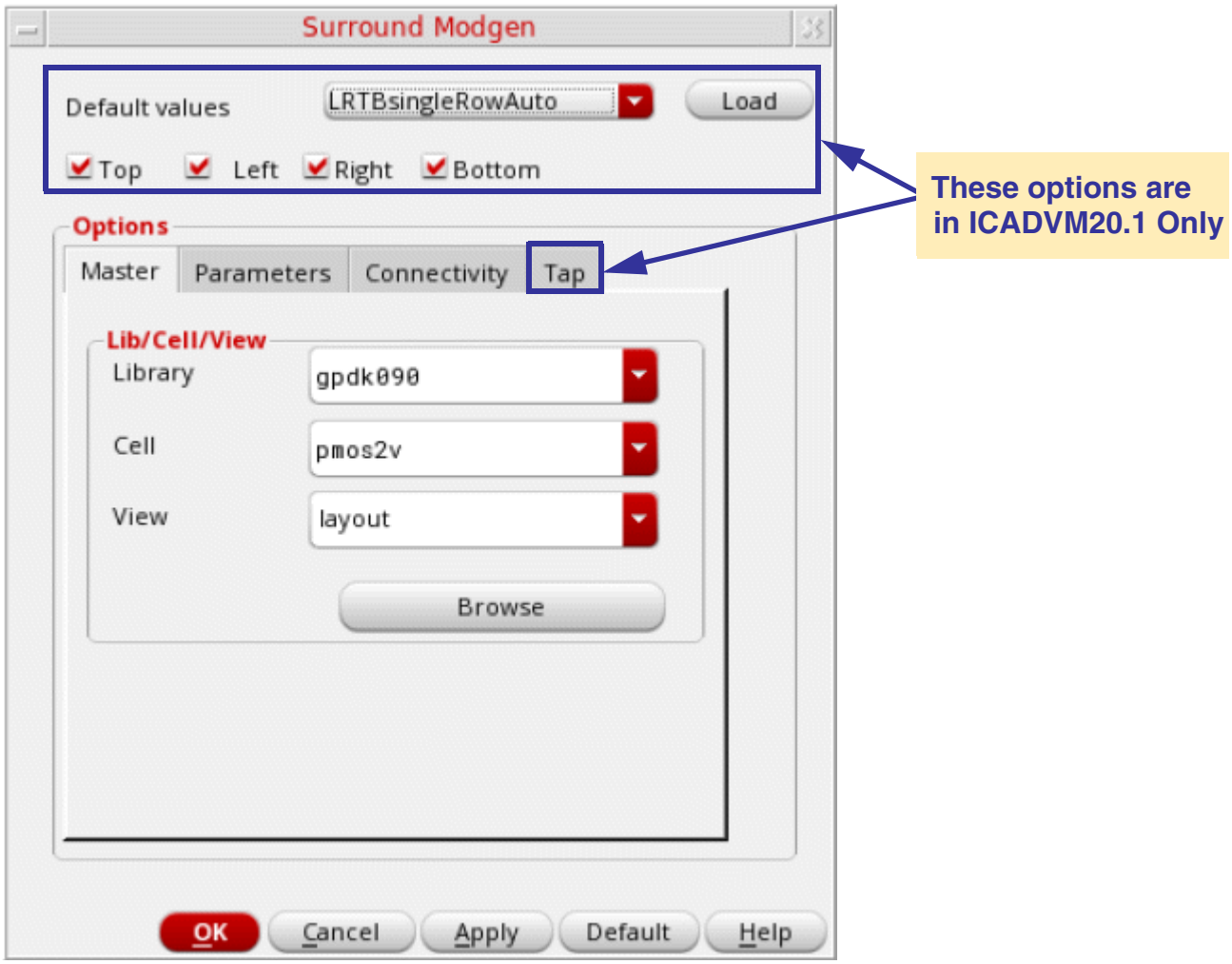
(ICADVM20.1 Only) For FinFET devices to support operations such as dummy creation, their component class must be set to NFIN or PFIN. To define these component types and assign devices to them, use either the Configure Physical Hierarchy command or the library and attributes mapping (LAM) file.

In addition, the *Number of Fins* parameter must be included in the value of transistorWidthParamNames environment variable:

```
envSetVal("layoutXL" "transistorWidthParamNames" 'string "nfin nFin
w wr")
```

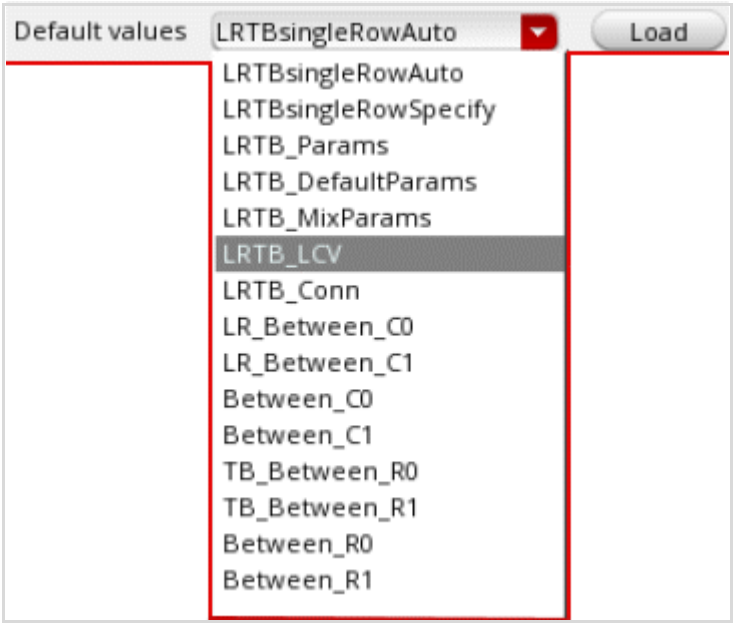
Adding Surround Dummies

To add surround dummies around the selected Modgen instances, select *Surround Dummies* from the *Add Dummies* drop-down list in the Modgen editor. The Surround Modgen form is displayed.



This form lets you perform the following tasks:

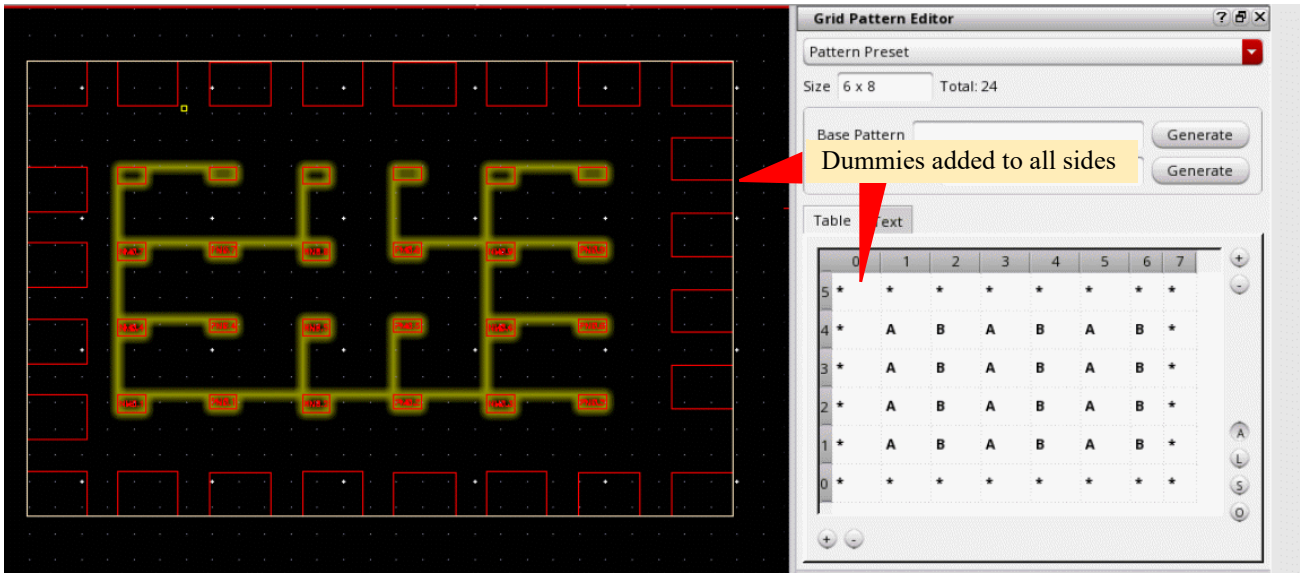
- (ICADVM20.1 Only) **Load default values:** *Default values* provides a list of pre-registerd SKILL call back functions from which you can load the values. This is helpful when you want to apply the same style across designs. You can load the values and edit them as per your requirement.



To load a preset:

1. Select the required preset.
2. Click *Load*.

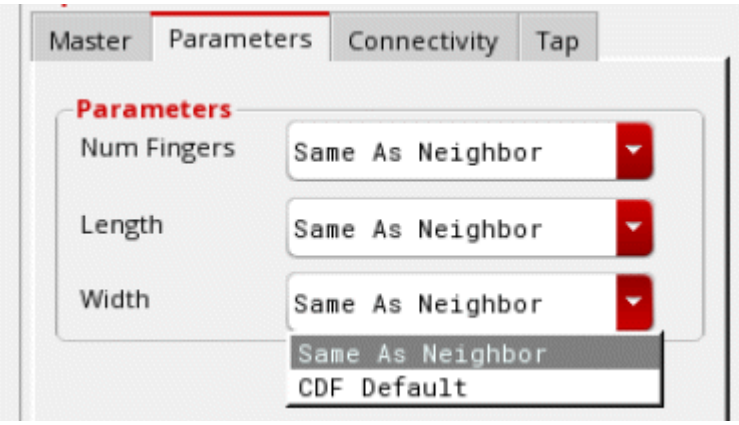
- (ICADVM20.1 Only) **Select the sides**: Choose one or more sides (*Top*, *Left*, *Right*, and *Bottom*) to add surround dummies. In the following example, dummies are added to all four sides of the Modgen constraint.



- **Specify the cellview**: On the *Master* tab (default), specify the *Library*, *Cell*, and *View* to be used to create custom dummy devices.

  Note: *View* lists only the `maskLayout` type views.

- **Set the dummy parameters:** The *Parameters* tab provides options to specify dummy parameters.

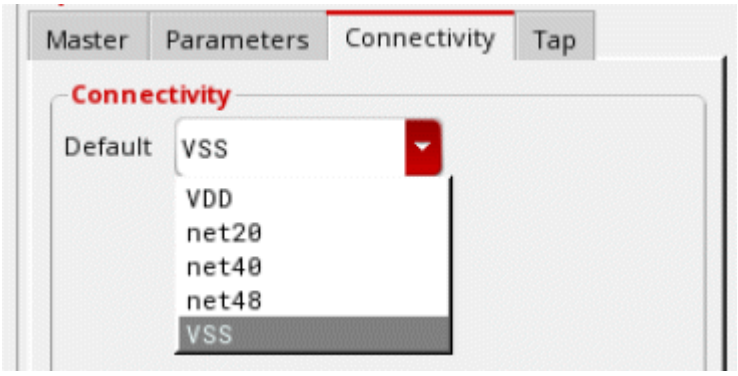


  Specify the following options:

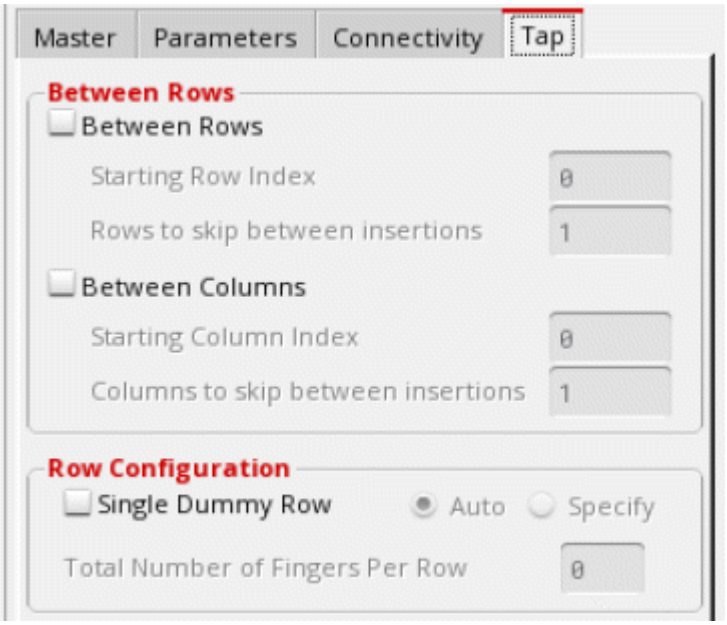
  - **Number of fingers** (*Num Fingers*) in each dummy. You can either type the required value in the field or choose from the following values:
    - ***Same As Neighbor***: The same number of fingers as the neighboring device are created.
    - ***CDF Default***: The default number of fingers, as specified in the CDF, are created.
  - ***Length*** of dummy fingers. You can either type the required value in the field or choose *Same as Neighbor* or *CDF Default*.
  - ***Width*** of dummy fingers. You can either type the required value in the field or choose *Same as Neighbor* or *CDF Default*.

    Note: In ICADVM20.1, the *Width* cyclic field is replaced by the *Number of Fins* cyclic field. Valid values are the same as *Width*.

- **Define Connectivity:** On the *Connectivity* tab, choose the net to which all dummy terminals must connect. If left blank (no net is selected), no terminals are created.

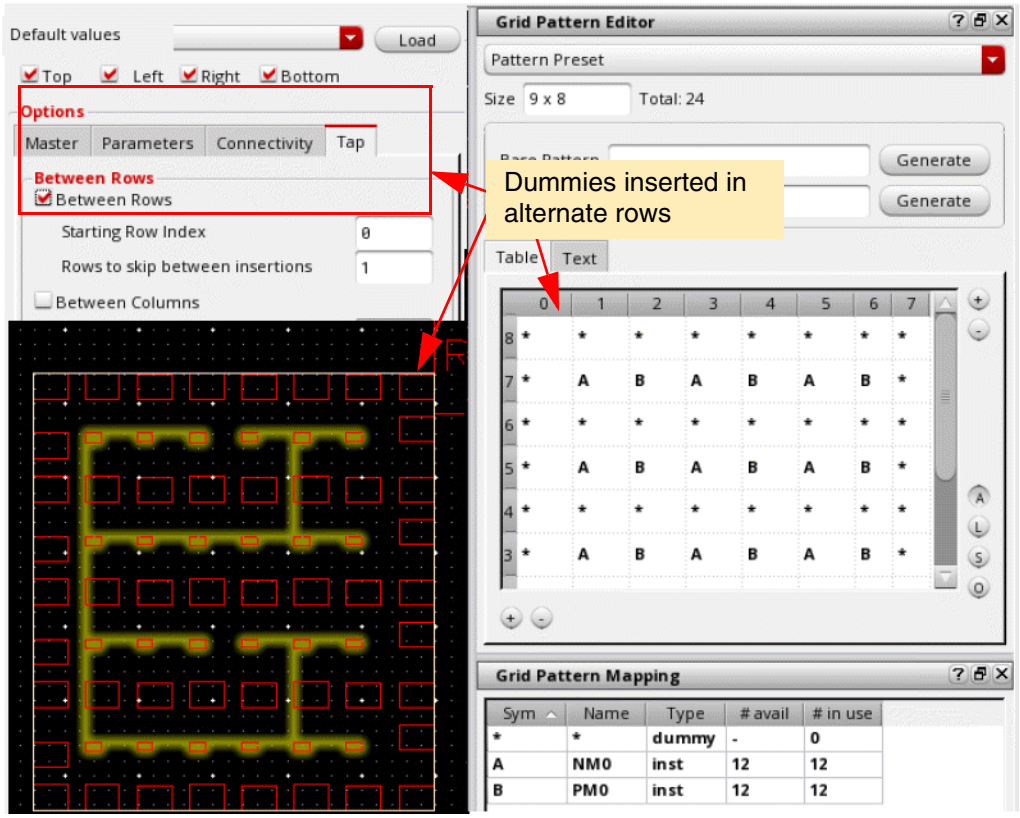


- (ICADVM20.1 Only) **Insert Tap Cells:** The *Tap* tab provides the following options:

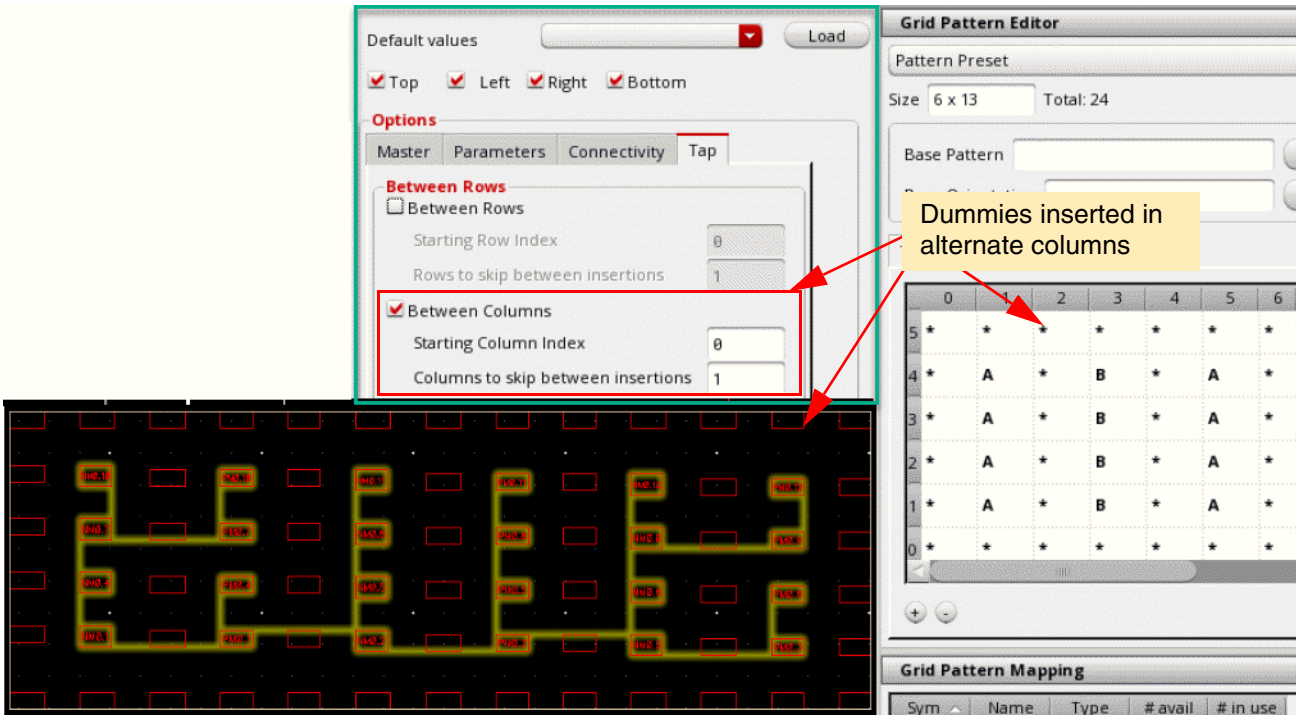


The options let you:

- Insert dummies *Between Rows* and *Between Columns* by specifying the starting row or column index and the number of rows or columns to be skipped between dummy insertions.



- Insert a *Single Dummy Row* instead of individual dummies for each device. You can also specify the number of fingers to be inserted in each dummy row.



Understanding the Neighbor Dummy Type

If the modgenMakeMinDummies environment variable is set to `nil`, then the dummy is created using the lib/cell/view of the neighboring device.

If the `modgenMakeMinDummies` environment variable is set to `t`, the following happens when a neighbor dummy is added to a MOSFET:

- If the dummy's location is on the right or left, a single finger, minimum length device is created to serve as the dummy.
- If the dummy's location is on the top or bottom, a minimum width device with the same number of fingers as the neighbor is created to serve as the dummy.

Note: When setting the parameters to create minimum length and minimum width dummies, the CDF callbacks will be invoked.

If added to a resistor:

- If the dummy's location is on the right or left, a single segment, minimum length device is created to serve as the dummy.

- If the dummy's location is on the top or bottom, a minimum width device with the same number of segments is created to serve as the dummy.

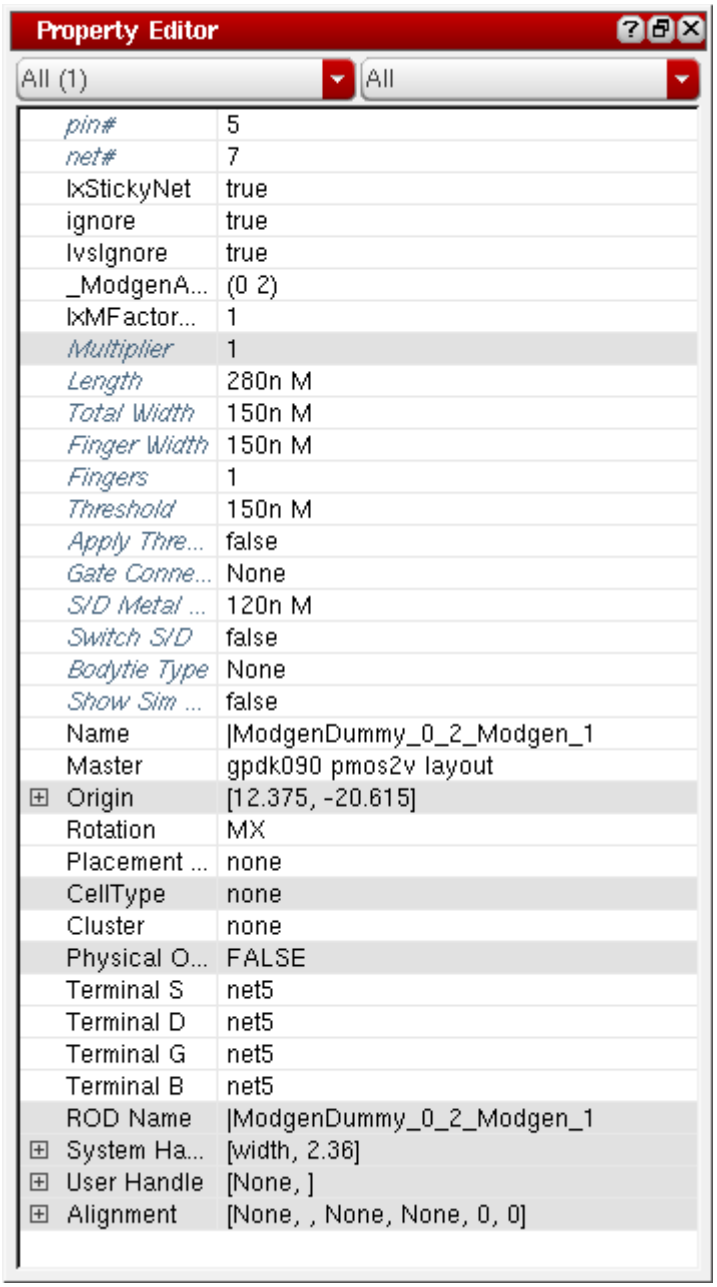
In this case, the parameters for fingers (`lxFingeringNames`), width (`transistorWidthParamNames`), and s-factor (`sfactorNames`) are used to create the device.

The `lxFingeringNames` environment variable specifies parameter names that define Pcell gate fingering in VLS -XL. In addition, this environment variable is used by the Modgen placer when adding dummy devices to MOSFETs with the modgenMakeMinDummies environment variable set to t.

For more information, refer to the *Virtuoso Layout Suite XL User Guide*.

Changing Dummy Properties

You can change parameters of dummy instances using the Property Editor assistant in the Modgen editor window. To open the Property Editor assistant, right-click the menu area and select the *Property Editor* option from the shortcut menu.



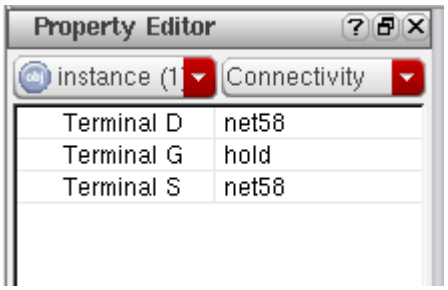
After changing the parameter values, these changes are retained in the `dummyParams` member parameter in the constraint.

Defining Connectivity for Dummies

Use one of the following methods to reset the connectivity of dummies.

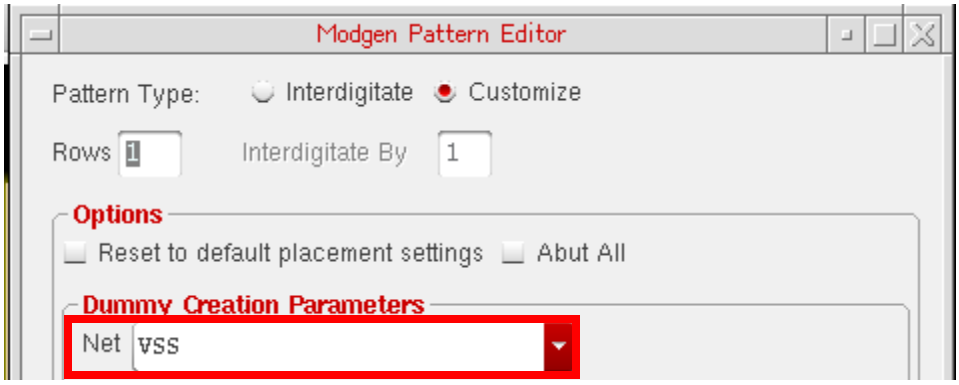
Using the Property Editor Assistant

Use this method to individually reset connectivity of terminals within a dummy. Each terminal can be connected to a different net. To reset connectivity using this method, open the Property Editor assistant by right-clicking the menu area and selecting the *Property Editor* option from the shortcut menu. Then, filter dummies by connectivity and change the net name.



Using the Modgen Pattern Editor

Use this method to connect all terminals within a dummy to a single net. Select the net name from the *Net* combo box in the Modgen Pattern Editor and click *OK*.



The current settings override all previously-defined connectivity settings.

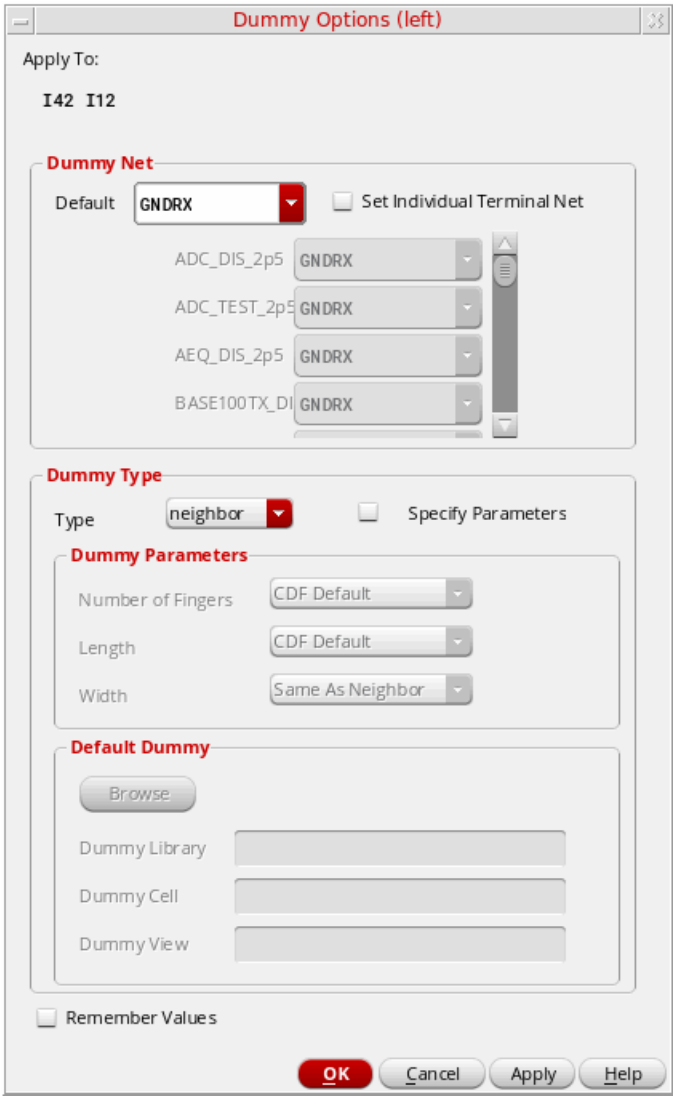
Note: If the *Net* combo box is left blank, then the connectivity definition in the Property Editor assistant remains valid.

Adding Dummy Device Rows or Columns

To add dummy devices around an array:

1. Select any device in an outer row or column of the array.

   Modgens provides the ability to add an entire row or column of dummy devices by selecting only one reference device. For instance, you can select any device in the left column and select *Add Dummy Row/Column* > *Left* to add an entire column of dummies to the left of the Modgen. The Dummy Options form is displayed.



   You can either set up the dummy options or accept the default options.

2. Select the *Specify Parameters* check box in the *Dummy Parameters* section to specify the number of fingers, length, and width for all dummy devices. If the *Specify Parameters* check box is not selected, then the default values for these fields is determined by the modgenMakeMinDummies environment variable. For more information, see Understanding the Neighbor Dummy Type.

Tip

The *Specify Parameters* check box is available only when the Dummy Options form is invoked by selecting *Add Dummy Row/Column* button.

3. Modify dummy device options as desired. For more information, see Adding Dummies to Sides.
4. Click *OK* or *Apply*.
   - Clicking *OK* applies the current settings and closes the form.
   - Clicking *Apply* commits the changes, but keeps the form open for further modifications.

   The Module Generator places the dummy devices as directed.

Adding Dummy Devices to the Array

You can also choose to add dummy devices to the entire array.

➤ Select any device in an outer row or column of the array.

➤ On the Modgen toolbar, click the arrow next to the *Add Dummy Row/Column*  button and choose the location.

Backannotating Dummy Devices

Modgen dummy devices in the layout can be backannotated to their corresponding schematic to keep the two views synchronized. If the dummy layout instance is already bound to a symbol in the schematic, no back annotation is performed for that instance.

Note: You can backannotate dummies even when the layout view is open in the read-only mode.

At any point, you can run *Check Against Source* to check for any instance mismatches with the schematic.

To know more about dummy backannotation, see BackAnnotating Dummy Instances in *Virtuoso Layout Site XL User Guide*.

Removing Dummy Devices

To remove a dummy device, do one of the following:

- Select the dummy device and click the *Delete*  button on the toolbar.
- Select the dummy device and press the *Delete* key.

Select *Place—Modgen—Remove Modgen Surround Dummies* to remove surround dummies in the layout canvas.

Note: All empty rows and columns in the Modgen are deleted.

Deleting Dummies

In the Modgen toolbar, click the arrow next to the *Add Dummy Row/Column*  button and click one of the following buttons:

- The *Delete All Dummy Rows/Columns* button to delete all dummy rows and columns of the Modgen. Alternatively, use the `mgDeleteAllDummyRowColumnCB` command.
- The *Delete All Dummies* button to delete all dummies in the design.

Note: Rows and columns with empty cells are deleted.

Adding Body Contacts

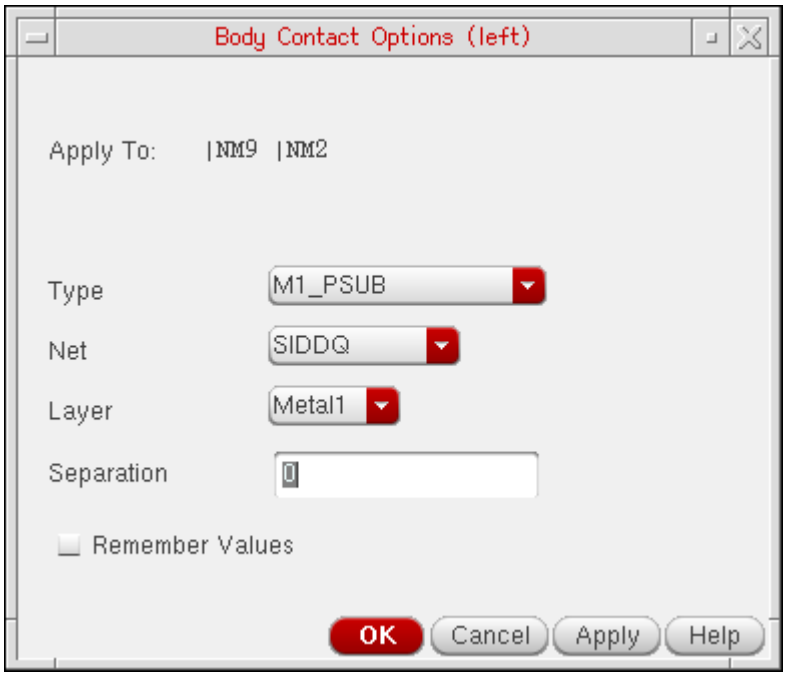
You can define properties for body contacts in the Body Contact Options form, then add body contacts to the array in the layout.

Defining Body Contact Properties

In the Body Contact Options form, you can specify the type, net, and separation distance for body contacts. Because the body contact created by the Modgen tool matches the height of the member instance, you also need to specify which reference layer the body contact matches. If you do not specify a reference layer, the Modgen tool will match the body contact to the height of the bounding box of the member instance. The Modgen tool also uses the reference layer to calculate the separation distance.

1. In the toolbar, click the *Add Body Contact* button.

   The Body Contact Options form appears.



2. From the *Type* cyclic field, choose the type of body contact you want.

   This field is populated from the technology file.

3. The *Net* combo box lists the nets that are connected to objects in the active Modgen. Choose the net to which you want body contacts attached to. To choose a net that is

present in the cellView, but not in the current Modgen, type the net name in the *Net* combo box. The entry is validated against all existing nets in the current cellView. If a matching net is not available in the current cellView, then a warning message is issued and the net is created.

4. From the *Layer* cyclic field, choose the reference layer for the body contacts.

   Environment Variable: modgenRememberBodyContactVals

5. In the *Separation* field, enter the distance in microns that you want between body contacts and devices. This value is added to minimum DRC to space the body contact.

   Note: You can also specify a negative value in this field. When you specify a negative value, the body contact is placed inside of the value that Modgen is calculating as a legal DRC distance.

6. Select the *Remember Values* check box to save these values for all dummy devices.

7. Click *OK* or *Apply*.

   - Clicking *OK* applies the current settings and closes the form.
   - Clicking *Apply* commits the changes, but keeps the form open for further modifications.

Adding Body Contacts

You have two options for adding body contacts. You can add them in the layout on a row or column basis, or you can add all body contacts using the Modgen toolbar.

Adding Body Contact Rows and Columns

You can add body contacts on a row or column basis in the layout.

1. Select a device in a row or column of the array.

   The body contacts you can add are dependent on which device you select.

2. Do one of the following:

   - Right-click and choose *Add Body Contact – <Location>*.
   - On the toolbar, click the arrow next to the *Add Body Contact* icon and choose a location option.

Adding Body Contacts to a Selected Device

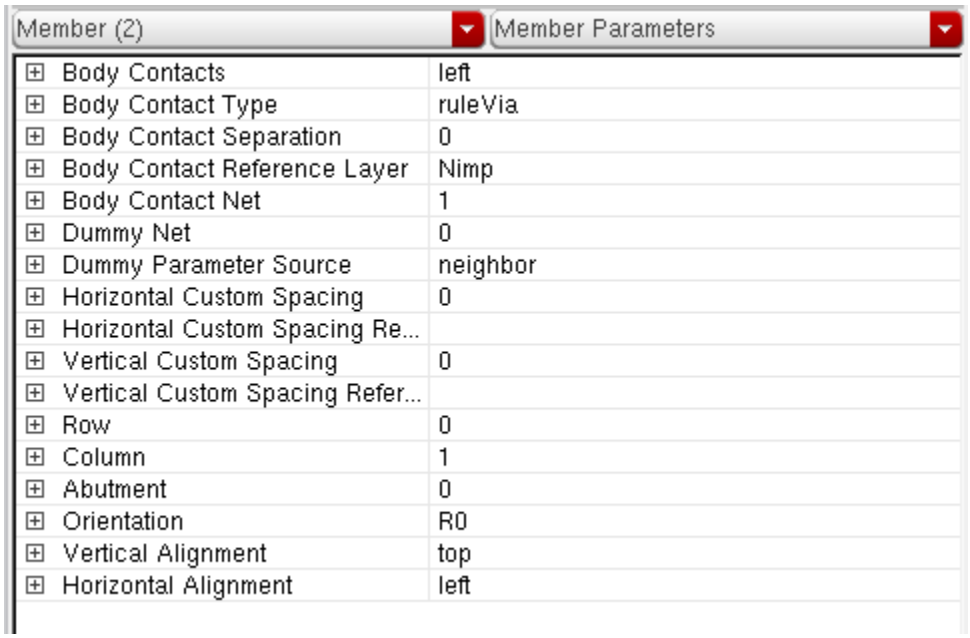
You can also add body contacts to the selected device(s).

