# Automated Device Placement and Routing Flow 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

This guide provides information about the various tasks in the automatic device-level layout flow and the tools that help you perform these tasks. The Auto P&R assistant, which is available in Layout EXL, lists these tasks in the correct sequence—from initializing the design and placing objects to routing the design automatically.

This flow 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.
- The Virtuoso design environment technology file.

This preface contains the following topics:



## Scope

The functionality described in this guide can be used ONLY in the advanced node ICADVM20.1 release. It is not available in mature node releases.

# Licensing Requirements

You can open the Auto Device P&R assistant only from the Layout EXL cockpit. The assistant requires the following licenses to be available in addition to the usual licenses required for the tools you are using:

- 95511: Virtuoso Advanced Node Options for Layout
- 95800: Virtuoso Layout Suite EXL

The Tree Router checks out 12 GXL tokens in addition to the above licenses.

For more information on the license checkout requirements for Layout EXL, see *Virtuoso Software Licensing and Configuration Guide*.

# Related Documentation

## What's New

- *Virtuoso Automated Device Placement and Routing Flow What's New*

## Installation, Environment, and Infrastructure

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

## Technology Information

- *Virtuoso Technology Data User Guide*
- *Virtuoso Technology Data ASCII Files Reference*
- *Virtuoso Technology Data SKILL Reference*
- *Virtuoso Technology Data Constraints 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 Concurrent Layout Editing User Guide*
- *Virtuoso Design Planner User Guide*
- *Virtuoso Multi-Patterning Technology User Guide*
- *Virtuoso Width Spacing Patterns User Guide*

### IC6.1.8 and ICADVM20.1

- *Virtuoso Abstract Generator User Guide*
- *Virtuoso Custom Digital Placer User Guide*
- *Virtuoso Floorplanner User Guide*
- *Virtuoso Layout Suite SKILL Reference*
- *Virtuoso Module Generator User Guide*
- *Virtuoso Parameterized Cell Reference*
- *Virtuoso Space-based Router User Guide*
- *Virtuoso Placer 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.

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.

# Understanding Cadence SKILL

Cadence SKILL is a high-level, interactive programming language based on the popular artificial intelligence language, Lisp. It lets you customize and extend your design environment. Using SKILL you can validate the steps of your algorithm incrementally before incorporating them into a larger program.

For more information about the SKILL language, see Getting Started in the *SKILL Language User Guide*.

## Using SKILL Code Examples

The SKILL APIs in this user manual are explained with illustrative code examples.

You can copy these examples from the manual and paste them directly into the Command Interpreter Window (CIW) or use the code in non-graphical SKILL mode.

## Sample SKILL Code

The following code sample shows the syntax of a SKILL API that accepts three arguments.

**axlGetRunStatus**

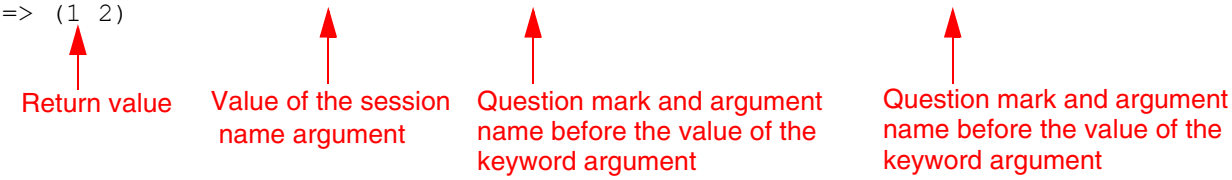
```
axlGetRunStatus(
    t_sessionName                          <-- Required argument
    [ ?optionName t_optionName ]           <-- Optional keyword argument
    [ ?historyName t_historyName ]         <-- Optional keyword argument
    )
    => l_statusValues                      <-- Return value
```

The first argument `t_sessionName` is a required argument, where `t` signifies the data type of the argument. The second and third arguments `?optionName t_optionName` and `?historyName t_historyName` are optional keyword arguments (identified by a question mark), which are specified in name-value pairs and can be placed in any order during the function call.

The return value is the value that the SKILL API returns after evaluating the expression. In this case, it is a list of status values, *l_statusValues*.

**Example**

```
axlSession=axlGetWindowSession( hiGetCurrentWindow())
=> "session0"
axlGetRunStatus("session0" ?historyName "Interactive.10" ?optionName "tests")
=> (1 2)
```




## Accessing API Help

Quick reference information for SKILL APIs is available from the CIW and the SKILL API Finder. To access the reference information for a particular SKILL API, do one of the following:

- Type `help <function_name>` in the CIW.
- Type `startFinder ( [ ?funcName t_functionName ] )` in the CIW.
- Start the SKILL API Finder from the CIW by choosing *Tools – Finder* or type `cdsFinder` on the UNIX command line.

  In the *Search in* field of the displayed Cadence SKILL API Finder window, type the SKILL API name for which you want to display the help information and click *Go*.

  The matches for the searched SKILL API appear in the *Results* area.

  To view the complete documentation of the searched SKILL API, select the API name in the *Results* area and click the *More Info* button. The complete documentation of the selected SKILL API appears in a new Cadence Help window.

# 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.<br>Used with brackets to indicate that you can specify zero or more arguments.<br>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.

# Identifiers Used to Denote Data Types

Data type identifiers are used to indicate the type of value required by an API argument. These data types are denoted by a single letter that is prefixed to the argument label and is separated from the argument by an underscore; for example, *t* is the data type in *t_viewName*. Data types and underscores are used only as identifiers; they must not be typed when specifying the argument in a function.

| Prefix | Internal Name | Data Type |
|---|---|---|
| *a* | array | array |
| *A* | amsobject | AMS object |
| *b* | ddUserType | DDPI object |
| *B* | ddCatUserType | DDPI category object |
| *C* | opfcontext | OPF context |
| *d* | dbobject | Cadence database object (CDBA) |
| *e* | envobj | environment |
| *f* | flonum | floating-point number |
| *F* | opffile | OPF file ID |
| *g* | general | any data type |
| *G* | gdmSpecIlUserType | generic design management (GDM) spec object |
| *h* | hdbobject | hierarchical database configuration object |
| *I* | dbgenobject | CDB generator object |
| *K* | mapiobject | MAPI object |
| *l* | list | linked list |
| *L* | tc | Technology file time stamp |
| *m* | nmpIlUserType | nmpIl user type |
| *M* | cdsEvalObject | cdsEvalObject |
| *n* | number | integer or floating-point number |
| *o* | userType | user-defined type (other) |
| *p* | port | I/O port |
| *q* | gdmspecListIlUserType | gdm spec list |

| Prefix | Internal Name | Data Type |
|---|---|---|
| *r* | defstruct | defstruct |
| *R* | rodObj | relative object design (ROD) object |
| *s* | symbol | symbol |
| *S* | stringSymbol | symbol or character string |
| *t* | string | character string (text) |
| *T* | txobject | transient object |
| *u* | function | function object, either the name of a function (symbol) or a lambda function body (list) |
| *U* | funobj | function object |
| *v* | hdbpath | hdbpath |
| *w* | wtype | window type |
| *sw* | swtype | subtype session window |
| *dw* | dwtype | subtype dockable window |
| *x* | integer | integer number |
| *y* | binary | binary function |
| *&* | pointer | pointer type |

For more information, see *Cadence SKILL Language User Guide*.

# 1

# Introduction to Automated Device Placement and Routing

This chapter provides information about the Virtuoso automated device placement and routing flow and the associated tools. The flow comprises a series of tasks that help generate automatically placed and routed layouts. The Virtuoso automated device placement and routing flow enables you to quickly generate placed and routed layouts that are constraint compliant and LVS correct, and follow DRCs as captured in the Virtuoso technology file. The layouts also incorporate base layer fill, as typically required in advanced nodes. These layouts can be used to extract parasitics for re-simulation to identify issues early on, without waiting for the final sign-off, and can be easily modified and updated to generate the final layout for sign-off.

The target designs for this flow are analog and mixed signal device-level blocks. The target process nodes are FinFET-based.

You can access the Auto Device P&R assistant from Layout EXL cockpit. This assistant provides a simple, yet powerful interface that guides you through the various tasks in the flow.

You can use environment variables to change the value of many aspects of your environment either for an individual design session or permanently until you change the value of the variable again. For more information about the environment variables, see Automated Device Placement and Routing Flow Environment Variables.

This chapter covers the following topics:

- Prerequisites
- Glossary of Terms
- Automated Device Placement and Routing Flow
- Auto Device P&R Assistant and Workspace
- Accessing the Auto Device P&R Assistant

## Prerequisites

To run the automated device placement and routing flow, you must have access to the following design environment capabilities:

- Virtuoso Release: ICADVM18.1 ISR4 or later
  - The flow is available only in the advanced node methodologies releases.

  For information about licensing in the Virtuoso design environment, see *Virtuoso Software Licensing and Configuration Guide*..
- Virtuoso Layout Suite EXL

  For information about the Layout Suite EXL editor, see *Virtuoso Layout Suite EXL Reference*.
- PDK Settings:
  - Support for Virtuoso Placer and Virtuoso Space-based Router

    For more information, see *Virtuoso Placer User Guide* and *Virtuoso Space-based Router User Guide*.
  - A LAM file that defines the component types for row templates

    For more information about LAM files, see Library and Attributes Mapping File Syntax in *Virtuoso Layout Suite XL: Connectivity Driven Editing User Guide*.
  - Device registration for circuit finders to recognize Circuit Prospector structures and create constraints

    Related SKILL Functions:
    - ciActiveSameCellAndSizeIterator
    - ciCascodeSeriesCurrentMirrorIterator
    - ciCommonGateAndSourceIterator
    - ciCommonGateIterator
    - ciCommonSourceIterator
    - ciHierarchicalSeriesIterator
  - (Optional, can be created as part of the flow) Width Spacing Patterns (WSPs) for placement and routing

- ❑ (Optional, can be created as part of the flow) Row templates
- ❑ (Optional, can be created as part of the flow) Poly Snap Patterns

*Important*

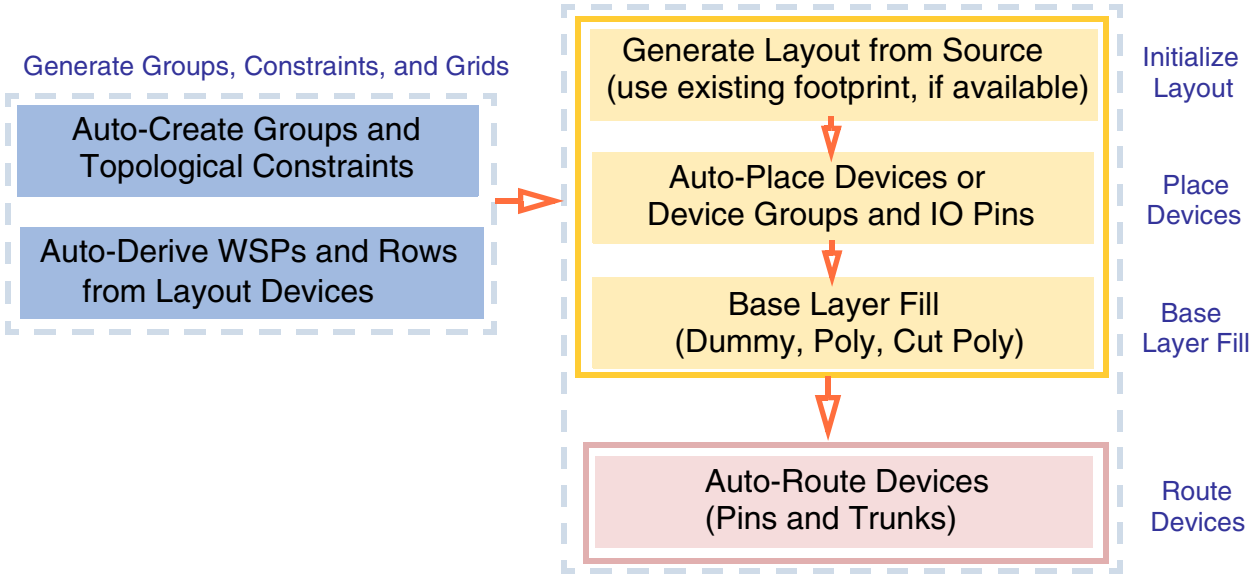
Ensure that the designs have uniform gate length to ensure uniform poly pitch for device snapping.

## Glossary of Terms

| Term | Explanation |
|---|---|
| Dummy | Devices that are structurally the same as regular devices, but their terminals are tied to supply or ground terminals (unless they are abutted to other devices). Dummies are inserted in the gaps between regular devices to achieve device matching. |
| Device Fill | Devices that are structurally similar to regular devices are inserted in the gaps between regular devices to ensure that the design meets density requirements. |
| Filler Cell | Devices that are structurally similar to regular devices are inserted in the gaps between regular devices to complete well connections so that there are no isolated wells. Adding filler cells enables the design to meet density requirements. |
| Tap Cell | A set of contacts (not actual cells) that connect to wells or substrates for latch-up protection. At advanced nodes, tap cells can resemble actual cells for matching and density reasons. |

## Automated Device Placement and Routing Flow

The following diagram summarizes the Virtuoso automated device placement and routing flow.




Flow steps:

- **Initialize Layout:** The first step is layout generation. Information about the PR boundary, instances, nets, and pins is generated in the target layout as per the source schematic.
- **Generate Groups, Constraints, and Grids:** Registered circuit finders are run on the schematic instances and nets to identify all compatible structures. These structures are then grouped into constraints and constraint groups and generated in the layout cellview. Information about any existing WSPs and row regions is also transferred.
- **Place Devices:** The automated device placement and routing flow supports two placement objectives—design compaction and better routability. In this step, you select a placement objective and then run the placer.
- **Base Layer Fill:** After placement, the gaps between devices are filled with either dummy fill or poly fill. Inserting fill helps maintain continuity of instances within each row, and ensures that the design is DRC-clean and meets density requirements.
- **Route Devices:** The final step is routing. The Tree router is used to connect the placed devices, while honoring all design rules.

## Auto Device P&R Assistant and Workspace

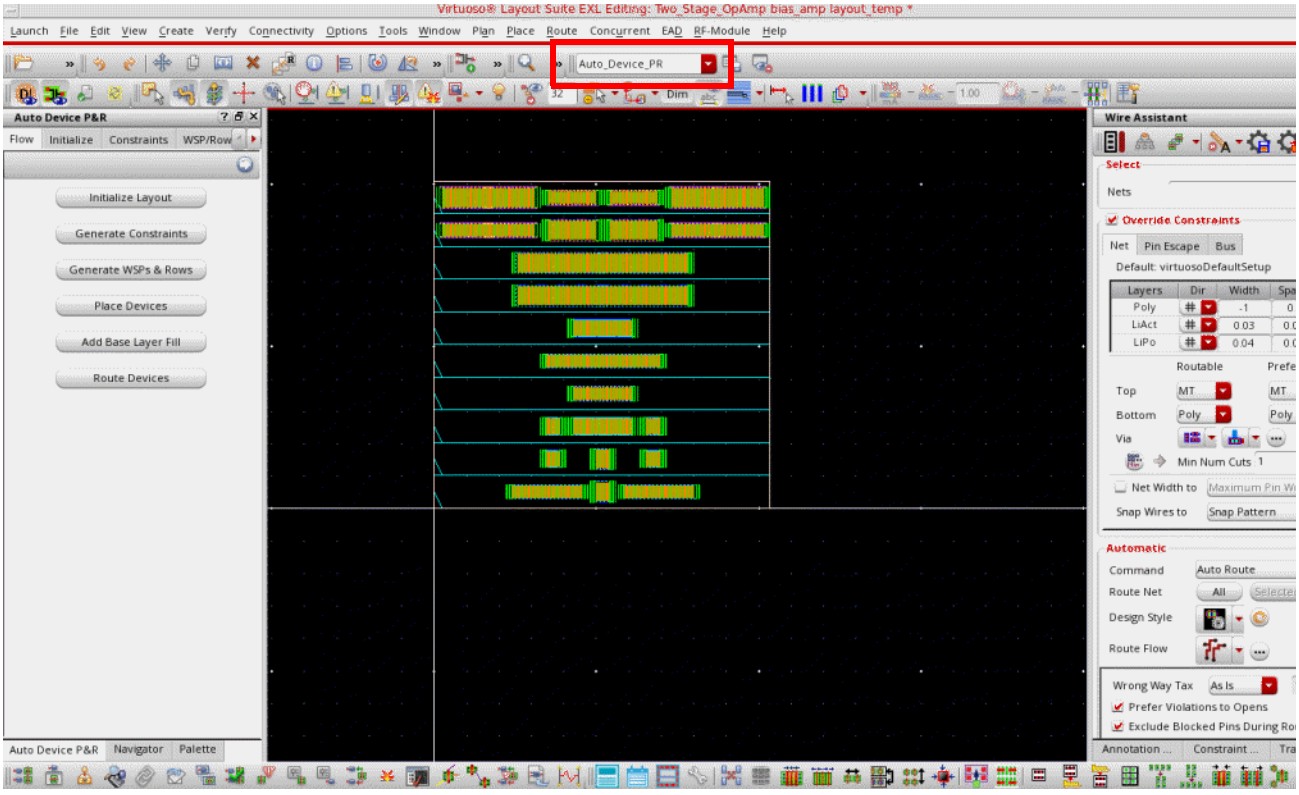
Auto Device P&R assistant is the integrated, automatic placement and routing solution available in Virtuoso. The options in this assistant let you initialize, generate, place, and route layout designs automatically, as per your requirements.