1. Select one or more devices.
2. Do one of the following:

   - Right-click and choose *Add Body Contact – <Location>*.
   - On the toolbar, click the arrow next to the *Add Body Contact*  button and choose a location option.

Editing Body Contact Properties

You can edit body contact parameters using the Property Editor assistant in the Modgen editor window. To open the Property Editor assistant, right-click the menu area and select the Property Editor option from the shortcut menu.



The modified parameter values are applied to the selected body contact.

Removing Body Contacts

To remove body contacts, do one of the following:

- Select the body contact and click the *Delete* button on the toolbar.
- Select the body contact and press the *Delete* key.

Defining Grid Placement

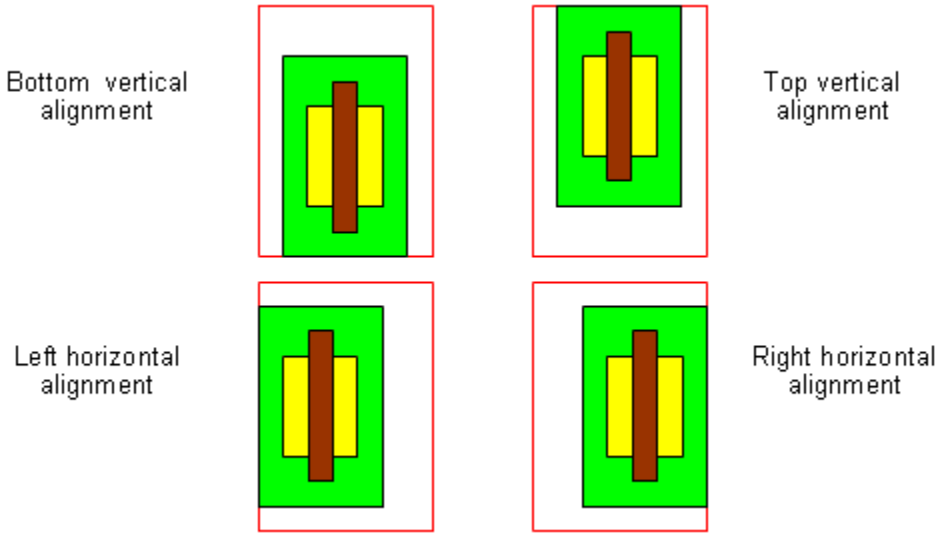
The Grid Placer is a back end placement engine for Modgen to generate the DRC-correct array placement. The Grid Placer places a set of instances according to their grid indexes into an array-like placement compacted to the minimum DRC and constraint-correct spacing. In addition, it enables you to specify the array size in terms of the number of rows and number of columns.

As compared to the previous Modgen placement engine, the Grid Placer provides fast and flexible placement, which supports the constraints between member instances, well merging, and complex guard ring creation.

Important

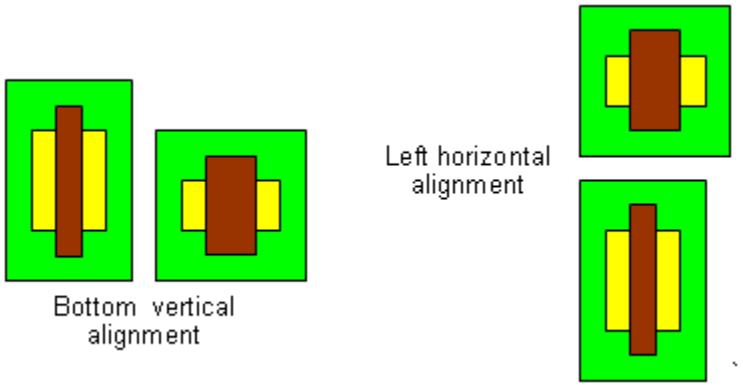
The Grid Placer algorithm scales better for large Modgens than the current cell by cell placement implementation.

With the introduction of the Grid Placer, the semantics of the Modgen's horizontal and vertical alignment member parameters has changed. Earlier, alignment was relative to the sides of a grid cell as shown below:



With the Grid Placer, the alignment is relative to a member's left or bottom neighbor. Therefore, setting a member's alignment to the left aligns its left edge with the left edge of the

member below it. If the member is in the bottom row, vertical alignment had no effect. This is illustrated in the image below:



Typically, Modgens are snapped to the manufacturing grid resolution as specified in the technology file when *Generate From Source* or *Generate Selected From Source* is used.

You can also snap Modgen origin to the snap spacing specified by the modgenUseSnapSpacing environment variable in the `.cdsinit` file.

Note: The Modgen origin is the origin of the figGroup, which is the lower-left corner of the `figGroup` bbox. The snapping is with respect to the Modgen origin.

Placing Body Contacts

Body contacts for a device in the grid are supported by the Grid Placer. The Grid Placer is not responsible for creating the body contact geometry, but only for the placement of the body contacts. You can define the relative position, such as left, right, top, or bottom and spacing between the body contacts and member instance of the grid. It also incorporates the DRC between the body contact and neighbor grid geometry as a hard constraint.

The Grid Placer also supports automatic minimum DRC spacing based placement for a body contact. This minimum DRC spacing would be obtained from the process rules on the database object.

Note: If there is a conflict between the DRC rules of neighbor member instances, the Grid Placer will use the maximum value as the DRC rule between the two instances.

Calculating DRC Rules

You can use the Process Rule Editor to query minimum spacing rules for a single or different layer to calculate the DRC rules between the neighboring grid objects.

For same layer, you need to use the constraint definition `minSpacing`, while for different layers you need to use `minClearance`. While setting DRC rules between two devices, you need to consider the maximum of the `minSpacing` or `minClearance` values obtained from the Process Rule Editor on each of the two devices.

Corner Case Conditions for Grid Placer

The parameters entered by users to define a Modgen can be put in two categories: one that defines the overall Module, and the other that defines each grid.

The module-level input parameters include the number of rows/columns, guard rings, row routing spacing estimation, and constraints between grid objects.

The grid-level input parameters define each grid object and its neighbors. This includes, row/column index, alignment, custom spacing, abutment, body contact, well layer, and DRC rules.

A combination of both these input parameters can create conflicts.

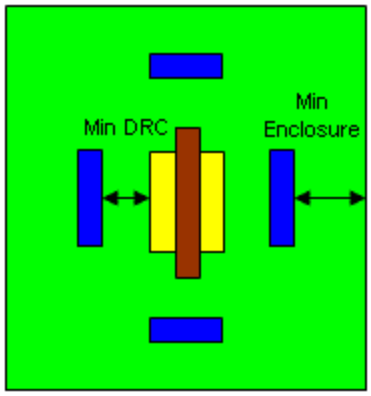
Note: In Grid Placer, if a member which is abutted to its neighbor also has custom spacing, then abutment will take precedence and the custom spacing will be ignored.

Body Contact v/s Well Merge

Let’s consider a scenario of a possible conflict between body contact and well merge.

Scenario 1:

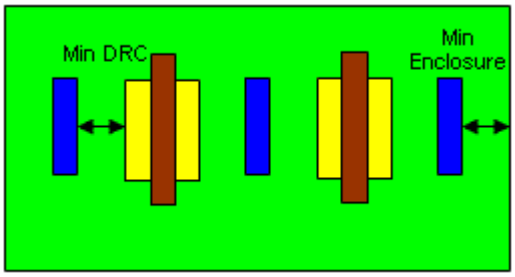
If the well-merge option is on, and the body contact has either the same well layer as the instance or no well layer in its geometry. Then, you enclose the body contact geometry with the well layer of the instance as shown in the image below.



Min-enclosure process rules need to be maintained between the well layer and the body contact geometry. However, if the body contact has a different well layer than the instance, you do not enclose the body contact with the well geometry of the instance.

Scenario 2:

If the well-merge option is on, and two neighboring instances in the grid sharing a common body contact also share the same well layer, then the well layers of both instances are merged maintaining minimum enclosure and minimum DRC rules as shown in the image below.



However, if the shared body contact happens to have a different well layer as the neighboring instances, well-merging would be disabled in this special case.

Note: If instance members in a Modgen have the same well layer but different bulk connectivity, then well merging is not allowed for these instances. Well merging is only applicable for instances that have the same well layer and same bulk connectivity.

Snapping Instances in Modgens (ICADVM20.1 Only)

To support FinFETs, the layout editor has been enhanced to automatically snap the Pcells in Modgens to the snap pattern. The entire Modgen block is, therefore, snapped to the top-level snap pattern shape. For more information about snap patterns, see Snapping Objects to Local Snap Pattern Shapes.

Specifying Guard Rings

Guard rings are used to enclose one or more objects such as devices or device chains. You can use the Modgen tool to create multipart path (MPP) guard rings and fluid guard rings. Before creating guard rings, ensure that they are defined in the technology file.

For more information about the procedures for creating the following types of guard rings, see:

- Creating Multipath Part (MPP) Guard Rings
- Creating Fluid Guard Rings
- Creating Identical Guard Rings (ICADVM20.1 Only)

Tip

You can define guard rings directly in the layout canvas without opening Modgen Editor. Select *Place—Modgen—Add Modgen Guard Ring* and choose the required guard ring type.

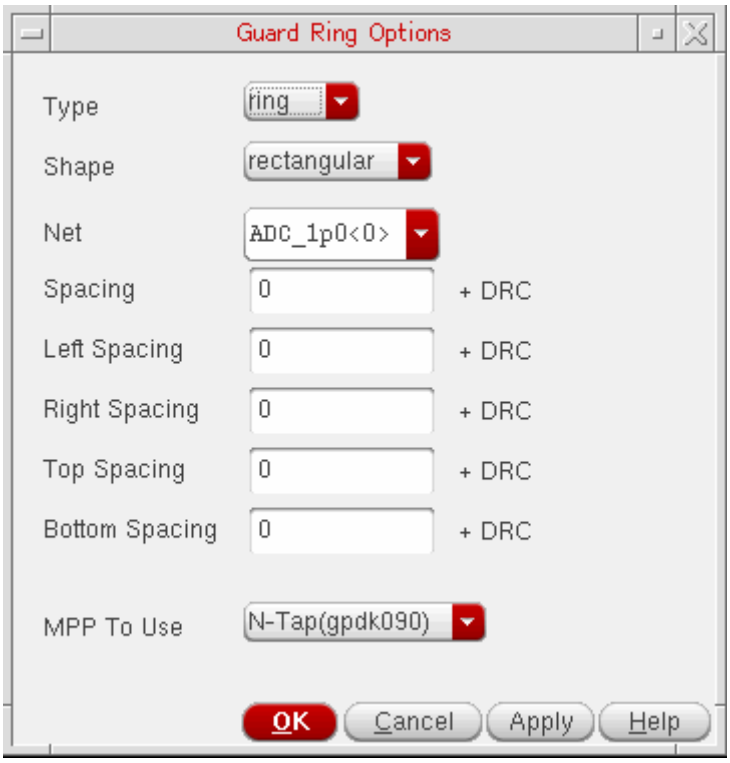
Creating Multipath Part (MPP) Guard Rings

The Modgen tool can generate a guard ring based on MPPs defined in the technology file. In the Guard Rings page of the Module Generator form, you can specify the guard ring type, shape, net, separation distance, and MPP.

You can either specify one spacing value that defines the separation between all four sides of the guard ring and the devices, or you can define a different value for each side. This spacing value is in addition to the minimum DRC distance. So, when creating a guard ring, the Modgen tool first finds the minimum DRC location at which to place the guard ring, examining each layer, then adds the specific spacing values to that distance in order to calculate the final location. To specify guard ring properties:

1. On the Modgen toolbar, click the *Guard Ring*  icon and choose *Create MPP Guard Ring*.

The Guard Ring Options form is displayed.



2. From the *Type* drop-down field, select the type of guard ring you want.

   The Type drop-down field has the following options

   a. *none*: Indicates no guard ring and can be used to delete an existing one.

   b. *ring*: Indicates a single guard ring around the entire Modgen.

   c. *pane*: Indicates the new window pane configuration that includes a guard ring around all devices as well as the entire Modgen.

3. From the *Shape* field, select the shape of guard ring.

4. From the *Net* combo box, select the net you want the guard ring attached to.

5. Specify the *Spacing* between the sides of the guard ring and the devices by doing one of the following:

   To specify the same *Spacing* value for all sides of the guard ring:

   a. Enter the distance in microns in the *Spacing* field. By default, the value is `0`.

   To specify different spacing values for each side of the guard ring:

a. Enter a value for the left side in the *Left Spacing* field.

b. Enter a value for the right side in the *Right Spacing* field.

c. Enter a value for the top side in the *Top Spacing* field.

d. Enter a value for the bottom in the *Bottom Spacing* field.

Important

The spacing value is in addition to the DRC spacing rule.

Tip

You can specify a negative *Spacing* value. In this case, the guard ring is placed inside of the value that Modgen calculates as a legal DRC distance.

Important

Once you have entered a value in any of the <Side> *Spacing* fields, all the <Side> Spacing fields apply, and the value in the *Spacing* field is ignored (as it applies to all sides).

6. From the *MPP To Use* cyclic field, choose an MPP for the guard ring.

7. Click *OK* or *Apply*.

- Clicking *OK* applies the current settings and closes the form.
- Clicking *Apply* commits the changes, but keeps the form open for further modifications.

The Module Generator creates a guard ring around the array using the parameters that you specified.

Note: Modgen pane guard ring stripes will honor existing abutment between member instances. Therefore, guard ring stripes will be inserted only between unabutted instances, and not between abutted instances. If user unabuts existing instances, then the pane will be regenerated accordingly, and stripes will be inserted between all instances.

Important

Cluster spacing to guard ring is the sum of spacing and DRC between instance Bbox and edge of guard ring edge.

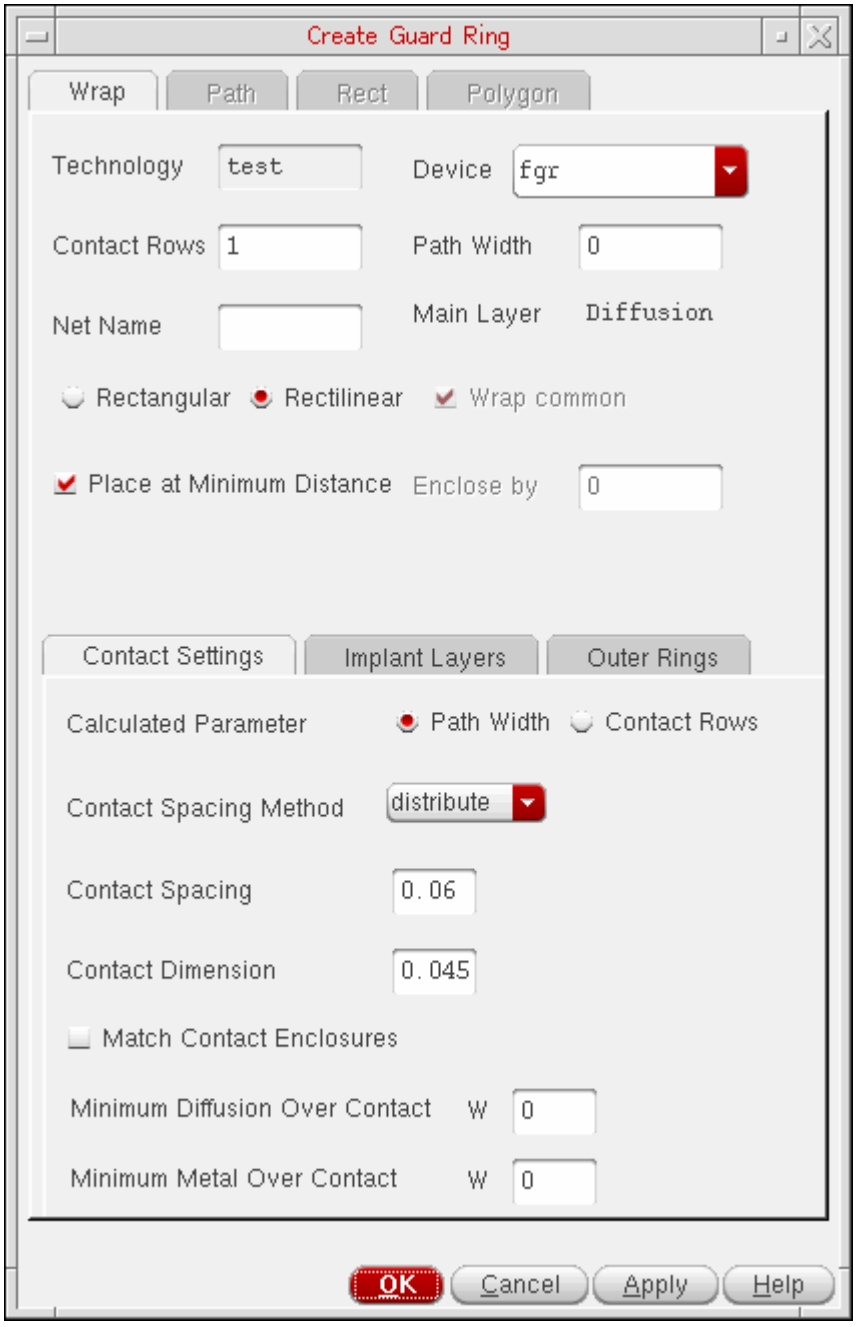
Modgen spacing to guard ring is the sum of spacing and DRC between outermost layer instance data to guard ring edge.

Creating Fluid Guard Rings

In addition to MPP guard rings, you can use the Modgen editor to create fluid guard rings (FGRs) around objects. FGRs are a type of fluid Pcells that can be used to enclose one or more objects such as devices or device chains. Before creating FGRs, ensure that the FGR is installed as a device class in the technology file. For more information about FGR installation, see Installing Fluid Guard Rings in the Virtuoso® Fluid Guard Ring User Guide.

To create an FGR, click the *Guard Ring*  icon on the Modgen toolbar and choose *Create Fluid Guard Ring*.

The Create Guard Ring form is displayed.



Typically, in Virtuoso Layout Suite L (Layout L), FGRs can be created in four different ways: by drawing a path, rectangle, or polygon, or by using the wrap mode. Each of these different modes represent a tab on the Create Guard Ring form. However, the Modgen Editor only supports creation of FGRs in the *Wrap* mode. In this mode, an FGR is created around the objects you select. The fluid guard ring parameters are stored in the Modgen constraint.

For information about options on the *Wrap* tab of the Create Guard Ring form, see Wrap Mode in the Virtuoso® Fluid Guard Ring User Guide.

Hand Editing Fluid Guard Rings

When a Modgen is edited, the fluid guard rings in the Modgen are regenerated automatically. As a result, all the manual edits made to the fluid guard rings are lost. To avoid this, switch to the hand edit mode for the fluid guard rings. In this mode, fluid guard rings are not regenerated each time the Modgen is updated. Therefore, all customizations remain.

To switch on the hand edit mode, either select *Enable Fluid Guard Ring Hand Edit Mode* from the *Guard Ring* menu, or use the `mgFGREnterHandEdit` SKILL function.

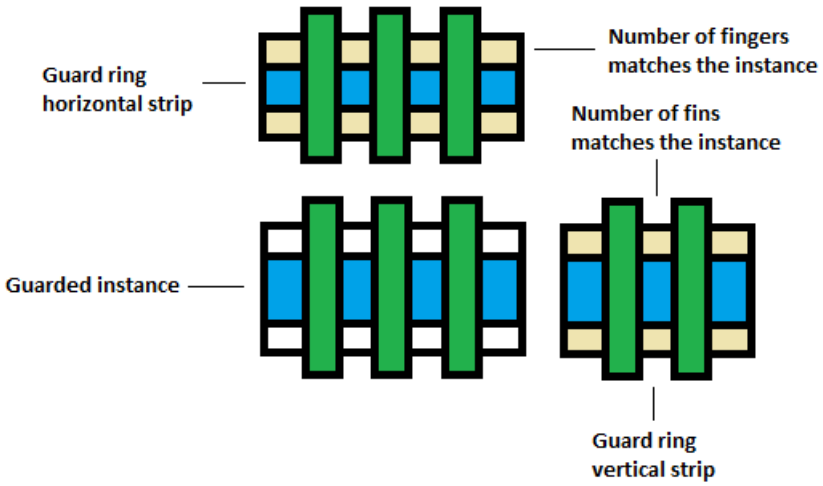
To switch off the hand edit mode, either select *Disable Fluid Guard Ring Hand Edit Mode* from the *Guard Ring* menu, or use the `mgFGRExitHandEdit` SKILL function.

Note: Fluid guard rings can be edited manually both in and out of the hand edit mode.

Creating Identical Guard Rings (ICADVM20.1 Only)

In addition to MPP and fluid guard rings, you can create identical guard rings in Modgens. Identical guard rings are composed of unit cells that match the guarded instances in terms of their number of fingers, number of fins, finger alignment, and other parameters.

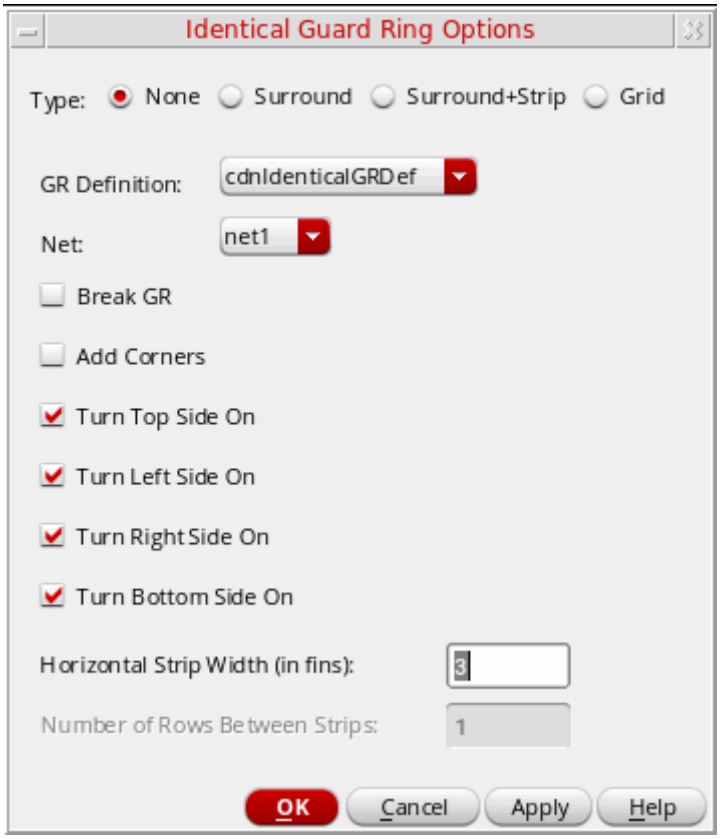
The following diagram depicts identical guard rings:



Select the *Create Identical Guard Ring* command from the *Guard Ring* menu to display the Identical Guard Ring Options form.

Important

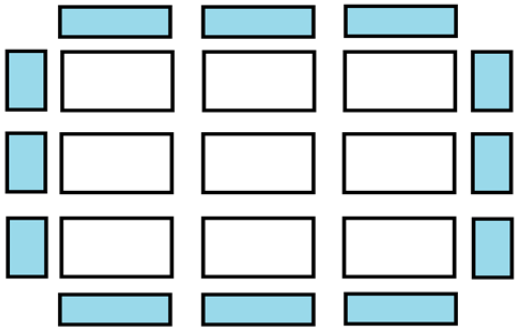
The *Create Identical Guard Ring* command is available only if the PDK that you are using is configured to support identical guard rings. For more information on this capability, contact your Cadence Customer Support representative.



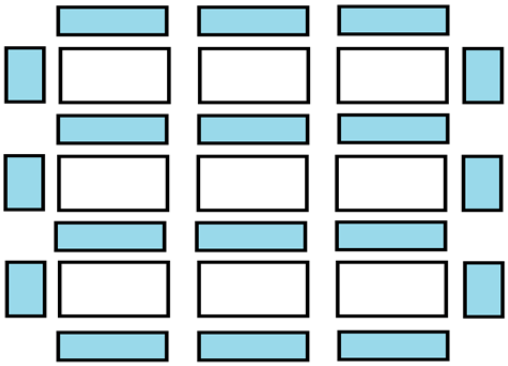
The Identical Guard Ring Options form contains the following options:

- *Type* defines the way in which the identical guard ring surrounds the adjoining Modgen instances. Choose one of the following guard ring types:
  - **None:** Deletes the identical guard ring.

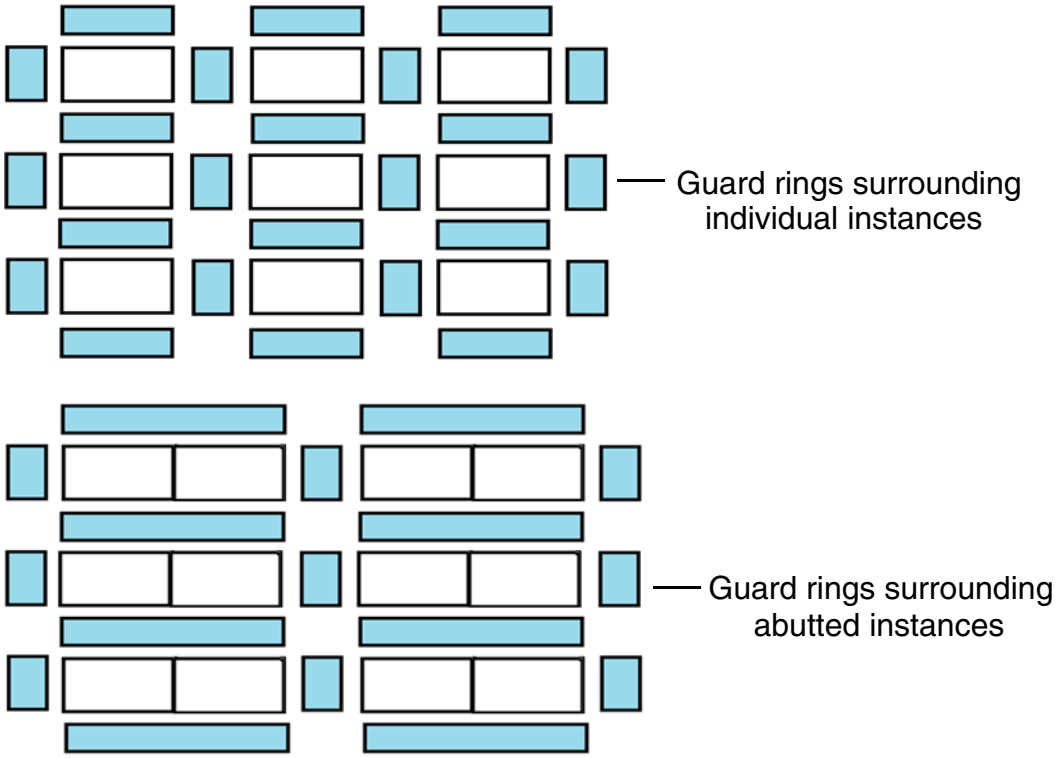
- **Surround**: The identical guard ring surrounds all the instances.



- **Surround+Strip**: The guard ring surrounds all the instances, and strips of guard ring instances are inserted between one or more Modgen rows.



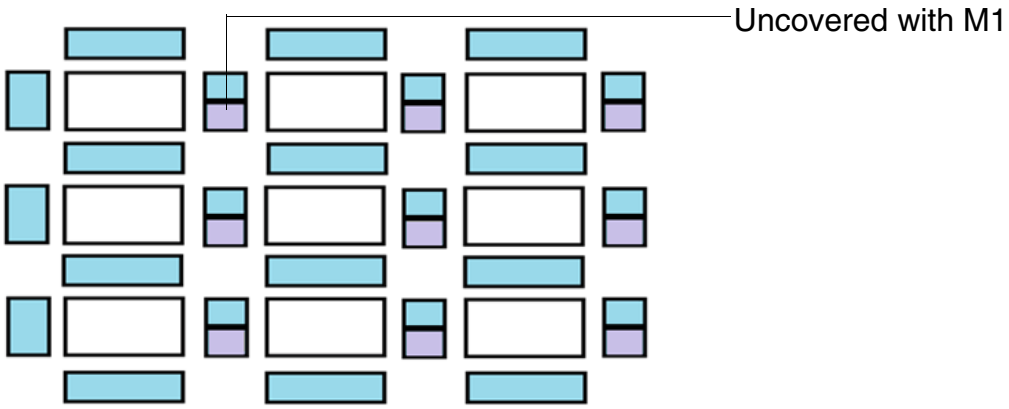
- **Grid:** The guard ring surrounds every instance or group of abutted instances separately.



- *Horizontal Strip Width (in fins)* value specifies the vertical width (as a number of fins) of the horizontal guard ring strips, represented by the horizontal blue rectangles in the diagram above.

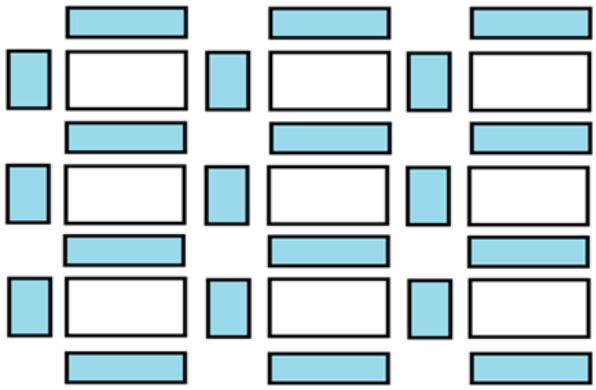
- *GR Definition* is a definition provided by the PDK that defines many aspects of the guard ring. It provides information such as the unit cell `lib:cell:view`, its parameters, and parameter callbacks, which helps Modgens instantiate the guard ring correctly.
- *Net* specifies the net to which the guard ring needs to be connected.

- *Break GR* affects the `Metal1` layer of all vertical guard ring strips except those on the left side of the guard ring. The effect is determined by the *GR Definition*, but usually the `Metal1` layer is trimmed from the bottom as depicted in the following diagram.



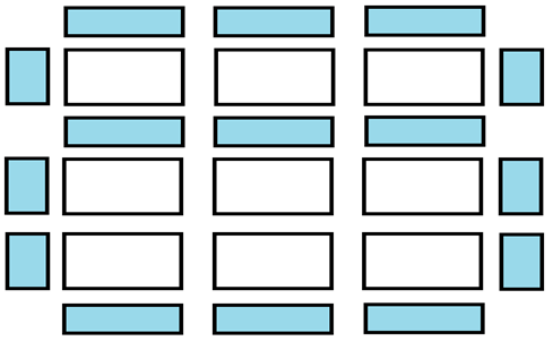
  The *Break GR* option is used to provide spaces for connecting guarded instances via the `Metal1` layer routes.

- *Add Corners* creates guard rings at all available corners around the selected instances.
- The side switches turn on or off the guard ring for the specified sides. For example, the following diagram depicts a *Grid* guard ring in which the right side is turned off.

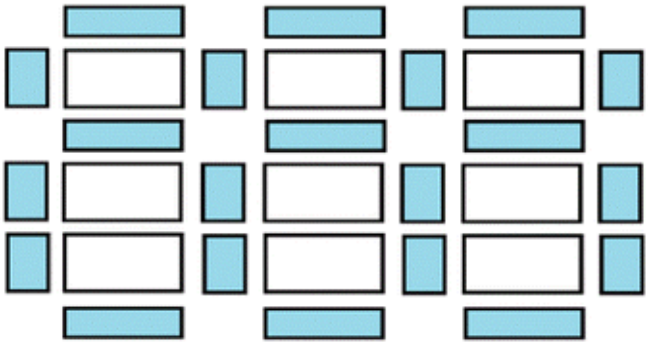


- *Number of Rows Between Strips* is available when *Type* is set to either *Surround+Strip* or *Grid*. It specifies the number of instance rows that are required between strips of guard rings, counting from the bottom row.

Example 1: *Type* is set to *Surround+Strip*; *Number of Rows Between Strips* is set to 2



Example 2: *Type* is set to *Grid*; *Number of Rows Between Strips* value is set to 2.



Modifying a Guard Ring

To modify an existing guard ring:

1. Do one of the following:
   - In the toolbar, click the *Guard Ring*  icon.
   - Right-click outside the array and choose *Edit Guard Ring*.

   The Guard Ring Options form appears.
2. Modify the guard ring options as desired.

   For more information, see Specifying Guard Rings.
3. Click *OK* when finished.

Snapping Fluid Guard Rings to Fin Grids (ICADVM20.1 Only)

When you create an FGR for a FinFET device, the FGR instance automatically snaps to the underlying fin grids if the *Snap Pattern Snapping* check box is selected in the Layout Editor Options form.

For detailed information about snap pattern grids in the layout canvas, refer to the FinFET Support in Layout L chapter of the *Virtuoso Layout Suite L* User Guide.

Removing a Guard Ring

Use one of the following methods to remove an existing guard ring:

Method 1:

1. On the Modgen toolbar, click the *Guard Ring*  icon.
2. Choose *Remove Guard Ring*.

Method 2:

1. Do one of the following:
   - In the toolbar, click the *Guard Ring*  icon.
   - Right-click outside the array and choose *Edit Guard Ring*.

   The Guard Ring Options form appears.
2. From the *Type* cyclic field, choose *none* as the type.
3. Click *OK* when finished.

 Video

For more information, see the Removing a Guard Ring from a Modgen video.

Abutting Devices

For MOSFET member instances, Modgen supports abutment of devices. When you abut a device, Modgen remains aware of connectivity and may flip the device to correctly abut.

You can also abut dummy devices to other devices, including other dummies. Internally, the dummy device's net may be changed from the default specified in the Dummy Options form, but if the dummy device is removed from abutment, the net will revert to the default. When two instances are abutted, any body contacts between them are deleted.

Modgen abutment uses the Layout XL chaining engine. Therefore, the abutment results are subject to the settings of the Layout XL chaining environment variables. For more information about these environment variables, see Layout XL Environment Variables. Notable examples are `chainMirror` and `chainMirrorEquivOrients`. These environment variables are enabled by default, and may therefore result in unintended changes to the orientation of instances.