Each button loads corresponding options for that step into the assistant. For example, *Initialize Layout* is linked to the *Initialize* tab. Proceed step-wise by clicking the arrow on the top-right corner to generate an automatically placed and routed layout design.

At any point, click *Flow* to view the *Flow* tab.

The steps or commands in this flow can also be used as standalone options in the regular layout editing environment, as a mix of interactive, assisted, and automated layout creation process.

Layout EXL supports the *Auto_Device_PR* workspace, which provides an interface suitable for initializing, placing, and routing designs. The *Auto_Device_PR* workspace includes the Auto Device P&R assistant and the Wire Assistant.



## Accessing the Auto Device P&R Assistant

To display the Auto Device P&R assistant, do one of the following:

- Choose *Window – Assistants – Auto Device P&R*.
- Right-click anywhere in the layout window menu bar and choose *Assistants – Auto Device P&R*.

# 2

# Initializing and Planning the Layout

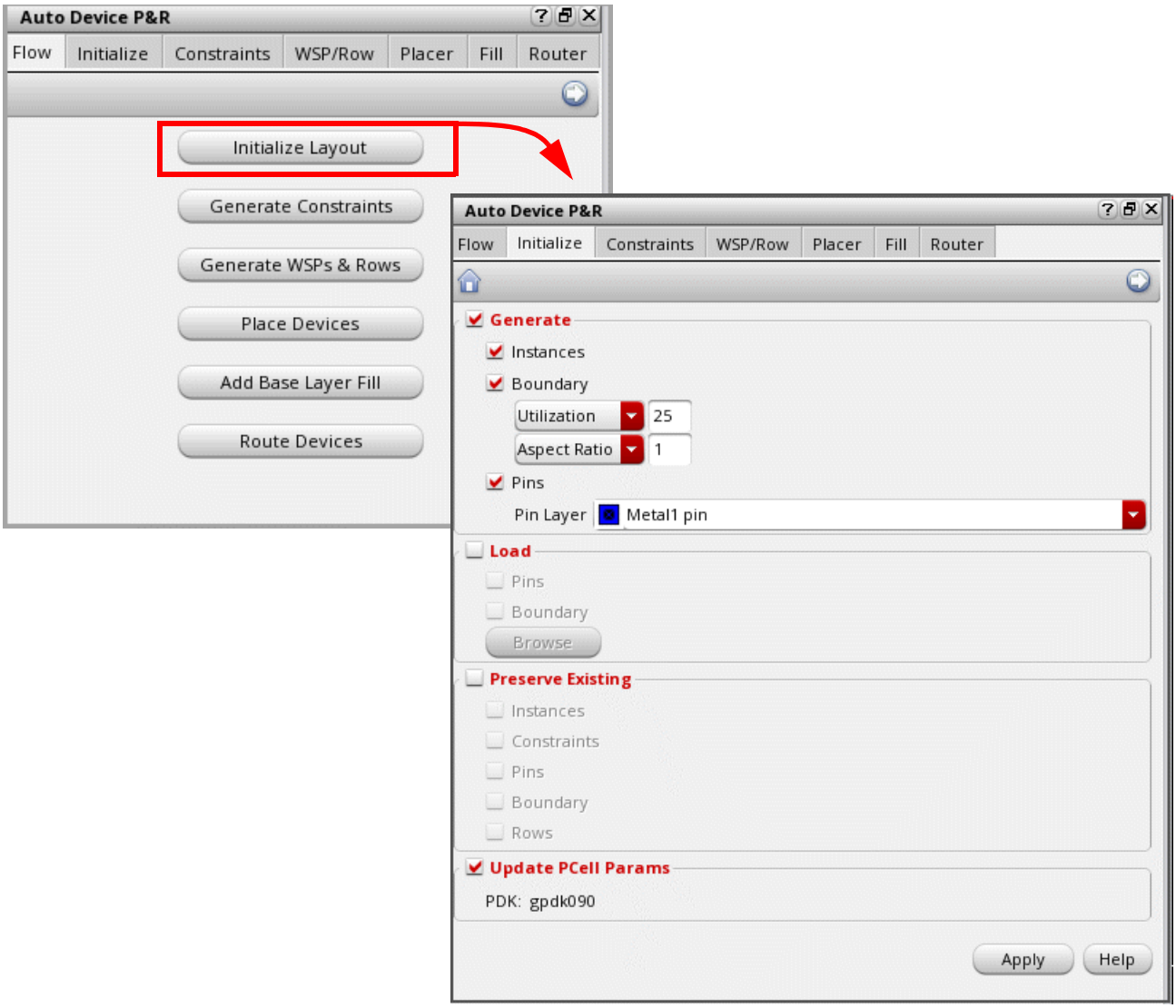
This section covers the following topics:

- Initializing a Layout
- Generating Constraints and Constraint Groups
- Deriving WSPs and Row Regions

# Initializing a Layout

The first step in the automated device placement and routing flow is to initialize your layout view. This involves the following steps.

1. Click *Initialize Layout* in the Auto Device P&R assistant to open the *Initialize* tab.

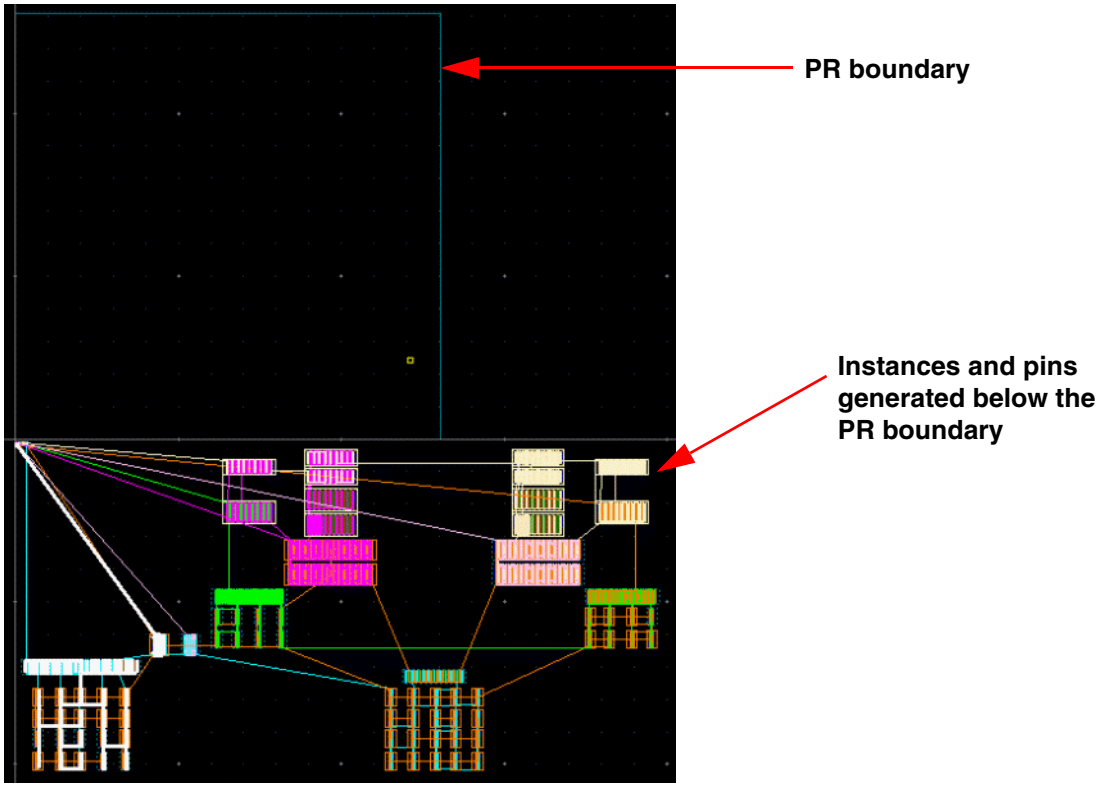



2. Select *Generate* to enable the device generation options.
3. Select *Instances* to generate all instances from the source schematic.
4. Select *Boundary* to generate a PR boundary as per the specified combination of *Utilization* and *Aspect Ratio* or *Width* and *Height*.

- Utilization and Aspect Ratio: *Utilization* specifies what percentage of the area within the PR boundary that can be filled with objects. The default is 25 percent. *Aspect Ratio* specifies the width-to-height ratio of the PR boundary. The default value is 1, which indicates a square boundary. An aspect ratio of 0.5 specifies a boundary twice as high as it is wide. A value of 2 specifies a boundary twice as wide as its height.
- Width and Height: *Width* specifies the exact width of the PR boundary. *Height* specifies the exact height of the PR boundary. *Width* can be accessed from the *Utility* drop-down list and *Height* from the *Aspect Ratio* drop-down list.

5. Select *Pins to* generate all the pins that are present on the selected *Pin Layer* in the source cellview. The default *Pin Layer* is the first metal layer in the layer stack.
6. Select *Load* to enable the options in the *Load* section.
7. Select *Pins* to load pins from another cellview. Click *Browse* to select the required cellview in the Library Browser.
8. Select *Boundary* to load the PR boundary from another cellview. Click *Browse* to select the required cellview in the Library Browser.
9. Select *Preserve Existing* to enable the options to preserve existing components.
10. Select *Instances* to preserve existing instances. Additional instances in the source schematic are incrementally generated in the target layout.
11. Select *Constraints* to preserve existing constraints when instances are not regenerated. This option is available only when *Instances* is selected.
12. Select *Pins* to preserve existing pins. Additional pins in the source schematic are incrementally generated in the target layout.
13. Select *Boundary* to preserve the existing boundary. The boundary is not regenerated.
14. Select *Row* to preserve existing rows. The rows are not regenerated.
15. Select *Update PCell Params* to update the parameters of the specified technology file. The PDK settings are loaded to prepare the Pcells to work in the placement and routing flow. Use this option to update the required layout-only Pcell parameters for specific PDKs and environment variables to facilitate placement and routing.

**16.** Click *Apply* to generate the selected objects in the layout canvas. All the instances and pins are generated below the PR boundary.




### *Environment Variable Settings*

The following environment variable settings are applied during layout initialization:

**Environment Variable Settings**

CAE is turned off:

```
envSetVal( "layoutXL" "constraintAwareEditing" 'boolean nil )
```

WSP-aware snapping is turned on:

```
envSetVal( "layoutXL.APAssist" "WSPAware" 'boolean t )
```

Modgen regeneration options are turned on:

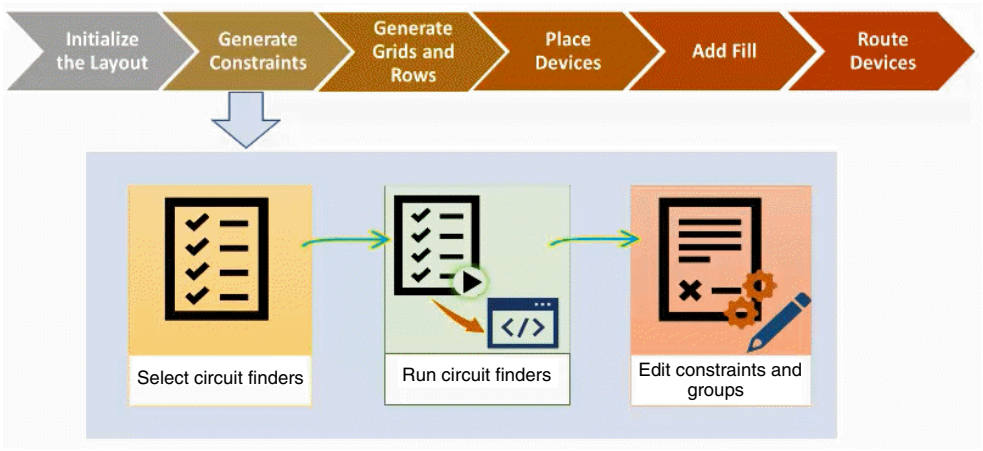
```
envSetVal( "layoutXL.AP" "regenModgenPostProcess" 'boolean t )
envSetVal( "layoutXL" "modgenRegenerateSnapToPlaceRow" 'cyclic
"WithinRowRegion" )
```

Wire Editor Snapping Mode is set to WSP Pattern:

```
envSetVal( "layout" "snapWireGrid" 'cyclic "Snap Pattern" )
```

# Generating Constraints and Constraint Groups

Device groups and constraints are central to the effective placement and routing of custom and analog layouts. After initializing the layout, the next step is to generate the required constraints. The automated device placement and routing flow supports automatic structure recognition using circuit finders, the organization of these structures into device groups, and the creation of corresponding constraints.

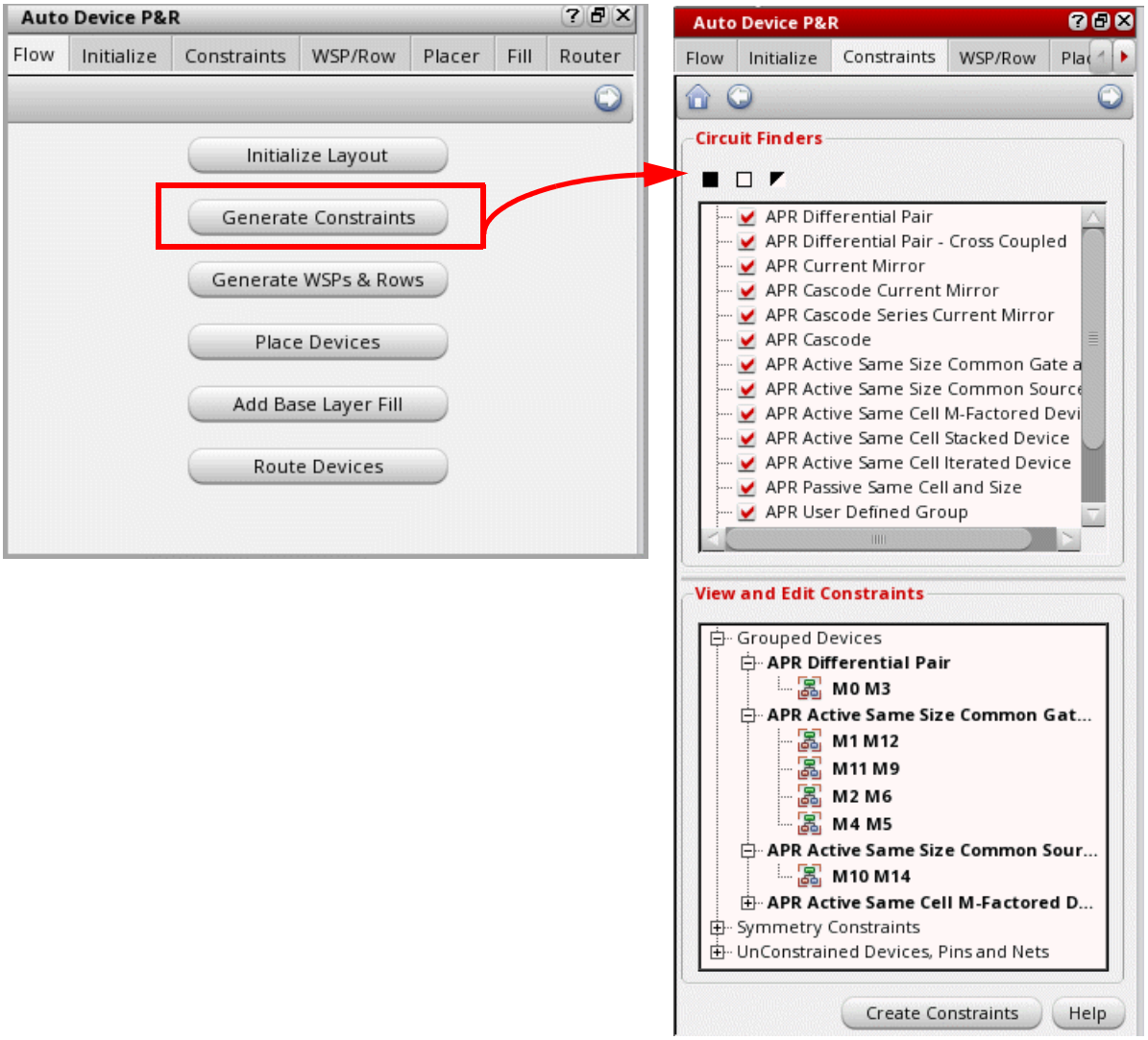



The automatic device placer and router honor these constraints during design placement and routing.

**Note:** Automatically created constraints can be overridden by creating groups and constraints manually, which are respected by the automatic tools.

To generate constraints:

1. Click *Generate Constraints* in the Auto Device P&R assistant to display the *Constraints* tab.




2. Select the *Circuit Finders* to be run to identify matching devices and device groups in the design. New topological constraints corresponding to the matching devices and device structures are generated.

   The Auto-Device P&R assistant is pre-loaded with a set of *Circuit Finders*. Each finder analyzes the source data (schematic cellview) and identifies all devices and device groups that match the given criteria. For example, the finder *Instances (Symmetry by Connectivity)* identifies all symmetric instances, nets, and pins in the source cellview and groups them into a device group. The finder names have the `APR-` prefix, which indicates that these finders are customized for the Auto-Device P&R flow. By default, all *Circuit Finders* are selected. Deselect the ones that you do not want to run.

You can use *Enable All* ■, *Disable All* □, and *Inverse All* ◤ to select the required circuit finders.

The sequence of the finders determines the sequence in which constraints are listed. You can reorder the finders by dragging them to the desired locations within the list.

**Note:** For devices to be added to device groups, they need to be registered for the Circuit Prospector assistant. For more information on device registration, see Constraint Manager Assistant Customization SKILL Commands.

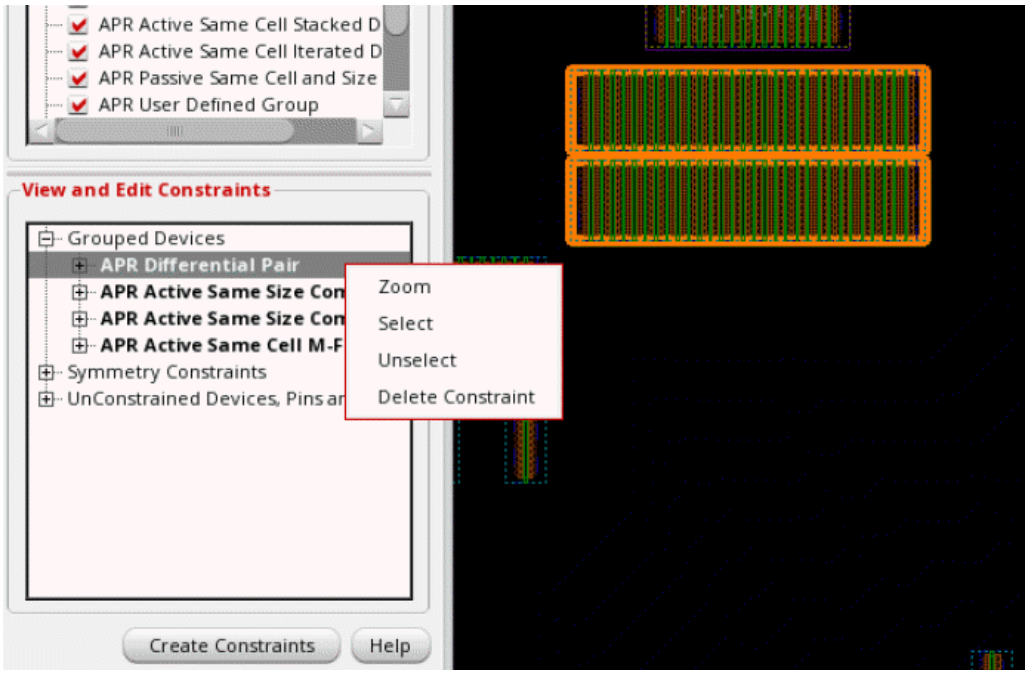
The following table describes the circuit finders.

| APR Finders | Description |
|---|---|
| *APR Differential Pair* | Groups devices that form a differential pair structure. |
| *APR Differential Pair - Cross Coupled* | Groups devices that form a cross-coupled differential pair structure. |
| *APR Current Mirror* | Groups devices that form a current mirror structure. |
| *APR Cascode Current Mirror* | Groups devices that form a cascoded current mirror structure. |
| *APR Cascode Series Current Mirror* | Groups devices that form a cascoded series current mirror structure. |
| *APR Cascode* | Groups cascoded MOS transistor structures. |
| *APR Active Same Size Common Gate and Source* | Groups same-sized devices with the same gate and source connections. |
| *APR Active Same Size Common Source* | Groups same-sized devices with the same source connection. |
| *APR Active Same Cell M-Factored Device* | Groups active devices with mfactor. |
| *APR Active Same Cell Stacked Device* | Groups transistors in series stacks. |
| *APR Active Same Cell Iterated Device* | Groups iterated devices. |
| *APR Active Same Cell and Size* | Groups active devices with same cell name and size. |
| *APR Passive Same Cell and Size* | Groups passive devices with same cell names or the same cell sizes and values. |

| | |
|---|---|
| *APR User Defined Group* | Groups device groups inside text boxes. |
| *APR Instances (Symmetry By Connectivity)* | Groups pairs of instances that are determined to be symmetric based on their connectivity. |
| *APR Instances (Symmetry By Connectivity with common source or drain)* | Groups pairs of instances that are determined to be symmetric based on their connectivity with a common source or drain. |
| *APR Nets (Symmetry By Connectivity)* | Groups nets that are determined to be symmetric based on their connectivity. |
| *APR Pins (Symmetry By Connectivity)* | Groups pins that are determined to be symmetric based on their connectivity. |

For more information about circuit finders, see Finders in the *Virtuoso Unified Custom Constraints User Guide*.

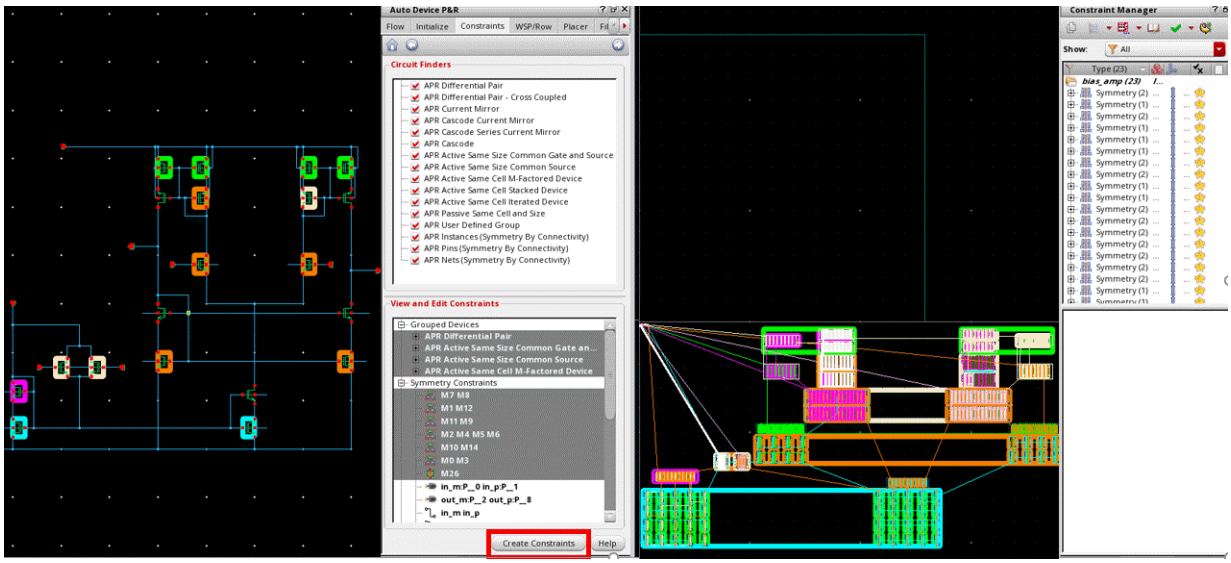
3. The *View and Edit Constraints* section lists all existing and new constraints and constraints groups based on the recognized structures. Click the + sign to expand each group. Right-click a constraint in this section to display a shortcut menu.



Use the options in the shortcut menu to perform the following tasks:

- *Zoom*: To magnify the selected constraints.
- *Select*: To select the constraints to be generated.
- *Unselect*: To unselect the constraints that you do not want to generate.
- *Delete Constraint*: To delete the constraint from the layout cellview.

4. Click *Create Constraints* to search for all structures that have been enabled in the Circuit Finders pane following the order in which they are listed. All identified structures are organized in groups or Modgen constraints, along with symmetry information for symmetric structures, and are listed in the Constraint Manager.




**Automatic Structure Recognition and Constraint Creation through the Constraints Tab**

**Note:** Structures with members in an already-found group are skipped.

Expand the following categories in the View and Edit Constraints pane to view the corresponding entries:

- *Grouped Devices*: Lists structures for which groups have been created. The selected groups are highlighted in the layout and schematic windows.
- *Symmetry Constraints*: Lists the instances, nets, and pins on which symmetry constraints have been created.
- *Unconstrained Devices, Nets and Pins*: Lists all unconstrained devices in the layout. You can select the required devices and create groups or symmetry constraints on them by using the options in the shortcut menu.

### Related SKILL Functions

You can use the following SKILL functions to instantiate the finders:

- ciActiveSameCellAndSizeIterator: Iterates over all the active same cell and size structures in the cellview and returns a list of corresponding devices.
- ciCascodeSeriesCurrentMirrorIterator: Iterates over all cascode series current mirror structures in the cellview and returns a list of corresponding devices.
- ciCommonGateAndSourceIterator: Iterates over all common gate and source structures in the cellview and returns a list of corresponding devices.
- ciCommonSourceIterator: Iterates over all common source structures in the cellview and returns a list of corresponding devices.
- ciHierarchicalSeriesIterator: Iterates over all the series structures in the cellview and returns a list of corresponding devices.
- ciCommonGateIterator: Iterates over all common gate structures in the cellview and returns a list of corresponding devices.

## Creating a Modgen

Instead of creating standard constraints, the automated device placement and routing flow lets you create Modgens for the device groups identified by the circuit finders.

To create Modgens in the Auto Device P&R assistant:

1. Set the aprCreateModgens environment variable to `t` as follows:

   ```
   envSetVal("layoutXL.AP" "aprCreateModgens" 'boolean t)
   ```

2. Open the *Constraints* tab of the Auto Device P&R assistant.
3. Select the circuit finders that you want to run to identify device groups in the design.

   For more information, see Generating Constraints and Constraint Groups.
4. Click *Create Constraints*.

The newly created Modgens are listed in the *View and Edit Constraints* section under the various categories.

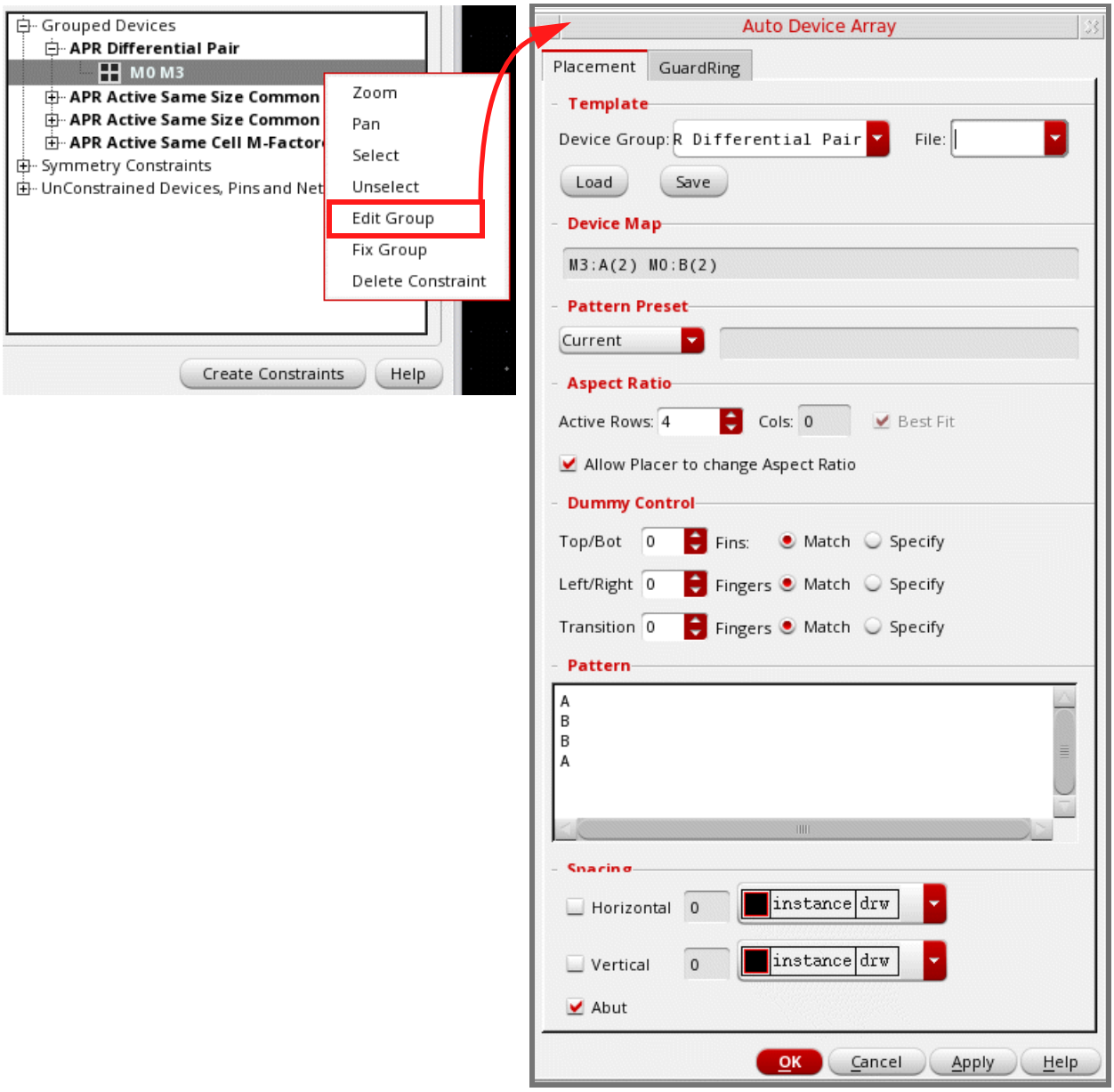
**Note:** When you create constraints on a design, existing Modgens are not automatically deleted. To delete them, right-click the required Modgen on the *Constraints* tab of the Auto Device P&R assistant and select *Delete Constraint*.

## Editing a Modgen

To edit a Modgen:

1. Expand the required category in the *View and Edit Constraints* section on the *Constraints* tab of the Auto Device P&R assistant.
2. Right-click the required Modgen.
3. Choose *Edit Group*.

The Auto Device Array form is displayed.




The Auto Device Array is a unified interface that allows you to quickly create and edit Modgens. The form integrates the key options from the Modgen Placement and Modgen Routing toolbars. For more information, see Generating Modgens Automatically.

# Deriving WSPs and Row Regions

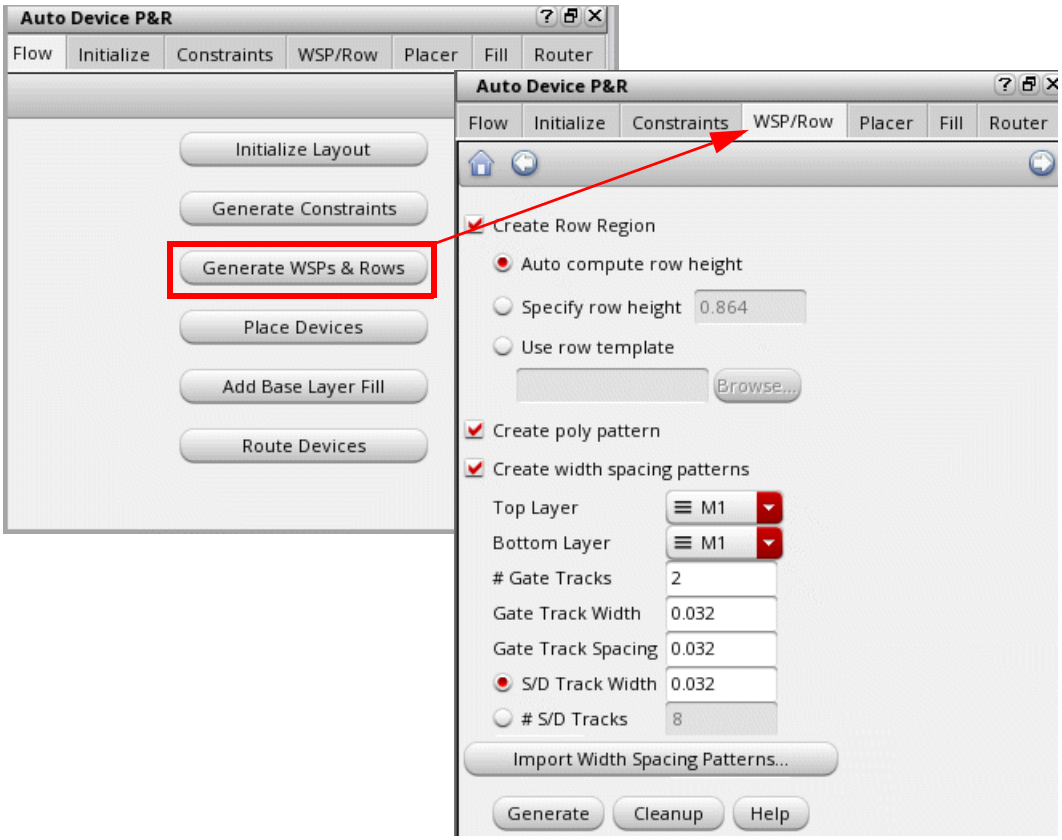
Generating grids is an optional task, but is highly recommended to obtain better results. At advanced nodes, grids—including fins, WSPs, poly snap patterns, and rows—are important for the optimum placement and routing of custom and analog layouts. While row regions are used for device placement, WSP information is used for device routing and snapping.