Prerequisites

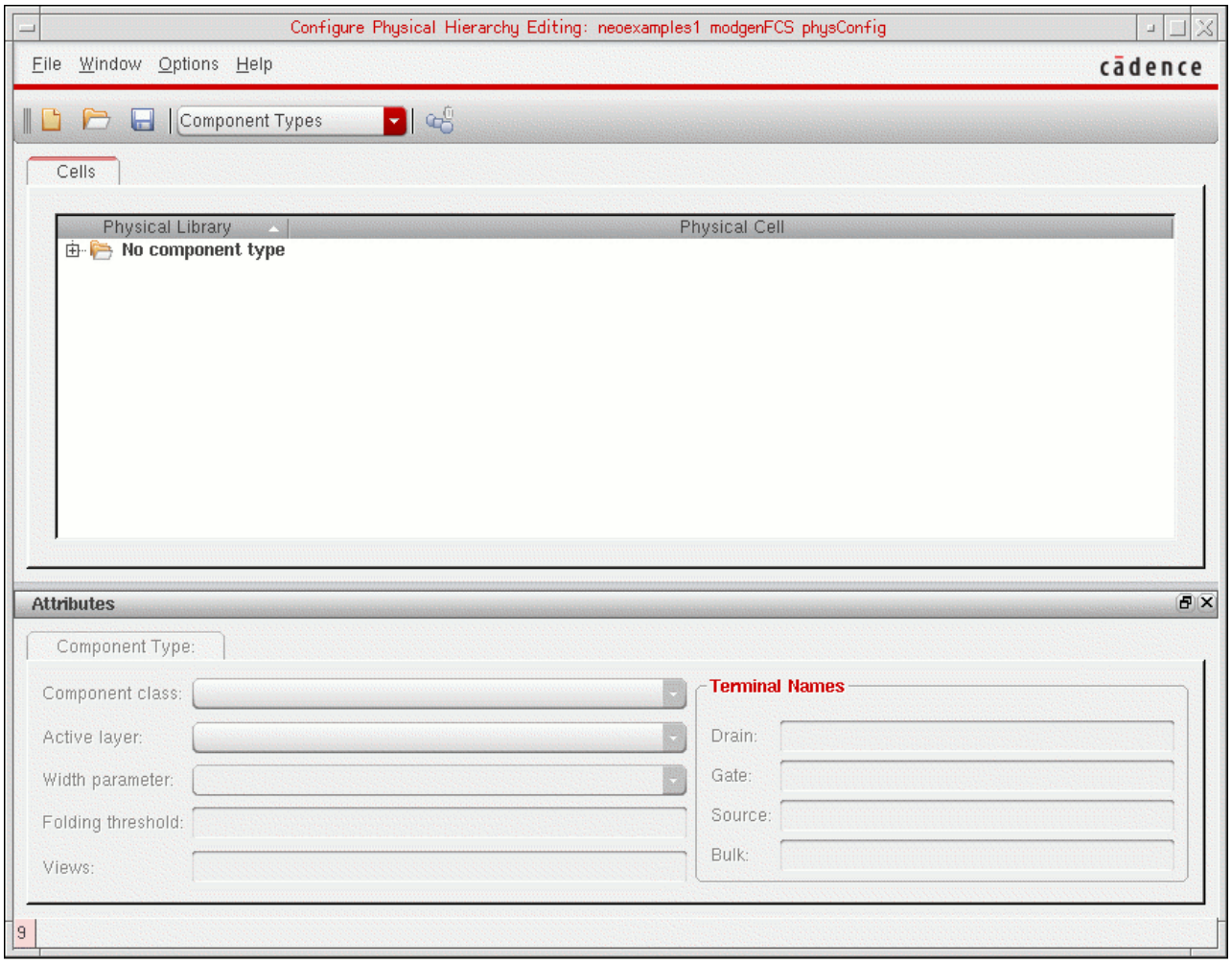
The devices to be abutted must be registered as component types. It addition to abutment, this is a key requirement for operations like chaining and folding devices, identifying pseudo-parallel nets, and identifying devices and structures using the Circuit Prospector assistant (this is a schematic XL feature) in Schematic view.

If the devices are not registered in the PDK, use one of the following methods to register devices as component types:

- Use the `ciRegisterDevice` function.
- Use the cph.lam file to assign the devices to component types NMOS, PMOS, NFIN, or PFIN. For more information, see Library and Attributes Mapping File Syntax.
- Use the *Component Type* mode of the Configure Physical Hierarchy window to assign the devices to component types NMOS, PMOS, NFIN, or PFIN.

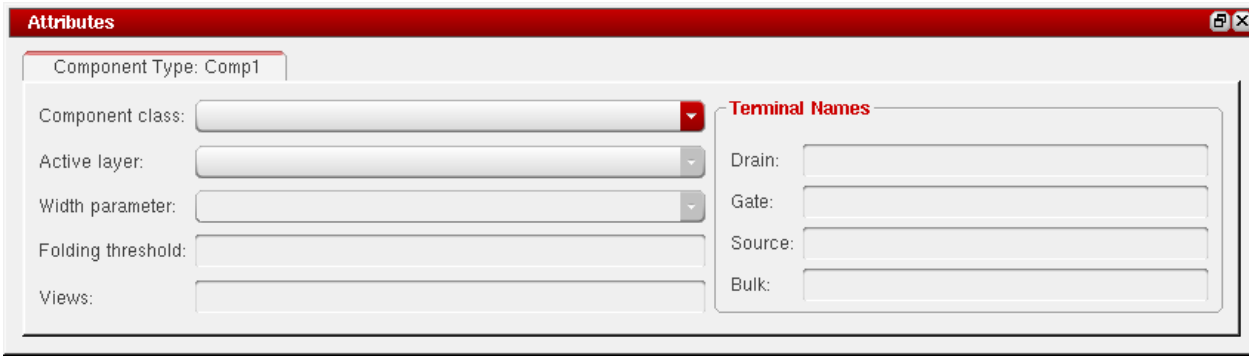
To set up the component types in the Configure Physical Hierarchy form:

1. Select *Edit – Component Type.* The *Configure Physical Hierarchy* window appears.

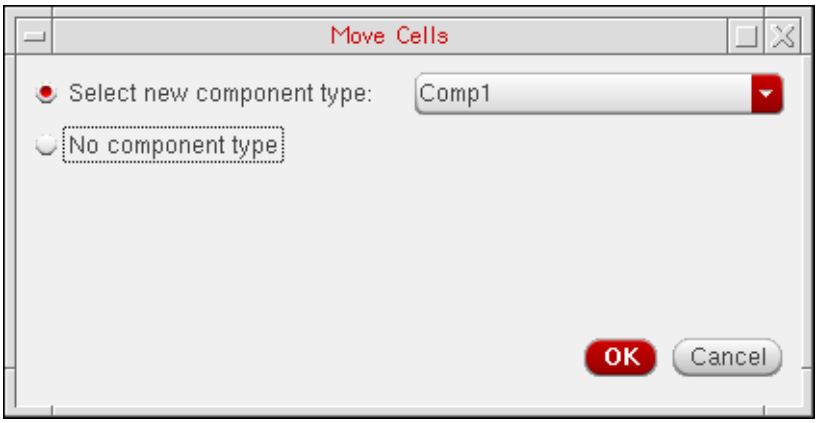


2. Right-click the *No component type* directory and select the *Add component type* option from the shortcut menu. The *Create Component Type* dialog box is displayed.

3. Type the name of the component type and click *OK*.
4. Select the newly added component type and specify its attributes in the *Attributes* section.



5. Click the + symbol next to *No component type* to expand it.
6. Right-click the master cell you want to move and select the *Move cells* option from the shortcut menu. The Move Cells dialog box is displayed.



7. From the *Select new component type* drop-down menu, select the component category you want to move the cell to and click the *OK* button.
8. Click the *Save*  icon.

Note: All the above mentioned steps need to be performed only once on a design before invoking the Modgen Editor.

Now, to abut devices, you need to perform the following steps:

1. Select two or more devices that you want to abut.

Note: Abutment in Modgen is a row-based operation. When you select multiple instances for abutment, the instances in each row are abutted separately.

2. Do one of the following:
   - In the Modgen toolbar, click the *Abut*  icon.
   - Right-click the array and choose *Abut*.
   - Select *Place—Modgen—Abut Modgen Instances*.

Important

By default, during abutment, the selected instances are mirrored, if needed. However, you can control whether to use mirroring instances or permutation of pins during the abutment process. To switch off mirroring and turn on pin permutation, set the chainPermutePins cdsenv variable to t.

Note: For a set of selected instances, abutment is performed only between the instances that can be abutted. For example, if there are body contacts between two neighboring instances that are selected, then those instances cannot abut.

Synchronized Abutment of Instances

To run the same abutment for all rows in the Modgen, before running the abutment command, turn on synchronized abutment by clicking the Synch Abut button  on the Modgen toolbar.

Instances in all rows of the Modgen are abutted in a synchronized operation along the column.

Turn off the Synch Abut button (default) to abut each row individually, without considering the other rows.

Removing Abutment from Devices

If you do not want devices in a module generator array abutted:

1. Select the devices you do not want abutted.
2. Do one of the following:
   - In the Modgen toolbar, click the *Unabut*  icon.
   - Right-click the array and choose *Unabut*.
   - Select *Place—Modgen—UnAbut Modgen Instances*.

Abutting All Devices

You can also choose to abut all devices.

1. To abut all devices, do one of the following:
   - In the Modgen toolbar, click the *Abut All*  icon.
   - Right-click the array and choose *Abut All*.

The modgenPatternFormAbutAll cdsenv variable enables you to abut all devices.

The modgenMergeWells cdsenv variable allows dummies to have different connectivity on source and drain.

Creating Multiple Abutment Scenarios

Each Modgen constraint can be associated with multiple abutment scenarios. The Abutment parameter can take multiple values (0, 1, 2, 3, and so on), where each value is assigned to a different abutment scenario. The default value 0 means no abutment.

Use the following SKILL functions to register, retrieve, and unregister abutment scenarios:

- `mgRegUserProc`: Registers abutment scenarios in the Modgen code to enable callbacks
- `mgGetRegUserProc`: Displays information about registered abutment scenarios
- `mgUnRegUserProc`: Unregisters the specified abutment scenarios

Removing All Abutment

You can also choose to remove abutment from all devices.

➤ Right-click the array and choose *Unabut All*.

Abutment of Dummy Shapes in Pcells (ICADVM20.1 Only)

The Modgen editor now supports the new Pcell abutment capability supported by Virtuoso Layout Suite XL. At advanced nodes, the Modgen editor supports the abutment of shapes in Pcell submasters that are not attached to pins. In addition to the usual abutment properties, these shapes also require an abutment name to be set in the Pcell SKILL code.

For more information about abutment at 20nm, see Advanced Node Abutment of Dummy Shapes in Pcells (ICADVM20.1 Only).

Specifying Device Alignment and Spacing

Since one Modgen module can contain instances with different Pcell masters, the instances can have different dimensions. To arrange instances with different heights within a module, you can specify the vertical bounding box alignment (top, center, or bottom).

When devices are aligned at the top, the top edges of the instances are aligned at the same Y-coordinate, while bottom edges are aligned at the same Y-coordinate for bottom alignment. For center alignment, the center of shorter instances are aligned with the center of the largest instance.

The default vertical alignment is bottom. If you want to modify this alignment:

1. Select the devices you want aligned.
2. Do one of the following:
   - Right-click and choose one of the following options:
     - *Align – Top*
     - *Align – Bottom*
     - *Align – Center (V)*
     - *Align – Center (H)*
     - *Align – Left*
     - *Align – Right*
   - In the toolbar, click the arrow next to the *Alignment*  icon and choose one of the following options:
     - *Align Left*
     - *Align Center (H)*
     - *Align Right*
     - *Align Top*
     - *Align Center (V)*
     - *Align Bottom*

You can also align using the Set Member Alignment and Spacing form.

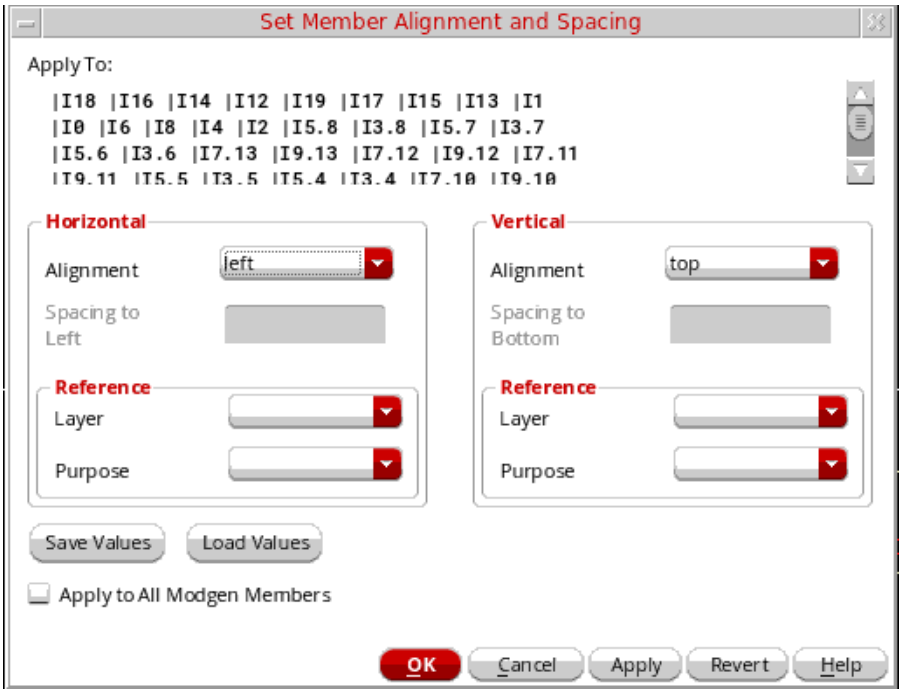
1. (Optional) Select the devices you want aligned.

   Note: If you do not select any device, then the settings are applied to all devices in the current Modgen module.

2. Do one of the following:

   - Right-click and choose *Set Member Alignment/Spacing*.
   - Click the Member Spacing/Alignment icon  on the Modgen toolbar.

   The Set Member Alignment and Spacing form appears.



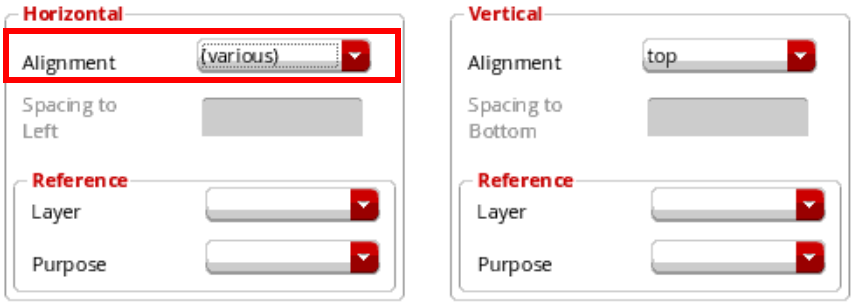
Tip

You can specify Modgen alignment and spacing parameters directly in the layout canvas. Select the required devices and choose *Place—Modgen—Set Member Alignment and Spacing* to display the Set Member Alignment and Spacing form.

3. Specify the horizontal and vertical spacing and alignment settings. Horizontal alignment refers to the alignment of instances along the horizontal axis, which is the x-axis. Vertical alignment refers to the alignment of instances along the vertical axis, which is the y-axis.

   The *Alignment* fields display the default *Horizontal* and *Vertical* alignments of the selected instances. In the above example, all selected instances have the same vertical and horizontal alignments, “left” and “top” respectively. In the example below, the

selected instances have the same vertical alignments, but their horizontal alignments are different. So, the horizontal *Alignment* is set to *(various)*.

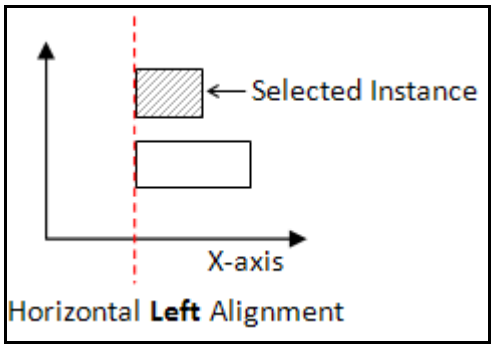


Note: When you change the *Alignment* from *various* to any other value, the *various* option disappears from the *Alignment* drop-down list.

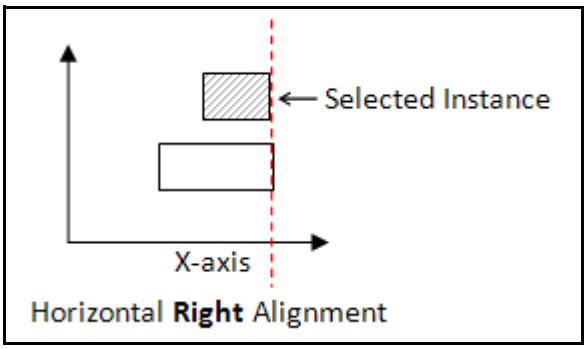
To specify different horizontal alignment values, in the *Horizontal* group box:

a. From the *Alignment* cyclic field, select one of the following options:

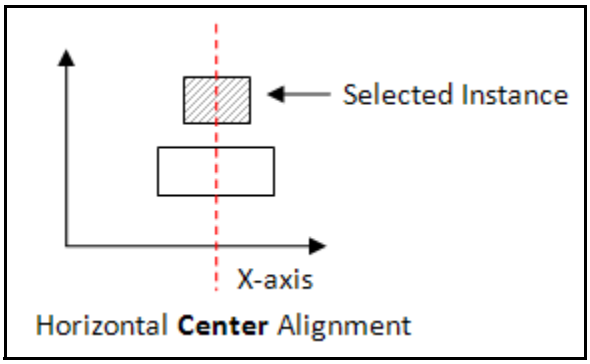
left: The left edge of the selected instance is aligned with the left edge of the instance below it.



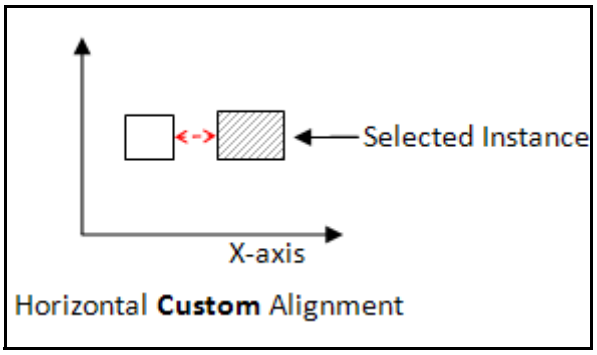
right: The right edge of the selected instance is aligned with the right edge of the instance below it.



center: The center of the selected instance is aligned with the center of the instance below it.



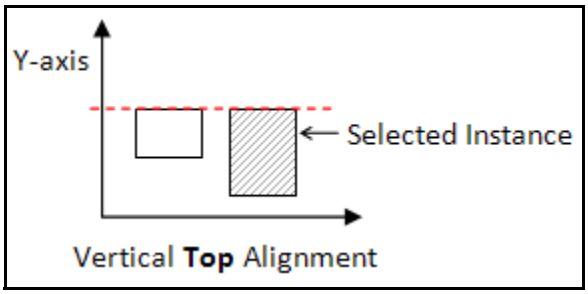
custom: The left edge of the selected instance is separated by the specified horizontal distance from the right edge of the neighboring instance.



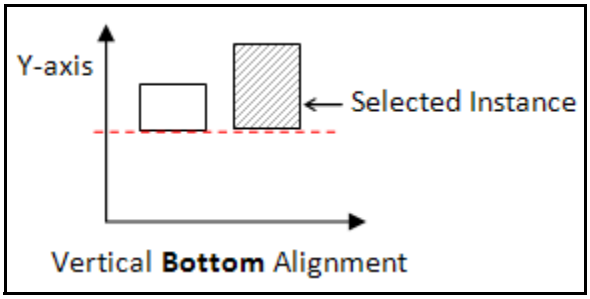
To specify a different vertical alignment, in the *Vertical* group box:

a. From the *Alignment* cyclic field, select one of the following options:

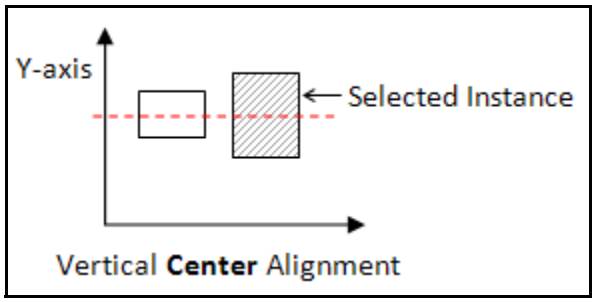
top: The top edge of the selected instance is aligned with the top edge of the instance to the left.



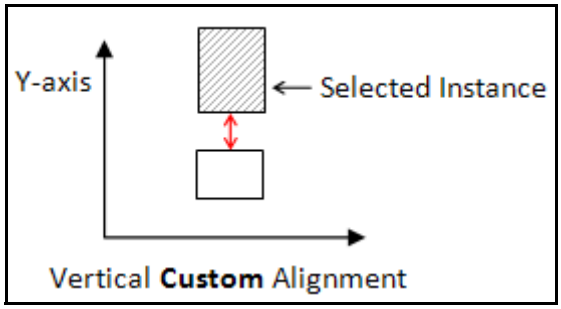
bottom: The bottom edge of the selected instance is aligned with the bottom edge of the instance to the left.



center: The center of the selected instance is aligned with the center of the instance to the left.



custom: The bottom edge of the selected instance is separated by the specified vertical distance from the top edge of the instance below it.



Note: Multiple instances can be selected and their horizontal and vertical alignments can be set. For example, if multiple instances in a row are selected and their vertical alignment is set to top, then all instances are top aligned.

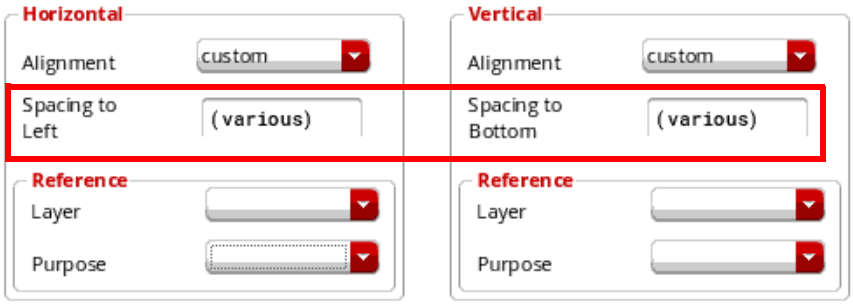
4. To specify a custom spacing, choose one of the following from the *Alignment* drop-down list:

   - *custom* (refers to `customLeft` in horizontal alignment and `customBottom` in vertical alignment)
   - *customRight* (for horizontal spacing)
   - *customTop* (for vertical spacing)

   With these selections, the spacing and reference options are made available.

   Note: The spacing field is available when the *Alignment* is set to *custom* or *customRight*/*customTop* in either one or both of the *Horizontal* and *Vertical* sections.

   The spacing fields display the default *Spacing to Left* and *Spacing to Bottom* values of the selected instances. If the spacing values are different for the selected instances, then *(various)* is displayed.



   You can reset the spacing value by specifying a new value in the *Spacing to Left* and *Spacing to Bottom* fields and clicking *OK* or *Apply*.

5. In the *Reference* section, select a reference *Layer* and *Purpose* based on which:

   - The horizontal or vertical spacing of devices must be calculated.
   - Instances must be aligned horizontally or vertically.

   Similar to *Alignment* and *Spacing* fields, if the layers and purposes are different for the selected instances, then *(various)* is displayed. You can reset these values and click *Apply* or *OK*.

   Note: Notice that the *Layer* cyclic field is blank by default. If no value is chosen in this field, then the default behavior is that the devices will be spaced using their instance bounding boxes.

6. Click *Save Values* to overwrite environment variables with the values specified in the Set Member Alignment and Spacing form.

   Note: At any point, you can use *Load Values* to reset all values in the form with values from corresponding environment variables (stored in the .cdsenv file). These values were saved earlier by using the *Save Values* button.

7. Select *Apply to All Modgen Members* to apply the custom spacing to all the devices in the module.

   If you do not select this option, the custom spacing is applied only to the selected devices.

8. Click *OK* or *Apply*. Clicking *Revert* resets the values in the fields with corresponding values from the database.

   Clicking *Apply* commits the changes, but keeps the form open for further modifications to the Modgen members. You can perform tasks such as:

   - Changing the selection objects and reapplying the current settings
   - Modifying the settings and applying to the current selection

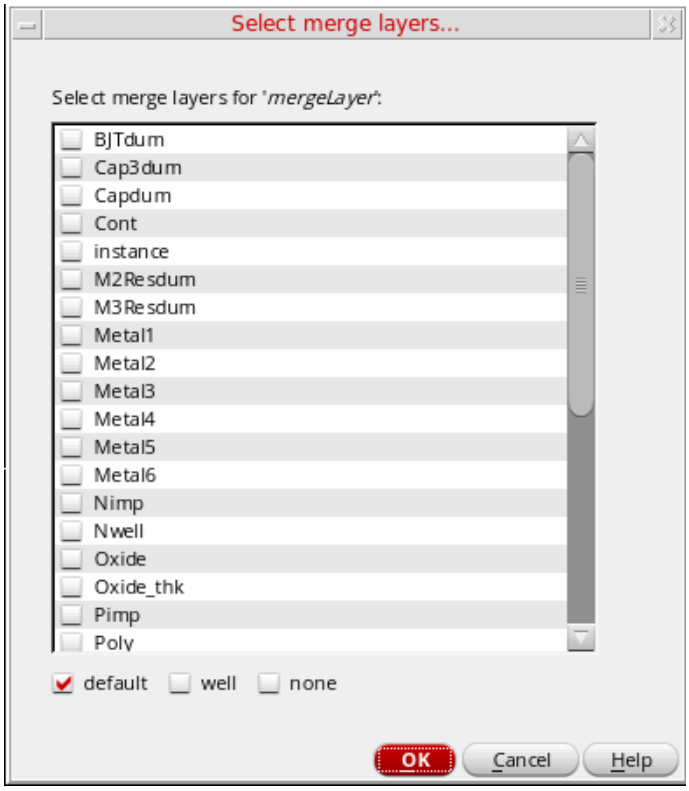
Removing a Custom Spacing Distance

Because custom spacing is interpreted internally as a custom alignment, in order to truly remove a custom spacing distance, you must choose an alignment for the devices. For more information, see Specifying Device Alignment and Spacing.

Merging Layers

Use the Select merge layers... form to specify the layers that need to be merged.

1. To invoke the Select merge layers... form, click the Merge Layers icon  on the Modgen toolbar. Alternatively, you can right-click anywhere in the Modgen Editor window and select *Merge Layers*. The Select merge layers... form is displayed.



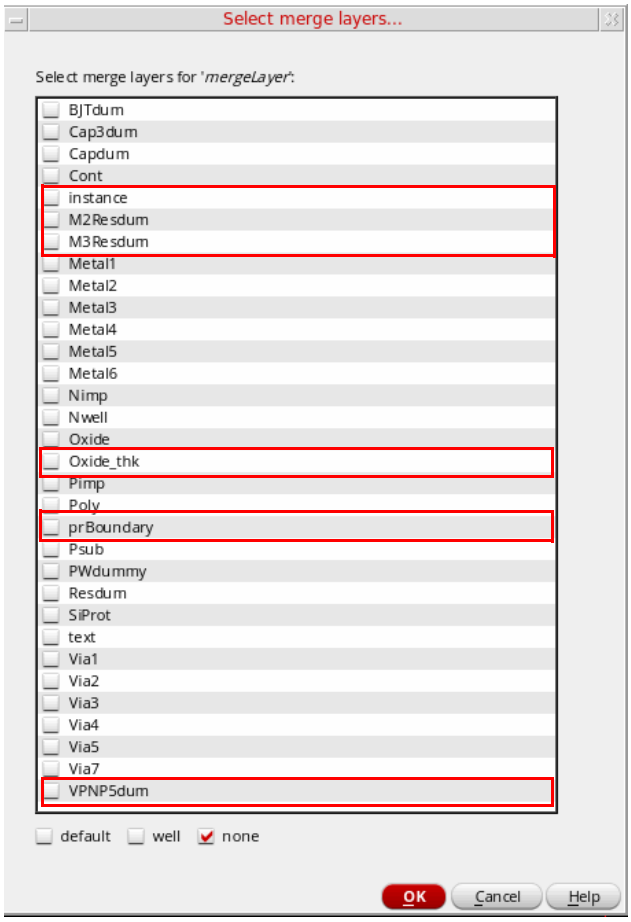
   Note: The Select merge layers... form can also be invoked from Constraint Manager with the `mergeLayer` parameter in the Modgen `ci` constraint.

2. Select the check boxes adjacent to the layers that you want to merge. Alternatively, select one of the Layers Preset: *default*, *well*, or *none* to select all layers of the selected type.
3. Click *OK*.

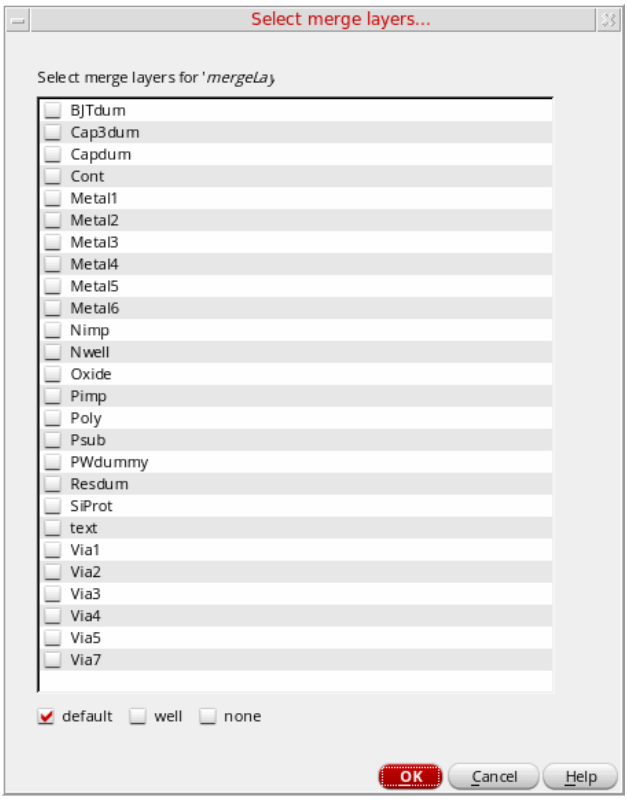
By default, all the used layers in the layout are listed in the merge layers list box. However, selecting the *Use Layer Palette LPPs Only* check box in the Layout Editor Options form enables the synchronization between the layers displayed in the list and the layers listed in the Palette assistant of Layout XL.

Note: To access the Layout Editor Options form, choose *Options - Editor...* from the Layout XL window. For more information, see Layout Editor Options form.

When you enable layer synchronization and change the filters in the Palette assistant by using the available check boxes (that is, *Used*, *Valid*, and *Routing*), you can ensure that layers of only a specific category are displayed in the merge layers list box. This change is visible when you update the filters in the Palette assistant and then launch the Select merge layers form, as illustrated in the figure below.



Before the Layer Palette LPP sync up
(the highlighted entries are not visible in the image on the right)



After the Layer Palette LPP sync up

Environment Variables: modgenMergeWells, modgenMergeLayers

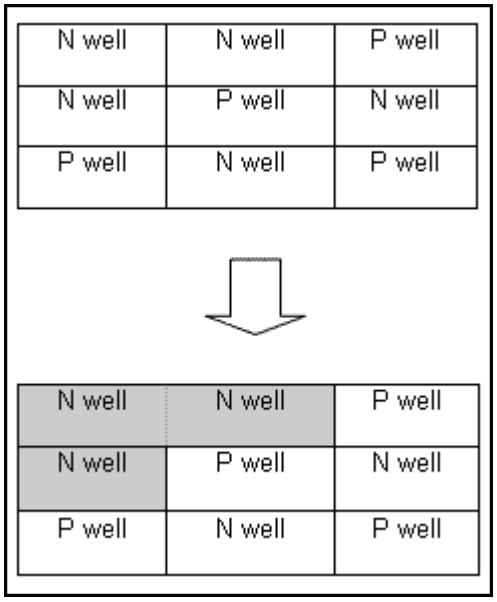
Merging of wells is controlled by the `mergeLayer` parameter in the Modgen `ci` constraint. The default value for the `mergeLayer` parameter in Modgen is `default`. For backward compatibility with the 614 Modgen constraints, the `modgenMergeWells` environment variable can be set in the `.cdsenv` file.

If this environment variable is set to `t` and the `mergeLayer` parameter is set to `default`, then the wells of the Modgen instances will be merged. However, if the environment variable

is not set, and the `mergeLayer` parameter is set to `default`, the wells in the Modgen layout are not merged.

Other possible values of the `mergeLayer` parameter are: `none`, `well`, and any other custom layer, which can be selected from the Constraint Manager GUI. If `mergeLayer` is set to `none`, then there will not be any well merging in the Modgen. If `mergeLayer` is set to `well`, then the wells of the Modgen devices will be merged and any DRCs between well layers and other layers of the devices will be ignored. If `mergeLayer` is set to a custom layer, such as `Oxide_thickness`, then the shapes on that layer will be merged and DRCs between that specified layer and other layers of the devices will be ignored.

In addition, you can merge multiple layers, such as `Nwell` and `Oxide_thk`. Before merging layers, only the top-level net connectivity between the layers is checked. So, layers across devices that have different master Pcells can also be merged. An example for well merging is shown below:



To specify whether the analog placer should have control over the well and body contact geometry, set the `aapDefaultWellsUnderPlacerControl` environment variable to `t`.

The merge layer functionality differs when there is a pane guard ring inside the Modgen structure. A layer across a pane guard ring is merged only if one of the following situations is applicable.

- The layer is defined as a recognition layer in the technology file.
- The layer is a well layer with the same net connectivity across all devices and the pane guard ring.

For more information about guard rings, see Specifying Guard Rings.

4

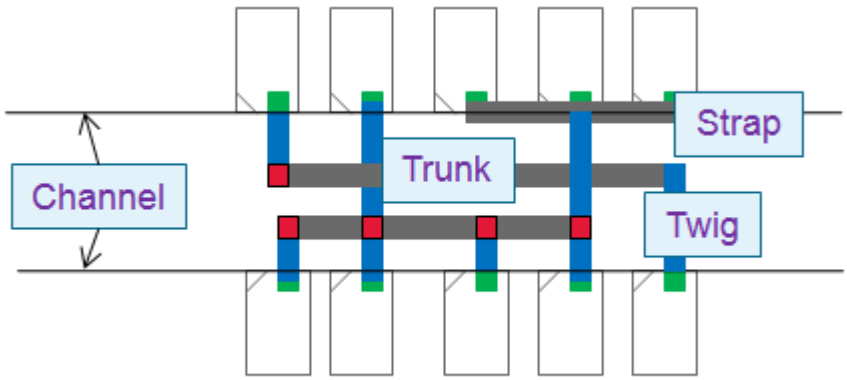
Creating Topology Patterns and pin-to-trunk Routing

A topology pattern is a routing plan for a design, which is used by the pin-to-trunk router while creating routes. Use the topology options in Modgen to visualize, configure, and store the routing information.

Topology patterns comprise the following topology objects:

- **Trunks:** Specify the location of the routes, including their anchor points and offsets, and the routing extents.
- **Twigs:** Define the connections between the trunks and the devices.
- **Channels:** Define the routing areas in a design.
- **Straps:** Define the connections between multiple objects. Straps are used for specific purposes, depending on the constraints.

The following diagram depicts the use of topology objects in a topology pattern:



The pin-to-trunk router in Modgen recognizes the topology objects and their constraints, and creates geometries that match the topology pattern.

This section provides information on the following topics:

- Creating and Editing Topology Patterns
- Creating Incremental Trunks
- Adding Straps
- Adding Twigs
- Creating Single Strap Topologies
- Setting the Channel Width
- Creating Pin-to-Trunk Routes
- Copying Topology and Routing Information
- Creating Matched Groups
- Deleting Topology and Routing Information
- Defining Manual Routes

Creating and Editing Topology Patterns

Use the topology options in the Modgen Editor to visualize, configure, and store advanced routing topologies.

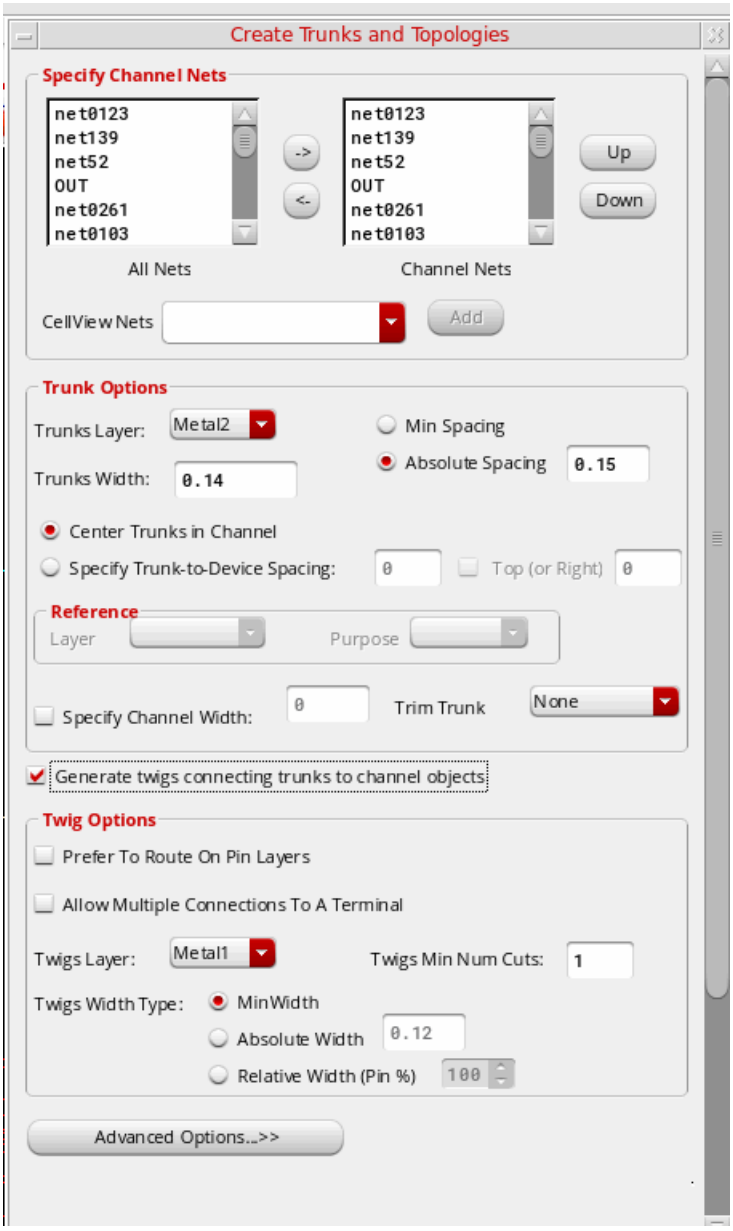
Tip

While creating a topology pattern, it would be useful to display the Navigator assistant, Property editor, and Palette assistant so that you can view, select, and edit the topology object attributes and the layer attributes. Instead of opening these windows for each session individually, you can use the modgenWindowConfigFile environment variable to set the default Modgen workspace, which will be automatically loaded whenever Modgen is invoked.