The automated device placement and routing flow lets you derive WSPs and row regions automatically based on the device footprint, layers, and DRCs, with minimal user input. A row template is created automatically for all devices types, with the WSP period determining the row height. Row regions are created in the layout based on the row templates.

Before generating rows and grids, ensure that your designs has a uniform gate length to ensure a uniform poly pitch for device snapping.

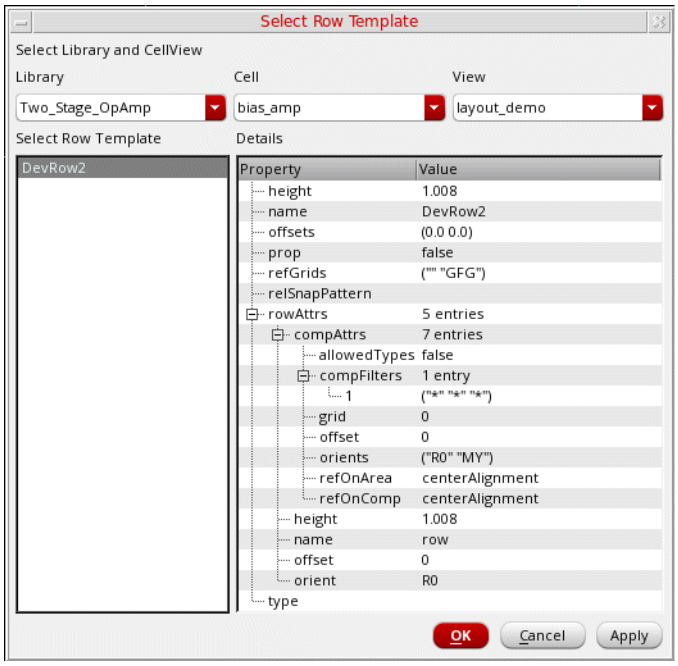
To derive WSPs and row regions:

1. Click *Generate WSPs & Rows* in the Auto Device P&R assistant to open the *WSP/Row* tab.




2. Select *Create Row Region* and specify the row height as follows:

- *Auto compute row height*: Automatically calculates the row height based on the maximum instance height and the heights of the gate, source, and drain tracks.
- *Specify row height*: Lets you specify a row height value.
- *Use row template*: Lets you select an existing row template. Click *Browse* to display the Select Row Template form.




**a.** Select the required *Library*, *Cell*, and *View*. All templates stored in the selected cellview are listed in the *Select Row Template* box.

**b.** Select the required row template.

**c.** Check the row template properties in the *Details* panel to ensure that they meet your requirements.

**d.** Click *OK*.

The name of the selected row template is displayed in the *Use row template* field.

The WSP period determines the height of rows in the auto-generated row template. All horizontal tracks have the same period.

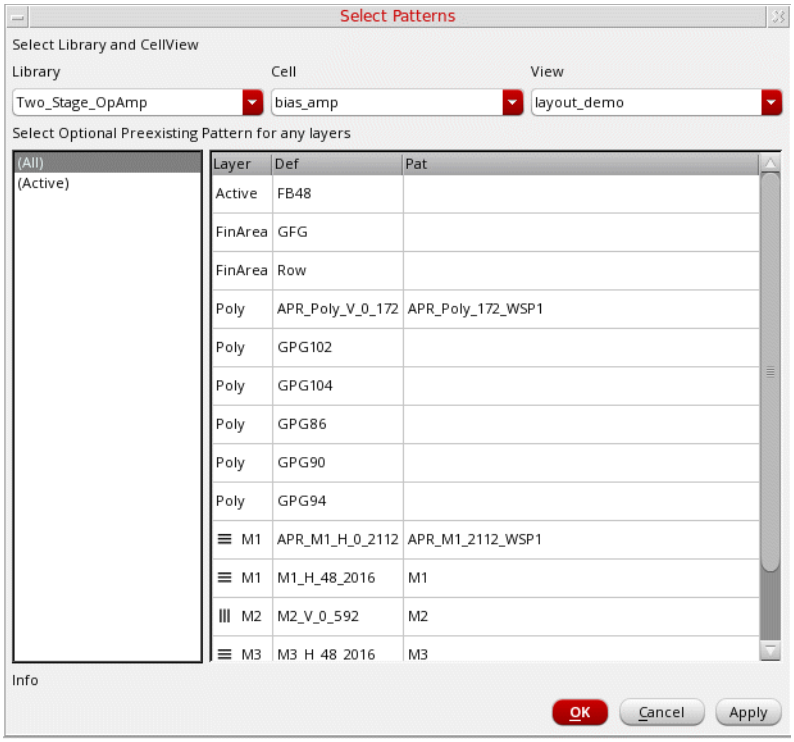
**3.** Select *Create Poly Pattern* to generate a poly WSP grid in the selected row region.

**4.** Select *Create Width Spacing Patterns* to generate WSPs in the selected row region and specify the following related details:

- *Top Layer* and *Bottom Layer*: Specify the routing layer range for which WSPs must be created. WSP tracks are inserted in these and all intermediate layers.
- *#Gate Tracks*: Specifies the number of gate tracks to be accommodated within a WSP track period. The default is `2`. The specified number of gate tracks impacts the WSP period. Gate tracks are required for poly gate routing.
- *Gate Track Width*: Specifies the number of gate tracks to be added at the top and bottom of each row. The default value is `2`, which specifies that two gate tracks are to be added to the top and bottom of each row.
- *Gate Track Spacing*: Specifies the spacing between gate tracks. Each WSP track must accommodate the given number of gate tracks of the specified width and spacing. If the specified track width is higher than the default, specify a wider spacing value.
- *S/D Track Width* and *#S/D Tracks:* Specifies the width and the number of source and drain tracks to be accommodated in each WSP track period. Select either *S/D Track Width* or *#S/D Tracks* and specify a value. The default value is automatically calculated based on the layer constraints and area of the S/D pins.

**Note:** The gate and source/drain parameters specified in this form override the default gate and track width, spacing, and count values.

5. Instead of specifying the WSP parameters, you can import a WSP pattern from an existing WSP pattern file. Click *Import Width Spacing Patterns* to open the *Select Patterns* form, which lets you choose the required WSP Pattern file.




Specify the following WSP parameters in the *Select Patterns* form:

   a. Select the required *Library*, *Cell*, and *View*. All WSP patterns stored in the selected cellview are listed in the following section.

   b. Select the required WSP pattern in the left panel. The *Layer*, definition (*Def*), and pattern (*Pat*) for the selected WSP are displayed in the right panel.

   c. Click *OK* to select the required pattern and close the form.

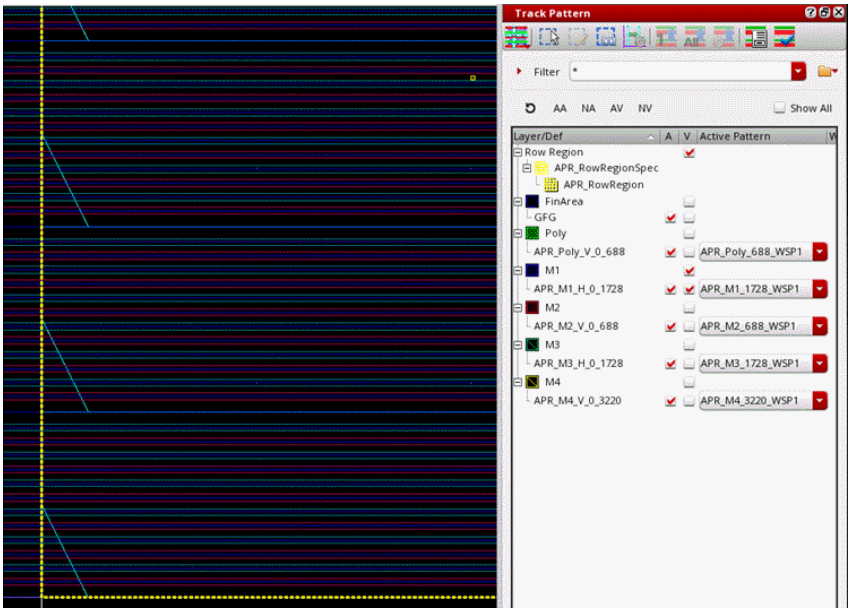
   WSP parameters from the selected WSP pattern file are loaded into the related fields on the *WSP/Row* tab.

   **Note:** In case of discrepancies between the WSP settings and row region settings, for example, a difference in the offset values, the row region settings are honored.

6. Click *Generate*.

   WSP grids are generated in the layout canvas as per your specifications.

The Track Pattern assistant lists all the auto-created WSPs and rows with the APR- prefix to differentiate them from existing WSPs.




**Note:** Existing WSPs that have no change in their specifications are not regenerated. The last set of auto-generated WSPs are made active.

# 3

# Placing and Routing the Design

This chapter covers the following topics:

# Placing Devices

After initializing the layout and generating the required constraints and tracks, the next task is to place the devices and pins within the PR boundary. The Virtuoso automated device placement and routing flow supports two placement models:

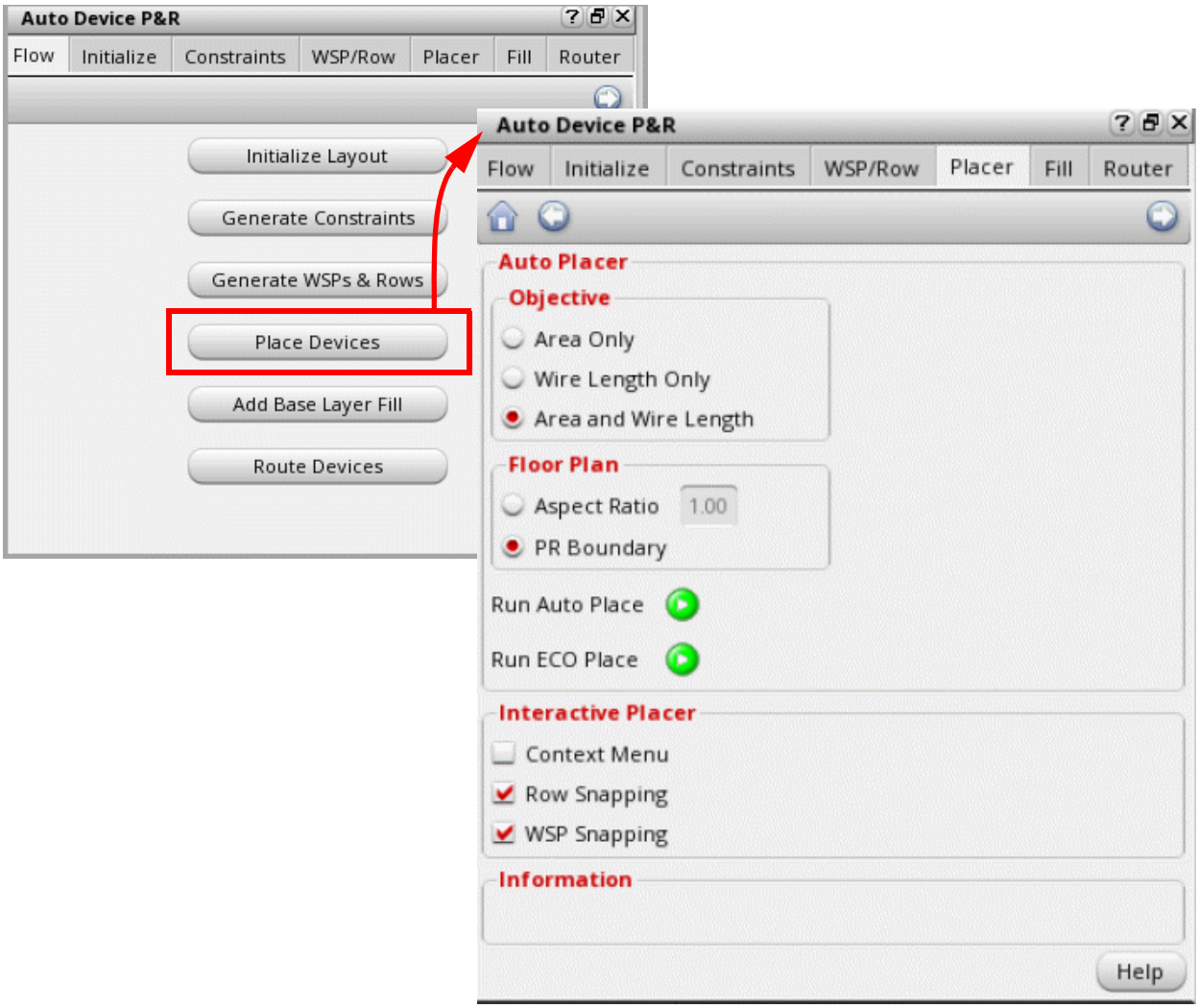
- Automatic Placement: Placement is performed by running the Virtuoso device-level automatic placer.
- Interactive Placement: Devices are placed semi-automatically. Depending on the placement needs and the complexity of the design, you can first run the Virtuoso device-level automatic placer, and then use the interactive placement options to refine the placement.

## Automatic Placement

The Virtuoso device-level placer analyzes the constraints from various sources, such as Modgens, Constraint Manager, figGroups, and circuit finder, and runs the global placer to achieve the required placement results without any design rule violations.

To run the Virtuoso device-level automatic placer:

1. Click *Place Devices* on the *Flow* tab of the Auto Device P&R assistant to open the *Placer* tab.




2. Select one of the following placement objectives:

   - *Area Only:* Place devices to optimize compactness.

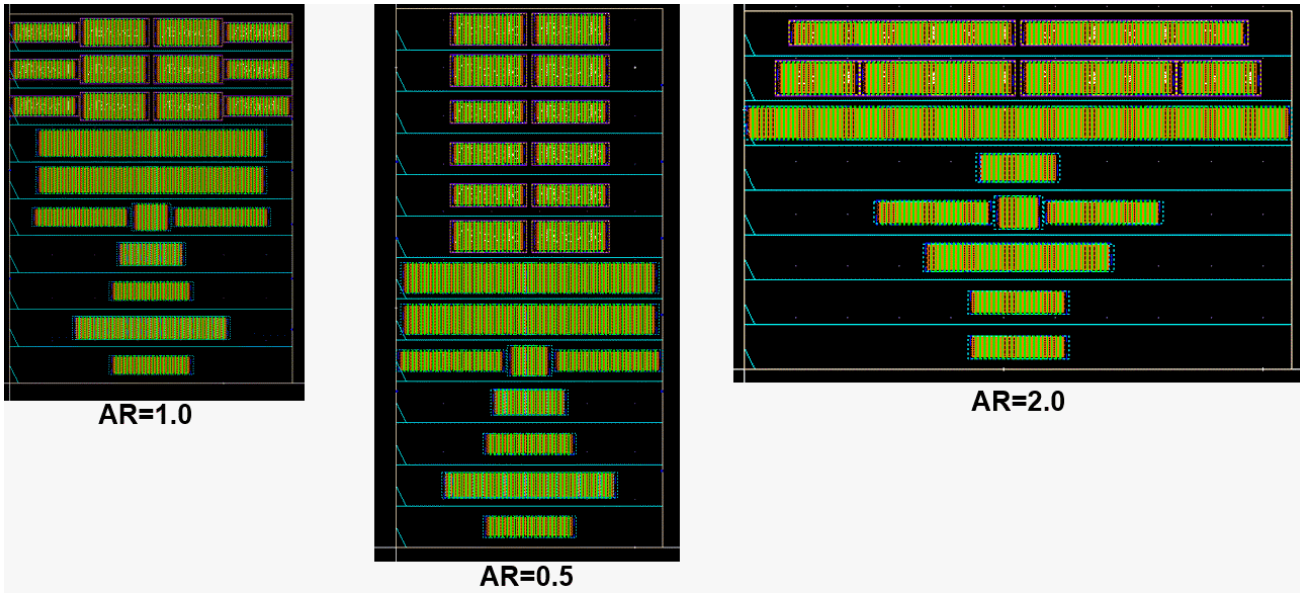
   - *Wire Length Only:* Minimizes the wire length for better routability.

   - *Area and Wire Length:* Achieve a balance between compact placement and reduced wire length.

3. Select one of the following options to specify the placement region:

   - *Aspect Ratio*: Generates a placement boundary as per the specified dimensions. The default value is 1.00, which specifies a square boundary. An aspect ratio of .50 specifies a boundary twice as high as it is wide. A value of 2.00 specifies a boundary twice as wide

as it is high. Multiple placement results can be obtained for different aspect ratios. Examples:




- *PR Boundary*: Fits all the objects inside the existing PR boundary. The placer does not adjust the boundary.

**Note:** In the *Aspect Ratio* mode, you can set autoPlaceAdjustBoundary to `t` to let the placer adjust the PR boundary to enclose all the components in the design. You can also set autoPlaceAdjustRowRegion to `t` to let the placer adjust the row regions during placement.