The Modgen Routing toolbar in the Modgen Editor provides options to create and edit topology patterns and to establish pin-to-trunk routing.

To define a new topology pattern or to edit an existing topology pattern:

1. Select the *Create Trunk and Topology* icon  from the Modgen Routing toolbar. The Create Trunks and Topologies form is displayed.



2. Use *Specify Channel Nets* to specify the channel nets to be included in the topology pattern. By default, all the available nets are selected. To remove nets, select the nets and move them from the *Channel Nets* box to the *All Nets* box. Use the *Up* and *Down* buttons to change the order of the channel nets. Use the *CellView Nets* drop-down list to add additional nets that already exist in the layout or schematic view.

Note: For the topology to be created, you need to select at least one channel net from the list.

Tip

You can add multiple instances of the same net to the Channel Nets box.

Note: The selected channel nets are remembered for the current session. So the next time you open the Create Trunks and Topologies form, the selected nets are automatically displayed in the Channel Nets box.

3. In the *Trunk Options* section, specify the *Trunks Layer* and *Trunks Width*. Either specify the *Min Spacing* based on which the trunk spacing needs to be calculated or an *Absolute Spacing* value for trunks. *Center Trunks in Channel* is selected by default. Therefore, trunks are centered in the channel such that their distances from devices is the same from both, the top and bottom edges.

   Specify the required values in the *Specify Trunk-to-Device Spacing*, *Top (or Right)*, *Reference*, and *Specify Channel Width* fields.

   From the *Trim Trunks* drop-down list, select the side from which trunks need to be trimmed while routing. For more information about these options, see Create Trunks and Topologies.

4. The *Generate twigs connecting trunks to channel objects* check box controls whether topological twigs, which connect trunks to channel objects (instances and body contacts), are generated when creating trunks.

   By default, the check box is selected, and so twigs will be generated. Use the options in the *Twig Options* section to customize the twigs. If the check box is cleared, twigs will not be generated. The options in the *Twig Options* section are disabled.

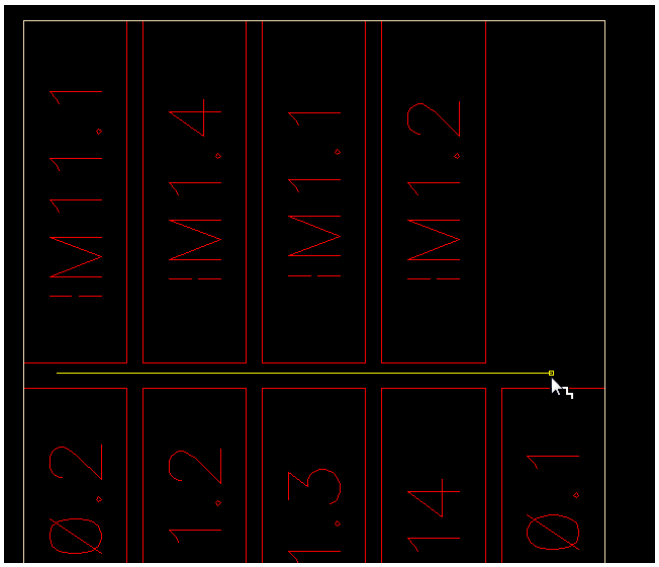
5. Click *Advanced Options* to specify options for controlling via configurations and pin cover settings within the topology pattern. The Advanced Options section includes the following tabs:

- **Via Controls:** Use the options on this tab to specify how vias need to be positioned and aligned on trunks and twigs. See Create Trunks and Topologies for more information about these options.

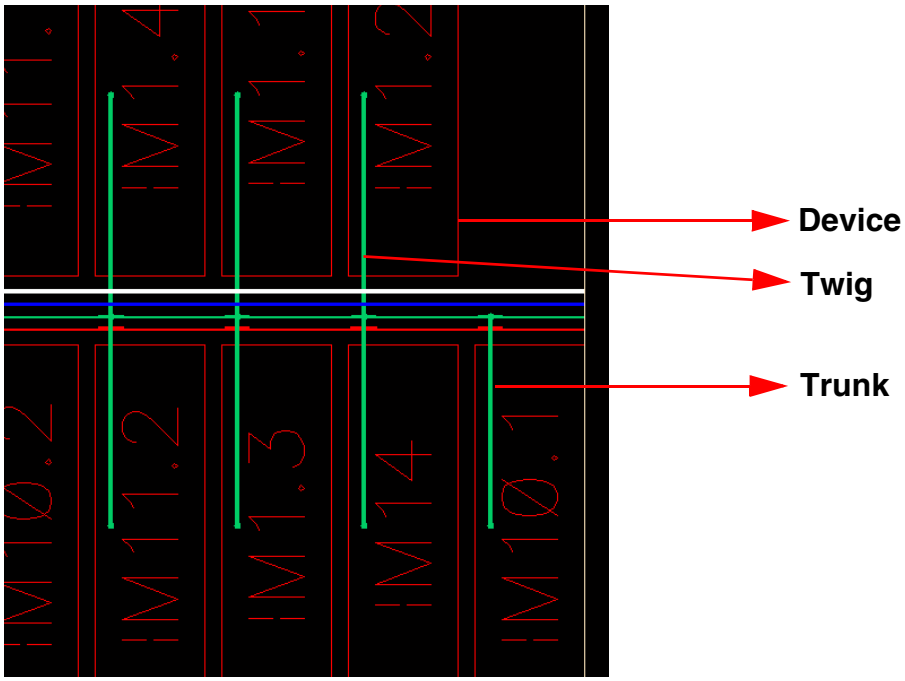
- **Pin Cover:** Use the options on this tab to control the devices that cover the pin shapes. See Create Trunks and Topologies for more information about these options.

6. Click *Hide* to go back to the layout canvas.

7. Click and drag in the Modgen Editor to create the topology pattern.

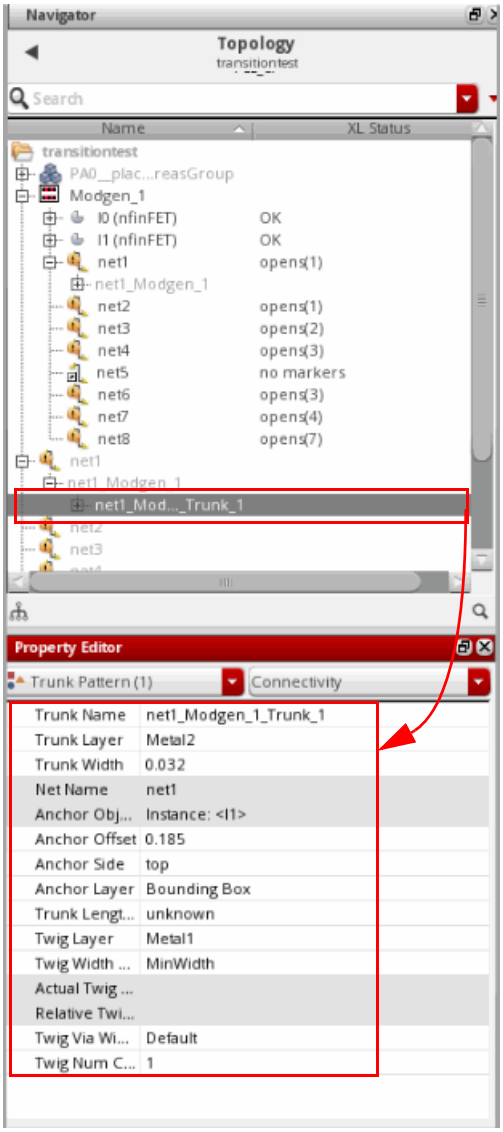


A topology pattern is created. The number of trunks in the topology pattern is the same as the number of channel nets specified in the Create Trunks and Topologies form. Twigs connect devices to the trunks.

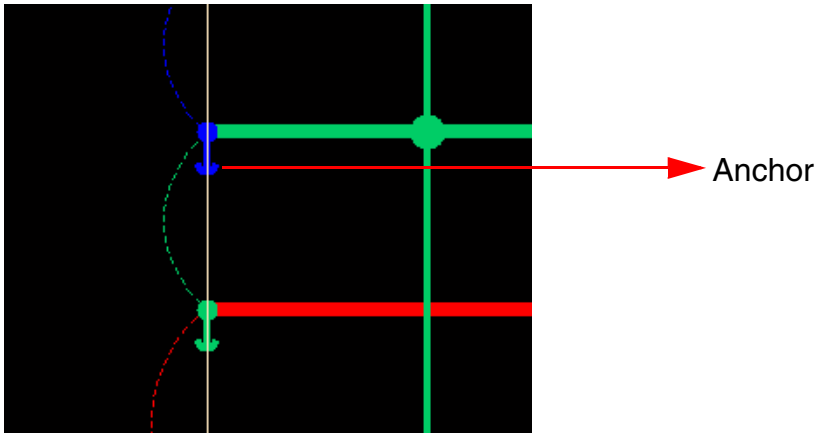


Note: The values specified in the Create Trunks and Topologies form will remain valid for the duration of the current session.

Note: When you select a trunk or a twig, it is highlighted in the Navigator assistant, and its properties are displayed in the Property assistant. Use the Property assistant to edit the properties.



Trunk offset references are indicated by anchors. Anchors are chained according to the defined offsets.



The anchor point is listed as a trunk property in the Property editor. When trunks are deleted, the corresponding anchors are also deleted. When trunks are moved, anchors are automatically moved and re-chained to the applicable offsets at their new locations.

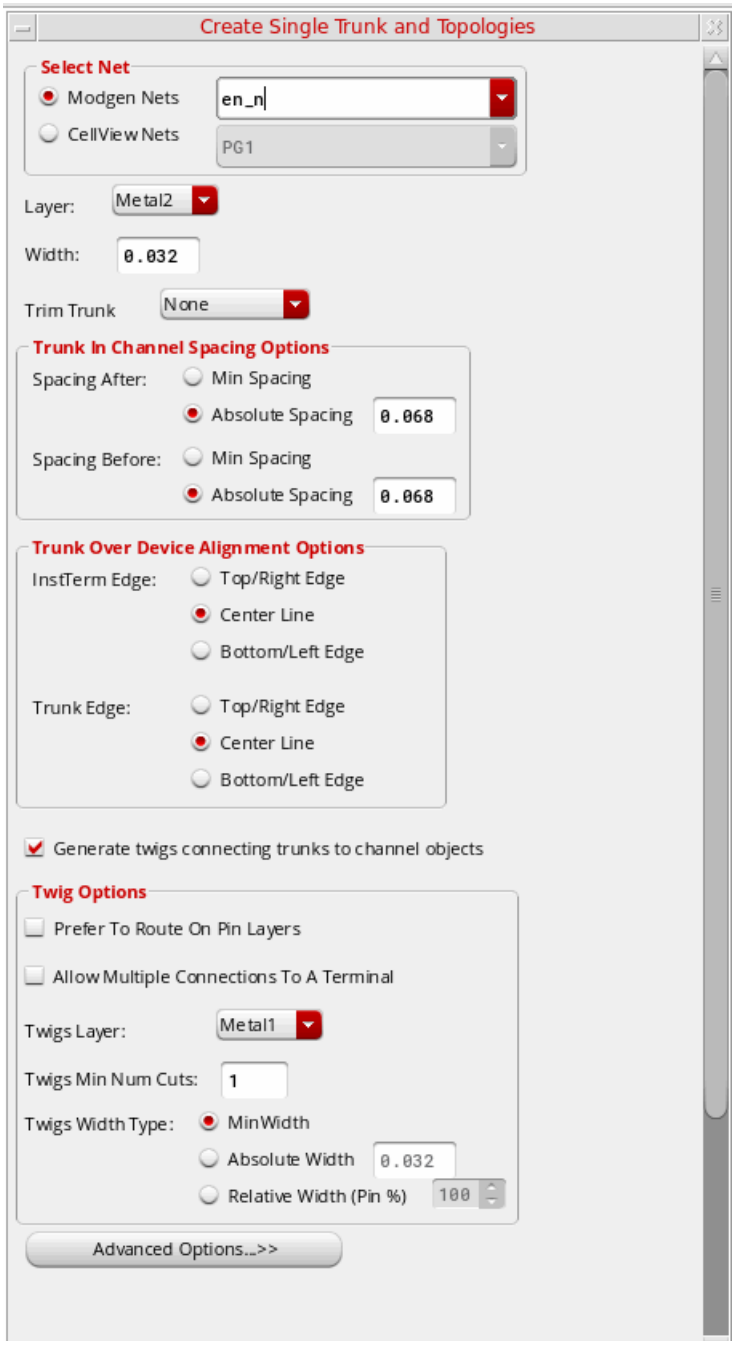
Related SKILL Functions:

- mgAddTopologyToModgen
- mgCreateMatchGroupInModgen
- mgGetTopologyFromModgen
- mgModgenHasTopology
- mgGetModgenConstraintFromTopology
- mgGetModgenFigGroupFromTopology
- mgIsTopologyInsideModgen

Creating Incremental Trunks

Use the Create Single Trunk and Topologies form to create incremental trunks, in addition to the ones created during topology creation. To create incremental trunks:

1. Choose the *Create Single Trunk and Topologies* icon on the Modgen Routing toolbar to display the Create Single Trunk and Topologies form.



2. Specify the *Net*, *Layer*, and *Width* of the new trunk.
3. Select the side for trimming trunks from the *Trim Trunk* drop-down list. Available options are: *None*, *Both*, *Left/Bottom*, and *Right/Top*. The default is *None*, in which case trunks are not trimmed.
4. In the *Trunk In Channel Spacing Options* section, specify the *Spacing Before* and *Spacing After* trunks and other devices.
5. Use the *InstTerm Edge* and *Trunk Edge* options to specify how instance and trunk edges need to be aligned when trunks are drawn over instances.
6. The *Generate twigs connecting trunks to channel objects* check box controls whether topological twigs, which connect trunks to channel objects (instances and body contacts), are generated when creating trunks.

   By default, the check box is selected, and so twigs will be generated. Use the options in the *Twig Options* section to customize the twigs. If the check box is cleared, twigs will not be generated. The options in the *Twig Options* section are disabled.

For more information about these options, see Create Single Trunk and Topologies.

Supported Constraint

- **topologyColorMask:** Specifies a mask color for the `pathSeg` that represents the trunk.

Related SKILL Functions

- `mgSetTrunkRefLayerPurpose`
- `mgGetTrunkRefLayerPurpose`

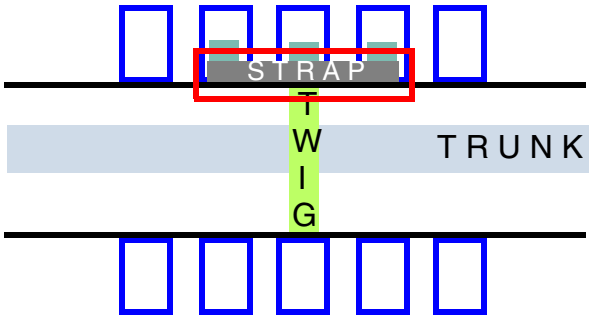
Setting Local Trunks

Use the `mgSetIsLocalTrunk` function to identify local trunks. Trunks inside a Modgen are called local trunks. These trunks can be connected only within the same Modgen, and not from outside the Modgen. So, the global router ignores these trunks during routing.

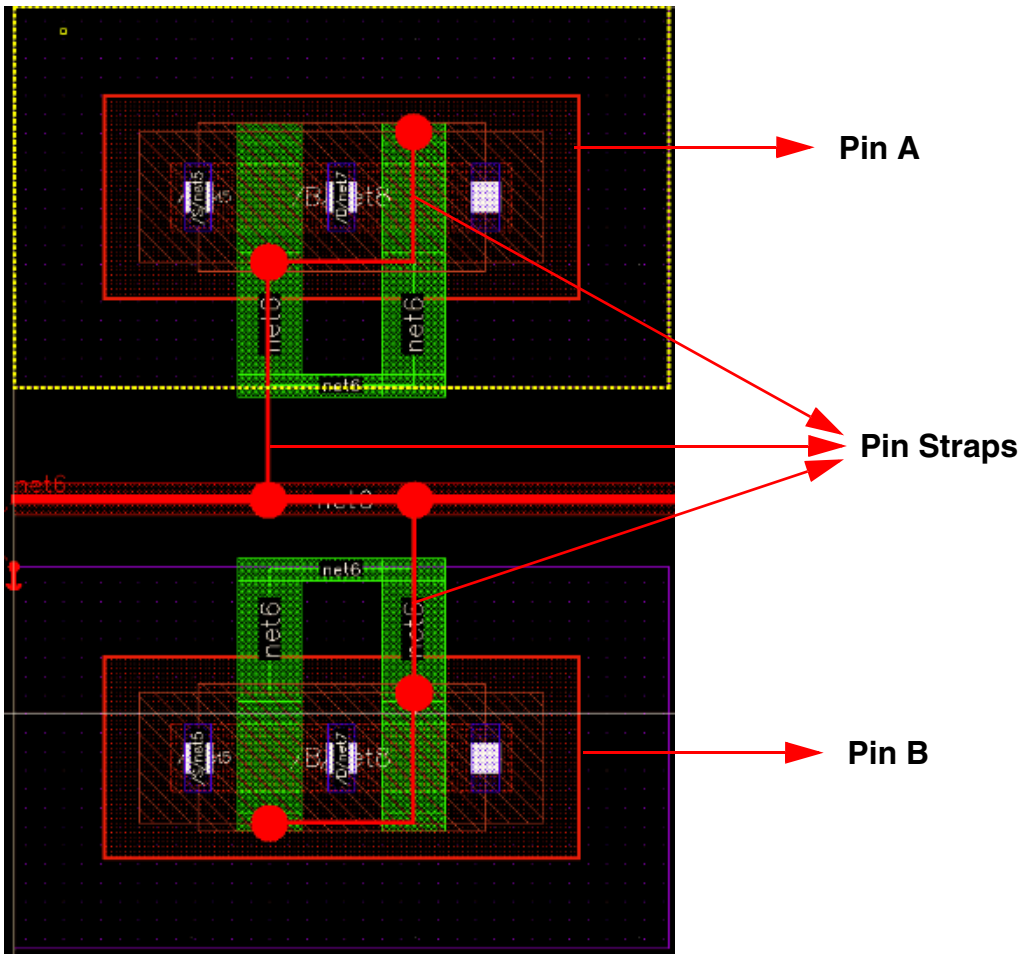
Related SKILL function: `mgGetIsLocalTrunk`

Adding Straps

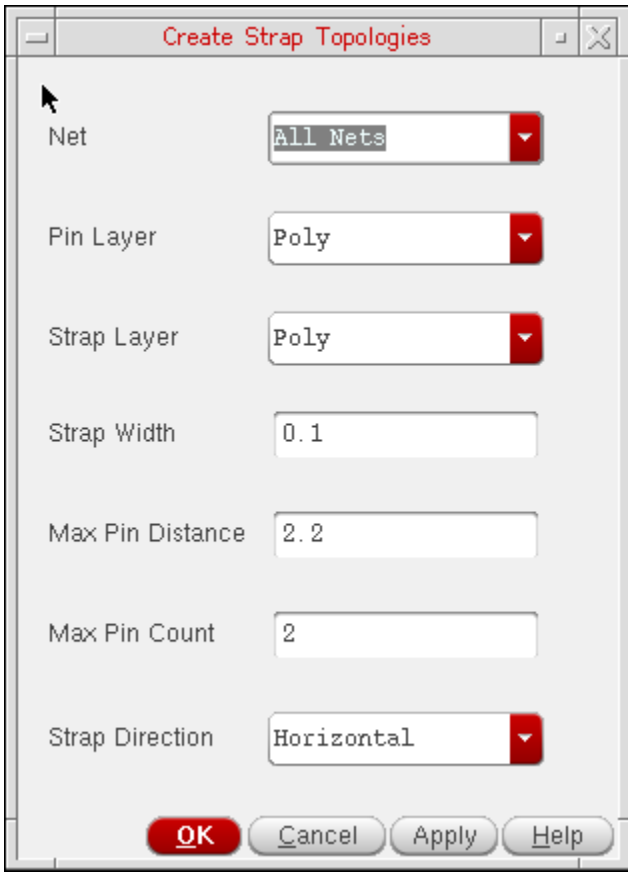
Straps are used to connect aligned pins within a Modgen constraint.



A strap is connected to a trunk using a twig. A strap can have multiple geometric segments.



Use the Create Strap Topologies form to define strap parameters.



To create straps:

1. Click the Create Strap Topologies button on the Modgen Routing toolbar to display the Create Strap Topologies form.
2. Specify the *Net*, *Pin Layer*, *Strap Layer*, and *Strap Width*.
3. Specify the *Max Pin Distance* and *Max Pin Count*.
4. Finally, select a *Strap Direction* and click *OK* to create the strap in the layout canvas.

For more information about the options in the Create Strap Topologies form, see Create Strap Topologies.

Supported Constraints

- **numStrands:** Specifies the number of strands to be used to route between pins.

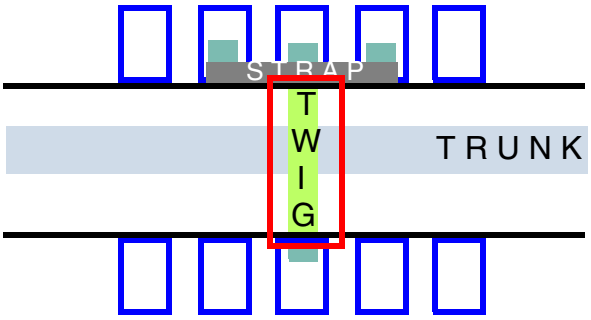
- **strandSpacing:** Specifies the spacing between individual wire strands.
- **strandStackStyle:** Specifies the layers to be used to create stranded stacks: `all` (default), `alternating`, or `adjacent`. The specified layers must be part of the `validStackLPPs` constraint list.
- **strandWidth:** Specifies the width of individual wire strands.
- **strapLocation:** Specifies whether the strap must be placed under a trunk. The strap must be set as valid targets for `strandSpacing`, `strandStackStyle`, `strandWidth`, and `validStackLPPs`.
- **validStackLPPs:** Specifies the layers to be used in the stranded stacks.

Related SKILL Functions

- `mgGetStrapDirection`
- `mgGetStrapHasDirection`
- `mgGetStrapHasOffset`
- `mgGetStrapOffset`
- `mgGetStrapLongOffset1`
- `mgGetStrapLongOffset2`
- `mgSetStrapDirection`
- `mgSetStrapHasDirection`
- `mgSetStrapHasOffset`
- `mgSetStrapLongOffset1`
- `mgSetStrapLongOffset2`
- `mgSetStrapOffset`

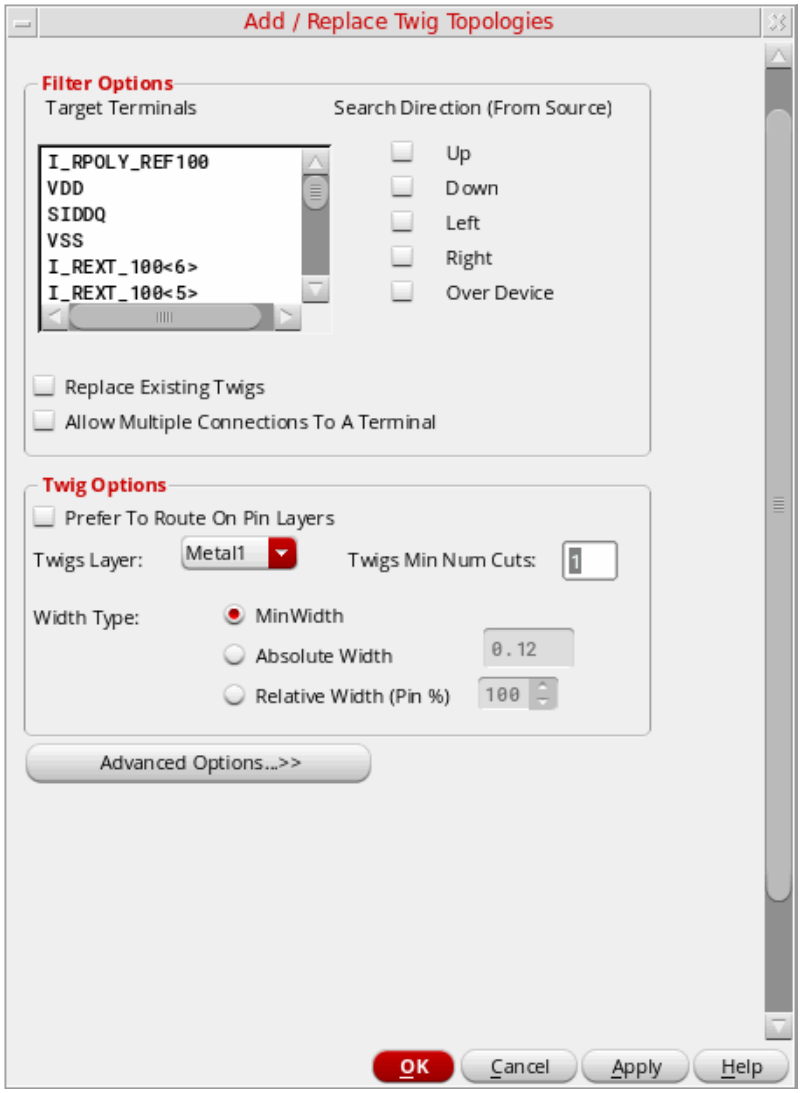
Adding Twigs

Twigs are important components of a topology design. They define the connections between the trunk and other components, such as another trunk, net, strap, or instance. In the following example, a twig is used to establish the connection between: strap --> trunk --> pin.



Typically you define twigs while creating the topology. The Create Trunks and Topologies form provides options for defining twigs.

It is possible that certain twigs were deleted from the design accidentally. In such situations, you can recreate the twigs using the Add/Replace Twig Topologies form. You can also use this form to create incremental twigs or to modify existing twigs.



You can use the Add / Replace Twig Topologies command to customize the topology of a Modgen by editing existing twigs and by adding new twigs. To create new twigs:

1. Select the instances, body contacts, trunks, or nets that need to be connected. Then, click the Add / Replace Twig Topologies button  on the Modgen Routing toolbar to display the Add/Replace Twig Topologies form.
2. In the Filter Options section:

a. Specify the Source or *Target Terminals* and *Search Direction*. You can select multiple terminals.

Note: If you do not select any source/target terminal, then the options are applied to all the listed terminals.

b. Select the appropriate *Search Direction* options to indicate the direction along which the connections need to be made. You can select one or more search directions: *Up*, *Down*, *Left*, *Right*, or *Over Device*. *Over Device* indicates that a twig must be created for a trunk located over the device or pin, which connects the trunk to an overlapping pin on the same net.

c. You can choose to *Replace Existing Twigs* with the new ones.

d. You can choose to *Allow Multiple Connections To A Terminal* to enable multiple trunk connections to a terminal.

3. In the Twig Options section, specify the twig parameters:

a. Select *Prefer to Route On Pin Layers* to indicate that routing should be done on the pin layers instead of the drawing layers.

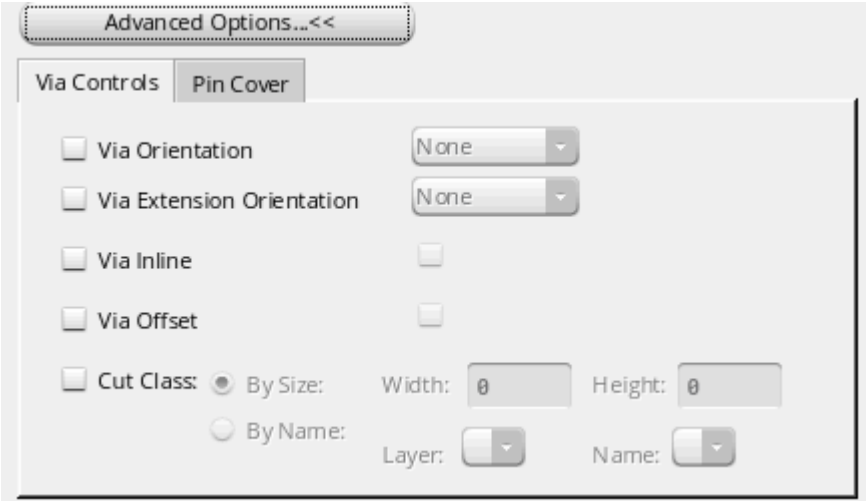
b. In the *Twigs Layer* field**,** specify the layer on which the twigs need to be created.

c. Specify the minimum number of cuts that the vias connecting the twigs to other objects need to have.

d. Choose a suitable *Twigs Width Type*.

4. Click *Advanced Options* to specify options for controlling via configurations and pin cover settings within the topology pattern. The Advanced Options section includes the following tabs:

- **Via Controls:** Use the options on this tab to specify how vias need to be positioned and aligned on trunks and twigs.



- **Pin Cover:** Use the options on this tab to control the devices that cover the pin shapes.

For more information about these options, see Add/Replace Twig Topologies.

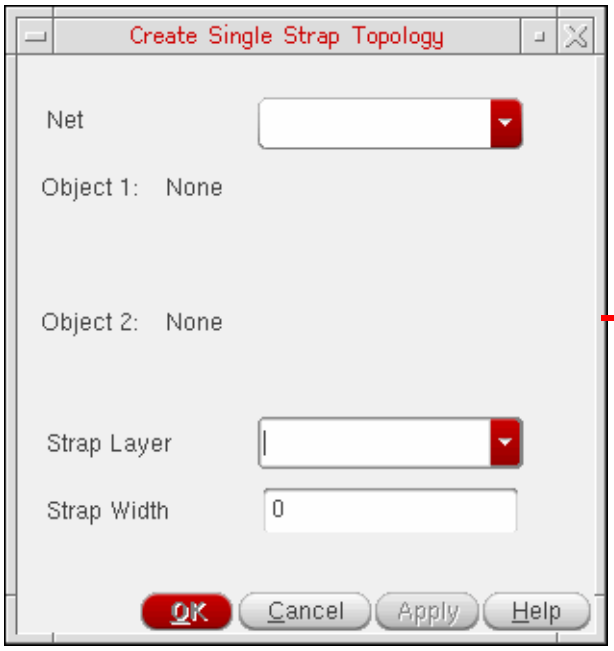
Creating Single Strap Topologies

Use the options in the Create Single Strap Topology form to define straps between individual pins. Pins can be strapped in Modgen only if the following conditions are met:

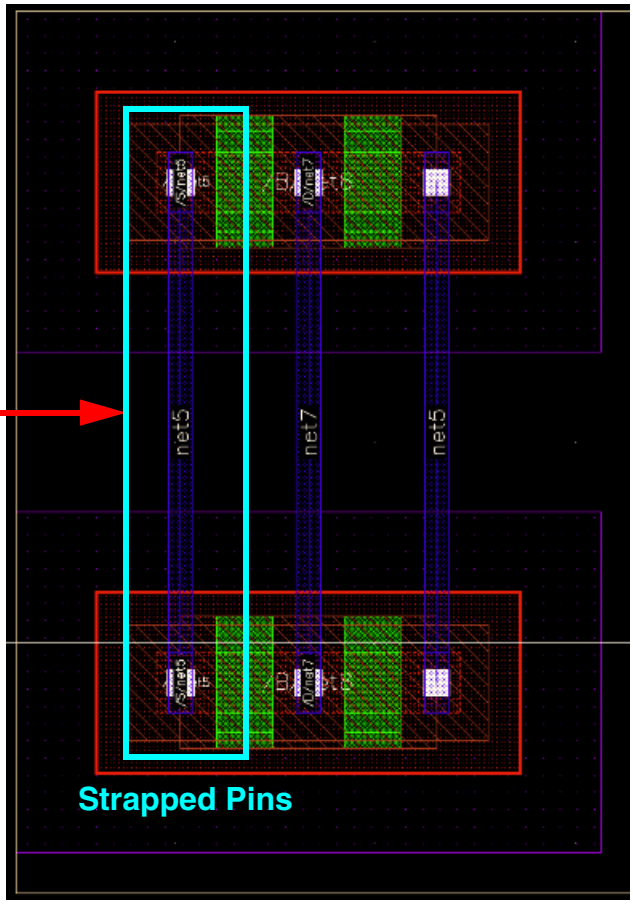
- The pins must be on the same layer.
- The pins must be aligned to the top and bottom rows of the channel.

To to define straps between individual pins:

1. Display the Create Single Strap Topology form by clicking the Create Single Strap Topology button  on the Modgen Routing toolbar.



Straps are defined



2. Specify the Net on which the straps need to be created.
3. Specify the objects, which are the names of the pins that need to be strapped during pin-to-trunk routing and their instances in the respective fields.
4. Specify the *Strap Layer* and *Strap Width*.

5. Click *OK* to create the strap in the layout canvas.

Note: Strap connections to guard rings is not supported.

For more information about these options, see Create Single Strap Topology.

Setting the Channel Width

Modgen provides multiple ways of setting the routing channel width.

- **Using the** Create Trunks and Topologies **form:** Specifies the channel width while defining a new topology pattern.
- **Using the** Create Single Trunk and Topologies **form:** Specifies the channel width while creating an incremental trunk for a topology.
- **Using the** Set the Channel Width **form:** Specifies the channel width before or after creating trunks. The channel width indicates the distance from either a device row to another device row, or a trunk to a device row.

The following section describes the procedure to set the routing channel width using the Set Channel Width form.

To specify the routing channel width values using the Set Channel Width form:

1. Select the *Set Channel Width* icon  on the Modgen Routing toolbar. The Set Channel Width form displays.

2. In the *Channel* section, choose a *Direction*:
   - *Horizontal* - Channel width indicates the device row-to-row spacing.
   - *Vertical* - Channel width indicates the device column-to-column spacing.
3. Specify the *Channel Number* for which the direction and width values need to be set. The corresponding channel is highlighted in the layout canvas.

   When the direction is set to *Horizontal*, *Number* refers to the row number. Row number 0 refers to the channel below Modgen, row number 1 indicates the first channel, row number 2 indicates the second channel, and so on.

   When the direction is set to *Vertical*, *Number* refers to the column number. Column number 0 refers to the channel to the left of Modgen, column number 1 indicates the first channel to the right, column number 2 indicates the second channel to the right, and so on.

   The value *All* indicates all the routing channels in the given direction.

   If the direction and width values are already set for a channel, then these values are displayed in the respective fields. You can modify the values.

4. Click *Center Trunks* to center trunks in the selected channel.

   Note: Center Trunks can be done only for the existing trunks in the channel.

5. Select the *Layer* and *Purpose* of the topological trunk that must be anchored to an instance. The bounding box of the specified layer and purpose inside the instance is used to determine the device reference edge for setting the channel width. In other words, the channel width is the edge-to-edge spacing between the bounding boxes of the associated instances.

   Note: If the *Layer* and *Purpose* are not specified, the bounding box of the instance determined by the Modgen is used.

6. Specify the channel width. For *Horizontal* direction, the channel width indicates the *Device Row To Row Spacing*; and for *Vertical* direction, the channel width indicates the *Device Column To Column Spacing*.

7. Click *Set* to save the channel width value.

8. Specify the *Trunk to Device Spacing*. This is an alternative way to specify the channel by specifying the distance between the trunks in the channel and the devices. *Top* is the spacing between the top edge of the top trunk to the bottom edge of the top device. *Bottom* is the spacing between the bottom edge of the bottom trunk to the top edge of the bottom device.

9. Click *Set* to save the trunk-to-device spacing value.

10. Click *Close* to close the form.

At any point, you can click *Remove Channel Width* to remove the channel width setting for the specified channel.

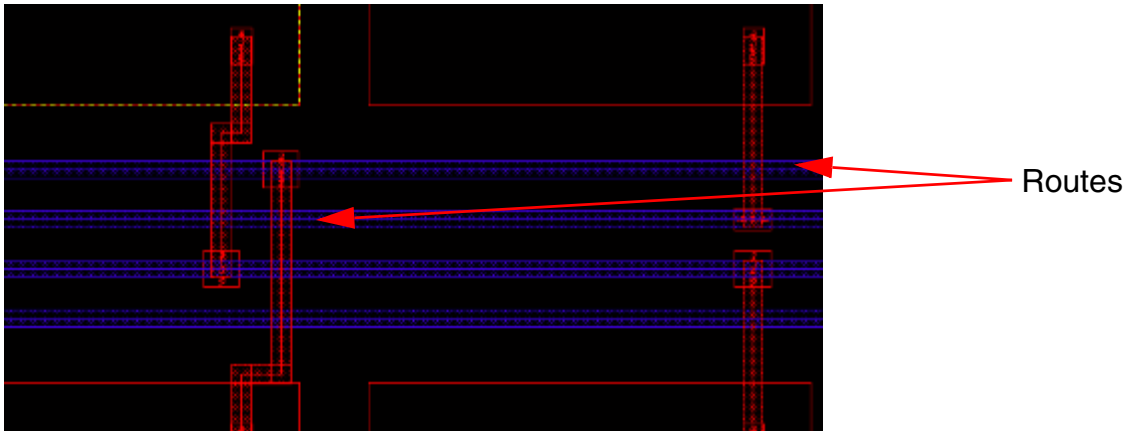
For more information, see Set the Channel Width.

Alternative SKILL functions:

- `mgSetColumnRoutingChannelWidth`
- `mgSetRowRoutingChannelWidth`
- `mgGetColumnRoutingChannelWidth`
- `mgGetRowRoutingChannelWidth`

Creating Pin-to-Trunk Routes

Select the Generate Routing from Topology button  from the Modgen Routing toolbar to establish pin-to-trunk connections between the terminals.



Use the Display/Hide Topology Objects button  to view or hide the topology pattern. By hiding the topology pattern, you can view only the routes.

Important

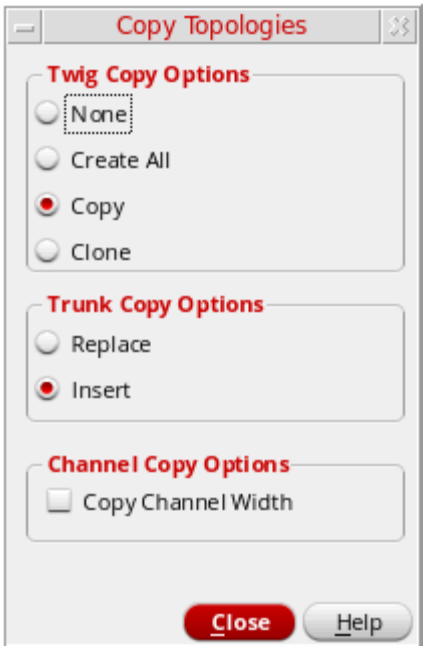
When the Modgen is launched from the Layout XL, Layout GXL, or Layout EAD cockpits, the pin-to-trunk router checks out 14 GXL tokens. Ensure that the required number of GXL tokes are available. No additional tokes are required when the tool is launched from the Layout EXL cockpit.

Copying Topology and Routing Information

When working on multi-row patterns, you can choose to copy the selected trunk topologies. While copying trunk topologies, the original configuration of the trunks is preserved. If a trunk located over a device is copied, then the new trunk will also be placed over the same device.

Note: Spacing values can be set to override the copied channel width values.

1. Select the *Copy Trunks Topologies* button  from the Modgen Routing toolbar to display the Copy Topologies form.



2. Choose the required twig copy option: *None*, *Create All*, *Copy*, or *Clone*.

Important

Only the *Copy* option copies constraint groups. The others options only create new twigs.

Note: Do not rotate trunks before copying them. If the source trunks are horizontal, then the destination must also be horizontal.

Note: Do not mix vertical and horizontal trunks while copying trunks.

3. Select the required *Trunk Copy Options*: *Replace* or *Insert.*
4. Choose *Copy Channel Width* to copy the channel width of the source trunks to the channel where the new trunks are placed.

   Note: This option is useful only if the new trunks are placed inside a channel, as opposed to over devices.

5. Click *Close*.

Note: What if the copied trunks are part of a matched group?

- If the source and destination channels are the same, then the matched trunks are added to the same matched group as the source.
- If trunks are copied to a different channel, then a new matched group is created at the target location with the new (copied) trunks as members. If only a few members of a matched group (more than one) are copied, then a new group is created with only the copied trunks as members. However, if only a single trunk that is part of a matched group is copied, then no group is created.

For more information about matched groups, see Creating Matched Groups.

For more information, see Copy Topologies.

Creating Matched Groups

A matched group is a set of topological trunks for which the specified properties (or match types) are the same. Creating a matched group helps maintain the same timing and resistance for all trunks within the set. Matched groups are stored in the Modgen storage.

The match type defines how the router modifies the trunk geometry to successfully match the selected trunks. You can create matched groups by matching twig geometries, trunk geometries, or both.

The available match types for twigs are:

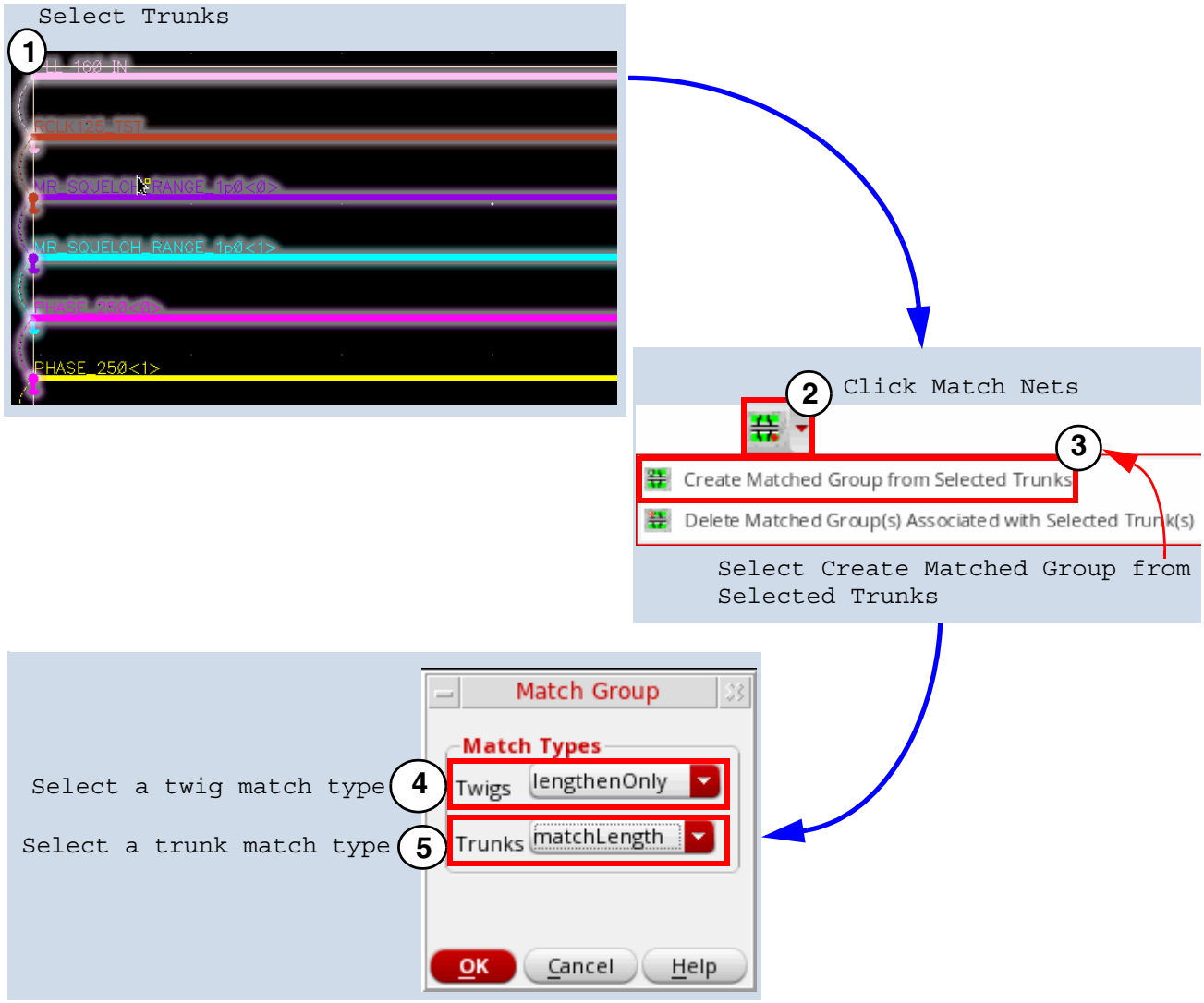
- **none:** No matching is done. This option lets you create a matched group by matching only the trunk geometries.
- **lengthenOnly:** Matching is done by lengthening the geometries associated with the twigs.
- **widenOnly:** Matching is done by widening the geometries associated with the twigs.
- **lengthenFirst:** Matching is done by first attempting to lengthen the geometries. If that is unsuccessful, then the geometries are widened.
- **widenFirst:** Matching is done by first attempting to widen the geometries. If that is unsuccessful, then the geometries are lengthened.

The available match types for trunks are:

- **none:** No matching is done. This option lets you create a matched group by matching only the twig parameters.
- **matchLength:** Matching is done by adjusting the length of the selected trunks such that they are equal.

To create a matched group:

1. Select the required trunks.
2. Click the Match Nets button on the Modgen Routing toolbar.
3. Select *Create Match Group from Selected Trunks*. The Matched Group form is displayed. You can also right-click and select *Create Matched Group from Selected Trunks*.
4. Select the required *Twigs* match type.
5. Select the required *Trunks* match type.



6. Click *OK*. The matched groups are displayed as shown below.

The selected trunks are grouped into a `matchGroup`.

To remove a trunk from its `matchGroup`, select the trunk, right-click, and select *Remove Selected Trunk(s) from Matched Group(s)*.

To highlight all trunks that belong to a particular matched group, select a trunk that belongs to the matched group, right-click, and select *Highlight Matched Group Members of Selected Trunk(s)*.

To delete a matched group:

1. Select a trunk that is part of the `matchGroup`.
2. Click the Match Nets button on the Modgen Routing toolbar.
3. Select *Delete Matched Group(s) Associated with Selected Trunk(s)*. Alternatively, you can right-click and select *Delete Matched Group(s) associated with Selected Trunk(s)*.

Related SKILL Functions:

- `mgCreateMatchGroupInModgen`
- `mgObjectHasMatchGroup`
- `mgGetMatchGroupMembers`
- `mgDestroyMatchGroupOnObject`
- `mgRemoveMemberFromMatchGroup`

Deleting Topology and Routing Information

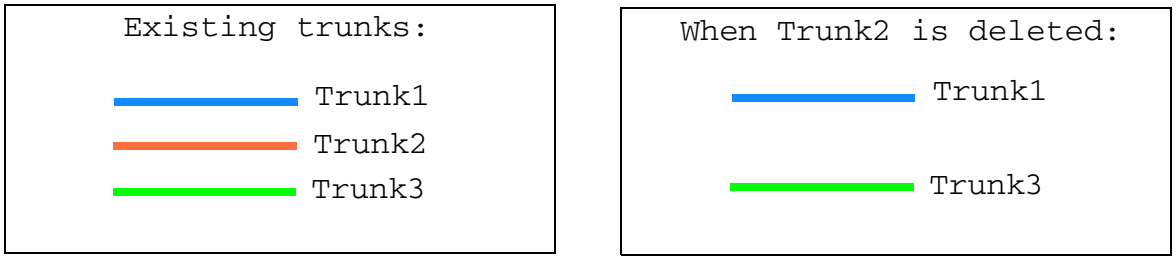
- Select the *Delete All Topology and Routing* button  from the Modgen Routing toolbar to delete all routing geometry and all topology in the Modgen.
- Select the *Delete All Trunks, Topologies, and Routing in Channel* button  from the Modgen Routing toolbar and draw a line (horizontal or vertical) in the desired channel to select the topology to be deleted. The selected topology is deleted from the specific channel without disturbing the settings of other channels.

  Related SKILL Function: `mgRemoveTopologyFromModgen`
- Select the *Delete All Strap Topologies and Routing* button  from the Modgen Routing toolbar to delete only the strap topologies and related routing information.
- Select the *Delete All Twig and Strap Routing* button  from the Modgen Routing toolbar to delete only the routes.

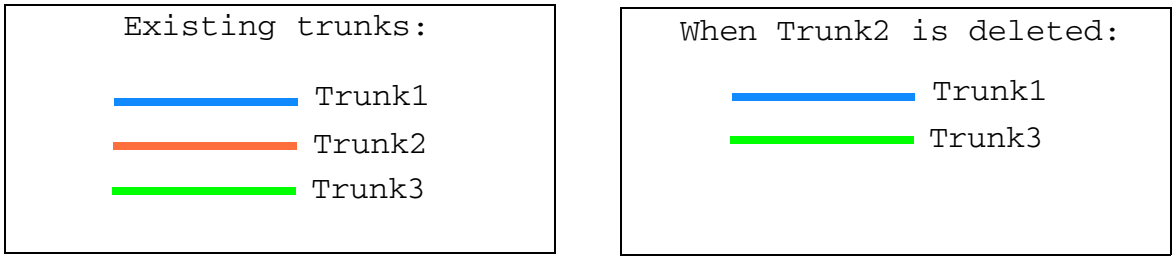
Using the Insert Mode

The Insert mode can be used to specify the trunk deletion behavior. Use the modgenPToTTwigLayer environment variable to switch ON (set to `t`) the Insert mode. Default is `nil`.

- **When the Insert Mode is turned OFF** (default): When a trunk is deleted, the existing trunks remain at their original positions even when some trunks are deleted.



- **When the Insert Mode is turned ON:** When a trunk is deleted, the positions of the remaining existing trunks are adjusted so that the distances between the trunks remain the same.



Here, when Trunk2 is deleted, the spacing between Trunk3 and Trunk1 is adjusted to be equal to the original offset from Trunk2 to Trunk3

Note: In this mode, when the Property Editor is used for adjusting the width of the trunk, the offsets for the Trunk2 to Trunk1 and Trunk3 to Trunk2 are adjusted so that the distance between the trunk geometries remains the same as before the width change was made.

Defining Manual Routes

After creating pin-to-trunk routes, you can make a few manual edits to the Modgen without disabling the Modgen.

Caution

Running the pin-to-trunk router after making manual edits may result in the loss of some of the manually created geometries.

Use the *Enable Hand Routing* command to enter the manual editing mode. After making the required modifications, the Modgen constraint automatically regenerates.

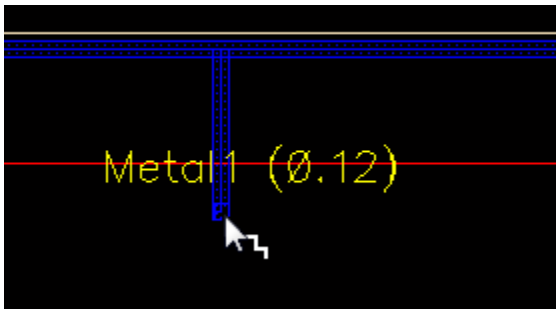
For example, after establishing pin-to-trunk routes, you can create additional routes manually.

To do this:

1. Select the *Enable Hand Routing* button  from the Modgen Routing toolbar.

   Environment Variable: modgenPToTOnCreateFigMode
2. Display the Create toolbar by right-clicking the title bar and selecting *Toolbars–Create*.
3. Select the Create Wire  icon.

4. Click to indicate the first point of the wire.
5. Continue to click at subsequent points of the wire.
6. Press Return or double-click to end the wire.



The Modgen automatically regenerates after the edit.

5

Placing Modgen Members Interactively

In this section you are going to learn placing Modgen members interactively in the Modgen Editor window. This section covers the following topics:

- Moving Instances
- Working With Rows and Columns
- Rotating and Flipping Instances
- Swapping Instances

Moving Instances

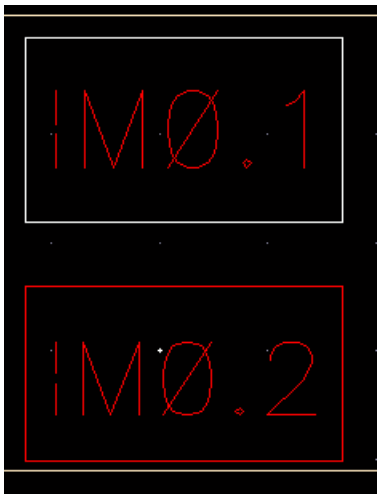
Modgen members can be moved interactively to other empty or non-empty cells on the same or a different row/column. If an instance is moved within the same row/column, the devices in that row/column are re-ordered. If an instance is moved to a different row/column, the positions of the moved instance and the device on which it is dropped are swapped.

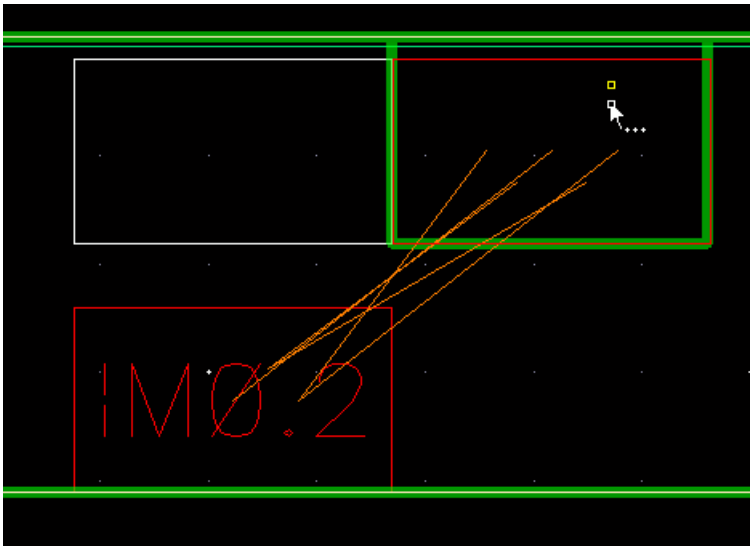
To move instances to an empty or non-empty cell, you can use one of the following methods:

Method 1

1. Click the instance you want to move

2. While holding down the mouse button, drag the instance to the required empty cell.





3. Release the mouse button to place the instance.

Method 2

Do one of the following:

- Right-click the instance. From the contextual menu, select *Move*. Click and release the left mouse button on the instance, drag it to the desired cell, and click again to place the instance.
- Select the instance. On the Modgen toolbar, click the *Move* button. Click and release the left mouse button on the instance, drag it to the desired cell, and click again to place the instance.

Working With Rows and Columns

This section includes the following topics:

- Adding Empty Rows and Columns
- Deleting Empty Rows and Columns
- Selecting Rows and Columns

Adding Empty Rows and Columns

This feature enables you add an empty row or column in the design display area.

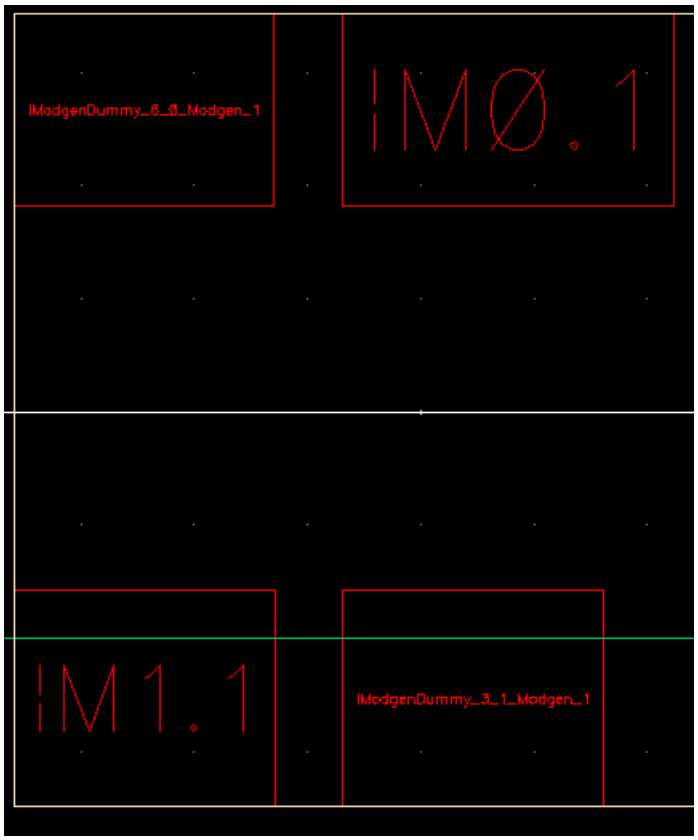
Adding An Empty Row

To add an empty row, select an instance in the row above or below which you want to add the row.

Do one of the following:

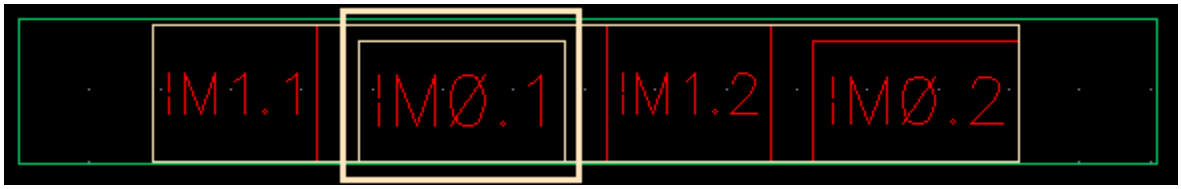
- On the Modgen toolbar, click the arrow next to the *Add Empty Row/Column* button and select *Add Empty Row Top* or *Add Empty Row Bottom*.
- Right-click the instance, navigate to *Add Empty Row/Column*, and select *Top* or *Bottom*.

A new empty row is added.



Adding an Empty Column

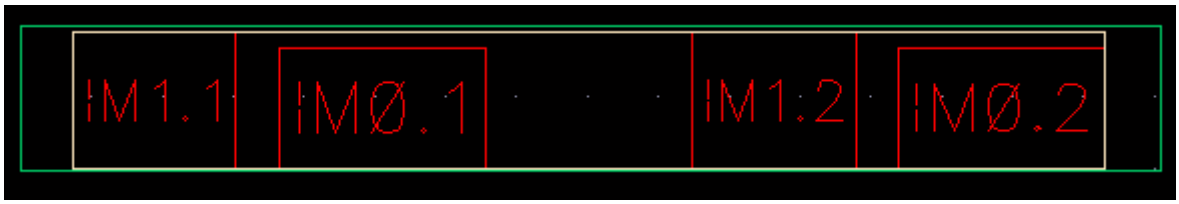
To add an empty column, select the instance to the right or left of which you want to add the column.



Do one of the following:

- On the Modgen toolbar, click the down arrow next to the *Add Empty Row/Column*  button and select *Add Empty Row Left* or *Add Empty Row Right*.

■ Right-click the instance, point to *Add Empty Row/Column*, and select *Left* or *Right*. An empty column is added.

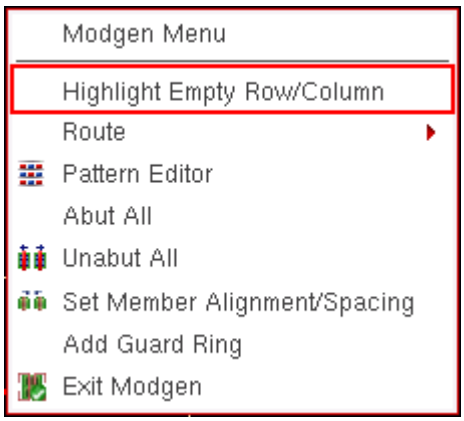


Instead of using the GUI, use the `mgAddEmptyRCCB` API to add empty rows and columns.

Highlighting Empty Rows and Columns

An empty row or column is highlighted in the Modgen editor window. The highlighting is on by default. When highlighting is on, the shortcut menu displays the *Remove Empty Row/column Highlight* option to turn the highlighting off as shown in the figure below.

When highlighting is off, the shortcut menu displays the *Highlight Empty Row/Column* option to turn the highlighting on, as shown in the figure below.



These options are displayed in the shortcut menu when:

- One or more instances are selected.
- No instance is selected.

However, the placement of this option in the shortcut menu is different in both these cases.

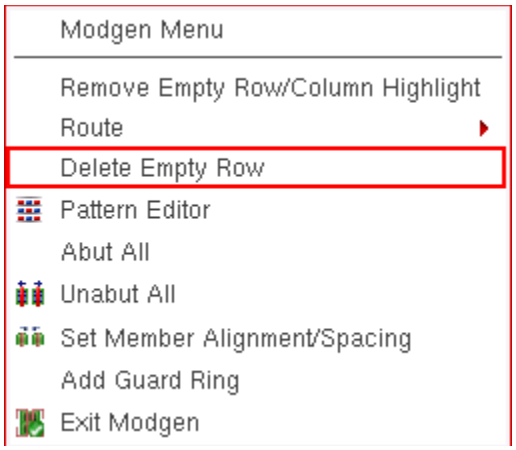
When you perform an *undo* operation in the Modgen editor window, the highlighting, if on, gets automatically turned off.

Alternative SKILL Function: `mgHilightEmptyRowColumnCB`

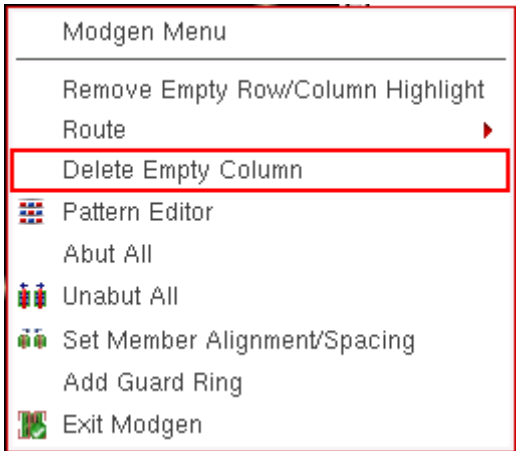
The Zoom to Fit  option also includes empty rows and columns in the Modgen editor window even if they are at the outer edges of the window.

Deleting Empty Rows and Columns

As a part of interactive Modgen editing, you can delete empty rows or columns from the Modgen editor window. To delete empty rows, you need to right-click the empty row and select the *Delete Empty Row* option from the shortcut menu, as shown in the figure below.



To delete empty columns, you need to right-click the empty columns and select the *Delete Empty Columns* option.



If you right-click at the location where an empty row and an empty column are overlapping, then both these options are displayed in the shortcut menu. You can select any of these options to either delete the row or column.

Note: If you right-click at the location where there is no empty row or columns, then the *Delete Empty Rows* or *Delete Empty Columns* option is not displayed in the shortcut menu.

These options are displayed in the shortcut menu when:

- One or more instances are selected.
- No instance is selected.

Alternative SKILL Function: mgDeleteEmptyRowColumnCB

To delete all empty rows and columns of the Modgen, use mgDeleteAllEmptyRowColumnCB.

Selecting Rows and Columns

To move or swap the entire row or column, you may need to select the entire row or column.

Selecting Rows

To select the entire row:

1. Select the instance in a row.
2. Right-click the instance, point to *Select Row/Column*, and select *Select Entire Row*.
3. The entire row will be selected.

Selecting Columns

To select the entire column:

1. Select the instance in a column.
2. Right-click the instance, point to *Select Row/Column*, and select *Select Entire Column*.
3. The entire column will be selected.

Alternative SKILL Function: mgSelectRowColCB

Rotating and Flipping Instances

You rotate or flip instances for the following reasons:

- To optimize placement
- To optimize routing; for example, to optimally connect resistors in a series
- To allow for matched routing

Rotating Instances

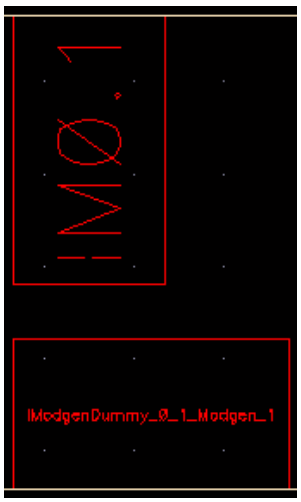
To rotate instances:

1. Select the instance that you want to rotate.

2. Do one of the following:
   - On the Modgen toolbar, click the *Rotate Left* or *Rotate Right* button.
   - Right-click the instance, point to *Rotate/Flip*, and select *Rotate Left* or *Rotate Right*.

     Alternative SKILL Functions: `mgRotateLeftCB`, `mgRotateRightCB`

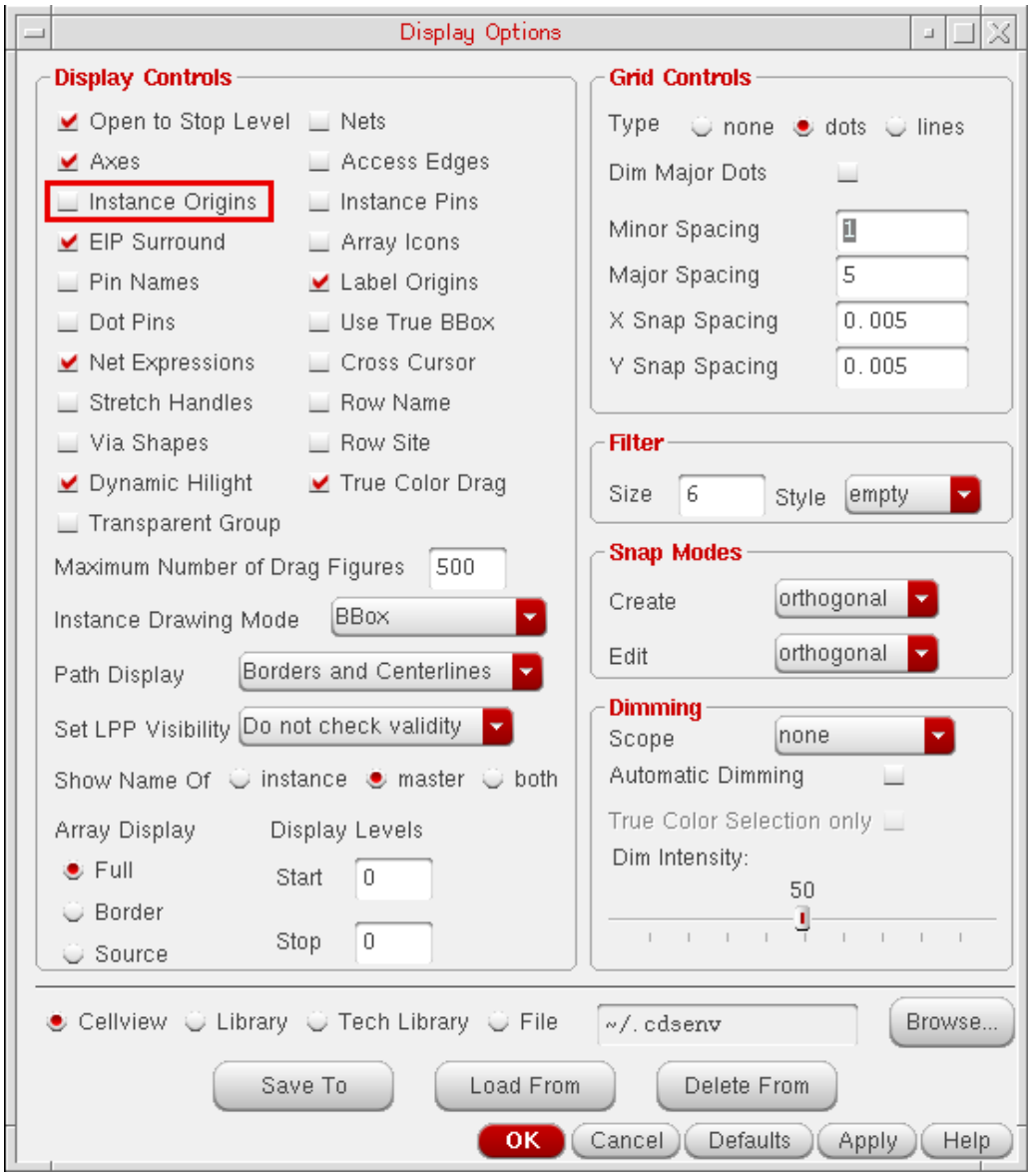
The selected instance is rotated.



Flipping Instances

Before you flip an instance:

1. In the layout window, choose *Options – Display* to view the Display Options form.

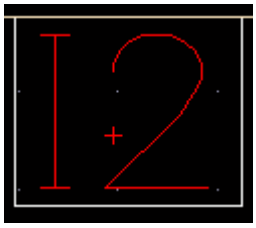


2. In the *Display Controls* group box, select the *Instance Origins* check box.
3. Click the *OK* button to save the settings.

Note: A plus sign appears on the instance, which indicates whether the instance was flipped.

To flip an instance:

1. Select the instance in the design display area.



2. Do one of the following:

   - On the Modgen toolbar, click the *Flip Horizontal* or *Flip Vertical* button.
   - Right-click the instance, point to *Rotate/Flip*, and select *Flip Horizontal* or *Flip Vertical*.

     Alternative SKILL Functions: `mgFlipHorizontalCB`, `mgFlipVerticalCB`

The selected instances is flipped. In the figure below, notice that the plus sign has moved upward.

Swapping Instances

By using the swap function, you can interchange the position of two instances, rows, or columns. This helps optimize routing. For example, by swapping instances, you can optimally connect resistors in a series.

To swap instances:

1. Select the two required instances in the design display area.

2. Do one of the following:

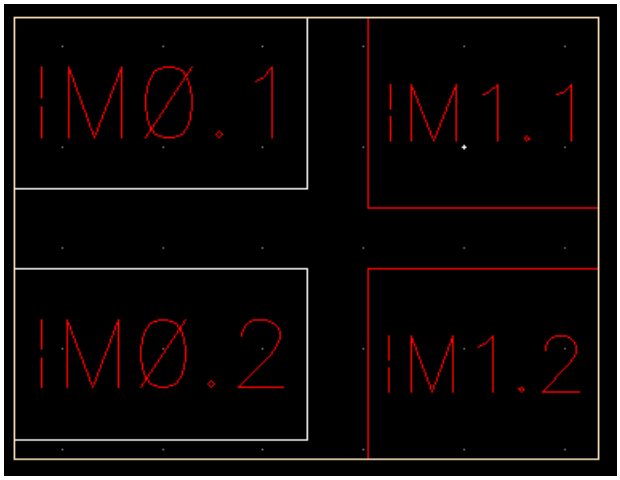
   - On the Modgen toolbar, click the arrow next to the *Swap*  button, and select *Swap Instances*.
   - Right-click the selected instances, point to *Swap*, and select *Swap Instances*.

     Alternative SKILL Function: `mgSwapCB`

The selected instances are swapped. In the figure below, notice that `M1.2` and `M0.2` are swapped.

To swap an entire row:

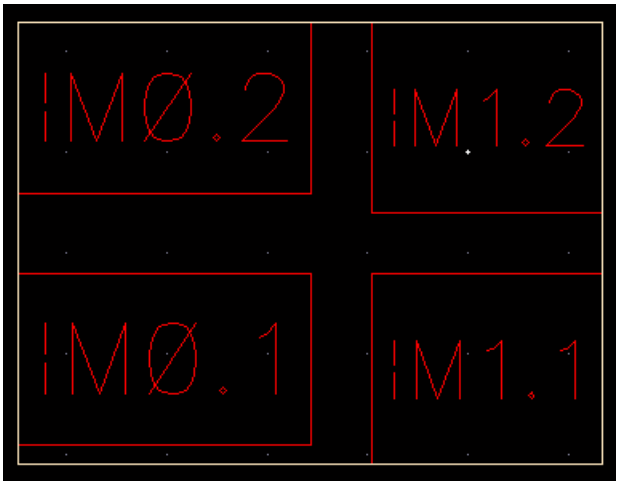
1. Select an instance each in the two rows that you want to swap.