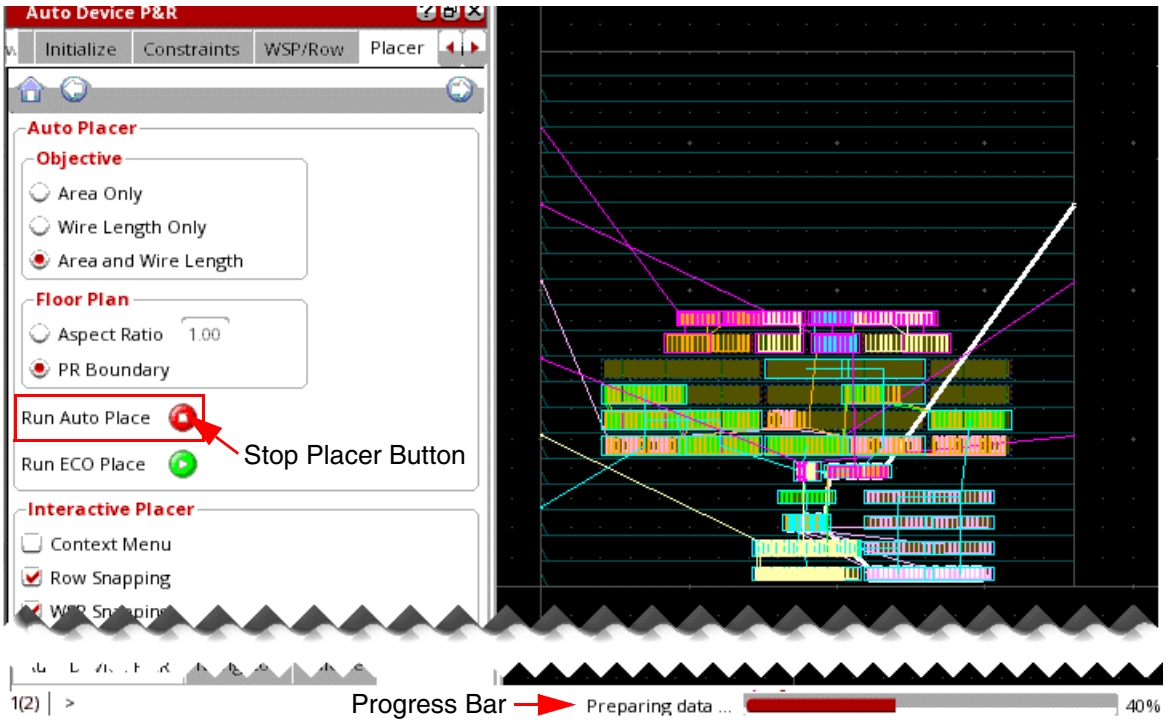
**Note:** To enable the placer to follow the aspect ratio specified in the initialization step, set *PR Boundary* as the placement region. Otherwise, the placer follows the aspect ratio specified in the placement options.

**Note:** To lock the aspect ratio of all Modgens while running the placer, set autoPlaceLockFGAR to `t`. The aspect ratio-related settings defined in the Auto Device Array form are ignored. When set to `nil` (default), the aspect ratio settings in the Auto Device Array form are honored while running the placer.

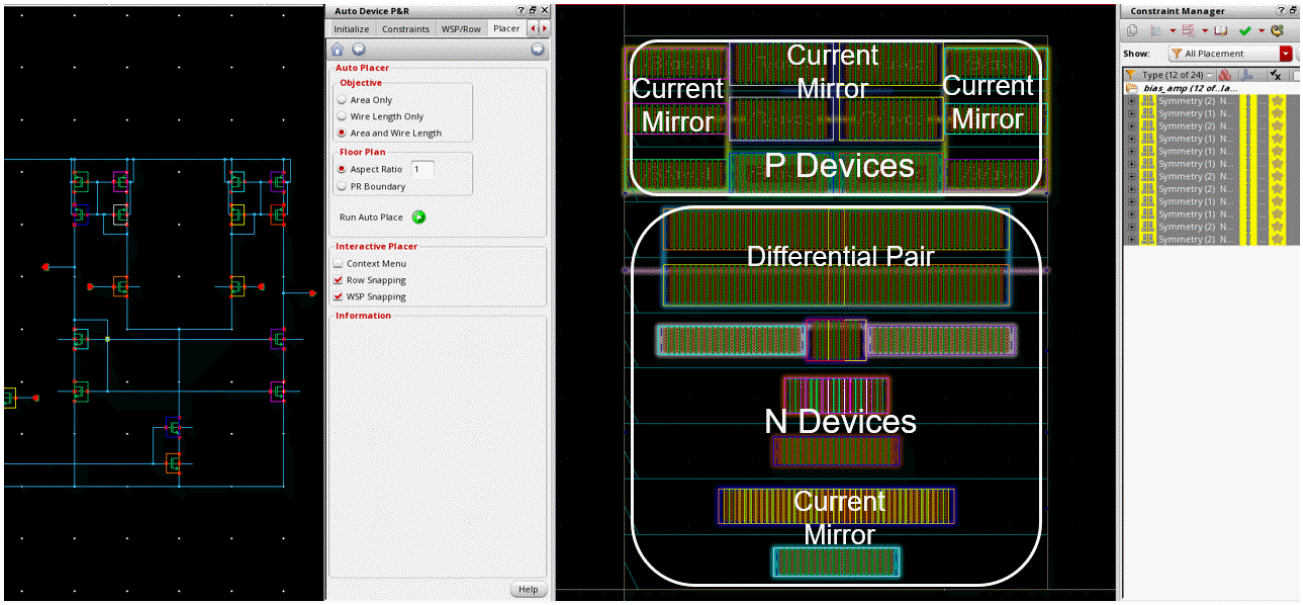
4. Click *Run Auto Place* to run the placer.

   Click *Run ECO Place* to run the placer to incrementally place the non-optimally placed devices, such as the unplaced, overlapping, non-grid compliant, and non-row compliant devices. These devices are placed based on their connectivity with the placed components, while honoring all constraints and grids. The incremental placer runs without changing the aspect ratio of the figGroups and Modgens. The incremental place run results depend on the starting layout.

When you run the placer, a progress bar that indicates the placement status is displayed at the bottom-right corner of the design canvas. The run placer button changes to the stop placer button, which lets you stop the placer and revert the design placement to its initial state.



The following image depicts the automatic row-based, constraint-compliant placement of devices and groups achieved after running the Virtuoso device-level automatic placer.



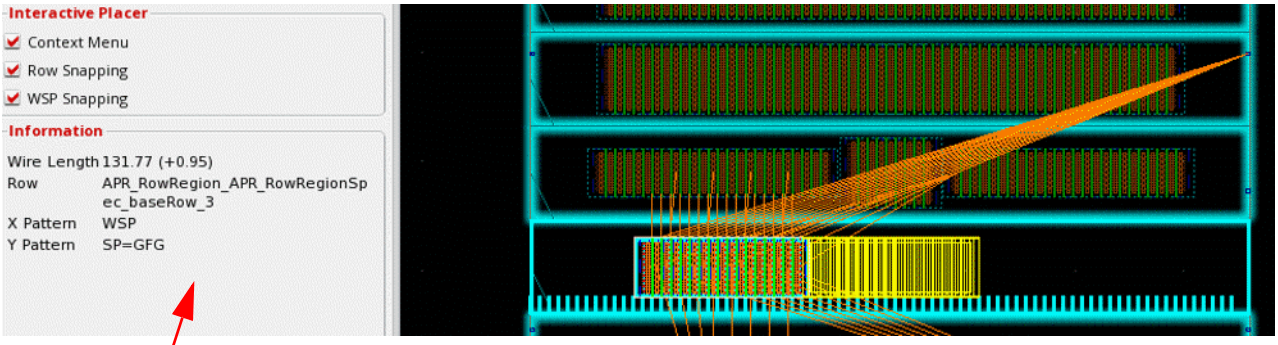

- Snapped to rows
- Snapped to WSPs (Poly)
- Optimized for area & wire length

- Symmetric placement of instances
- Symmetric placement of pins

## Interactive Placement

To run the device-level interactive placer:

1. Select *Context Menu* to display a shortcut menu that lets you perform context-sensitive operations on devices and device groups directly in the layout canvas. For more information, see Interactive Placement Using the Context Menu.
2. Select *Row Snapping* to display row snapping-related details in the Information pane during interactive placement of devices.
3. Select *WSP Snapping* to display SP and WSP snapping-related details in the Information pane during interactive placement of devices.

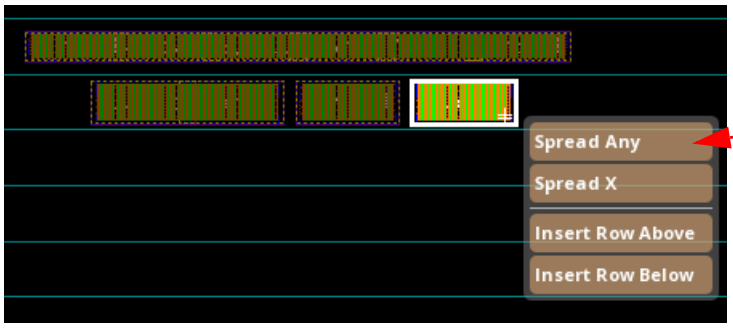



*Information* section displaying information about row snapping and SP and WSP snapping

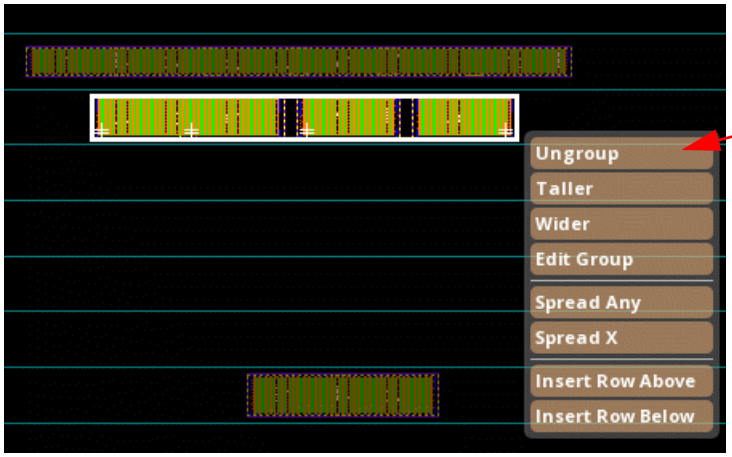
### Interactive Placement Using the Context Menu

Select *Context Menu* in the *Interactive Placer* section to display a shortcut menu when you select devices and device groups in the layout canvas.

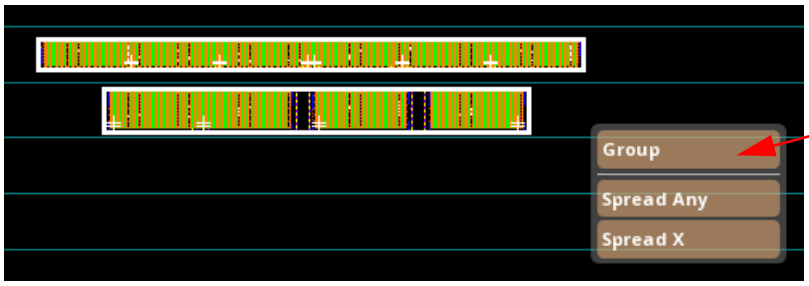
The following images show the context-sensitive menus for devices and device groups.




Context-sensitive menu for devices




Context-sensitive menu for a device group

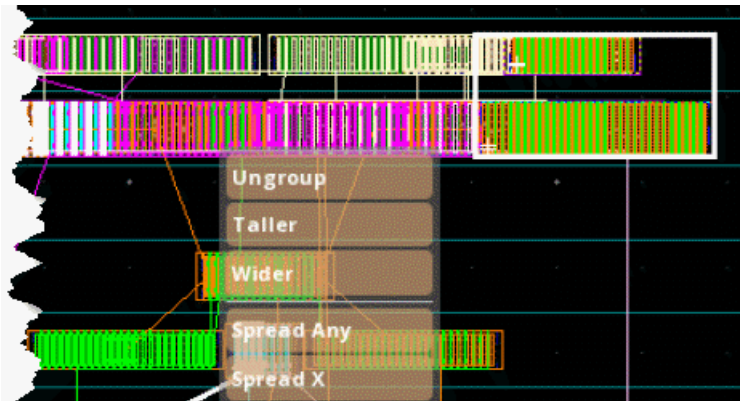



Context-sensitive menu for two or more device groups

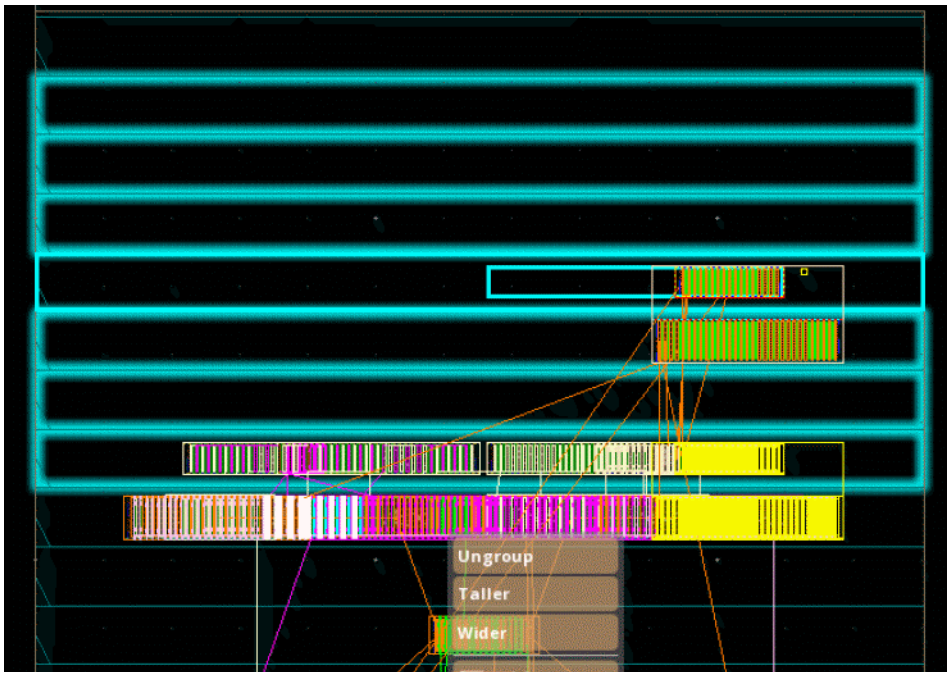
The context menu lists the following interactive placement options:

- **Group and Ungroup Devices:** Lets you group and ungroup the selected devices. The relative positions of the grouped devices are always maintained. Grouped devices can be in the same or different rows. Devices in a group are abutted horizontally.

The following image shows the vertical grouping of devices across two different rows.



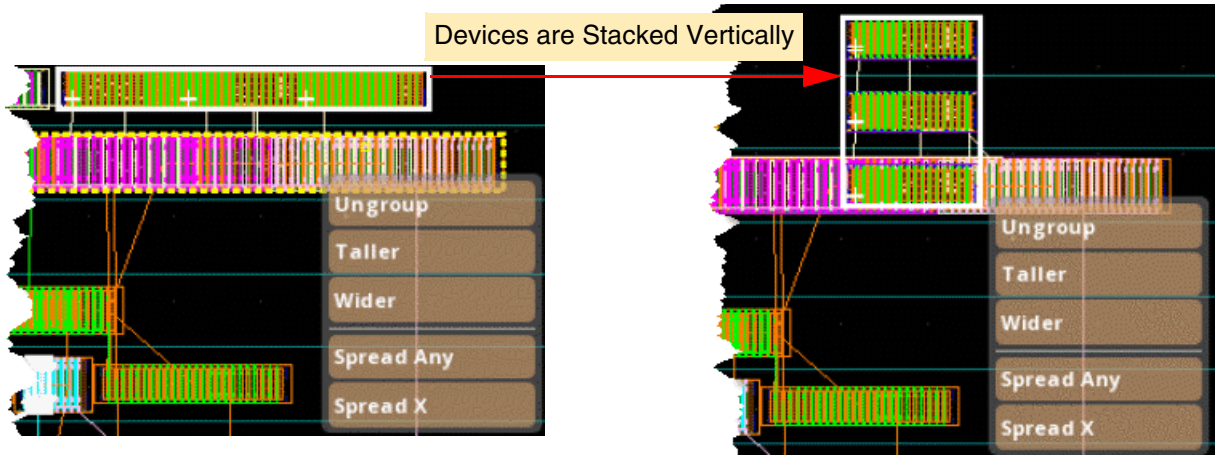

In the following image, a device is moved to a new location. The other devices in the group are also moved to new locations to maintain the relative positions of the grouped devices.




You can merge two or more device groups by selecting the groups and then selecting *Group* from the context menu.