2. Do one of the following:

   - On the Modgen toolbar, click the arrow next to the *Swap*  button, and select *Swap Rows*.
   - Right-click a selected instance, point to *Swap*, and select *Swap Rows*.

3. The selected rows are swapped. In the figure below, notice that the row (`M0.2 | M1.2`) is swapped with the row (`M0.1 | M1.1`).



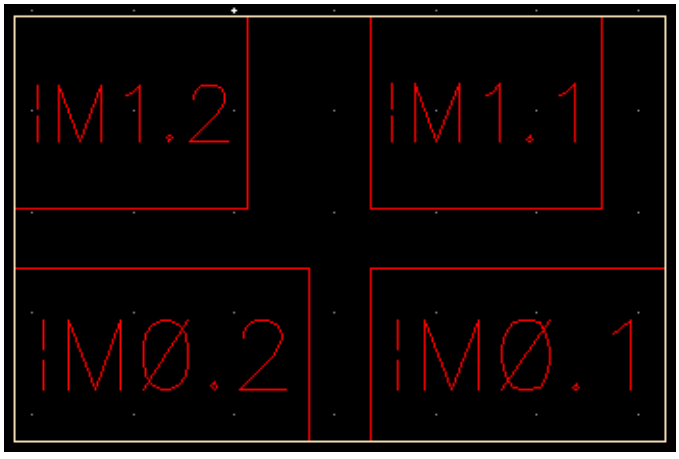
To swap entire columns:

1. Select an instance each in the two columns that you want to swap.

2. Do one of the following:

   - On the Modgen toolbar, click the arrow next to the *Swap*  button, and select *Swap Columns*.
   - Right-click a selected instance, point to *Swap*, and select *Swap Columns*.

3. The selected columns are swapped. In the figure below, notice that the column `(Mi.2 | M0.2)` is swapped with the column `(M1.1 | M0.1)`.



6

Modgen Forms

This chapter lists the controls in the Modgen forms.

Note: Many of the options described in this section have a corresponding environment variable. For a full listing of these environment variables, see List of Modgen Environment Variables.

- Add/Replace Twig Topologies
- Apply Reuse Template
- Body Contact Options
- Copy Topologies
- Create Single Strap Topology
- Create Single Trunk and Topologies
- Surround Modgen
- Create Strap Topologies
- Create Trunks and Topologies
- Dummy Options
- Grid Pattern Form
- Guard Ring Options
- Identical Guard Ring Options
- Matched Group
- Reuse Template Exaction
- Modgen Pattern Editor
- Set the Channel Width
- Select Merge Layers

- Set Member Alignment and Spacing
- Surround Modgen

Add/Replace Twig Topologies

Use the **Add / Replace Twig Topologies** form to add new twigs or recreate twigs that were accidentally deleted.

Filter Options specifies the Source or *Target Terminals* and *Search Direction*, which indicates the direction along which the connections need to be made. Corresponding environment variables: modgenPToTTwigDirectionDown, modgenPToTTwigDirectionLeft, modgenPToTTwigDirectionOver, modgenPToTTwigDirectionRight, modgenPToTTwigDirectionUp,

Replace Existing Twigs replaces existing twigs with the new ones.

Allow Multiple Connections To A Terminal enables Multiple trunk connections to each terminal.

Twig Options specifies the twig parameters.

Prefer to Route On Pin Layers specifies that routing should be done on the pin layers instead of the drawing layers.

Twigs Layer lets you specify the layer on which the twigs need to be created. Corresponding environment variable: modgenPToTTwigLayer

Twigs Min Num Cuts specifies the minimum number of cuts that the vias connecting the twigs to other objects need to have. Corresponding environment variable: modgenPToTTwigMinNumCuts

Twigs Width Type lets you choose one of the following options:

> ***minWidth*:** Considers the minWidth for the layer, which is defined in the technology file, as the default twig width.
>
> ***Absolute Width*:** Specifies an absolute twig width value.
>
> ***Relative Width (Pin%)*:** Specifies the percentage of the twig width value with respect to the pin width value. For example for value 80%, the twig width is 80% of the pin width value.
>
> Corresponding environment variables: modgenPToTTwigWidthType, modgenPToTTwigRelativeWidth

Advanced Options provides options for controlling via configurations and pin cover settings within the topology pattern. This section comprises the following tabs:

Via Controls includes the following options to specify how vias need to be positioned and aligned on trunks and twigs:

Via Orientation specifies the default via orientation: *Horizontal*, *Vertical*, or *None*. The bBox of the via cuts is aligned in the specified direction. For example, a via with two cuts is rendered top-down if the preferred via orientation is *Vertical*.

Corresponding environment variables: modgenPToTViaControlOrient, modgenPToTViaControlOrientEnable

Via Extension Orientation specifies the preference of the via extension or enclosure orientations. For multiple cut vias, a higher preference is given to vias with cut boxes lining up in a certain direction. Therefore, a vertical-cut bBox via may have a horizontal extension. This means that the extended metal portion over the cut bBox in the horizontal direction is larger than in the vertical direction.

Corresponding environment variables: modgenPToTViaControlExtensionOrient, modgenPToTViaControlExtensionOrientEnable

Via Inline, when selected, places a higher preference for vias that are fully enclosed or inline with the wire.

For more information, see Adding Twigs.

Corresponding environment variables: modgenPToTViaControlInline, modgenPToTViaControlInlineEnable

Via Offset, when selected, honors the via offset values.

Corresponding environment variables: modgenPToTViaControlOffset, modgenPToTViaControlOffsetEnable

Cut Class specifies the following cut class parameters:

By Size specifies the *Width* and *Height* of the associated cut classes. For more information about the cut class and the via definition values, see Cut Class and Via Definitions in the Create Via Options form documentation in Virtuoso Layout Suite L User Guide.

Corresponding environment variables: modgenPToTViaControlCutClass1, modgenPToTViaControlCutClass2, modgenPToTViaControlCutClassEnable

By Name specifies the *Layer* and *Name* of the associated cut classes.

Pin Cover tab provides the following options to control the devices that cover the pin shapes:

Tap Via Percent specifies the percentage of a pin shape to be covered by the tapping via on all layers. The percentage is measured in the direction perpendicular to the trunk.

Corresponding environment variable: modgenPToTSpecifyChannelWidth

Tap Lower Via Percent specifies the percentage of a pin shape to be covered by the lower layers of the tapping vias. This value applies all layers expect the top-most via. The percentage is measured in the direction perpendicular to the trunk.

Corresponding environment variable: modgenPToTSpecifyChannelWidth

Via Mode Trunk Over specifies the via mode to be applied when the trunk or a pin is covered by the via stack. Available options are:

- *matchTrunk* (default): Covers the width of the trunk

- *full*: Covers the entire pin shape

- *fullExceptTop*: Covers the entire trunk on the top via layer and the entire pin on the lower via layers.

Corresponding environment variables:
modgenPToTPinCoverTapLowerViaPercentEnable,
modgenPToTPinCoverViaModePercentTrunkOver

Via Mode Percent Trunk Over controls the percentage of a trunk or a pin to be covered by the via stack. The percentage is measured in the direction. This option can be used only when *Via Mode Trunk Over* is set to *full* or *fullExceptTop*.

Corresponding environment variable: modgenPToTSpecifyChannelWidth

Via Percent (%) specifies the depth of via coverage when the via overlaps with a pin.

Related Topics

- Adding Twigs

Apply Reuse Template

(ICADVM20.1 EXL Only) Use the **Add Reuse Template** form to generate a new Modgen constraint using a Modgen template file.

Device Group specifies the device group under which the required Modgen pattern file is located.

Pattern File specifies the name of the Modgen reuse template file.

Rows specifies the number of rows to be included in the new Modgen. The default value is `1`.

Related Topics

- Reusing the Modgen Template File

Body Contact Options

Use the **Body Contact Options** form to specify the parameters for body contacts created by Modgens.

Apply To lists the instances for which body contacts need to be created.

Type provides a list of body contacts defined in the technology file. Select the required type of body contact.

Net lists the nets that are connected to objects in the active Modgen. Choose the net to which you want body contacts attached to. To choose a net that is present in the cellview, but not in the current Modgen, type the net name in the *Net* combo box. The entry is validated against all existing nets in the current cellView. If a matching net is not available in the current cellView, then a warning message is issued and the net is created.

Layer lets you select the reference layer for the body contacts.

Environment Variable: modgenDefVCSRefPurpose

Separation specifies the distance (in microns) between the body contacts and devices. This value is added to minimum DRC to space the body contact.

Note: You can also specify a negative value in this field. When you specify a negative value, the body contact is placed inside of the value that Modgen is calculating as a legal DRC distance.

Remember Values ave the specified values for all dummy devices.

Related Topics

- Defining Body Contact Properties
- Removing Body Contacts
- Placing Body Contacts
- Corner Case Conditions for Grid Placer

Copy Topologies

Use the **Copy Topologies** form to copy only the trunk topologies or the entire channel topology.

Twig Copy Options specifies the twig creation behavior when trunks are copied into a channel. Choose one of the following options:

None: No twigs are created. Use the *Add / Replace Twig Topologies* command to add new twigs.

Create All: All existing twigs that have instTerms that match the net, and that do not have a twig attached to the destination channel are created in the new channel.

Copy: Copies the constraint group on the twig. This option can be used only when the source and destination channels have matching surrounding instances. This means that the instMaster, terminal, net, and relative location of the source and destination channels should match.

Clone: Creates twigs that are similar to the surrounding instances. Twigs are cloned only when the instMaster, terminal, and net of the surrounding instance matches the source twig.

Trunk Copy Options specifies the behavior when trunks are copied into a channel. Select a suitable option:

Replace: All existing trunks in the channel are deleted before adding the copied trunks.

Insert: The copied trunks are inserted into the channel. For channels above channel 0, the trunks are inserted below the instance above the channel. For channel 0, the trunks are inserted at the maximum distance away from the row 0 or column 0 instance.

Copy Channel Width lets you copy the channel width of the source trunks to the channel where the new trunks are placed.

Related Topics

- Copying Topology and Routing Information

Create Single Strap Topology

Use the **Create Single Strap Topology** form to define straps between individual pins.

Net specifies the net on which the straps need to be created during pin-to-trunk routing.

Object 1 and **Object 2** display the names of the pins that need to be strapped during pin-to-trunk routing.

InstTerm1 and **InstTerm2** indicate the instances in Objects 1 and 2 to be strapped.

Strap Layer specifies the layer on which strapping should take place during pin-to-trunk routing.

Strap Width specifies the width of the strap.

Related Topics

- Creating Single Strap Topologies

Create Single Trunk and Topologies

Use the **Modgen Create Single Trunk and Topologies** form to create incremental trunks for a topology.

Net specifies the net for which a trunk needs to be added.

Layer specifies the layer on which the trunk needs to be created.

Width specifies the trunk width.

Trim Trunk lets you select the side from which trunks need to be trimmed while routing:

None: Trunks are not trimmed.

Both: Trunks are trimmed from both ends.

Left/Bottom: Trunks are trimmed along the left and bottom edges. Trunks are trimmed to the last twig connection.

Right/Top: Trunks are trimmed along the right and top edges. Trunks are trimmed to the last twig connection.

Trunk In Channel Spacing Options lets you specify the *Spacing Before* and *Spacing After* values between the trunk and other trunks or devices.

Note: The trunk spacing value cannot be smaller than the `minSpacing` value specified in the technology file.

Trunk Over Device Alignment Options includes the *InstTerm Edge* and *Trunk Edge* options, which are used together to specify how instance and trunk edges need to be aligned when trunks are drawn over instances, and not inside any channel. For example, if *InstTerm Edge* is set to *Bottom/Left Edge* and *Trunk Edge* is set to *Top/Right Edge*, then the resulting trunk's top edge is aligned with the instTerm's bottom edge. If the trunk is vertical, then the resulting trunk's right edge is aligned with the instTerm's left edge.

Generate twigs connecting trunks to channel objects controls whether topological twigs, which connect trunks to channel objects (instances and body contacts), are generated when creating trunks. To create twigs, select the check box (default). The options in the *Twig Options* section are enabled. When the check box is cleared, options in the *Twig Options* section are disabled.

Corresponding environment variable: modgenPToTGenTrunkTwigs.

Twig Options specifies options for defining twigs for the new trunk:

Prefer to Route On Pin Layers: Specifies that routing should be done on the pin layers instead of the drawing layers.

Allow Multiple Connections To A Terminal: Allows more than one twig to be connected to a terminal.

Twigs Layer: The layer on which the twigs needs to be created.

Twigs Min Num Cuts: The minimum number of cuts to be used for the vias connecting the twigs to the terminals and trunks.

Twigs Width Type: Width of the twigs. The width can be set to be equal to the layer *MinWidth*, an *Absolute Width* specified in the corresponding field, or *Relative Width* (as a percent) of the pin width for the layer.

Advanced Options provides options for controlling via configurations and pin cover settings within the topology pattern. This section comprises the following tabs:

Via Controls includes the following options to specify how vias need to be positioned and aligned on trunks and twigs:

Via Orientation specifies the default via orientation: *Horizontal*, *Vertical*, or *None*. The bBox of the via cuts is aligned in the specified direction. For example, a via with two cuts is rendered top-down if the preferred via orientation is *Vertical*.

Corresponding environment variables: modgenPToTViaControlOrient, modgenPToTViaControlOrientEnable

Via Extension Orientation specifies the preference of the via extension or enclosure orientations. For multiple cuts vias, a higher preference is given to vias with cut boxes lining up in a certain direction. Therefore, a vertical-cut bBox via may have a horizontal extension. This means that the extended metal portion over the cut bBox in the horizontal direction is larger than in the vertical direction.

Corresponding environment variables: modgenPToTViaControlExtensionOrient, modgenPToTViaControlExtensionOrientEnable

Via Inline, when selected, places a higher preference for vias that are fully enclosed or inline with the wire.

Corresponding environment variables: modgenPToTViaControlInline, modgenPToTViaControlInlineEnable

Via Offset, when selected, honors the via offset values.

Corresponding environment variables: modgenPToTViaControlOffset, modgenPToTViaControlOffsetEnable

Cut Class Width and Height Specifies the *Width* and *Height* of the associated cut classes. For more information about the cut class and the via definition values, see Cut Class and Via Definitions in the Create Via Options form documentation in Virtuoso Layout Suite L User Guide.

Corresponding environment variables: modgenPToTViaControlCutClass1, modgenPToTViaControlCutClass2, modgenPToTViaControlCutClassEnable

Pin_Cover tab provides the following options to control the devices that cover the pin shapes:

Tap Via Percent specifies the percentage of a pin shape to be covered by the tapping via on all layers. The percentage is measured in the direction perpendicular to the trunk.

Tap Lower Via Percent specifies the percentage of a pin shape to be covered by the lower layers of the tapping vias. This value applies all layers expect the top-most via. The percentage is measured in the direction perpendicular to the trunk.

Via Mode Trunk Over specifies the via mode to be applied when the trunk or a pin is covered by the via stack. Available options are:

- *matchTrunk* (default): Covers the width of the trunk

- *full*: Covers the entire pin shape

- *fullExceptTop*: Covers the entire trunk on the top via layer and the entire pin on the lower via layers.

Via Mode Percent Trunk Over controls the percentage of a trunk or a pin to be covered by the via stack. The percentage is measured in the direction. This option can be used only when *Via Mode Trunk Over* is set to *full* or *fullExceptTop*.

Via Percent (%) specifies the depth of via coverage when the via overlaps with a pin.

Related Topics

- Creating Incremental Trunks

Create Strap Topologies

Use the **Create Strap Topologies** form to define strap parameters.

Net specifies the net to which the to-be-strapped pins belong.

Pin Layer indicates the layer on which the pins are located.

Strap Layer indicates the layer on which strapping should take place during pin-to-trunk routing.

Strap Width specifies the width of the strap.

Max Pin Distance lets you specify the maximum edge-to-edge distance between the adjacent to-be-strapped pins.

Max Pin Count lets you specify the maximum number of pins that can be strapped together.

Strap Direction indicates the direction along which the straps need to be created.

Related Topics

- Adding Straps

Create Trunks and Topologies

Use the **Create Trunks and Topologies** form to define a new topology pattern or to edit an existing topology pattern.

Specify Channel Nets specifies the channel nets to be included in the topology pattern. By default, all the available nets are selected. To remove nets, select the nets and move them from the *Channel Nets* box to the *All Nets* box. Use the *Up* and *Down* buttons to change the order of the channel nets.

CellView Nets allows you to add nets that exist in the layout or schematic views. The selected nets are added to the Add Nets box. You can choose to move these to the Channel Nets box.

Trunk Options lets you specify the following options:

Trunks Layer specifies the layer on which trunks are to be created.

Trunks Width specifies the trunk width.

Note: The trunk width value cannot be smaller than the `minWidth` value specified in the technology file.

Min Spacing specifies the `minSpacing` value to be used to calculate the trunk spacing.

Absolute Spacing specifies the absolute trunk spacing value. Ensure that the value is equal to or greater than *Min Spacing*.

Center Trunks in Channel lets you choose if trunks need to be centered in the channel such that the distance between the trunks and devices is the same from both, the top and bottom edges. This option is selected by default.

Specify Trunk-to-Device Spacing specifies the default distance between a trunk and a device.

Top (or Right) specifies the default distance between the trunk and any device located at the top or right of the trunk.

Reference specifies the layer and purpose of a topological trunk that is anchored to an instance. The bounding box of the specified layer and purpose inside the instance is used to determine the trunk's location. In other words, the trunk's orthogonal offset is from one side of the reference layer and purpose bounding box.

Alternative SKILL functions: `mgSetTrunkRefLayerPurpose`, `mgGetTrunkRefLayerPurpose`

Specify Channel Width specifies the channel width.

Trim Trunk lets you select the side from which trunks need to be trimmed while routing:

None: Trunks are not trimmed.

Both: Trunks are trimmed from both ends.

Left/Bottom: Trunks are trimmed along the left and bottom edges. Trunks are trimmed to the last twig connection.

Right/Top: Trunks are trimmed along the right and top edges. Trunks are trimmed to the last twig connection.

Generate twigs connecting trunks to channel objects controls whether topological twigs, which connect trunks to channel objects (instances and body contacts), are generated when creating trunks. To create twigs, select the check box (default). The options in the *Twig Options* section are enabled. When the check box is cleared, options in the *Twig Options* section are disabled.

Corresponding environment variable: modgenPToTGenTrunkTwigs.

Twig Options lets you specify the following options:

Prefer to Route On Pin Layers specifies that routing should be done on the pin layers instead of the drawing layers.

Allow Multiple Connections To A Terminal allows more than one twig to be connected to a terminal.

Twigs Layer specifies the layer on which twigs need to be created.

Twigs Min Num Cuts specifies the minimum number of cuts that the twigs need to generate.

Twigs Width Type lets you choose one of the following width types:

MinWidth: Considers the minWidth for the layer, which is defined in the technology file, as the default twig width.

Absolute Width: Specifies an absolute twig width value.

Relative Width (Pin%): Specifies the percentage of the twig width value with respect to the pin width value. For example for value 80%, the twig width is 80% of the pin width value.

Advanced Options provides options for controlling via configurations and pin cover settings within the topology pattern. This section comprises the following tabs:

Via Controls includes the following options to specify how vias need to be positioned and aligned on trunks and twigs:

Via Orientation specifies the default via orientation: *Horizontal*, *Vertical*, or *None*. The bBox of the via cuts is aligned in the specified direction. For example, a via with two cuts is rendered top-down if the preferred via orientation is *Vertical*.

Corresponding environment variables: modgenPToTViaControlOrient, modgenPToTViaControlOrientEnable

Via Extension Orientation specifies the preference of the via extension or enclosure orientations. For multiple cuts vias, a higher preference is given to vias with cut boxes lining up in a certain direction. Therefore, a vertical-cut bBox via may have a horizontal extension. This means that the extended metal portion over the cut bBox in the horizontal direction is larger than in the vertical direction.

Corresponding environment variables: modgenPToTViaControlExtensionOrient, modgenPToTViaControlExtensionOrientEnable

Via Inline, when selected, places a higher preference for vias that are fully enclosed or inline with the wire.

Corresponding environment variables: modgenPToTViaControlInline, modgenPToTViaControlInlineEnable

Via Offset, when selected, honors the via offset values.

Corresponding environment variables: modgenPToTViaControlOffset, modgenPToTViaControlOffsetEnable

Cut Class Width and Height Specifies the *Width* and *Height* of the associated cut classes. For more information about the cut class and the via definition values, see Cut Class and Via Definitions in the Create Via Options form documentation in Virtuoso Layout Suite L User Guide.

Corresponding environment variables: modgenPToTViaControlCutClass1, modgenPToTViaControlCutClass2, modgenPToTViaControlCutClassEnable

Pin Cover tab provides the following options to control the devices that cover the pin shapes:

Tap Via Percent specifies the percentage of a pin shape to be covered by the tapping via on all layers. The percentage is measured in the direction perpendicular to the trunk.

Tap Lower Via Percent specifies the percentage of a pin shape to be covered by the lower layers of the tapping vias. This value applies all layers expect the top-most via. The percentage is measured in the direction perpendicular to the trunk.

Via Mode Trunk Over specifies the via mode to be applied when the trunk or a pin is covered by the via stack. Available options are:

- *matchTrunk* (default): Covers the width of the trunk

- *full*: Covers the entire pin shape

- *fullExceptTop*: Covers the entire trunk on the top via layer and the entire pin on the lower via layers.

Via Mode Percent Trunk Over controls the percentage of a trunk or a pin to be covered by the via stack. The percentage is measured in the direction. This option can be used only when *Via Mode Trunk Over* is set to *full* or *fullExceptTop*.

Via Percent (%) specifies the depth of via coverage when the via overlaps with a pin.

Related Topics

- Creating and Editing Topology Patterns

Dummy Options

Use the **Dummy Options** form to specify the net to which dummy devices need to be attached and the type of dummy devices.

Default in the **Dummy Net** section lists the nets that are connected to objects in the active Modgen. Choose the net to which you want the dummy devices attached. To choose a net that is present in the cellview but not in the current Modgen, type the net name in the *Default* combo box. The entry is validated against all existing nets in the current cellView. If a matching net is not available, then a warning message is issued and the net is created.

Environment Variable: modgenDummyNet

Set Individual Terminal Net sets individual nets for dummy terminals, as opposed to the same net for all four terminals.

Type in the **Dummy Type** section specifies the default type for dummy devices to be added. If set to *neighbor*, the dummy device type depends on the location of the dummy and the setting of the modgenMakeMinDummies environment variable. If set to *default*, then use the **Default Dummy** section (below) or the modgenDummyLib, modgenDummyCell, and modgenDummyView environment variables to specify the default dummy device type. If set to *copy*, then a dummy that is similar to the source dummy is added.

Environment Variable: modgenPhysConfigs

Specify Parameters activates the options in the *Dummy Parameters* section.

Dummy Parameters specifies the *Number of Fingers*, *Length*, and *Number of Fins* for the dummies. When set to *CDF Default*, values specified in the CDF is considered. When set to *Same As Neighbor*, corresponding value of the neighboring device is considered. When set to *Specify*, you can specify the required value.

Environment Variables:
modgenDummyNumFingersOptions,modgenDummyNumFingersValue,
modgenDummyWidthOptions, modgenDummyLengthValue,
modgenDummyWidthOptions, modgenDummyWidthValue

Default Dummy specifies the default dummy to be used when **Dummy Type** (above) is set to *default*. Click *Browse* to select the required library, cell, and view for the dummy devices.

Environment Variable: modgenDummyLib, modgenDummyCell, modgenDummyView

Remember Values specifies Modgen to save the specified values for all dummy devices and to load them until they are overwritten with new values.

Environment Variable: modgenRememberDummyVals

Related Topics

- Adding Dummy Device Rows or Columns

Grid Pattern Form

Use the **Grid Pattern Form** to edit the default preset values before generating a grid pattern.

Rows specifies the number of rows in the grid pattern.

Cols specifies the number of columns in the grid pattern.

Auto automatically generates a square grid. When this option is selected, the *Rows* and *Cols* fields are turned inactive and their existing values are ignored.

Pattern specifies the interdigitation pattern for devices on the grid.

Orientation specifies the orientation of devices on the grid.

Group By includes the following options:

All places all instances of each symbol.

Interdigitate By specifies the integer to use for interdigitation. Instances are placed in accordance with the specified interdigitation pattern.

Placement specifies whether the devices should be placed sequentially by the rows (*Horizontal*) or by the columns (*Vertical*).

Related Topics

- Working with Presets

Guard Ring Options

Use the **Guard Ring Options** form to create multipath Part (MPP) guard rings.

Type lets you select the type of guard ring to be created. Here, *none* indicates no guard ring. Use this option to delete an existing one, *ring* indicates a single guard ring around the entire Modgen, and *pane* indicates the new window pane configuration, which indicates that the guard ring needs to cover all devices as well as the entire Modgen.

Shape specifies the shape of the guard ring.

Net lets you select the net to which the guard ring needs to be attached.

Spacing specifies the spacing between the guard ring and other devices.

Left Spacing, **Right Spacing**, **Top Spacing**, and **Bottom Spacing** specify the spacing of the guard ring from the respective sides.

MPP to Use specifies the MPP to be used for the guard ring

Related Topics

- Creating Multipath Part (MPP) Guard Rings

Identical Guard Ring Options

(ICADVM20.1 Only) Use the **Identical Guard Ring Options** form to define identical guard rings in your design.

Important

The *Create Identical Guard Ring* command is available only if the PDK that you are using is configured to support identical guard rings. For more information on this capability, contact your Cadence Customer Support representative.

Type defines the way in which the identical guard ring surrounds the adjoining Modgen instances. For more information about the different types, see Creating Identical Guard Rings (ICADVM20.1 Only).

GR Definition lists the available guard ring definitions provided by the PDK, which define various aspects of the guard ring. They provide information such as the unit cell `lib:cell:view`, its parameters, and parameter callbacks, which helps Modgens instantiate the guard ring correctly.

Net specifies the net to which the guard ring needs to be connected.

Break GR provides spaces for connecting guarded instances via the **`Metal1`** layer routes.

Add Corners creates guard rings at all available corners around the selected instances.

Turn Top Side On, **Turn Left Side On**, **Turn Right Side On**, **Turn Bottom Side On** are the side switches that can be used turn on or off the guard ring for the specified sides.

Horizontal Strip Width (in fins) specifies the vertical width (as a number of fins) of the horizontal guard ring strips.

Number of Rows Between Strips lets you set the number of instance rows that are required between strips of guard rings, counting from the bottom row.

Related Topics

- Creating Identical Guard Rings (ICADVM20.1 Only)

Matched Group

Use the **Matched Group** form to create groups of topological trunks by matching the specified twig or trunk geometries.

Twigs lets you select a twig match type based on which the matched group must be created. Valid values are:

- **none:** No matching is done. This option lets you create a matched group by matching only the trunk geometries.
- **lengthenOnly:** Matching is done by lengthening the geometries associated with the twigs.
- **widenOnly:** Matching is done by widening the geometries associated with the twigs.
- **lengthenFirst:** Matching is done by first attempting to lengthen the geometries. If that is unsuccessful, then the geometries are widened.
- **widenFirst:** Matching is done by first attempting to widen the geometries. If that is unsuccessful, then the geometries are lengthened.

Trunks lets you select a trunk match type based on which the matched group must be created. Valid values are:

- **none:** No matching is done. This option lets you create a matched group by matching only the twig parameters.
- **matchLength:** Matching is done by adjusting the length of the selected trunks such that they are equal.

Related Topics

- Creating Matched Groups

Reuse Template Exaction

(ICADVM20.1 EXL Only) Use the **Reuse Template Extraction** form to extract reusable templates from the given source layout or Modgen.

Device Group specifies the group under which the pattern file must be created.

Pattern File specifies the name of the Modgen reuse template file (with the `.txt` file extension).

Related Topics

- Generating a Modgen Template File

Modgen Pattern Editor

Use the **Modgen Pattern Editor** to specify the interdigitation pattern for the Modgen.

Pattern Type lets you select a pattern type: *Interdigitate* or *Customize*.

Rows specifies the number of rows in the pattern.

Interdigitate By lets you specify the integer to use for interdigitation.

Reset to default placement settings applies default placement settings to all devices in the Modgen. For example, selecting this option unabuts all devices, removes custom spacing and body contacts, and changes all alignments to top and left.

Abut all turns on auto abutment for all devices in the Modgen once the pattern editor has been applied.

Net in the **Dummy Creation Parameters** section lets you select the net to which all the dummies in the pattern for the Modgen need to be connected.

Specify Parameters in the **Dummy Creation Parameters** section lets you specify the number of *Fingers*, *Length*, and *Width* of the dummy devices.

Pattern Name lets you specify a unique pattern name.

Save lets you save the pattern.

Base Pattern lets you specify the base interdigitation pattern for dummy devices. Click **Generate** to generate the interdigitation pattern based on the specified base pattern and number of **Rows**. The resulting pattern is displayed in the *Base Pattern* box.

Base Orientation specifies an orientation for each instance in the order in which they are listed in the *Base Pattern* box. Click **Update** to update values in the *Base Orientation* box.

Change Name Mapping lets you change the mapping of pattern names to instance master names.

Related Topics

- Using the Modgen Pattern Editor

Set the Channel Width

Use the options in the **Set Channel Width** form to set the routing channel width. The channel width can be specified either from the device to device row, or from trunk to device row.

The **Channel** section includes the following options:

Direction lets you choose the direction along which channel width must be applied. Valid values are *Horizontal* and *Vertical*.

Number specifies the channel number to which the channel width must be applied.

Center Trunks centers trunks in the specified channel.

The **Channel Width** section includes the following options:

Layer and **Purpose** specifies the layer and purpose of a topological trunk that is anchored to an instance. The bounding box of the specified layer and purpose inside the instance is used to determine the device reference edge for setting the channel width. In other words, the channel width is measured from one side of the reference layer and purpose bounding box.

Device Row To Row Spacing (when *Direction* is set to *Horizontal*) and **Device Column To Column Spacing** (when *Direction* is set to *Vertical*) specify the channel width value.

Trunk to Device Spacing specifies the distance between a trunk and a device. This is an alternative way to specify the channel by specifying the distance between the trunks in the channel and the devices. *Top* is the spacing between the top edge of the top trunk to the bottom edge of the top device. *Bottom* is the spacing between the bottom edge of the bottom trunk to the top edge of the bottom device.

Remove Channel Width removes the channel width setting from the channel.

Related Topics

- Setting the Channel Width

Select Merge Layers

Use the **Select merge layers** form to specify the layers that need to be merged.

Select merge layers for 'mergeLayer' displays a list of all designs in the design.

default to select all default layers in the *Select merge layers for 'mergeLayer'* list.

well to select all well layers in the *Select merge layers for 'mergeLayer'* list.

none to deselect all layers in the *Select merge layers for 'mergeLayer'* list.

Related Topics

- Setting the Channel Width

Set Member Alignment and Spacing

Use the **Set Member Alignment and Spacing** form to specify alignment and spacing values for instances in Modgens.

Apply To lists the Modgen members to which the alignment and spacing settings need to be applied.

Alignment displays the default *Horizontal* and *Vertical* alignments of the selected instances.

Spacing to Left and **Spacing to Bottom** lets you specify the spacing values for the selected instances in the corresponding direction. For more information about the various alignment and spacing values, see Specifying Device Alignment and Spacing.

Layer and **Purpose** in the **Reference** section specify the reference layer and purpose for calculating custom spacing value and for specifying the alignment of instances.

Save Values overwrites environment variables with the values specified in the Set Member Alignment and Spacing form

Load Values resets all values in the form with values from their corresponding environment variables.

Apply to All Modgen Members applies the custom spacing to all the devices in the module.

Related Topics

- Specifying Device Alignment and Spacing

Surround Modgen

Use the **Surround Modgen** form to add surround dummies around Modgen instances.

(ICADVM20.1 Only) **Default values** provides a list of presets that have been registered using a SKILL call back function. Select the required preset and click **Load** to load values.

(ICADVM20.1 Only) **Top**, **Left**, **Right**, and **Bottom** specify the sides to which dummies must be added.

Master tab specifies the **Library**, **Cell**, and **View** to be used to create dummies.

Parameters tab specifies the **Number of Fingers**, **Length**, and **Width** for the dummies. Valid values are:

Custom value specified in the text box.

CDF Default: Values specified in the CDF is considered.

Same As Neighbor (default): Corresponding value of the neighboring device is considered.

Note: In ICADVM20.1, the *Width* cyclic field is replaced by the *Number of Fins* cyclic field. Valid values are the same as *Width*.

Connectivity tab includes the **Default** list of all available nets. Choose the net to which all dummy terminals must be connected.

(ICADVM20.1 Only) **Tap** tab provides the following options:

Between Rows inserts dummies between rows based on the following settings:

Starting Row Index specifies the first reference row for inserting dummies.

Rows to skip between insertions specifies the gap (number of rows) after which dummies must be inserted.

Between Columns inserts dummies between columns based on the following settings:

Starting Column Index specifies the first reference column for inserting dummies.

Columns to skip between insertions specifies the gap (number of columns) after which dummies must be inserted.

Single Dummy Row inserts a single dummy row, instead of individual dummies, in each direction.

Auto automatically calculates the number of fingers to be included in a dummy row.

Specify lets you specify the **Total Number of Fingers Per Row**.

Related Topics

- Adding Surround Dummies

7

Modgen Environment Variables

This appendix provides information on the names, descriptions, and graphical user interface equivalents for the Modgen environment variables.

Note: Only the environment variables documented in this section are supported for public use. All other Layout XL environment variables, regardless of their name or prefix, and undocumented aspects of the environment variables described below, are private and are subject to change at any time.

For information about the supported inherited environment variables, see Environment Variables in the Virtuoso Layout Suite documentation.

Related Topics

List of Modgen Environment Variables

List of Modgen Environment Variables

- modgenDummyWidthOptions
- modgenDummyWidthValue
- modgenGuardRingSep
- modgenInterdigitationFactor
- modgenMakeMinDummies
- modgenMergeLayers
- modgenMergeWells
- modgenMPPGuardRingToUseCB
- modgenPatternFormAbutAll
- modgenPhysConfigs
- modgenPlacementConstraintGroup
- modgenPreviewIgnoreXLConnVios
- modgenReferencePoint
- modgenRememberBodyContactVals
- modgenRememberDummyVals
- modgenPToTChannelWidth
- modgenPToTGenTrunkTwigs
- modgenPToTOnCreateFigMode
- modgenPToTPinCoverTapLowerViaPercent
- modgenPToTPinCoverTapLowerViaPercentEnable
- modgenPToTSpecifyChannelWidth
- modgenPToTPinCoverTapViaPercentEnable
- modgenPToTPinCoverViaModePercentTrunkOver
- modgenPToTPinCoverViaModePercentTrunkOverEnable
- modgenPToTPinCoverViaModeTrunkOver
- modgenPToTPinCoverViaModeTrunkOverEnable
- modgenPToTSpecifyChannelWidth

- modgenPToTSpecifyTrunk2DevSpacing
- modgenPToTTrunk2DevSpacing
- modgenPToTTrunkInsertMode
- modgenPToTTrunkLayer
- modgenPToTTrunkNets
- modgenPToTTrunkSpacing
- modgenPToTTrunkWidth
- modgenPToTTwigAbsoluteWidth
- modgenPToTTwigDirectionDown
- modgenPToTTwigDirectionLeft
- modgenPToTTwigDirectionOver
- modgenPToTTwigDirectionRight
- modgenPToTTwigDirectionUp
- modgenPToTTwigLayer
- modgenPToTTwigMinNumCuts
- modgenPToTTwigRelativeWidth
- modgenPToTTwigWidthType
- modgenPToTViaControlCutClass1
- modgenPToTViaControlCutClass2
- modgenPToTViaControlCutClassLayer
- modgenPToTViaControlCutClassName
- modgenPToTViaControlCutClassType
- modgenPToTViaControlCutClassEnable
- modgenPToTViaControlExtensionOrientEnable
- modgenPToTViaControlInline
- modgenPToTViaControlInlineEnable
- modgenPToTViaControlOffset

- modgenPToTViaControlOffsetEnable
- modgenPToTViaControlOrient
- modgenPToTViaControlOrientEnable
- modgenPToTViaWidthPercent
- modgenPToTViaWidthPercentEnable
- modgenSaveOnClosePreviewWindow
- modgenTransferDiffInstaces
- modgenTransferIgnoreParamsList
- modgenUseSnapSpacing
- modgenUseIteratedAsMfactor
- modgenWidthParamProportionalToFingers
- modgenWindowConfigFile
- chainPermutePins

Grid Pattern Editor Environment Variables

- moveAsSwap

modgenAllowPinPermutation

```
layoutXL modgenAllowPinPermutation boolean { t | nil }
```

Description

Enables automatic pin permutation within the Modgen figGroup. Pin permutation refers to the exchange of connectivity or net connections between the pins of a component. For more information about pin permutation, see Pin Permutation in *Virtuoso Layout Suite XL User Guide*.

The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenAllowPinPermutation")
envSetVal("layoutXL" "modgenAllowPinPermutation" 'boolean t)
envSetVal("layoutXL" "modgenAllowPinPermutation" 'boolean nil)
```

modgenBodyContactNetToUse

```
layoutXL modgenBodyContactNetToUse string "net_name"
```

Description

Specifies the default net to use when creating body contacts.

GUI Equivalent

Command	Modgen Editor – *Body Contacts icon on the Modgen Placement Toolbar*
Field	*Net* (Body Contact Options)

Examples

```
envGetVal("layoutXL" "modgenBodyContactNetToUse")
envSetVal("layoutXL" "modgenBodyContactNetToUse" 'string "myNet")
```

Related Topics

List of Modgen Environment Variables

modgenBodyContactReferenceLayer

```
layoutXL modgenBodyContactReferenceLayer string "layer_name"
```

Description

Specifies the default layer for body contact creation.

GUI Equivalent

Command	Modgen Editor – *Body Contacts icon on the Modgen Placement Toolbar*
Field	Layer (Body Contact Options)

Examples

```
envGetVal("layoutXL" "modgenBodyContactReferenceLayer")
envSetVal("layoutXL" "modgenBodyContactReferenceLayer" 'string "myLayer")
```

Related Topics

List of Modgen Environment Variables

modgenBodyContactSep

`layoutXL modgenBodyContactSep float` *`float_number`*

Description

Specifies the default separation distance for body contact creation.

The default is `0`.

GUI Equivalent

Command	Modgen Editor – *Body Contacts icon on the Modgen Placement Toolbar*
Field	*Separation* (Body Contact Options)

Examples

```
envGetVal("layoutXL" "modgenBodyContactSep")
envSetVal("layoutXL" "modgenBodyContactSep" 'float 1.0)
```

Related Topics

List of Modgen Environment Variables

modgenBodyContactType

```
layoutXL modgenBodyContactType string "type"
```

Description

Specifies the default type for body contact creation.

GUI Equivalent

Command	Modgen Editor – *Body Contacts icon on the Modgen Placement Toolbar*
Field	Type (Body Contact Options)

Examples

```
envGetVal("layoutXL" "modgenBodyContactType")
envSetVal("layoutXL" "modgenBodyContactType" 'string "myType")
```

Related Topics

List of Modgen Environment Variables

modgenCreatePreserveSpacing

```
layoutXL modgenCreatePreserveSpacing boolean { t | nil }
```

Description

Specifies whether the relative spacing of the Modgen members must be respected when creating a Modgen from a layout selection set. The Modgen respects the existing spacing value by setting the custom spacing values between members. These spacing values may vary slightly from the spacings in the selection set because the Modgen aligns parallel member edges that are within a certain proximity and border member edges.

When set to `nil`, a Modgen with minDRC spacing values is created.

The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenCreatePreserveSpacing")
envSetVal("layoutXL" "modgenCreatePreserveSpacing" 'boolean t)
envSetVal("layoutXL" "modgenCreatePreserveSpacing" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenCreateReferenceLPP

```
layoutXL modgenCreateReferenceLPP string { "layer" | "layer_purpose" | "" }
```

Description

This setting impacts how Modgens are created when modgenCreatePreserveSpacing is set to `t`. When a layer (or layer purpose pair) is specified, the layer's geometry is used to determine the Modgen's pattern. The default value is `""`. In this state, the bounding box of the selected instances is used as reference to determine the Modgen's pattern.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenCreateReferenceLPP")
envSetVal("layoutXL" "modgenCreateReferenceLPP" 'string "Metal2")
envSetVal("layoutXL" "modgenCreateReferenceLPP" 'string "Poly drawing")
```

Related Topics

List of Modgen Environment Variables

modgenDefHCSRefLayer

```
layoutXL modgenDefHCSRefLayer string { "layer_name" | "" }
```

Description

Specifies the reference layer for horizontal spacing of devices. The spacing value is the distance between the bounding boxes of all shapes on this layer in the devices.

GUI Equivalent

Command	Modgen Editor – *Member Alignment/Spacing* icon on the Modgen Placement Toolbar
Field	*Horizontal:Reference* (Set Member Alignment and Spacing)

Examples

```
envGetVal("layoutXL" "modgenDefHCSRefLayer")
envSetVal("layoutXL" "modgenDefHCSRefLayer" 'string "Metal2")
envSetVal("layoutXL" "modgenDefHCSRefLayer" 'string "Poly")
```

Related Topics

List of Modgen Environment Variables

modgenDefHCSRefPurpose

```
layoutXL modgenDefHCSRefPurpose string "purpose_name"
```

Description

Specifies the purpose to be set for horizontal spacing of devices. The spacing value is the distance between the bounding boxes of all shapes on this layer in the devices.

GUI Equivalent

Command	Modgen Editor – *Member Alignment/Spacing* icon on the Modgen Placement Toolbar
Field	*Horizontal: Reference: Purpose* (Set Member Alignment and Spacing)

Examples

```
envGetVal("layoutXL" "modgenDefHCSRefPurpose")
envSetVal("layoutXL" "modgenDefHCSRefPurpose" 'string "drawing")
envSetVal("layoutXL" "modgenDefHCSRefPurpose" 'string "label")
envSetVal("layoutXL" "modgenDefHCSRefPurpose" 'string "pin")
```

Related Topics

List of Modgen Environment Variables

modgenDefHoriAlignment

```
layoutXL modgenDefHoriAlignment string { "left" | "right" | "center" | "custom" |
     "customRight" }
```

Description

Specifies the default value for the Horizontal Alignment cyclic field in the Set Member Alignment and Spacing form. If “custom” or “customRight” is specified, the value defined in the modgenDefHoriSpacing environment variable is used.

The default is `left`.

GUI Equivalent

Command	Modgen Editor – *Member Alignment/Spacing* icon on the Modgen Placement Toolbar
Field	*Horizontal: Alignment* (Set Member Alignment and Spacing)

Examples

```
envGetVal("layoutXL" "modgenDefHoriAlignment")
envSetVal("layoutXL" "modgenDefHoriAlignment" 'string "left")
envSetVal("layoutXL" "modgenDefHoriAlignment" 'string "right")
envSetVal("layoutXL" "modgenDefHoriAlignment" 'string "center")
envSetVal("layoutXL" "modgenDefHoriAlignment" 'string "custom")
envSetVal("layoutXL" "modgenDefHoriAlignment" 'string "customRight")
```

Related Topics

List of Modgen Environment Variables

modgenDefHoriSpacing

```
layoutXL modgenDefHoriSpacing float float_number
```

Description

Specifies the default value for the Horizontal Spacing field in the Set Member Alignment and Spacing form. This value is used only if the modgenDefHoriAlignment variable is set to `custom` or `customRight`.

The default is `0.0`.

GUI Equivalent

Command	Modgen Editor – *Member Alignment/Spacing* icon on the Modgen Placement Toolbar
Field	*Horizontal: Spacing to Left* (Set Member Alignment and Spacing)

Examples

```
envGetVal("layoutXL" "modgenDefHoriSpacing")
envSetVal("layoutXL" "modgenDefHoriSpacing" 'float 1.0)
```

Related Topics

List of Modgen Environment Variables

modgenDefVCSRefLayer

```
layoutXL modgenDefVCSRefLayer string { "layer_name" | "" }
```

Description

Specifies the reference layer for vertical spacing of devices. The spacing value is the distance between the bounding boxes of all shapes on this layer in the devices.

GUI Equivalent

Command	Modgen Editor – *Member Alignment/Spacing* icon on the Modgen Placement Toolbar
Field	*Vertical: Reference* (Set Member Alignment and Spacing)

Examples

```
envGetVal("layoutXL" "modgenDefVCSRefLayer")
envSetVal("layoutXL" "modgenDefVCSRefLayer" 'string "Metal2")
envSetVal("layoutXL" "modgenDefVCSRefLayer" 'string "Poly")
```

Related Topics

List of Modgen Environment Variables

modgenDefVCSRefPurpose

```
layoutXL modgenDefVCSRefPurpose string "purpose_name"
```

Description

Specifies the purpose to be set for vertical spacing of devices. The spacing value is the distance between the bounding boxes of all shapes on this layer in the devices.

GUI Equivalent

Command	Modgen Editor – *Member Alignment/Spacing* icon on the Modgen Placement Toolbar
Field	*Vertical: Reference: Purpose* (Set Member Alignment and Spacing)

Examples

```
envGetVal("layoutXL" "modgenDefVCSRefPurpose")
envSetVal("layoutXL" "modgenDefVCSRefPurpose" 'string "drawing")
envSetVal("layoutXL" "modgenDefVCSRefPurpose" 'string "label")
envSetVal("layoutXL" "modgenDefVCSRefPurpose" 'string "pin")
```

Related Topics

List of Modgen Environment Variables

modgenDefVertAlignment

```
layoutXL modgenDefVertAlignment string { "top" | "bottom" | "center" | "custom" |
     "customTop" }
```

Description

Specifies the default value for the Vertical Alignment cyclic field in the Set Member Alignment and Spacing form. If "`custom`" or "`customTop`" is specified, the value defined in the modgenDefVertSpacing environment variable is used.

The default is `top`.

GUI Equivalent

Command	Modgen Editor – *Member Alignment/Spacing* icon on the Modgen Placement Toolbar
Field	*Vertical: Alignment* (Set Member Alignment and Spacing)

Examples

```
envGetVal("layoutXL" "modgenDefVertAlignment")
envSetVal("layoutXL" "modgenDefVertAlignment" 'string "top")
envSetVal("layoutXL" "modgenDefVertAlignment" 'string "bottom")
envSetVal("layoutXL" "modgenDefVertAlignment" 'string "center")
envSetVal("layoutXL" "modgenDefVertAlignment" 'string "custom")
envSetVal("layoutXL" "modgenDefVertAlignment" 'string "customTop")
```

Related Topics

List of Modgen Environment Variables

modgenDefVertSpacing

`layoutXL modgenDefVertSpacing float` *`float_number`*

Description

Specifies the default value for the Vertical Spacing field in the Set Member Alignment and Spacing form. This value is used only if the modgenDefVertAlignment variable is set to `custom` or `customTop`.

The default is `0.0`.

GUI Equivalent

Command	Modgen Editor – *Member Alignment/Spacing* icon on the Modgen Placement Toolbar
Field	*Vertical: Spacing to Bottom* (Set Member Alignment and Spacing)

Examples

```
envGetVal("layoutXL" "modgenDefVertSpacing")
envSetVal("layoutXL" "modgenDefVertSpacing" 'float 1.0)
```

Related Topics

List of Modgen Environment Variables

modgenDummyCell

```
layoutXL modgenDummyCell string "cellName"
```

Description

Specifies the default cell to use when creating custom dummy devices.

GUI Equivalent

Command	Modgen Editor – *Dummies icon on the Modgen Placement Toolbar*
Field	*Default Dummy: DummyCell* (Dummy Options)

Examples

```
envGetVal("layoutXL" "modgenDummyCell")
envSetVal("layoutXL" "modgenDummyCell" 'string "myCell")
```

Related Topics

List of Modgen Environment Variables

modgenDummyLengthOptions

```
layoutXL modgenDummyLengthOptions cyclic { "CDF Default" | "Same As Neighbor" |
     "Specify" }
```

Description

Specifies valid values for the dummy *Length* field.

- `CDF Default` – The default finger length, as specified in the CDF, is considered.
- `Same As Neighbor` – The length of fingers of the neighboring device is considered.
- `Specify` – You can specify the length of fingers.

GUI Equivalent

Command	Modgen Editor – *Dummies icon on the Modgen Placement Toolbar*
Field	*Dummy Type: Dummy Parameters: Length* (Dummy Options)

Examples

```
envGetVal("layoutXL" "modgenDummyLengthOptions")
envSetVal("layoutXL" "modgenDummyLengthOptions" 'cyclic "CDF Default")
envSetVal("layoutXL" "modgenDummyLengthOptions" 'cyclic "Same as Neighbor")
envSetVal("layoutXL" "modgenDummyLengthOptions" 'cyclic "Specify")
```

Related Topics

List of Modgen Environment Variables

modgenDummyLengthValue

```
layoutXL modgenDummyLengthValue string "Default_Length"
```

Description

Specifies default value for the dummy *Length* field. This variable can be used only if the modgenDummyWidthOptions environment variable is set to `Specify`.

GUI Equivalent

Command	Modgen Editor – *Dummies icon on the Modgen Placement Toolbar*
Field	*Length* (Dummy Options)

Examples

```
envGetVal("layoutXL" "modgenDummyLengthValue")
envSetVal("layoutXL" "modgenDummyLengthValue" 'string "2")
```

Related Topics

List of Modgen Environment Variables

modgenDummyLib

```
layoutXL modgenDummyLib string "library_name"
```

Description

Specifies the default library to use when creating custom dummy devices.

GUI Equivalent

Command	Modgen Editor – *Dummies icon on the Modgen Placement Toolbar*
Field	*Default Dummy: DummyLibrary* (Dummy Options)

Examples

```
envGetVal("layoutXL" "modgenDummyLib")
envSetVal("layoutXL" "modgenDummyLib" 'string "myLib")
```

Related Topics

List of Modgen Environment Variables

modgenDummyNet

```
layoutXL modgenDummyNet string "net_name"
```

Description

Specifies the default net to use when creating dummy devices.

GUI Equivalent

Command	Modgen Editor – *Dummies icon on the Modgen Placement Toolbar*
Field	*Dummy Net: Default* (Dummy Options)

Examples

```
envGetVal("layoutXL" "modgenDummyNet")
envSetVal("layoutXL" "modgenDummyNet" 'string "myNet")
```

Related Topics

List of Modgen Environment Variables

modgenDummyNumFingersOptions

```
layoutXL modgenDummyNumFingersOptions cyclic { "CDF Default" | "Same As Neighbor"
     | "Specify" }
```

Description

Specifies valid values for the dummy *Number of Fingers* field.

- `CDF Default` – The default number of fingers, as specified in the CDF, is created
- `Same As Neighbor` – The same number of fingers as the neighboring device is created
- `Specify` – You can specify the number of fingers to be created in the box beside the *Number of Fingers* cyclic field

GUI Equivalent

Command	Modgen Editor – *Dummies icon on the Modgen Placement Toolbar*
Field	*Dummy Type: Dummy Parameters: Number of Fingers* (Dummy Options)

Examples

```
envGetVal("layoutXL" "modgenDummyNumFingersOptions")
envSetVal("layoutXL" "modgenDummyNumFingersOptions" 'cyclic "CDF Default")
envSetVal("layoutXL" "modgenDummyNumFingersOptions" 'cyclic "Same as Neighbor")
envSetVal("layoutXL" "modgenDummyNumFingersOptions" 'cyclic "Specify")
```

Related Topics

List of Modgen Environment Variables

modgenDummyNumFingersValue

```
layoutXL modgenDummyNumFingersValue string "Number_of_Fingers"
```

Description

Specifies valid values for the dummy *Number of Fingers* field. This variable can be used only if the modgenDummyNumFingersOptions environment variable is set to `Specify`.

GUI Equivalent

Command	Modgen Editor – *Dummies icon on the Modgen Placement Toolbar*
Field	*Number of Fingers* (Dummy Options)

Examples

```
envGetVal("layoutXL" "modgenDummyNumFingersValue")
envSetVal("layoutXL" "modgenDummyNumFingersValue" 'string "2")
```

Related Topics

List of Modgen Environment Variables

modgenDummySpecifyParams

```
layoutXL modgenDummySpecifyParams boolean { t | nil }
```

Description

Specifies whether the number of fingers, length, and width values for dummies need to be specified. When set to `nil`, the default values are as specified in the modgenMakeMinDummies environment variable.

The default is `nil`.

GUI Equivalent

Command	Modgen Editor – *Pattern icon on the Modgen Placement Toolbar*
Field	*Specify Parameters* (Modgen Pattern Editor)

Examples

```
envGetVal("layoutXL" "modgenDummySpecifyParams")
envSetVal("layoutXL" "modgenDummySpecifyParams" 'boolean t)
envSetVal("layoutXL" "modgenDummySpecifyParams" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenDummyType

```
layoutXL modgenDummyType cyclic { "neighbor" | "default" | "copy" }
```

Description

Specifies the default type for dummy devices.

If the default type is `neighbor`, the type of device created is dependent on the location of the dummy and the setting of the modgenMakeMinDummies environment variable.

If the default type is `default`, then the type of device can be specified using modgenDummyLib, modgenDummyCell, and modgenDummyView variables.

If the default type is `copy`, then identical dummies of the selected instances are created. In this mode, the dummy parameters and default values of the source instances are used. Different values cannot be specified.

GUI Equivalent

Command	Modgen Editor – *Dummies icon on the Modgen Placement Toolbar*
Field	*Dummy Type: Type* (Dummy Options)

Examples

```
envGetVal("layoutXL" "modgenDummyType")
envSetVal("layoutXL" "modgenDummyType" 'cyclic "neighbor")
envSetVal("layoutXL" "modgenDummyType" 'cyclic "default")
envSetVal("layoutXL" "modgenDummyType" 'cyclic "copy")
```

Related Topics

List of Modgen Environment Variables

modgenDummyView

```
layoutXL modgenDummyView string "viewName"
```

Description

Specifies the default cell view to use when creating custom dummy devices.

GUI Equivalent

Command	Modgen Editor – *Dummies icon on the Modgen Placement Toolbar*
Field	*Default Dummy: DummyView* (Dummy Options)

Examples

```
envGetVal("layoutXL" "modgenDummyView")
envSetVal("layoutXL" "modgenDummyView" 'string "myView")
```

Related Topics

List of Modgen Environment Variables

modgenDummyWidthOptions

```
layoutXL modgenDummyWidthOptions cyclic { "CDF Default" | "Same As Neighbor" |
     "Specify" }
```

Description

Specifies the default value for the dummy *Number of Fins* field. Valid values are:

- `CDF Default` – The default finger width, as specified in the CDF, is considered
- `Same As Neighbor` – The width of fingers of the neighboring device is considered
- `Specify` – You can specify the width of fingers in the box beside the Width cyclic field

GUI Equivalent

Command	Modgen Editor – *Dummies icon on the Modgen Placement Toolbar*
Field	*Number of Fins* (Dummy Options)

Examples

```
envGetVal("layoutXL" "modgenDummyWidthOptions")
envSetVal("layoutXL" "modgenDummyWidthOptions" 'cyclic "CDF Default")
envSetVal("layoutXL" "modgenDummyWidthOptions" 'cyclic "Same as Neighbor")
envSetVal("layoutXL" "modgenDummyWidthOptions" 'cyclic "Specify")
```

Related Topics

List of Modgen Environment Variables

modgenDummyWidthValue

```
layoutXL modgenDummyWidthValue string "Default_Width"
```

Description

Specifies the default values for the dummy *Number of Fins* field. This variable can be used only if the modgenDummyWidthOptions environment variable is set to `Specify`.

GUI Equivalent

Command	Modgen Editor – *Dummies icon on the Modgen Placement Toolbar*
Field	*Number of Fins* (Dummy Options)

Examples

```
envGetVal("layoutXL" "modgenDummyWidthValue")
envSetVal("layoutXL" "modgenDummyWidthValue" 'string "2")
```

Related Topics

List of Modgen Environment Variables

modgenGuardRingSep

```
layoutXL modgenGuardRingSep float float_number
```

Description

Specifies the default separation distance for all sides of a guard ring.

The default value is `0.0`.

GUI Equivalent

Command	Modgen Editor – *Body Contacts* icon on the Modgen Placement Toolbar
Field	Separation (Body Contact Options)

Examples

```
envGetVal("layoutXL" "modgenGuardRingSep")
envSetVal("layoutXL" "modgenGuardRingSep" 'float 1.0)
```

Related Topics

List of Modgen Environment Variables

modgenInterdigitationFactor

```
layoutXL modgenInterdigitationFactor int integer_number
```

Description

Specifies the interdigitation pattern for the device, which is specific to the current module. In case no interdigitation in the Modgen is required, specify `0` as the interdigitation value.

The default value is `1`.

GUI Equivalent

Command	Modgen Editor – *Pattern* icon on the toolbar
Field	*Interdigitate By* (Modgen Pattern Editor)

Examples

```
envGetVal("layoutXL" "modgenInterdigitationFactor")
envSetVal("layoutXL" "modgenInterdigitationFactor" 'int 0)
envSetVal("layoutXL" "modgenInterdigitationFactor" 'int 1)
```

Related Topics

List of Modgen Environment Variables

modgenMakeMinDummies

```
layoutXL modgenMakeMinDummies boolean { t | nil }
```

Description

Determines the behavior of the Modgen when dummies are added to MOSFETs or resistors.

If true, the following happens when a neighbor dummy is added to a MOSFET:

- If the dummy's location is on the right or left, a single finger device is created to serve as the dummy.
- If the dummy's location is on the top or bottom, a minimum width device with the same number of fingers as the neighbor is created to serve as the dummy.

If true, the following happens when a neighbor dummy is added to a resistor:

- If the dummy's location is on the right or left, a single segment device is created to serve as the dummy.
- If the dummy's location is on the top or bottom, a minimum width device with the same number of segments as the neighbor is created to serve as the dummy.

In this case, the parameters for fingers (`lxFingeringNames`), width (`transistorWidthParamNames`), and s-factor (`sfactorNames`) are used to create the device.

If it is set to false, then the dummy is created as identical to the neighboring device (same number of fingers or segments).

The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenMakeMinDummies")
envSetVal("layoutXL" "modgenMakeMinDummies" 'boolean t)
envSetVal("layoutXL" "modgenMakeMinDummies" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenMergeLayers

```
layoutXL modgenMergeLayers string "layer_names"
```

Description

Specifies the layers that need to be merged in the Modgens that have been created interactively. The string value is delimited by a colon (:), where the characters on both sides of the delimiter are layer names. Other valid values are `well` and `default`.

GUI EquivalentExamples

Command	Modgen Editor – *Merge Layers* icon on the Modgen Placement Toolbar
Field	*Select merge layers for 'mergeLayer'* (Select Merge Layers)

```
envGetVal("layoutXL" "modgenMergeLayers")
envSetVal("layoutXL" "modgenMergeLayers" 'string "Nimp:Pimp")
```

Related Topics

List of Modgen Environment Variables

modgenMergeWells

```
layoutXL modgenMergeWells boolean { t | nil }
```

Description

Specifies whether shared well layer shapes can be created for member instances that have the same well layer.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenMergeWells")
envSetVal("layoutXL" "modgenMergeWells" 'boolean t)
envSetVal("layoutXL" "modgenMergeWells" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenMPPGuardRingToUseCB

```
layoutXL modgenMPPGuardRingToUseCB string { "instance_names" }
```

Description

Sets the default MPP guard ring when the MPP Guard Ring Options form is launched.

The environment variable accepts a user-defined callback function with a list of instances in the Modgen, and returns the name of the valid MPP guard ring that surrounds the Modgen.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenMPPGuardRingToUseCB")
envSetVal("layoutXL" "modgenMPPGuardRingToUseCB" 'string "mgGRProc")
```

Related Topics

List of Modgen Environment Variables

modgenPatternFormAbutAll

```
layoutXL modgenPatternFormAbutAll boolean { t | nil }
```

Description

Allows you to set the default state of the *Abut All* parameter in the *Modgen Pattern Editor* form. The last used state of this option is saved as the default for subsequent uses.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Pattern* icon on the Modgen Placement Toolbar
Field	*Abut All* (Modgen Pattern Editor)

Examples

```
envGetVal("layoutXL" "modgenPatternFormAbutAll")
envSetVal("layoutXL" "modgenPatternFormAbutAll" 'boolean t)
envSetVal("layoutXL" "modgenPatternFormAbutAll" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPhysConfigs

```
layoutXL modgenPhysConfigs string { "physConfig" }
```

Description

Specifies the physConfig view name to be used by the Modgen when run from the schematic, if the physConfig view exists with this name. The default value is "".

When none of the specified physConfigs exists, the Modgen creates a temporary default physConfig and uses it. You can specify multiple physConfig view names by separating them by a space.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenPhysConfigs")
envSetVal("layoutXL" "modgenPhysConfigs" 'string "physConfig")
envSetVal("layoutXL" "modgenPhysConfigs" 'string "viewName1 viewName2")
envSetVal("layoutXL" "modgenPhysConfigs" 'string "")
```

Related Topics

List of Modgen Environment Variables

modgenPlacementConstraintGroup

```
layoutXL modgenPlacementConstraintGroup string { "Modgen_ConstraintGroup_Name"
     | t }
```

Description

Specifies the name of the tool Constraint Group to be used when creating, generating, or updating Modgens. The tool Constraint Group comprises a set of process rules that control creation of Modgens. If you change the `modgenPlacementConstraintGroup` setting before regenerating a Modgen, then the new Constraint Group is automatically used, potentially resulting in a different placement of the Modgen.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenPlacementConstraintGroup")
envSetVal("layoutXL" "modgenPlacementConstraintGroup" 'string "My_ConstraintGp")
```

Related Topics

List of Modgen Environment Variables

modgenPreviewIgnoreXLConnVios

```
layoutXL modgenPreviewIgnoreXLConnVios boolean { t | nil }
```

Description

Selectively reports or hides the number of open violations in the design in the Modgen previewer window.

The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenPreviewIgnoreXLConnVios")
envSetVal("layoutXL" "modgenPreviewIgnoreXLConnVios" 'boolean t)
envSetVal("layoutXL" "modgenPreviewIgnoreXLConnVios" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenReferencePoint

```
layoutXL modgenReferencePoint cyclic { "center" | "interactive" | "lowerLeft" |
     "upperRight" }
```

Description

Specifies the reference point for aligning Modgens when they are either created or repositioned, for example when a Modgen is moved or when the number of rows in a Modgen is increased. Valid values are:

- `center` – Aligns the center of the new or edited Modgen with the center of the existing Modgen.
- `interactive` – Aligns the center of the new or edited Modgen with the center of the existing Modgen. However, the center excludes any user-moved instance.

  Example:

  Instances in an existing Modgen are positioned as following:

  ```
  M1

  M2   M3  M4
  ```

  Instance `M1` is moved to the right of the three instances. The revised alignment of the Modgen is follows:

  ```
  M2   M3   M4   M1
  ```

  Note:

  When set to `center`, all instances, `M1 M2 M3 M4`, are considered for determining the center of the Modgen.

  When set to `interactive`, only instances, `M2 M3 M4`, are considered.
- `lowerLeft` – Aligns the lower left corner of the new or edited Modgen with that of the existing Modgen.
- `upperRight` – Aligns the upper right corner of the new or edited Modgen with that of the existing Modgen.

The default is `interactive`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenReferencePoint")
envSetVal("layoutXL" "modgenReferencePoint" 'cyclic "center")
envSetVal("layoutXL" "modgenReferencePoint" 'cyclic "interactive")
envSetVal("layoutXL" "modgenReferencePoint" 'cyclic "lowerLeft")
envSetVal("layoutXL" "modgenReferencePoint" 'cyclic "upperRight")
```

Related Topics

List of Modgen Environment Variables

modgenRememberBodyContactVals

```
layoutXL modgenRememberBodyContactVals boolean { t | nil }
```

Description

Specifies whether the values from the body contacts form should be saved.

The default is `nil`.

GUI Equivalent

Command	Modgen Editor – *Body Contacts* icon on the Modgen Placement Toolbar
Field	*Remember Values* (Body Contact Options)

Examples

```
envGetVal("layoutXL" "modgenRememberBodyContactVals")
envSetVal("layoutXL" "modgenRememberBodyContactVals" 'boolean t)
envSetVal("layoutXL" "modgenRememberBodyContactVals" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenRememberDummyVals

```
layoutXL modgenRememberDummyVals boolean { t | nil }
```

Description

Specifies whether the values from the Dummy Options form should be saved.

The default is `nil`.

GUI Equivalent

Command	Modgen Editor – *Dummies icon on the Modgen Placement Toolbar*
Field	*Remember Values*

Examples

```
envGetVal("layoutXL" "modgenRememberDummyVals")
envSetVal("layoutXL" "modgenRememberDummyVals" 'boolean t)
envSetVal("layoutXL" "modgenRememberDummyVals" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTChannelWidth

```
layoutXL modgenPToTChannelWidth float float_number
```

Description

Specifies the width of channel nets in the current Modgen. This option can be set only if the modgenPToTSpecifyChannelWidth environment variable is set to `t`. The default value is `0.0`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies icon on the Modgen Routing Toolbar*
Field	*Trunk Options: Specify Channel Width* (Create Trunks and Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTChannelWidth")
envSetVal("layoutXL" "modgenPToTChannelWidth" 'float 1.2)
```

Related Topics

List of Modgen Environment Variables

modgenPToTGenTrunkTwigs

```
layoutXL modgenPToTGenTrunkTwigs boolean { t | nil }
```

Description

Controls whether topological twigs, which connect trunks to channel objects (instances and body contacts), need to be generated when creating trunks. The default is `t`, and so twigs will be generated. The options in the *Twig Options* section are enabled.

When set to `nil`, twigs will not be generated. The options in the *Twig Options* section are disabled.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies* and *Create Single Trunk and Topologies icons on the Modgen Routing Toolbar*
Field	*Generate twigs connecting trunks to channel objects* (Create Trunks and Topologies, Create Single Trunk and Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTGenTrunkTwigs")
envSetVal("layoutXL" "modgenPToTGenTrunkTwigs" 'boolean t)
envSetVal("layoutXL" "modgenPToTGenTrunkTwigs" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTOnCreateFigMode

```
layoutXL modgenPToTOnCreateFigMode cyclic { "ManualRoute" | "Ignore" }
```

Description

Specifies whether the manual editing mode should be turned on. When set to `ManualRoute` (default), the manual editing mode is turned on. Shapes and vias added in this mode are honored.

When set to `Ignore`, any new shape added is immediately deleted, and an appropriate message is displayed.

- `ManualRoute`: Manual editing mode is turned on.
- `Ignore`: Manual editing mode is turned off.

GUI Equivalent

Command	Modgen Editor – *Modgen Routing* Toolbar
Field	*Enable Hand Routing*

Examples

```
envGetVal("layoutXL" "modgenPToTOnCreateFigMode")
envSetVal("layoutXL" "modgenPToTOnCreateFigMode" 'cyclic "ManualRoute")
envSetVal("layoutXL" "modgenPToTOnCreateFigMode" 'cyclic "Ignore")
```

Related Topics

List of Modgen Environment Variables

modgenPToTPinCoverTapLowerViaPercent

`layoutXL modgenPToTPinCoverTapLowerViaPercent int` *`integer_number`*

Description

Specifies the percentage of a pin shape to be covered by the lower layers of the tapping vias. This value applies to all the layers expect the top-most via. The percentage is measured in the direction perpendicular to the trunk.

The default value is `100`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Pin Cover – Tap Lower Via Percent* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTPinCoverTapLowerViaPercent")
envSetVal("layoutXL" "modgenPToTPinCoverTapLowerViaPercent" 'int 150)
envSetVal("layoutXL" "modgenPToTPinCoverTapLowerViaPercent" 'int 133)
```

Related Topics

List of Modgen Environment Variables

modgenPToTPinCoverTapLowerViaPercentEnable

```
layoutXL modgenPToTPinCoverTapLowerViaPercentEnable boolean { t | nil }
```

Description

Specifies whether the percentage of a pin shape, which is to be covered by the lower layers of the tapping vias, can be specified.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Pin Cover – Tap Lower Via Percent* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTPinCoverTapLowerViaPercentEnable")
envSetVal("layoutXL" "modgenPToTPinCoverTapLowerViaPercentEnable" 'boolean t)
envSetVal("layoutXL" "modgenPToTPinCoverTapLowerViaPercentEnable" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTPinCoverTapViaPercent

```
layoutXL modgenPToTPinCoverTapViaPercent int integer_number
```

Description

Specifies the percentage of a pin shape to be covered by the tapping via on all layers. The percentage is measured in the direction perpendicular to the trunk.