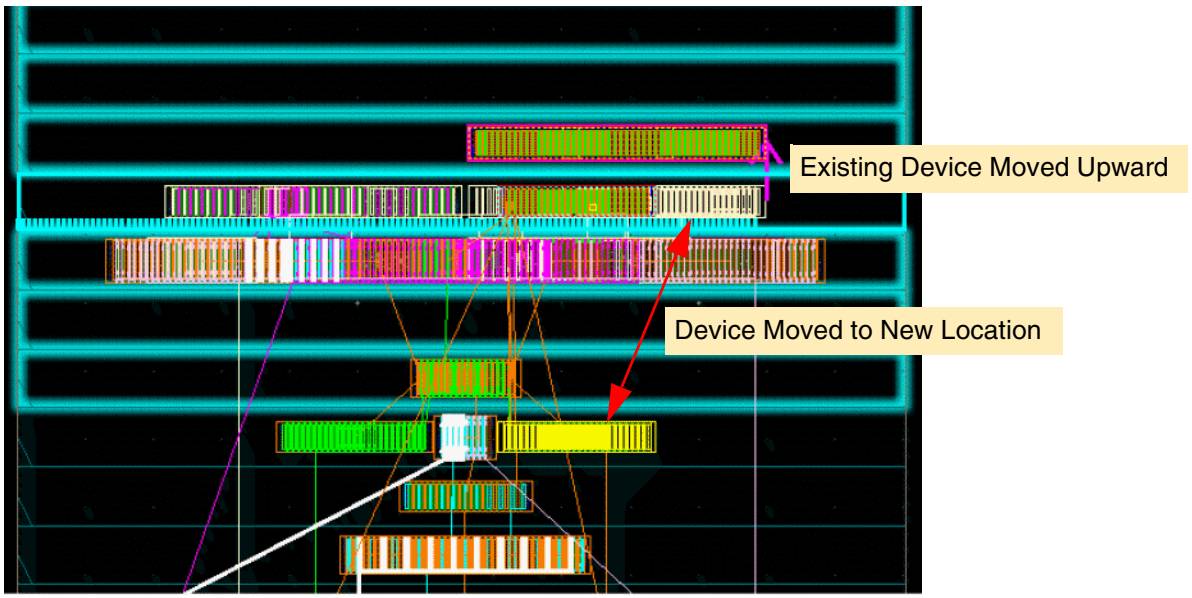
- **Taller:** Stacks the selected devices vertically. This option is applicable only for grouped devices.

The following example shows the effect of choosing the *Taller* setting for a horizontal group of devices.

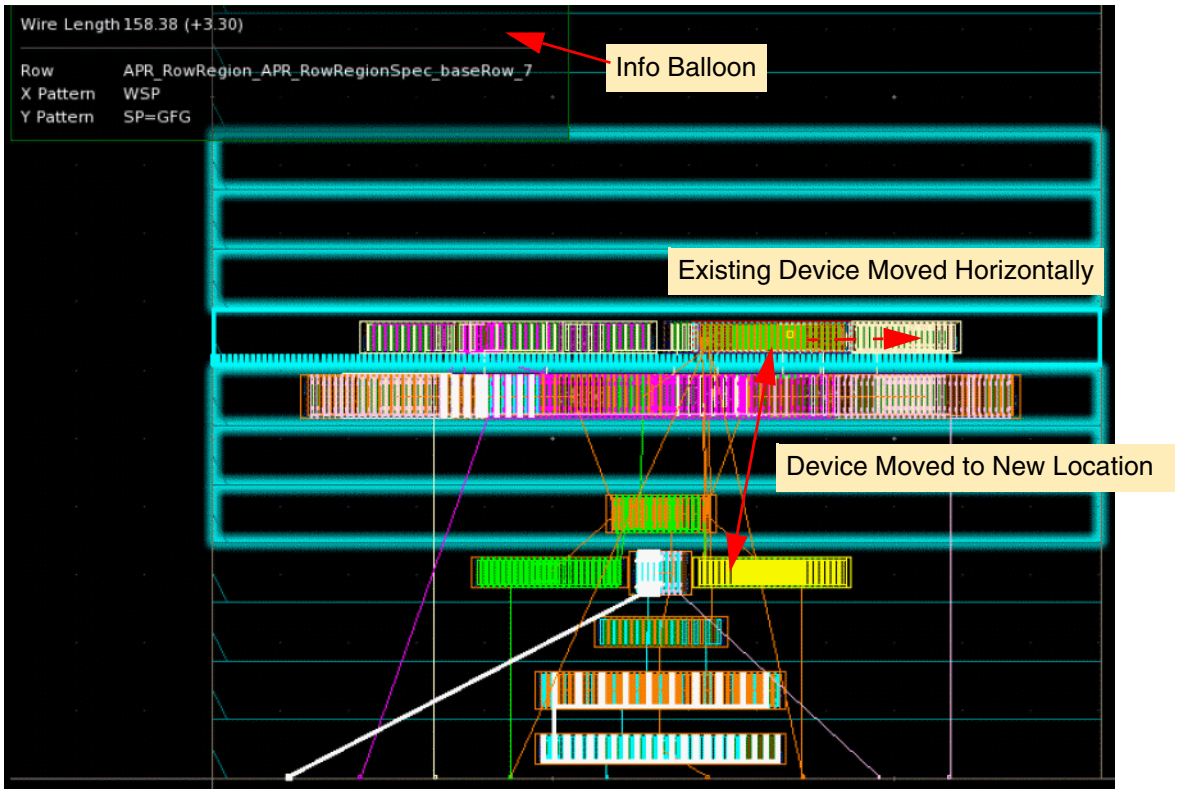



When you move a stacked device, the associated grouped devices are also moved to new locations so that their relative positions are maintained.

- **Wider:** Arranges the selected devices horizontally. This option is applicable only for grouped devices. Therefore, when you move a grouped device, the relative positions of the other devices in the group are also updated.
- **Spread Any:** Moves existing devices in any direction to avoid overlaps. This option is applicable when you move a device to a position where there is another (overlapping) device. The device that you moved is placed at the new location, while the existing device is moved to avoid overlaps. An arrow indicating the direction of movement is displayed.



- **Spread X:** Moves existing devices in the X-direction to avoid overlaps. This option is applicable when you move a device to a position where there is another (overlapping) device. The device that you moved is placed at the new location, while the existing device is moved to avoid overlaps.




An info balloon that summarizes the following information is displayed:

- Total wire length

- Change in wire length

- Row to which the device has snapped

- Snap pattern to which the device has snapped

- **Edit Group:** Displays the Auto Device Array form. Use the options in the form to create and edit Modgens. For more information about the form, see Generating Modgens Automatically.

- **Insert Row Above:** Inserts a blank row above the selection.

- **Insert Row Below:** Inserts a blank row below the selection.

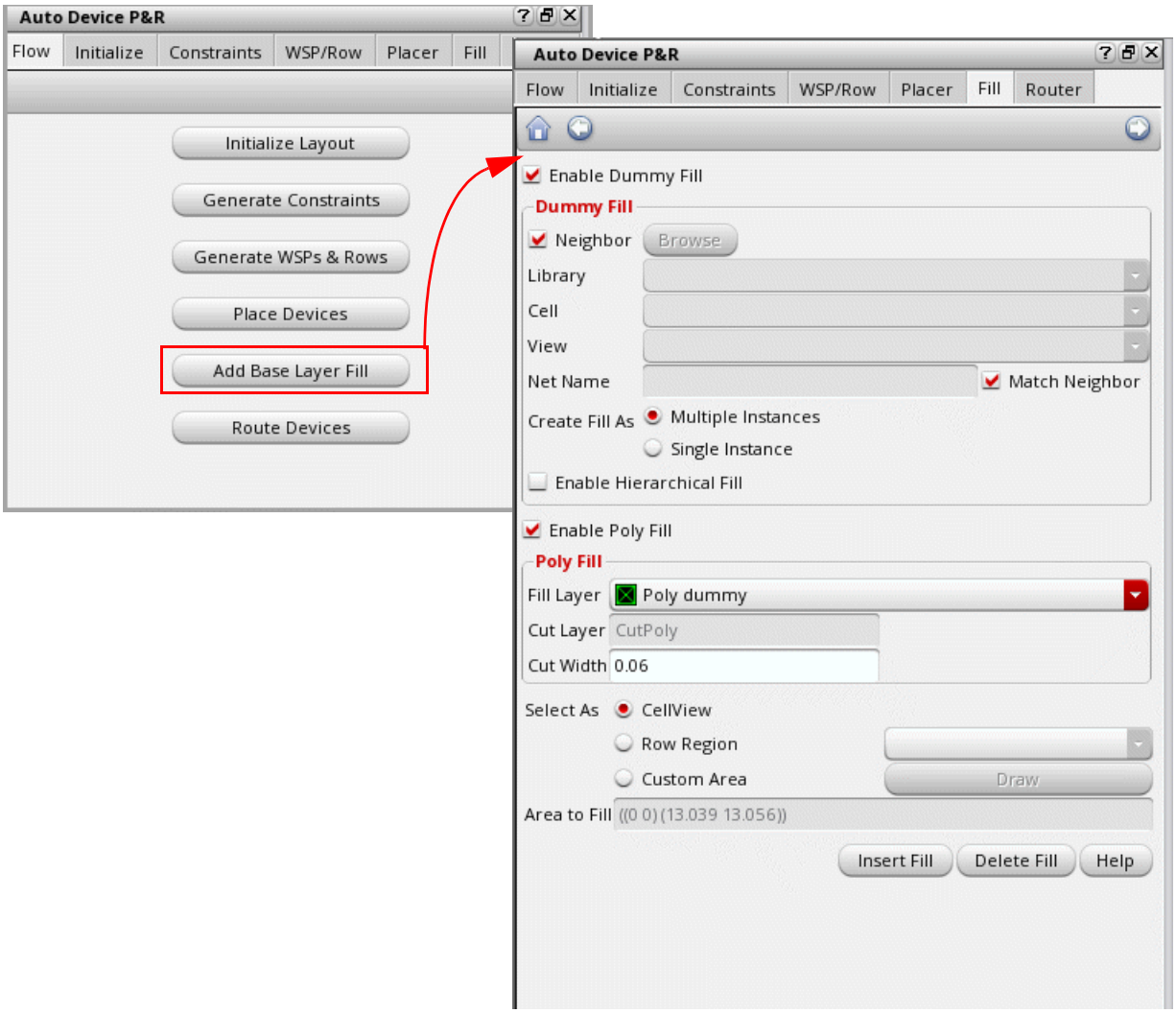
**Note:** An extended outline around grouped devices indicates that the number of open connections have exceeded the given certain threshold, which have resulted in unverified

connectivity areas. These open connections are visible in the Annotation Browser. To address this issue, either re-extract the design or extract the unverified area in the Annotation Browser.

# Adding Base Layer Fill

After device placement, there might be some gaps in the layout design. The unfilled area might also result from the *Utilization* setting on the *Initialize* tab. Running a DRC check on the design at this point could report several spacing violations. To avoid these violations, you must generate base layer fill in the unfilled area, between devices. Base layer fill, which includes device, poly, and cut poly fill, are critical for density (manufacturing and matching) reasons at advanced nodes. The automated device placement and routing flow provides a unified interface for adding all types of base layer fill. To generate device fill:

1. Click *Add Base Layer Fill* in the Auto Device P&R assistant to open the *Fill* tab.




2. Ensure that *Enable Dummy Fill* is selected to add dummy fill.
3. Specify the required options in the *Dummy Fill* section.

**a.** Select one of the following options to specify the instance to be considered as the master for creating dummy fill.

**b.** Select *Neighbor* to create dummies that match the neighboring active devices. This option is selected by default.

**c.** Specify the *Library*, *Cell*, and *View* of the master cellview when *Neighbor* is not selected.

**d.** Specify the *Net Name* to which the dummy fill must be connected.

**e.** Select *Match Neighbor* to match the connectivity of the dummy devices to the bulk net of the neighbor. The bulk net of the neighbor is assigned to the gate, source, and drain terminals of the dummy devices, unless the source and drain terminals overlap (abut) the active device terminals on a different net. *Match Neighbor* is selected by default. When unchecked and *Net Name* is blank, the dummy device terminals are not assigned any net unless they overlap (abut) an active device terminal to inherit its net.

**f.** Select the required option from the *Create Fill As* section:

**g.** *Multiple Instance* mode inserts multiple dummy fill as per the setting of the dummyFillNeighborMultipleMode environment variable. When set to `singleFinger`, single-fingered multiple dummy instances are created. When set to `exactCopy`, dummies are created as exact copies of their neighboring instances.

**h.** *Single Instance* mode inserts each dummy fill as a single instance with multiple fingers to fill the gaps and empty rows.

**i.** Select *Enable Hierarchical Fill* to look through the hierarchy to identify gaps that need to be filled.

4. Ensure that *Enable Poly Fill* is selected to insert poly fill.

5. Specify the required options in the *Poly Fill* section. Poly fill extend the gate poly and add cut poly rails as per the DRC rules.

   **Note:** As part of poly fill, the tool supports poly extensions, cut poly rails, and local interconnect layers, along with rails on the associated cut layer.

   **Note:** Poly fill does not fill diffusion areas and areas with placement blockages.

*Important*

It is recommended that you do not place the same type of devices with different bulk nets in the same row. If present, extra space is left while inserting dummy fill to avoid shorts.

To insert poly fill and cut poly:

**a.** Select the *Poly Layer* to derive cut-poly rails. The default value is the first layer-purpose pair that has its layer function set to `Poly`.

**b.** Specify the maximum *Poly Fill Width* by which the fill can be extended. The default value is set to the minimum width value for the cut layer as defined in the technology file.

**c.** Specify the *Cut-Poly Layer Width*. The default value is `0.0u`.

**6.** Specify the area to be filled using one of the following options:

- **CellView** (default): Inserts device fill in all row regions in the current cellview.
- **Row Region**: Inserts device fill only in the row region you select from the drop-down list.
- **Custom Area**: Inserts device fill in the custom area drawn. You can either use *Area to Fill* to specify the area coordinates or click *Draw* to draw the required area in the canvas.

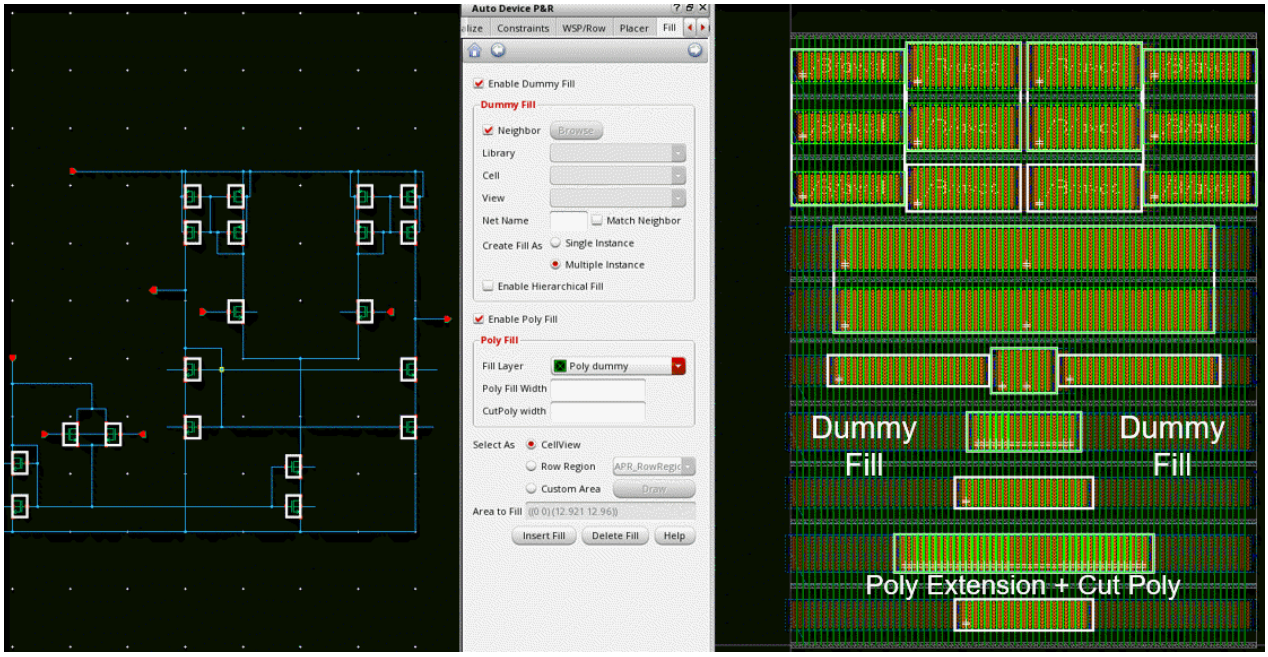
  **Note:** Device fill is not inserted in areas outside row regions.

**7.** Click *Insert Fill* to generate the required dummy and poly fill as per your specifications.

The following image depicts a placed design for which automatic DRC-compliant base layer fill (Dummy and Poly) have been added.

- The dummy fill cells match the neighboring active devices.

- The poly fill cells extend the gate poly and add cut poly as per the DRC.




Select instances in the schematic to view the dummy fill around these devices. You can also see the gate poly of the devices being extended by the poly fill and the cut poly rails between rows of devices.

**Note:** You can select a local interconnect layer for diffusion from the *Fill Layer* drop-down list in the *Poly Fill* section to add fill for these layers. The associated cut layer is picked automatically. You can choose to fill the entire cellview or only the selected areas.

### *Why are poly fill not generated?*

Poly fill might not be generated due to various design scenarios and constraints. To generate poly fill correctly, check the following settings:

- Ensure that there are no diffusion areas. Poly fill do not fill the diffusion areas.
- Check for any spacing rules between diffusion areas and poly shapes. Ensure that these rules are adhered to.
- Ensure that the selected area is inside the cellview boundary.
- Ensure that the poly shapes are aligned properly along the poly grid. If not aligned, a warning message is displayed and poly fill are not inserted.

- Check whether the `allowedLengthRanges` rule is set for the specified fill layer in the technology file. If set, ensure that the spacing left for the poly fill in the selected area is sufficient to accommodate the poly fill.

### Deleting Fill

To delete device fill:

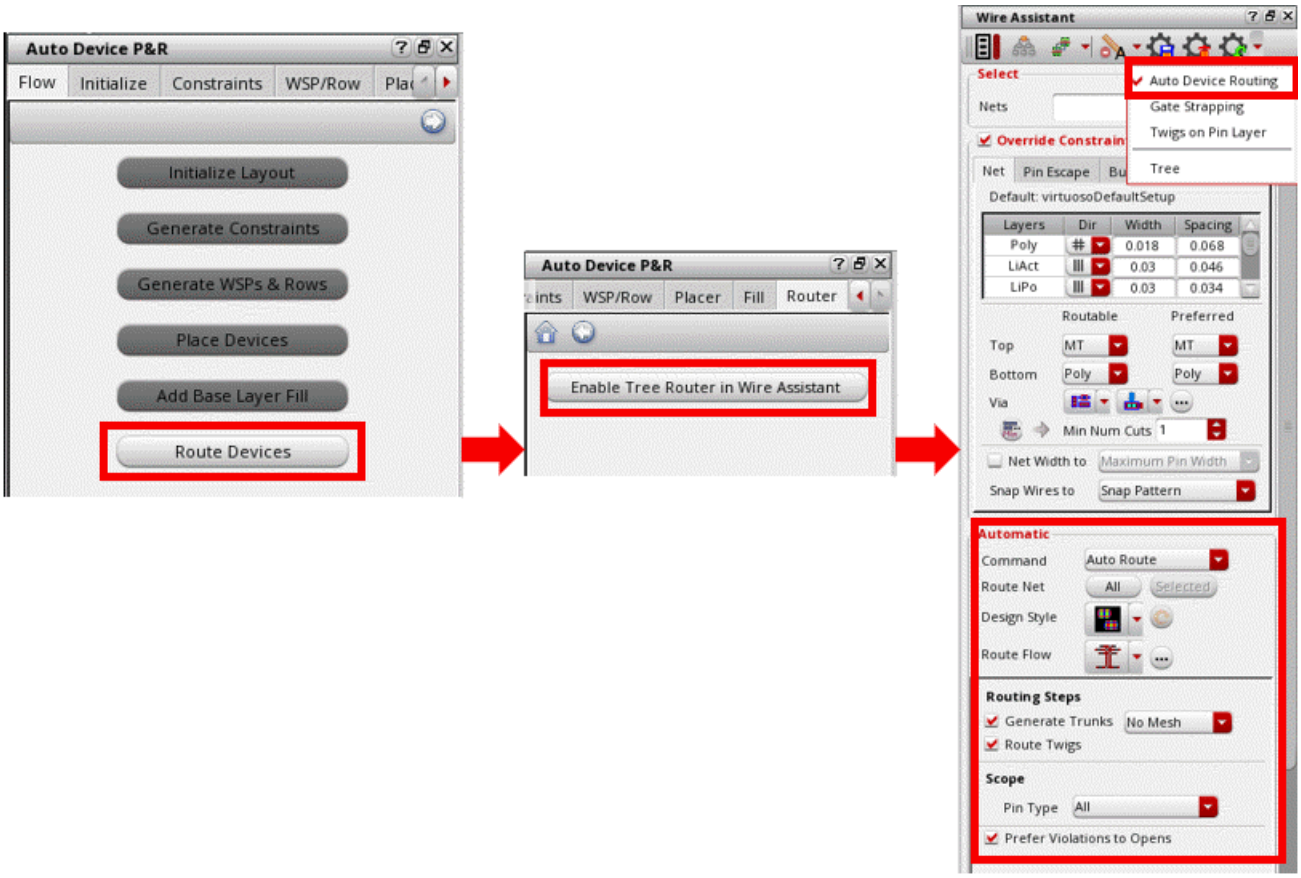
1. Select the required options in the *Dummy Fill* and *Poly Fill* areas to indicate the device fill to be deleted.
2. Click *Delete Fill*.

# Routing the Design

The final step in the automated device placement and routing flow is device routing. You can use the Tree Router, which is the automatic device-level router to route your design.

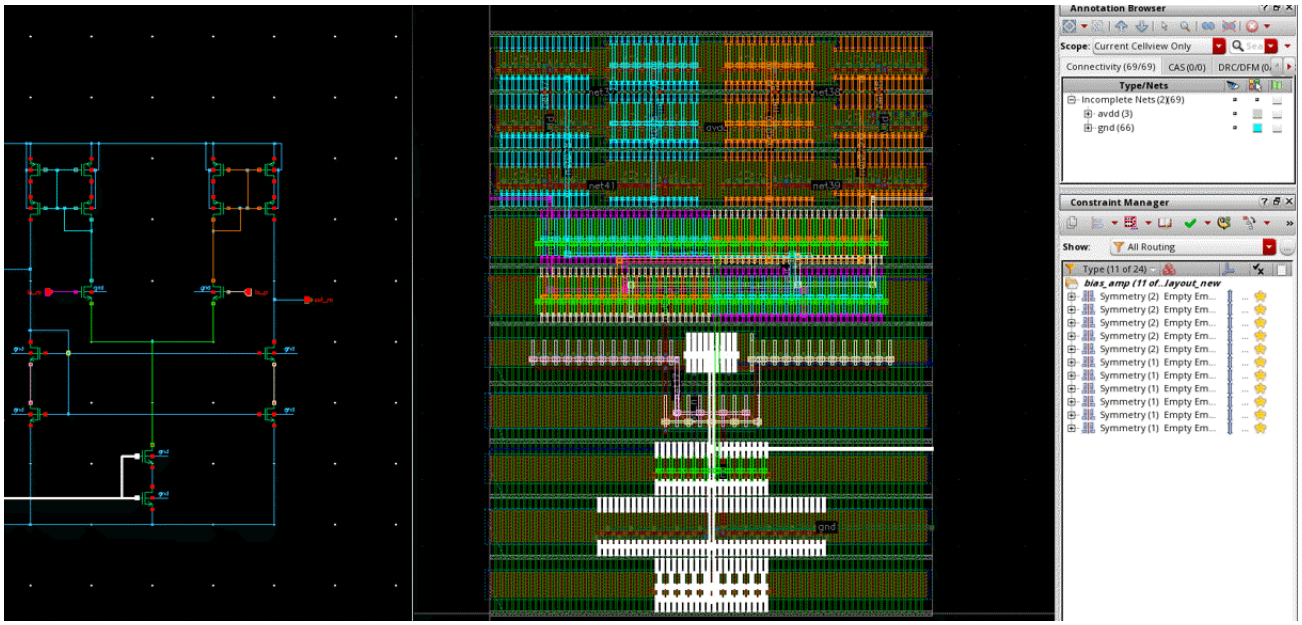
To route a design, click *Route Devices* in the Auto Device P&R assistant to open the *Router* tab.

Click *Enable Tree Router in Wire Assistant* to open the Wire Assistant.




The Auto Device Routing preset is selected by default. In this mode, the Tree router is used for routing. The Tree router automatically identifies and routes mesh, ring (tap cell rings), and tree structures in the design.

The following image shows an automatically placed and routed design that is WSP-based, DRC-correct, and constraint-compliant.




For more information about running the tree router, see Using the Tree Route Flow (ICADVM 18.1 Only) in *Virtuoso Space-based Router User Guide*.

# 4

# Generating Modgens Automatically

The Auto Device Array form is a unified interface that allows you to quickly create and edit Modgens without invoking the Modgen Editor. The form integrates the key options from the Modgen Placement and Modgen Routing toolbars. Select the instances that you want to include in the Modgen and use the options in the Auto Device Array form to generate a placed and routed Modgen. The options in the form are organized in three tabs that let you specify the placement settings, guard ring options, and topology and routing settings.

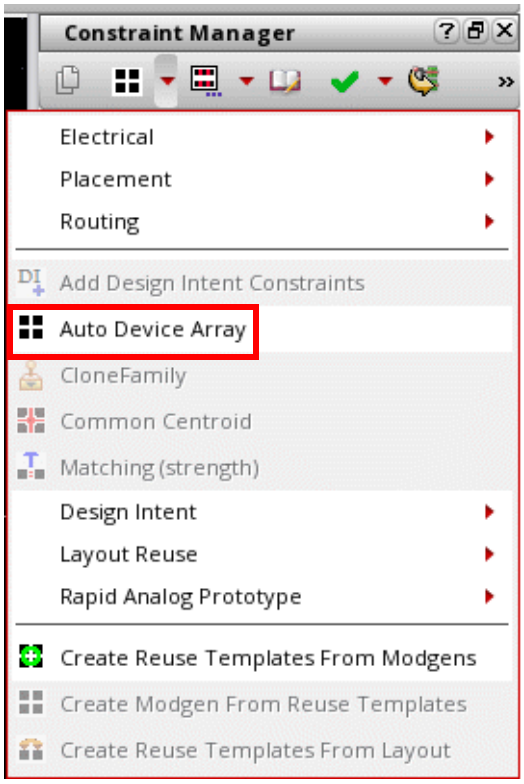
This chapter includes the following sections:

## Opening the Auto Device Array Form

The Auto Device Array form is accessible from both the Constraint Manager assistant and the Auto Device P&R assistant.

To open the form from the Constraint Manager assistant:

1. Select either a Modgen in the Constraint Manager assistant or the required devices in the layout canvas.
2. Open the constraint list, which is the second drop-down menu in the Constraint Manager assistant.
3. Choose *Auto Device Array*. The Auto Device Array form appears.




**Note:** Alternatively, you can select a Modgen or the required instances and press `Control + M`.

You can open the Auto Device Array form from the Auto Device P&R assistant either in the constraint generation step or after running the placer. To invoke the Auto Device Array form during constraint generation:

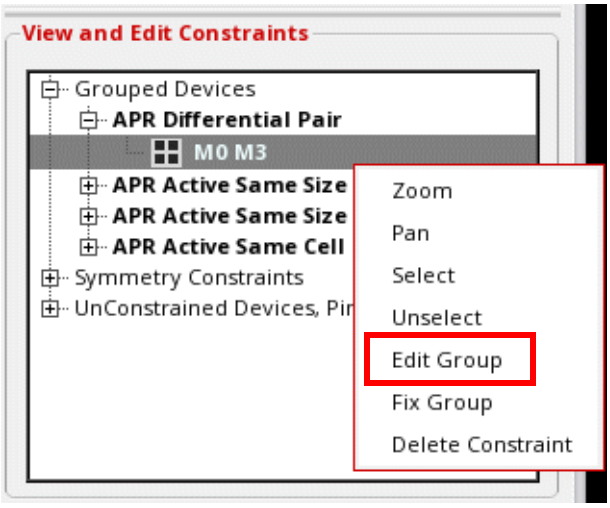
1. Set the `aprCreateModgens` to `t` as follows:

```
envSetVal("layoutXL.AP" "aprCreateModgens" 'boolean t)
```

In this mode, grouped devices are generated as Modgens in the automated device placement and routing flow.

2. Initialize the layout. For more information, see Initializing a Layout.
3. Generate the required device groups. For more information, see Generating Constraints and Constraint Groups.
4. Right-click the required device group in the *View and Edit Constraints* pane.
5. Choose *Edit Group*.

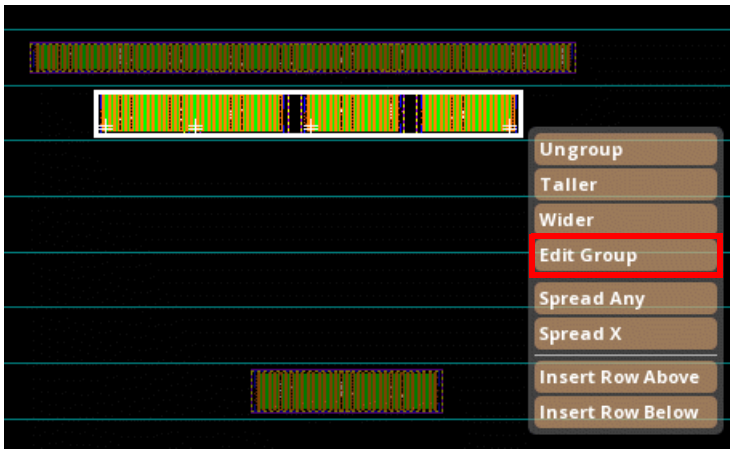
   The Auto Device Array form is displayed.




In addition to the above method, the Auto Device P&R assistant lets you create Modgens after running the placer—as part of interactive placement. To do this:

1. Select *Context Menu* on the *Placer* tab of the Auto Device P&R assistant.
2. Select the required Modgens or devices in the layout canvas. A context menu is displayed.
3. If you have selected devices in the layout canvas, select *Group* to create a group that includes the selected devices. If you have selected Modgens, skip this step.

4. Select *Edit Group* from the context menu. The Auto Device Array form appears.




The Auto Device Array form provides a unified interface to create and edit Modgens. The Auto Device Array form includes the following tabs:

- **Placement**: Lets you specify the Modgen pattern settings. For more information, see Defining Modgen Placement Settings.
- **GuardRing**: Lets you create guard rings around the Modgens. For more information, see Creating Guard Rings around Modgens.
- **Routing**: Lets you define Modgen topology patterns and set routing preferences. For more information, see Defining Modgen Topology Patterns and Routing Options.

*Important*

The *Routing* tab is available only when the Auto Device Array form is launched from the Constraint Manager assistant. In the Virtuoso automated device placement and routing flow, the tree router is used for routing. Therefore, the *Routing* tab is not available when the Auto Device P&R assistant.

## Defining Modgen Placement Settings

When you open the Auto Device Array form, the *Placement* tab is displayed by default.


To define Modgen placement settings:

1. Load the Modgen settings from an existing template file.

The uneditable *Device Map* field displays the device-to-symbol mapping as in the Grid Pattern Mapping (GPM) assistant.

2. In the *Template* section of the *Placement* page:

   a. Select a *Device Group*. The default value is the device group to which the selected instances belong. If the selected instances are not part of any device group, then *Device Group* is set to *Generic Group*.

   b. Select a Modgen template file name in the *File* field.

   c. Click *Load* to load settings from the selected Modgen template file.

   **Note:** The *Template* section also lets you save the current settings to a Modgen template file. To do this, specify a unique name in the *File* field and click *Save*.

3. Select one of the following *Pattern Preset* options to specify the pattern in which the devices are to be placed within the Modgen.

   - *Current*: Uses the current Modgen pattern.
   - *Uniform*: Displays a pattern based on the bottom-up approach, with row-based split.
   - *Interdigitated*: Applies an interdigitation of 1.
   - *Compact*: Adjusts the pattern to achieve the maximum abutment of devices.
   - *Custom*: Lets you specify a base pattern for the Modgen in the adjoining text box.

4. Specify the *Aspect Ratio* of the Modgen by setting *Active Rows* to one of the following:

   - *Best Fit*: This option is available only when *Pattern* is set to *Custom* and a value is entered in the custom pattern text box. In this mode, an optimal placement of the Modgen devices is achieved according to the specified pattern.
   - *Specify*: Lets you specify the number of active rows in the Modgen. The number of columns is automatically calculated and displayed.
   - *Allow Placer to change Aspect Ratio*: Lets the Virtuoso device-level automatic placer adjust the aspect ratio of the Modgen to achieve optimized placement for the given floorplan. The rows and columns of the Modgen are modified, while retaining the base pattern. By default, this option is selected, implying that the placer can reshape the Modgen.