The default value is `100`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Pin Cover – Tap Via Percent* (Create Trunks and Topologies*,* Create Single Trunk and Topologies*,* Add/ Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTPinCoverTapViaPercent")
envSetVal("layoutXL" "modgenPToTPinCoverTapViaPercent" 'int 150)
envSetVal("layoutXL" "modgenPToTPinCoverTapViaPercent" 'int 133)
```

Related Topics

List of Modgen Environment Variables

modgenPToTPinCoverTapViaPercentEnable

```
layoutXL modgenPToTPinCoverTapViaPercentEnable boolean { t | nil }
```

Description

Specifies whether the percentage of the pin shape to be covered by the tapping via on all layers can be specified.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Pin Cover – Tap Via Percent* (Create Trunks and Topologies*,* Create Single Trunk and Topologies*,* Add/ Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTPinCoverTapViaPercentEnable")
envSetVal("layoutXL" "modgenPToTPinCoverTapViaPercentEnable" 'boolean t)
envSetVal("layoutXL" "modgenPToTPinCoverTapViaPercentEnable" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTPinCoverViaModePercentTrunkOver

```
layoutXL modgenPToTPinCoverViaModePercentTrunkOver int integer_number
```

Description

Controls the percentage of a trunk or a pin to be covered by the via stack.

The default value is `100`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Pin Cover – Via Mode Percent Trunk Over* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTPinCoverViaModePercentTrunkOver")
envSetVal("layoutXL" "modgenPToTPinCoverViaModePercentTrunkOver" 'int 150)
envSetVal("layoutXL" "modgenPToTPinCoverViaModePercentTrunkOver" 'int 133)
```

Related Topics

List of Modgen Environment Variables

modgenPToTPinCoverViaModePercentTrunkOverEnable

```
layoutXL modgenPToTPinCoverViaModePercentTrunkOverEnable boolean { t | nil }
```

Description

Specifies whether the percentage of a trunk or a pin to be covered by the via stack can be specified.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Pin Cover – Via Mode Percent Trunk Over* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTPinCoverViaModePercentTrunkOverEnable")
envSetVal("layoutXL" "modgenPToTPinCoverViaModePercentTrunkOverEnable" 'boolean
t)
envSetVal("layoutXL" "modgenPToTPinCoverViaModePercentTrunkOverEnable" 'boolean
nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTPinCoverViaModeTrunkOver

```
layoutXL modgenPToTPinCoverViaModeTrunkOver cyclic { "matchTrunk" | "full" |
     "fullExceptTop" }
```

Description

Specifies the via mode to be applied when a trunk or a pin is covered by a via stack. The following options are available:

- `matchTrunk` (default): Covers the width of the trunk.
- `full`: Covers the entire pin shape.
- `fullExceptTop`: Covers the entire trunk on the top via layer and the entire pin on the lower via layers.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Pin Cover – Via Mode Trunk Over* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTPinCoverViaModeTrunkOver")
envSetVal("layoutXL" "modgenPToTPinCoverViaModeTrunkOver" 'cyclic "full")
envSetVal("layoutXL" "modgenPToTPinCoverViaModeTrunkOver" 'cyclic
"fullExceptTop")
```

Related Topics

List of Modgen Environment Variables

modgenPToTPinCoverViaModeTrunkOverEnable

```
layoutXL modgenPToTPinCoverViaModeTrunkOverEnable boolean { t | nil }
```

Description

Specifies whether the via mode, which is to be applied when the trunk or a pin is covered by the via stack, can be specified.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Pin Cover – Via Mode Trunk Over* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTPinCoverViaModeTrunkOverEnable")
envSetVal("layoutXL" "modgenPToTPinCoverViaModeTrunkOverEnable" 'boolean t)
envSetVal("layoutXL" "modgenPToTPinCoverViaModeTrunkOverEnable" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTSpecifyChannelWidth

```
layoutXL modgenPToTSpecifyChannelWidth boolean { t | nil }
```

Description

Specifies whether the width of channel nets can be specified for the current Modgen. The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies icon on the Modgen Routing Toolbar*
Field	*Trunk Options: Specify Channel Width* (Create Trunks and Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTSpecifyChannelWidth")
envSetVal("layoutXL" "modgenPToTSpecifyChannelWidth" 'boolean t)
envSetVal("layoutXL" "modgenPToTSpecifyChannelWidth" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTSpecifyTrunk2DevSpacing

```
layoutXL modgenPToTSpecifyTrunk2DevSpacing string { "Center Trunks in Channel" |
     nil }
```

Description

Specifies whether trunks must be centered in the channel such that their distances from devices is the same from both, the top and bottom edges.

The default value is `Center Trunks in Channel`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies icon on the Modgen Routing Toolbar*
Field	*Trunk Options: Center Trunks in Channel* (Create Trunks and Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTSpecifyTrunk2DevSpacing")
envSetVal("layoutXL" "modgenPToTSpecifyTrunk2DevSpacing" 'boolean t)
envSetVal("layoutXL" "modgenPToTSpecifyTrunk2DevSpacing" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTTrimTrunks

```
layoutXL modgenPToTTrimTrunks cyclic { "None" | "Both" | "Left/Bottom" | "Right/
     Top" }
```

Description

Specifies the side from which trunks need to be trimmed while routing. The following options are available:

- `None` (default): Trunks are not trimmed.
- `Both`: Trunks are trimmed from both the ends.
- `Left/Bottom`: Trunks are trimmed along the left and bottom edges. Trunks are trimmed to the last twig connection.
- `Right/Top`: Trunks are trimmed along the right and top edges. Trunks are trimmed to the last twig connection.

GUI Equivalent

Command	Modgen Editor – *Modgen Routing* Toolbar – *Create Trunk and Topology*
Field	*Trunk Options – Trim Trunk* (Create Trunks and Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTrimTrunks")
envSetVal("layoutXL" "modgenPToTTrimTrunks" 'cyclic "Both")
envSetVal("layoutXL" "modgenPToTTrimTrunks" 'cyclic "Left/Bottom")
```

Related Topics

List of Modgen Environment Variables

modgenPToTTrunk2DevSpacing

```
layoutXL modgenPToTTrunk2DevSpacing float float_number
```

Description

Specifies the trunk-to-device spacing value for the current Modgen.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies icon on the Modgen Routing Toolbar*
Field	*Trunk Options: Specify Trunk-to-Device Spacing* (Create Trunks and Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTrunk2DevSpacing")
envSetVal("layoutXL" "modgenPToTTrunk2DevSpacing" 'float 1.2)
```

Related Topics

List of Modgen Environment Variables

modgenPToTTrunkInsertMode

```
layoutXL modgenPToTTrunkInsertMode boolean { t | nil }
```

Description

With this option set to `t`, whenever a trunk is deleted, the adjacent (existing) trunks are moved to ensure that the relative distances between them remain the same.

When the option is set to `nil` (default state), the adjacent (existing) trunks remain at their original positions, irrespective of whether trunks are deleted.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenPToTTrunkInsertMode")
envSetVal("layoutXL" "modgenPToTTrunkInsertMode" 'boolean t)
envSetVal("layoutXL" "modgenPToTTrunkInsertMode" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTTrunkLayer

```
layoutXL modgenPToTTrunkLayer string "layer_name"
```

Description

Specifies the name of the layer on which trunks need to be created for the current Modgen. The layer name must be enclosed in quotation marks; for example, `"metal1"`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies* and *Create Single Trunk and Topologies icons on the Modgen Routing Toolbar*
Field	*Trunk Options: Trunks Layer* (Create Trunks and Topologies)
	Layer (Create Single Trunk and Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTrunkLayer")
envSetVal("layoutXL" "modgenPToTTrunkLayer" 'string "metal1")
```

Related Topics

List of Modgen Environment Variables

modgenPToTTrunkNets

```
layoutXL modgenPToTTrunkNets string "net_name"
```

Description

Specifies the names of the nets to which trunks need to be added. The net name must be enclosed in quotation marks; for example, `"(\"GND\")"`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies icon on the Modgen Routing Toolbar*
Field	*Specify Channel Nets* (Create Trunks and Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTrunkNet")
envSetVal("layoutXL" "modgenPToTTrunkNet" 'string "(\"GND\")")
envSetVal("layoutXL" "modgenPToTTrunkNet" 'string "(\"GND\" \"INP\" \"OUT\")")
```

Related Topics

List of Modgen Environment Variables

modgenPToTTrunkSpacing

```
layoutXL modgenPToTTrunkSpacing float float_number
```

Description

Specifies the trunk spacing for the current Modgen.

The default value is `0.0`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies icon on the Modgen Routing Toolbar*
Field	*Trunk Options*: *Absolute Spacing* (Create Trunks and Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTrunkSpacing")
envSetVal("layoutXL" "modgenPToTTrunkSpacing" 'float 1.2)
```

Related Topics

List of Modgen Environment Variables

modgenPToTTrunkWidth

```
layoutXL modgenPToTTrunkWidth float float_number
```

Description

Specifies the width of trunks in the current Modgen.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies* and *Create Single Trunk and Topologies icons on the Modgen Routing Toolbar*
Field	*Trunk Options: Trunks Width* (Create Trunks and Topologies)
	Width (Create Single Trunk and Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTrunkWidth")
envSetVal("layoutXL" "modgenPToTTrunkWidth" 'float 1.2)
```

Related Topics

List of Modgen Environment Variables

modgenPToTTwigAbsoluteWidth

```
layoutXL modgenPToTTwigAbsoluteWidth float float_number
```

Description

Specifies the width of twigs in the current Modgen. This option can be used only if the modgenPToTTwigWidthType environment variable is set to `Absolute Width`. The default value is `0.0`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Twig Options: Absolute Width* (Create Trunks and Topologies, Create Single Trunk and Topologies, and Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTwigAbsoluteWidth")
envSetVal("layoutXL" "modgenPToTTwigAbsoluteWidth" 'float 1.2)
```

Related Topics

List of Modgen Environment Variables

modgenPToTTwigDirectionDown

```
layoutXL modgenPToTTwigDirectionDown boolean { t | nil }
```

Description

Specifies the direction (down) along which twigs should be searched, starting from the selected source. It also indicates the direction in which twigs must be created.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Add/Replace Twig Topologies*
Field	*Search Direction from Source – Down* (Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTwigDirectionDown")
envSetVal("layoutXL" "modgenPToTTwigDirectionDown" 'boolean t)
envSetVal("layoutXL" "modgenPToTTwigDirectionDown" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTTwigDirectionLeft

```
layoutXL modgenPToTTwigDirectionLeft boolean { t | nil }
```

Description

Specifies the direction (left) along which twigs should be searched, starting from the selected source. It also indicates the direction in which twigs must be created.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Add/Replace Twig Topologies*
Field	*Search Direction from Source – Left* (Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTwigDirectionLeft")
envSetVal("layoutXL" "modgenPToTTwigDirectionLeft" 'boolean t)
envSetVal("layoutXL" "modgenPToTTwigDirectionLeft" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTTwigDirectionOver

```
layoutXL modgenPToTTwigDirectionOver boolean { t | nil }
```

Description

Specifies that for a trunk located over the device or pin, a twig must be created, which connects the trunk to an overlapping pin on the same net.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Add/Replace Twig Topologies*
Field	*Search Direction from Source – Over* (Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTwigDirectionOver")
envSetVal("layoutXL" "modgenPToTTwigDirectionOver" 'boolean t)
envSetVal("layoutXL" "modgenPToTTwigDirectionOver" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTTwigDirectionRight

```
layoutXL modgenPToTTwigDirectionRight boolean { t | nil }
```

Description

Specifies the direction (right) along which twigs should be searched, starting from the selected source. It also indicates the direction in which twigs must be created.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Add/Replace Twig Topologies*
Field	*Search Direction from Source – Right* (Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTwigDirectionRight")
envSetVal("layoutXL" "modgenPToTTwigDirectionRight" 'boolean t)
envSetVal("layoutXL" "modgenPToTTwigDirectionRight" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTTwigDirectionUp

```
layoutXL modgenPToTTwigDirectionUp boolean { t | nil }
```

Description

Specifies the direction (up) along which twigs should be searched, starting from the selected source. It also indicates the direction in which twigs must be created.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Add/Replace Twig Topologies*
Field	*Search Direction from Source – Up* (Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTwigDirectionUp")
envSetVal("layoutXL" "modgenPToTTwigDirectionUp" 'boolean t)
envSetVal("layoutXL" "modgenPToTTwigDirectionUp" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTTwigLayer

```
layoutXL modgenPToTTwigLayer string "layer_name"
```

Description

Specifies the name of the layer on which twigs need to be created for the current Modgen. The layer name must be enclosed in quotation marks; for example, `"metal1"`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Twig Options: Twigs Layer* (Create Trunks and Topologies, Create Single Trunk and Topologies, and Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTwigLayer")
envSetVal("layoutXL" "modgenPToTTwigLayer" 'string "metal1")
```

Related Topics

List of Modgen Environment Variables

modgenPToTTwigMinNumCuts

```
layoutXL modgenPToTTwigMinNumCuts int integer_number
```

Description

Specifies the minimum number of cuts for twigs in the current Modgen. Value must be a non-zero, positive integer that specifies the number of cuts for twigs. The default value is `1`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Twig Options: Twigs Min Num Cuts* (Create Trunks and Topologies, Create Single Trunk and Topologies, and Add/ Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTwigMinNumCuts")
envSetVal("layoutXL" "modgenPToTTwigMinNumCuts" 'int 2)
```

Related Topics

List of Modgen Environment Variables

modgenPToTTwigRelativeWidth

```
layoutXL modgenPToTTwigRelativeWidth int integer_number
```

Description

Specifies the relative width of twigs in the current Modgen. This option can be used only if the modgenPToTTwigWidthType environment variable is set to `Relative Width (Pin %)`. Value must be a non-zero, positive integer that specifies the relative width of twigs.

Default value is `100`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Twig Options: Relative Width (Pin %)* (Create Trunks and Topologies, Create Single Trunk and Topologies, and Add/ Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTwigRelativeWidth")
envSetVal("layoutXL" "modgenPToTTwigRelativeWidth" 'int 70)
```

Related Topics

List of Modgen Environment Variables

modgenPToTTwigWidthType

```
layoutXL modgenPToTTwigWidthType cyclic { "MinWidth" | "Absolute Width" | "Relative
     Width (Pin %)" }
```

Description

Specifies the mode in which the twig width can be specified for the current Modgen. This environment variable can take three values. Default is `MinWidth`.

- `MinWidth`: Minimum width of twigs as per the defaults is used if this option is specified.
- `Absolute Width`: Absolute width of twigs can be specified (using the modgenPToTTwigAbsoluteWidth environment variable), if this option is specified.
- `Relative Width (Pin %)`: Relative width of twigs can be specified (using the modgenPToTTwigRelativeWidth environment variable), if this option is specified.GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Twig Options: Twigs Width Type* (Create Trunks and Topologies, Create Single Trunk and Topologies, and Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTTwigWidthType")
envSetVal("layoutXL" "modgenPToTTwigWidthType" 'cyclic "MinWidth")
envSetVal("layoutXL" "modgenPToTTwigWidthType" 'cyclic "Absolute Width")
envSetVal("layoutXL" "modgenPToTTwigWidthType" 'cyclic "Relative Width (Pin %)")
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlCutClass1

```
layoutXL modgenPToTViaControlCutClass1 float float_number
```

Description

Stores the last used cut class width value.

The default value is `0.0`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Via Controls – Cut Class Width* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlCutClass1")
envSetVal("layoutXL" "modgenPToTViaControlCutClass1" 'float 1.2)
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlCutClass2

`layoutXL modgenPToTViaControlCutClass2 float` *`float_number`*

Description

Stores the last used cut class height value.

The default value is `0.0`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Via Controls – Height* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlCutClass2")
envSetVal("layoutXL" "modgenPToTViaControlCutClass2" 'float 1.2)
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlCutClassLayer

```
layoutXL modgenPToTViaControlCutClassLayer string "layer_name"
```

Description

Stores the layer name of the last used cut class.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Via Controls – Cut Class – By Name – Layer* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlCutClassLayer")
envSetVal("layoutXL" "modgenPToTViaControlCutClassLayer" 'string metal1)
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlCutClassName

```
layoutXL modgenPToTViaControlCutClassName string "cutClass_name"
```

Description

Stores the name of the last used cut class.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Via Controls – Cut Class – By Name – Name* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlCutClassName")
envSetVal("layoutXL" "modgenPToTViaControlCutClassName" 'string CutClass1)
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlCutClassType

```
layoutXL modgenPToTViaControlCutClassType string "cutClass_type"
```

Description

Specifies how the last used cut class must be stored. Valid values are the following:

- `By Size`: Lets you specify the width and height of the associated cut classes.
- `By Name`: Lets you specify the name and layer of the associated cut classes.

The default value is `By Size`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Via Controls – Cut Class* (Create Trunks and Topologies*,* Create Single Trunk and Topologies*,* Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlCutClassType")
envSetVal("layoutXL" "modgenPToTViaControlCutClassType" 'string "By Name")
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlCutClassEnable

```
layoutXL modgenPToTViaControlCutClassEnable boolean { t | nil }
```

Description

When set to `t`, enables the *Cut Class Width* and *Height* options on the *Via Controls* tab.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*\
Field	*Advanced Options – Via Controls – Cut Class Width, Height* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlCutClassEnable")
envSetVal("layoutXL" "modgenPToTViaControlCutClassEnable" 'boolean t)
envSetVal("layoutXL" "modgenPToTViaControlCutClassEnable" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlExtensionOrient

```
layoutXL modgenPToTViaControlExtensionOrient cyclic { "None" | "Horizontal" |
     "Vertical" }
```

Description

Specifies the preference of the via extension or enclosure orientations. Default is `None`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Via Controls – Via Extension Orientation* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlExtensionOrient")
envSetVal("layoutXL" "modgenPToTViaControlExtensionOrient" 'cyclic "None")
envSetVal("layoutXL" "modgenPToTViaControlExtensionOrient" 'cyclic "Horizontal")
envSetVal("layoutXL" "modgenPToTViaControlExtensionOrient" 'cyclic "Vertical")
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlExtensionOrientEnable

```
layoutXL modgenPToTViaControlExtensionOrientEnable boolean { t | nil }
```

Description

When set to `t`, enables the *Via Extension Orientation* option on the *Via Controls* tab.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Via Controls – Via Extension Orientation* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlExtensionOrientEnable")
envSetVal("layoutXL" "modgenPToTViaControlExtensionOrientEnable" 'boolean t)
envSetVal("layoutXL" "modgenPToTViaControlExtensionOrientEnable" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlInline

```
layoutXL modgenPToTViaControlInline boolean { t | nil }
```

Description

When selected, places a higher preference for vias that are fully enclosed or inline with the wire.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Via Controls – Via Inline* (Create Trunks and Topologies*,* Create Single Trunk and Topologies*,* Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlInline")
envSetVal("layoutXL" "modgenPToTViaControlInline" 'boolean t)
envSetVal("layoutXL" "modgenPToTViaControlInline" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlInlineEnable

```
layoutXL modgenPToTViaControlInlineEnable boolean { t | nil }
```

Description

When set to `t`, enables the *Via Inline* option on the *Via Controls* tab.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Via Controls – Via Inline* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlInlineEnable")
envSetVal("layoutXL" "modgenPToTViaControlInlineEnable" 'boolean t)
envSetVal("layoutXL" "modgenPToTViaControlInlineEnable" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlOffset

```
layoutXL modgenPToTViaControlOffset boolean { t | nil }
```

Description

When set to `t`, honors the via offset values.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Via Controls – Via Offset* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlOffset")
envSetVal("layoutXL" "modgenPToTViaControlOffset" 'boolean t)
envSetVal("layoutXL" "modgenPToTViaControlOffset" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlOffsetEnable

```
layoutXL modgenPToTViaControlOffsetEnable boolean { t | nil }
```

Description

When set to `t`, enables the *Via Offset* option.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Via Controls – Via Offset* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlOffsetEnable")
envSetVal("layoutXL" "modgenPToTViaControlOffsetEnable" 'boolean t)
envSetVal("layoutXL" "modgenPToTViaControlOffsetEnable" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlOrient

```
layoutXL modgenPToTViaControlOrient cyclic { "None" | "Horizontal" | "Vertical" }
```

Description

Specifies the default via orientation. Default is `None`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Via Controls – Via Orientation* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/ Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlOrient")
envSetVal("layoutXL" "modgenPToTViaControlOrient" 'cyclic "None")
envSetVal("layoutXL" "modgenPToTViaControlOrient" 'cyclic "Horizontal")
envSetVal("layoutXL" "modgenPToTViaControlOrient" 'cyclic "Vertical")
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaControlOrientEnable

```
layoutXL modgenPToTViaControlOrientEnable boolean { t | nil }
```

Description

When set to `t`, enables the *Via Orientation* option.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Via Controls – Via Orientation* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/ Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaControlOrientEnable")
envSetVal("layoutXL" "modgenPToTViaControlOrientEnable" 'boolean t)
envSetVal("layoutXL" "modgenPToTViaControlOrientEnable" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaWidthPercent

```
layoutXL modgenPToTViaWidthPercent int integer_number
```

Description

Specifies the depth (in percentage) of via coverage when the via overlaps with a pin.

The default value is `100`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced – Pin Cover – Via Percent (%)* (Create Trunks and Topologies, Create Single Trunk and Topologies, Add/Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaWidthPercent")
envSetVal("layoutXL" "modgenPToTViaWidthPercent" 'int 120)
envSetVal("layoutXL" "modgenPToTViaWidthPercent" 'int 170)
```

Related Topics

List of Modgen Environment Variables

modgenPToTViaWidthPercentEnable

```
layoutXL modgenPToTViaWidthPercentEnable boolean { t | nil }
```

Description

Specifies whether the depth (in percentage) of via coverage, when the via overlaps a pin, can be specified.

The default value is `nil`.

GUI Equivalent

Command	Modgen Editor – *Create Trunks and Topologies, Create Single Trunk and Topologies,* and *Add / Replace Twig Topologies icons on the Modgen Routing Toolbar*
Field	*Advanced Options – Pin Cover – Via Percent (%)* (Create Trunks and Topologies*,* Create Single Trunk and Topologies*,* Add/ Replace Twig Topologies)

Examples

```
envGetVal("layoutXL" "modgenPToTViaWidthPercentEnable")
envSetVal("layoutXL" "modgenPToTViaWidthPercentEnable" 'boolean t)
envSetVal("layoutXL" "modgenPToTViaWidthPercentEnable" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenSaveOnClosePreviewWindow

```
layoutXL modgenSaveOnClosePreviewWindow cyclic { "prompt" | "yes" | "no" }
```

Description

Specifies the behavior when the Modgen Previewer window is closed. When set to `prompt`, a pop-up message is displayed requesting for confirmation whether changes to the Modgen need to be saved. Choose *Yes* to save changes, and *No* to discard changes. Click *Cancel* to reject the closing of the window.

When set to `yes`, no pop-up is displayed. Instead, all modifications are saved and the Modgen Previewer window is closed.

When set to `no`, no pop-up is displayed. Instead, all modifications are discarded and the Modgen Previewer window is closed.

The default is `prompt`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenSaveOnClosePreviewWindow")
envSetVal("layoutXL" "modgenSaveOnClosePreviewWindow" 'cyclic "prompt")
envSetVal("layoutXL" "modgenSaveOnClosePreviewWindow" 'cyclic "yes")
envSetVal("layoutXL" "modgenSaveOnClosePreviewWindow" 'cyclic "no")
```

Related Topics

List of Modgen Environment Variables

modgenTransferDiffInstaces

```
layoutXL modgenTransferDiffInstances boolean { t | nil }
```

Description

When set to `t`, verifies the layout and schematic parameters during Modgen transfer from the schematic to layout or vice versa.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenTransferDiffInstaces")
envSetVal("layoutXL" "modgenTransferDiffInstaces" 'boolean t)
envSetVal("layoutXL" "modgenTransferDiffInstaces" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenTransferIgnoreParamsList

```
layoutXL modgenTransferIgnoreParamsList string "parameters_list"
```

Description

Indicates the parameters to be ignored while verifying parameters between the layout and the schematic during transfer of the Modgen from schematic to layout or vice versa. This environment variable can be used only when modgenTransferDiffInstaces is set to `t`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenTransferIgnoreParamsList")
envSetVal("layoutXL" "modgenTransferIgnoreParamsList" 'string "paramName")
```

Related Topics

List of Modgen Environment Variables

modgenUseSnapSpacing

```
layoutXL modgenUseSnapSpacing boolean { t | nil }
```

Description

Allows to snap Modgen origin to snap spacing.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenUseSnapSpacing")
envSetVal("layoutXL" "modgenUseSnapSpacing" 'boolean t)
envSetVal("layoutXL" "modgenUseSnapSpacing" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenUseIteratedAsMfactor

```
layoutXL modgenUseIteratedAsMfactor boolean { t | nil }
```

Description

Allows Modgen to consider iterated instances, with same or different connectivity, equivalent to an m-factor (multiplier).

The default value is `t`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenUseIteratedAsMfactor")
envSetVal("layoutXL" "modgenUseIteratedAsMfactor" 'boolean t)
envSetVal("layoutXL" "modgenUseIteratedAsMfactor" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

modgenWidthParamProportionalToFingers

```
layoutXL modgenWidthParamProportionalToFingers cyclic { "default" | "yes" | "no" }
```

Description

Specifies whether the finger width of Modgen dummy instances must be proportional to the number of fingers. The default value is `default`. For FinFET devices, the `default` is `no`, and for non-FinFET devices, the `default` is `yes`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "modgenWidthParamProportionalToFingers")
envSetVal("layoutXL" "modgenWidthParamProportionalToFingers" 'cyclic "default")
envSetVal("layoutXL" "modgenWidthParamProportionalToFingers" 'cyclic "yes")
envSetVal("layoutXL" "modgenWidthParamProportionalToFingers" 'cyclic "no")
```

Related Topics

List of Modgen Environment Variables

modgenWindowConfigFile

```
layoutXL modgenWindowConfigFile string "workspace_name"
```

Description

Specifies the workspace that Modgen needs to load during startup.

GUI Equivalent

None

Example

```
envGetVal("layoutXL" "modgenWindowConfigFile")
envSetVal("layoutXL" "modgenWindowConfigFile" 'string "modgenUserWorkspace")
```

Related Topics

List of Modgen Environment Variables

chainPermutePins

```
layoutXL chainPermutePins boolean { t | nil }
```

Description

Specifies whether to use mirroring instances or permutation of pins during the abutment process.

The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL" "chainPermutePins")
envSetVal("layoutXL" "chainPermutePins" 'boolean t)
envSetVal("layoutXL" "chainPermutePins" 'boolean nil)
```

Related Topics

List of Modgen Environment Variables

Grid Pattern Editor Environment Variables

moveAsSwap

```
gpe moveAsSwap boolean { t | nil }
```

Description

Specifies the behavior when instances are moved in the Grid Pattern Editor. With this environment variable set to `t`, the instances in the source and target cells are swapped. When the environment variable is set to `nil`, the target instance first shifts horizontally and then vertically, until the source location is backfilled.

The default value is `t`.

GUI Equivalent

None

Examples

```
envGetVal("gpe" "moveAsSwap")
envSetVal("gpe" "moveAsSwap" 'boolean t)
envSetVal("gpe" "moveAsSwap" 'boolean nil)
```

Related Topics

Grid Pattern Editor Environment Variables

8

Modgen Topology Constraints

The Modgen pin-to-trunk router recognizes topology objects and their constraints and creates geometries that match the topology pattern. You can add topology constraints to customize certain routing attributes. All topology constraints have a name, a layer, and a value. The layer can be set to `nil` if it is not applicable.

The Modgen topology constraints can be passed as arguments to certain Modgen place and route SKILL APIs such as `gpeCreateTrunkEntries`, `gpeCreateTwigEntries`, and `gpeCreateStrapEntries`. These constraints are honored while creating the respective topology objects.

In the following example, values for `twigConstraints` are first instantiated. These values are then passed as an argument to `gpeCreateTwigEntries`.

```
twigConstraints= list(
list("validLayers" nil list("Metal1" "Metal2"))
list("minWidth" list("Metal1") 0.14)
list("minWidth" list("Metal2") 0.12)
)
twigEntry = gpeCreateTwigEntries(
?instTermEntries list(iterm)
?constraints twigConstraints
)
```

For more information about Modgen topologies, see Creating Topology Patterns and pin-to-trunk Routing.

For more information about Modgen topology and pin-to-trunk routing SKILL APIs, see Modgen Placement and Routing Functions section of the Virtuoso Layout Suite SKILL Reference manual.

This chapter provides information about the following Modgen topology constraints.

- minNumCut
- minWidth
- numStrands
- pinCover
- properTwigTrunkOverlap
- strandSpacing
- strandWidth
- trunkAccessingNumCuts
- validLayers
- validStackLPPs
- viaControl
- viaPercent
- wirePercent

minNumCut

Definition

Specifies the minimum number of cuts that a via object or a via instance, which is created between a pin and the topology connecting that pin, must contain. Use constraint trunkAccessingNumCuts for the vias that are created between trunk pathSegs and the topology connecting that trunk.

Wide wires are capable of carrying more current, and a sufficient number of via cuts is required to carry that additional current. Using multiple via cuts increases redundancy, and therefore reliability.

Values

- *l_layerNames*

  Specifies the layers to which the constraint must be applied.

  Type: List of strings (layer and purpose names) or integers (layer numbers)

- *x_minNumCut*

  The minimum number of cuts that a via object or a via instance must contain.

  Type: Integer

Parameters

None

Applies To

- Twig constraints
- Strap constraints

Example

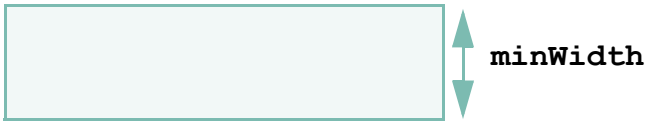
```
'("minNumCut" '("Metal1" "Metal2" "Metal3") 4 )
```

Specifies that the vias on the specified layers must contain a minimum of four cuts.

minWidth

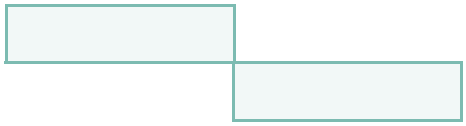
Definition

Specifies the minimum orthogonal width of shapes on the specified layers.

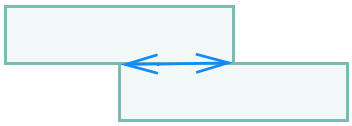


The minimum width constraint does not apply in the following situations:

- Two shapes touch at a point, as shown in the following figure:



- Two shapes abut for a sufficient overlap, as shown in the following figure:



Values

- *l_layerNames*

  Specifies the layers to which the constraint must be applied.

  Type: String (layer name), list of strings (layer and purpose names), or integers (layer numbers)

- *x_minWidth*

  The minimum permissible orthogonal width of shapes.

  Type: Integer

Parameters

None

Applies To

- Trunk constraints
- Twig constraints
- Strap constraints

Example

```
'("minWidth" '("Metal1") 0.14)
```

Sets the minimum width constraint to `0.14`.

```
'("minWidth" '("Metal1" "Metal2" "Metal3") 0.11)
```

Sets the minimum width constraint for the specified layers to `0.14`.

numStrands

Definition Specifies the number of strands to be used to route the topology object.

Values

- `nil`

  Layer specification is not required.

- *`x_numStrands`*

  Specifies the number of strands to be used to route the topology object.

  Type: Integer

Parameters None

Applies To

- Trunk constraints
- Twig constraints
- Strap constraints

Example

```
'("numStrands" nil 3)
```

Specifies that three strands must be used to route pins in the given topology object.

pinCover

Definition

Defines parameters to control how vias must cover pins.

Values

- `nil`

  Layer specification is not required.

- *`g_pinCoverStatus`*

  Specifies whether the tap must cover the pin area.

  Type: Boolean (`t` or `nil`) or Integer (`0` or `1`)

  Default: `100`

Parameters

- **`pinCoverTapViaPercent`**

  Specifies the percentage of pin shape to be covered by the tapping via on all the layers. The percentage is measured in the direction perpendicular to the trunk.

  Type: Integer

  Valid Values: `0` through `100`

  Default: `100`

- **`pinCoverTapLowerViaPercent`**

  Specifies the percentage of a pin shape to be covered by the lower layers of the tapping vias. This value applies to all layers expect the top layer. The percentage is measured in the direction perpendicular to the trunk.

  Type: Integer

  Valid Values: `0` through `100`

  Default: `100`

- **pinCoverViaModeTrunkOver**

  Specifies the via mode to be applied when a trunk or pin is covered by a via stack. Valid values are:

  - `matchTrunk`: Covers the width of the trunk.
  - `full`: Covers the entire pin shape.
  - `fullExceptTop`: Covers the entire trunk on the top via layer and the entire pin on the lower via layers.

  Type: Enum

  Valid Values: `(matchTrunk, full, fullExceptTop)`

  Default: `matchTrunk`

- **pinCoverViaModePercentTrunkOver**

  Specifies the percentage of trunk or pin to be covered by the via stack. The percentage is measured in the direction perpendicular to the trunk. This option can be used only when `pinCoverViaModeTrunkOver` is set to `full` or `fullExceptTop`.

  Type: Integer

  Valid Values: `0` through `100`

  Default: `100`

Applies To

- Twig constraints

Example

```
list("pinCover" nil 1
        list("pinCoverTapViaPercent" 70)
        list("pinCoverTapLowerViaPercent" 70)
        list("pinCoverViaModeTrunkOver" full)
        list("pinCoverViaModePercentTrunkOver" 80)
        ))
```

Specifies the `pinCover` parameters.

properTwigTrunkOverlap

Definition	Specifies whether the twig must extend to cover the width of the trunk.
Values	■ `nil` Layer specification is not required. ■ *`g_overlapStatus`* Specifies whether the twig must extend to cover the width of the trunk. *Type:* Boolean (`t` or `nil`) or Integer (`0` or `1`)
Parameters	None
Applies To	■ Twig constraints

Example

```
'("properTwigTrunkOverlap" nil 1 )
'("properTwigTrunkOverlap" nil t )
```

The `properTwigTrunkOverlap` value is an integer in the first example, and a Boolean in the second example.

strandSpacing

Definition

Specifies the required exact spacing between individual wire strands on the given layers.

Values

- *l_layerNames*

  Specifies the layers to which the constraint must be applied.

- *x_strandSpacing*

  Specifies the exact spacing between individual wire strands on the given layer.

  Type: Integer

Parameters

None

Applies To

- Trunk constraints
- Twig constraints
- Strap constraints

Example

```
'("strandSpacing" "Metal1" 0.06)
```

Sets `0.06` user units as the required spacing between individual strands on layer `Metal1`.

strandWidth

Definition	Specifies the required exact width of individual wire strands on the given layer.
Values	■ *l_layerNames* Specifies the layers to which the constraint must be applied. ■ *x_strandWidth* Specifies the width of individual wire strands on the given layers. *Type:* Integer
Parameters	None
Applies To	■ Trunk constraints ■ Twig constraints ■ Strap constraints

Example

```
'("strandWidth" "Metal1" 0.03)
```

Sets `0.03` user units as the required width of individual strands on layer `Metal1`.

trunkAccessingNumCuts

Definition

Specifies the number of cuts a via must contain when connecting a trunk to a twig.

Values

- *l_layerNames*

  Specifies the layers to which the constraint must be applied.

  Type: List of strings (layer and purpose names) or integers (layer numbers)

- *x_numCuts*

  Number of cuts that the via must contain.

  Type: Integer

Parameters

None

Applies To

- Twig constraints
- Strap constraints

Example

```
'("trunkAccessingNumCuts" '("Metal1") 2)
```

Specifies that vias with two cuts can be used to connect trunks to twigs.

validLayers

Definition Defines a list of valid routing layers or layer-purpose pairs.

Values

- `nil`

  Layer specification is not required.

- *`tx_layer`*

  Lists the layers that can be used for routing. If you do not specify the purpose, then the constraint applies to all purposes.

  Type: String (layer name) or integer (layer number)

- (*`tx_layer tx_purpose`*)

  Lists the layer-purpose pairs that can be used for routing.

  Type: String (layer and purpose names) or integer (layer and purpose numbers)

Parameters None

Applies To

- Trunk constraints
- Twig constraints
- Strap constraints

Examples

```
'("validLayers" nil '("Metal1" "Metal2" "Metal3"))
```

Specifies the valid layers for routing.

```
'("validLayers" nil '("Metal1 pin" "Metal2 drawing" "Metal3 pin"))
```

Specifies the valid layer-purpose pairs for routing.

validStackLPPs

Definition	Specifies a list of layer-purpose pairs that can be used for routing. The router uses all the listed layer-purpose pairs for routing.
Values	■ `nil` Layer specification is not required. ■ *`l_validStackLPPs`* List of layer-purpose pairs that can be used for routing. *Type:* list
Parameters	None
Applies To	■ Trunk constraints ■ Twig constraints ■ Strap constraints

Example

```
'("validStackLPPs" nil '("Metal1 drawing" "Metal2 pin" "Metal3 drawing"))
```

Defines valid layer-purpose pairs for routing.

viaControl

Definition

Specifies the via control parameters that define how vias must be positioned and aligned on trunks and twigs.

Values

- *l_layerNames*

  Specifies the layers to which the constraint must be applied.

- *x_viaControlStatus*

  Specifies whether the via control parameters can be specified.

Parameters

- **viaControlOrient**

  Specifies the default via orientation. The bounding box of the via cuts is aligned along the specified direction. For example, a via with two cuts is rendered top-down if the via orientation is set to `Vertical`.

  Type: String

  Valid Values: `(Horizontal, Vertical, None)`

  Default: `None`

- **viaControlInline**

  Prefers vias that are fully enclosed or in line with the wire.

  Type: Boolean

  Valid Values: `(1, 0)` or `(t, nil)`

  Default: `0`

- **viaControlOffset**

  Honors the via offset values specified.

  Type: Boolean

  Valid Values: `(1, 0)` or `(t, nil)`

  Default: `0` or `nil`

- **viaControlExtensionOrient**

  Specifies the preference of the via extension or enclosure orientations. For vias with multiple cuts, a higher preference is given to vias with cut boxes lining up in a certain direction. Therefore, a vertical-cut bounding box via may have a horizontal extension. This means that the extended metal portion over the cut bounding box in the horizontal direction is larger than that in the vertical direction.

  Type: String

  Valid Values: `(Horizontal, Vertical, None)`

  Default: `None`

- **viaControlCutClass**

  Specifies the width and height of the associated cut classes.

  Type: List

  Valid Values: A list of two floating-point numbers, for example `'(0.032 0.032)`.

  Default: `'(0 0)`

Applies To

- Twig constraints

Example

```
list("viaControl" list("M1" "M2" "M3") 1
    list(
        list("viaControlOrient" "horizontal")
        list("viaControlInline" 1)
        list("viaControlOffset" 1)
        list("viaControlExtensionOrient" "horizontal")
        list("viaControlCutClass" '(0.07 0.07))
        )
    )
```

Specifies the various via control parameters.

viaPercent

Definition

Specifies the allowed wire width as a percentage of the width of the associated pin. The setting applies in the direction perpendicular to the trunk.

Values

- `nil`

  Layer specification is not required.

- *`x_viaPercent`*

  Specifies the allowed wire width as a percentage of the width of the associated pin.

  Type: Integer

  Valid Values: `0` to `100`

  Default: `100`

Parameters

None

Applies To

- Twig constraints
- Strap constraints

Example

```
'("viaPercent" nil 60)
```

Specifies the allowed wire width as `60` percent of the width of the associated pin.

wirePercent

Definition	Specifies the allowed wire width as a percentage of the pin width. The value is calculated along the pin edge parallel to the direction of the trunk.
Values	■ `nil` Layer specification is not required. ■ *`x_wirePercent`* Specifies the wire width as a percentage of the pin width. *Type:* Integer
Parameters	None
Applies To	■ Twig constraints ■ Strap constraints

Example

```
'("wirePercent" nil 90)
```

Specifies that the allowed wire width is `90` percent of the width of the associated pin.