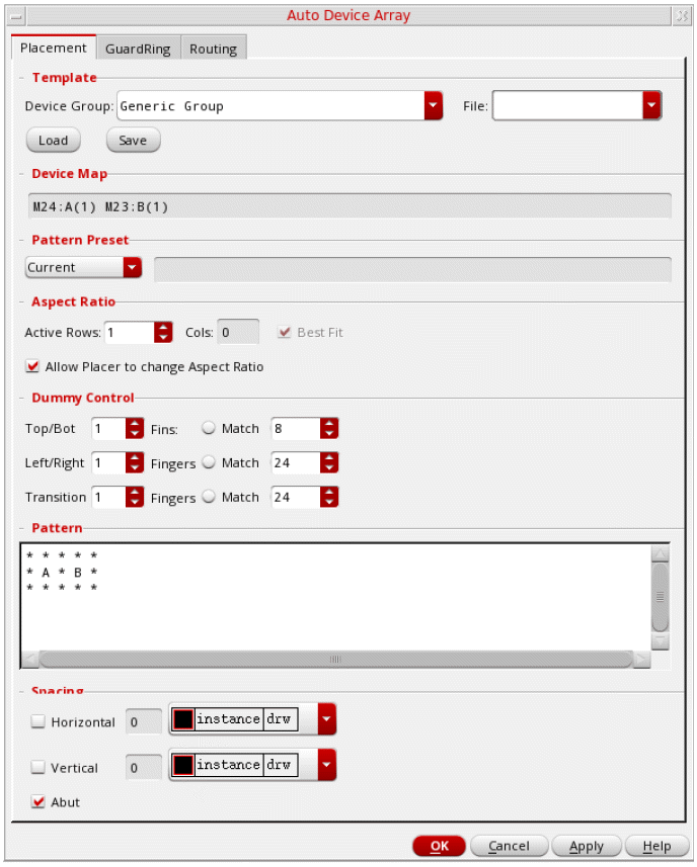
5. Add dummy rows and columns by specifying the following settings in the *Dummy Control* section:

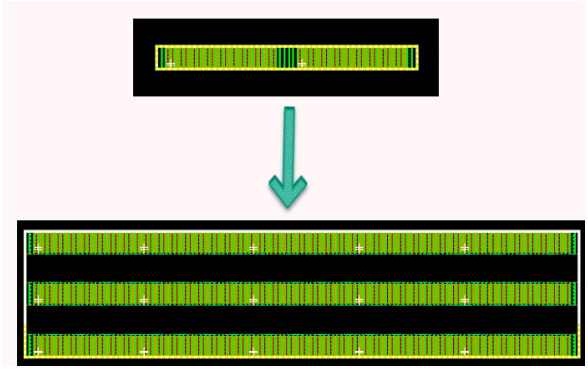
- *Top/Bot*: Inserts the specified number of dummy rows at the top and bottom of the Modgen.
- *Left/Right*: Inserts the specified number of dummy columns on either side of the Modgen.

  **Note:** To add surround dummies, specify both *Top/Bottom* and *Left/Right* values.
- *Transition*: Inserts a dummy column after the specified number of active device columns in the Modgen. For example, if *Transition* is set to `1`, every alternate column is a dummy column.

6. Specify the number of fins or fingers to be added to the dummy rows and columns:
   - *Match*: Creates the same number of fins or fingers as the neighboring device.
   - *Specify*: Lets you specify the required number of fins or fingers.
7. Verify that the Modgen pattern displayed in the *Pattern* box matches your requirements. The pattern is derived based on the specified preset, active rows, and dummies. You cannot edit the pattern in the *Pattern* box. Use the options in the previous sections to customize the pattern as per your requirements.
8. Specify the spacing between the Modgen devices:
   - *Horizontal*: Specifies the spacing between devices in rows. Select a reference layer for calculating the horizontal spacing.
   - *Vertical*: Specifies the spacing between devices in columns. Select a reference layer for calculating the vertical spacing.
9. Select *Abut* to abut all devices in the Modgen.
10. Click *OK*.

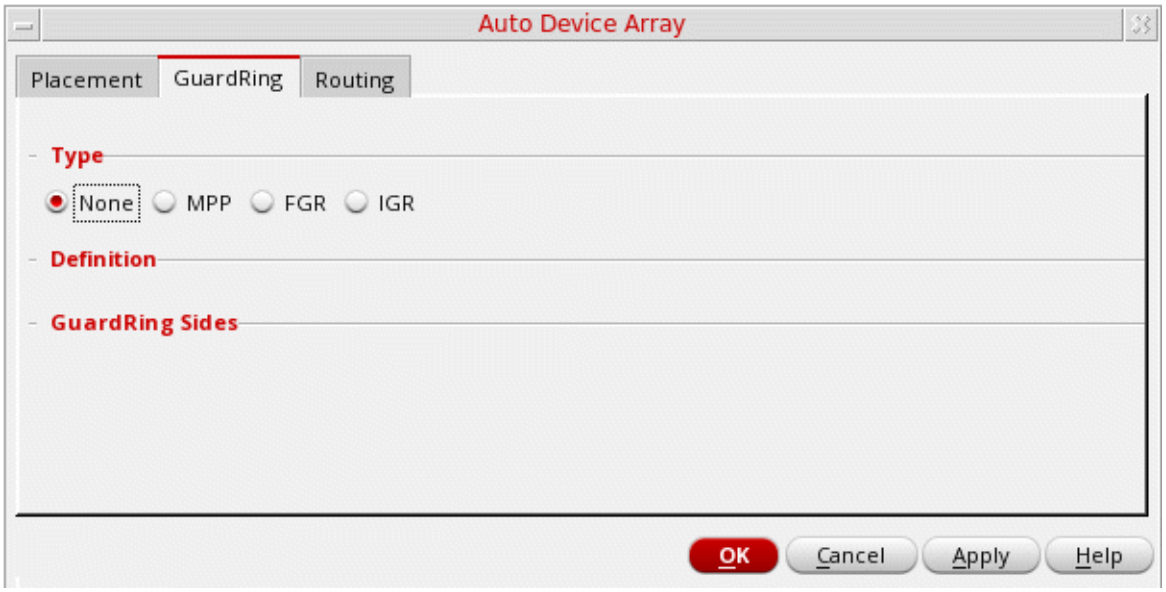
The following images show how the placement settings are applied to a Modgen.



## Creating Guard Rings around Modgens

Use the options on the *GuardRing* tab of the Auto Device Array form to generate guard rings around Modgens.




By default, *Type* is set to *None*, which indicates that no guard rings are to be created. You can create the following types of guard rings:

- Multipath part (MPP) guard rings
- Fluid guard rings (FGR)
- Identical guard rings (IGR)

*Important*

Before creating guard rings, ensure that their definitions are added to the technology file. The options to create IGR are available only if the PDK that you are using is configured to support identical guard rings.

To create a guard ring:

1. Open the *GuardRing* tab of the Auto Device Array form.
2. Select a *Net* connection for the guard ring. This option is available only when *Type* is set to *MPP*, *FGR*, or *IGR*.
3. Select one of the following types of guard rings:

- *MPP*: Related options are displayed in the *Definition* and *GuardRing Sides* sections.


**a.** Select the required MPP guard ring definition from the *MPP* list. All MPP guard rings that are defined in the technology file are listed here.

**b.** Select the sides around the Modgen on which you want to add guard rings: *Left*, *Right*, *Bottom*, or *Top*. By default, all sides are selected.

- *FGR*: Related options are displayed in the *Definition* and *GuardRing Sides* sections.


Select the required FGR guard ring definition from the *FGR* list, which lists the FGRs defined in the technology file.

FGRs are added on all sides of the Modgen. Therefore, all the four options in the *GuardRing Sides* sections, *Left*, *Right*, *Bottom*, *Top*, are selected by default. You cannot modify these settings.

- *IGR*: Related options are displayed in the *Definition* and *GuardRing Sides* sections.


**a.** Select an IGR definition from the *IGR* list.

The IGR definition provided by the PDK defines all aspects of the guard ring, for example the unit cell `lib:cell:view`, its parameters, and parameter callbacks, which helps the Modgen instantiate the guard ring correctly.

**b.** Select one of the following IGR types:

- *Surround*: The IGR surrounds all the instances.
- *Surround+Strip*: The IGR surrounds all the instances, and strips of guard ring instances are inserted between one or more Modgen rows.
- *Grid*: The IGR surrounds every instance or group of abutted instances separately.

**c.** Select the sides around the Modgen on which you want to add guard rings: *Left*, *Right*, *Bottom*, *Top*. By default, all sides are selected.

**d.** Specify the strip width by specifying the number of fins in the *Strip Width in Fins* field.

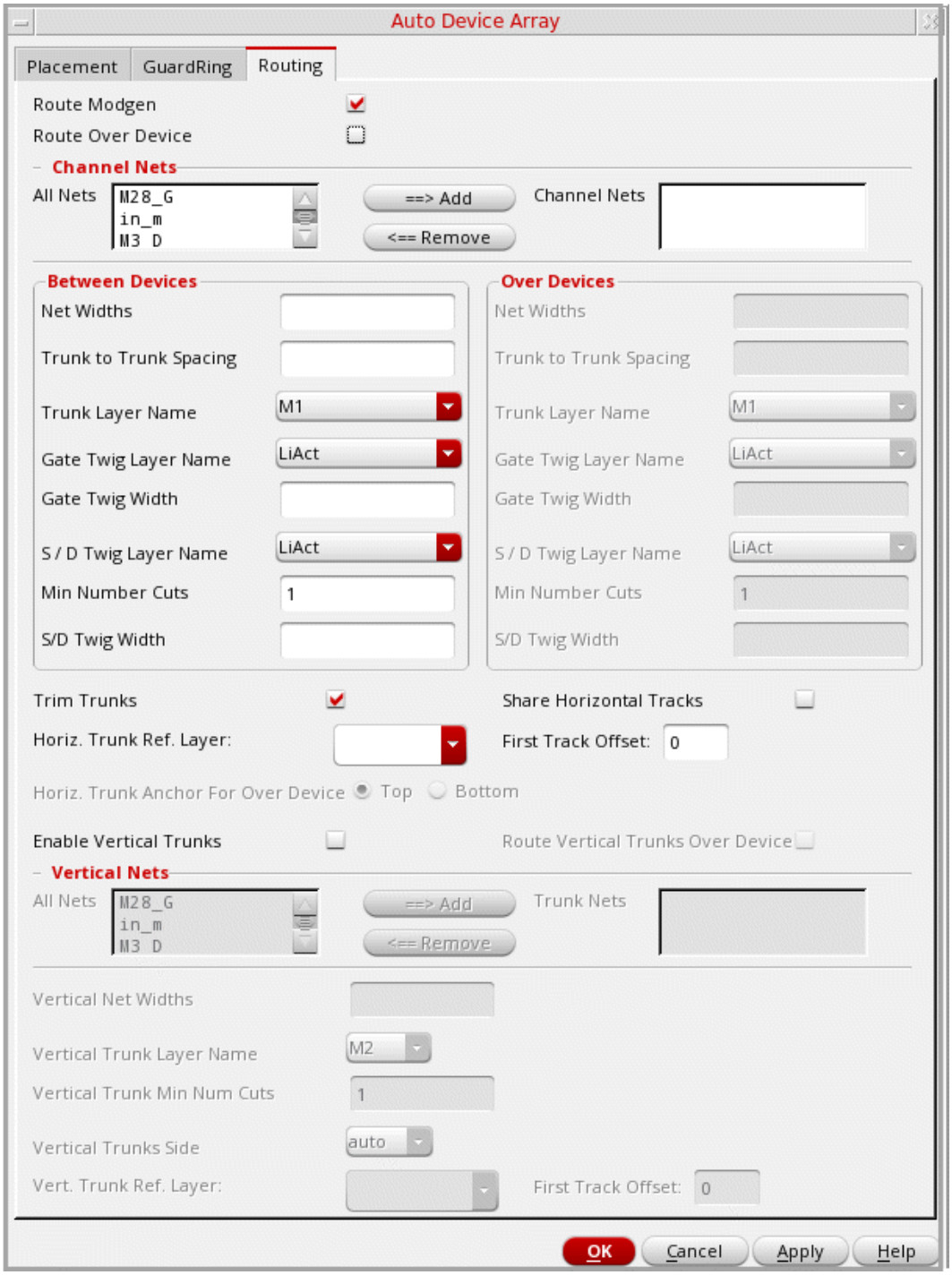
**e.** Specify the number of rows to be inserted between strips of IGRs in the *Rows Between Strips* field.

**4.** Click *OK*.

A guard ring as per your definition is generated in the layout canvas.

## Defining Modgen Topology Patterns and Routing Options

The *Routing* tab provides options to use the pin to trunk router to define Modgen topology patterns. These patterns help you to visualize, configure, and store routing information.



**Note:** Omit this task if you want to use any other router, such as the tree router, to route the Modgen devices.

To define the Modgen topology pattern and routing options:

1. Open the *Routing* tab of the Auto Device Array form.
2. Select *Route Modgen* to use the pin to trunk router to route the Modgen devices.
3. Specify routing preferences for horizontal and vertical nets as indicated in the following topics:
   - Specifying Routing Preferences for Horizontal Trunks
   - Specifying Routing Preferences for Vertical Trunks
4. Click *OK*.

A routed Modgen as per your definition is generated in the layout canvas.

### Specifying Routing Preferences for Horizontal Trunks

To define routing preferences for horizontal trunks, ensure that *Route Modgen* is selected.

1. Select *Route Over Device* to allow horizontal routes to be generated over devices. When selected, the *Over Devices* section is available for edit. If not selected, the *Between Devices* section is available.
2. Specify the channel nets for the horizontal trunks. Select the required nets in the *All Nets* box and click *Add* to move them to the *Channel Nets* box. The sequence in which the nets are listed in the *Channel Nets* box determines the net order.

   **Note:** You can route a net multiple times by adding multiple instances of the channel net in the *Channel Nets* box.
3. Specify a space-separated list of net widths in the *Net Widths* field for the selected channel nets. If not specified, the default values from either the technology file or a predefined WSP are used. If a single value is specified, it is applied to all channel nets.
4. Specify the spacing between trunks in the *Trunk to Trunk Spacing* field.
5. Select a *Trunk Layer Name* to specify the layer on which horizontal trunks are to be generated.
6. Select a layer from the *Gate Twig Layer Name* list. The twigs that are connected to the gate terminal are generated in this layer.

7. Specify the width of the gate twigs in the *Gate Twig Width* field.
8. Select a layer from the *S/D Twig Layer Name* list. The twigs connected to the source and drain terminals are generated in this layer.
9. Specify the minimum number of cuts (*Min Number Cuts*) for the vias connecting the twigs to other objects. The default value is `1`.
10. Specify the source and drain twig width values in the *S/D Twig Width* field.
11. Select *Trim Trunks* to trim the ends of the horizontal trunks while routing.
12. Select *Share Horizontal Tracks* to share the horizontal trunks that are on the same layer and have the same connectivity.
13. Specify the trunk offset by selecting the reference layer from the *Horiz. Trunk Ref. Layer* list and specifying the offset value in the *First Track Offset* field. The default value is `0`.
14. Set the trunk anchor point of the trunks in a device row by selecting one of the following options in the *Horizontal Trunk Anchor For Over Devices* section:
    - *Top* (default): The trunks are anchored to the top-left vertex of the device row.
    - *Bottom*: The trunks are anchored to the bottom-left vertex of the device row.

### Specifying Routing Preferences for Vertical Trunks

To define routing preferences for vertical trunks:

1. Select *Enable Vertical Trunks*. The options to define routing preferences for vertical trunks are available for edit.
2. Select *Route Vertical Trunks Over Device* to allow vertical routes to be generated over devices.
3. Specify the nets to be connected to the vertical trunks. By default, the nets that you have selected for the horizontal trunks are listed in the *All Nets* box. Select the required nets in the *All Nets* box and click *Add* to move them to the *Trunk Nets* box.
4. Specify a space-separated list of vertical net widths for the selected trunk nets. If not specified, the default values from either the technology file or a predefined WSP are used. If a single value is specified, it is applied to all trunk nets.

5. Select a *Vertical Trunk Layer Name* to specify the layer on which the vertical trunks are to be generated.
6. Specify the *Vertical Trunk Min Number Cuts* for the vias that connect the twigs to other objects. The default value is 1.
7. Select a *Vertical Trunk Side* to specify the side along which vertical trunks are to be generated. The available options are: *left*, *right*, *both*, and *auto* (default).
8. Specify the trunk offset by selecting the reference layer from the *Vert. Track Ref. Layer* list and specifying the device-to-trunk offset of the first device in the *First Track Offset* field.

# A

# Automated Device Placement and Routing Flow Tabs

This chapter provides information about the following tabs that can be invoked through the Auto Device P&R assistant:

- Flow
- Initialize
- Constraints
- WSP/Row
- Placer
- Fill

## Flow

The **Flow** tab lists the following automatic placement and routing tasks in the correct sequence.

**Initialize Layout** opens the Initialize tab, which lets you generate the PR boundary, instances, nets, and pins in the target layout as per the source schematic.

**Generate Constraints** opens the Constraints tab, which lets you run registered circuit finders on the schematic cellview to identify compatible structures, group these structures into constraints and constraint groups, and generate these constraints and constraint groups in the layout cellview.

**Generate WSPs and Rows** opens the WSP/Row tab provides options to derive WSPs and row regions automatically, based on the device footprint, layers, and DRCs.

**Place Devices** opens the Placer tab, which lets you place the devices and pins within the PR boundary.

**Add Base Layer Fill** opens the Fill tab, which provides a unified interface for adding all types of base layer fill.

**Route Devices** opens the Router tab, which, in turn, opens the Tree Router in the Wire Assistant.

## Initialize

Use the *Initialize* tab to generate layout representations of the schematic design components. Any existing components in the layout view are deleted and are regenerated from scratch. The *Initialize* tab comprises the following sections:

**Generate** lets you choose the objects to be generated in the layout.

- **Instances** generates all the instances in the schematic that do not have any ignore properties attached to them.
- **Boundary** generates a PR boundary. Use the following options to specify the method used to calculate the rectangular boundary:
  - **Utilization** specifies the percentage of area within the cell boundary that must be filled. The default is 25%.
  - **Aspect Ratio** is the width-to-height ratio of the design boundary. The default value is 1 (a square boundary).
  - **Width** specifies the width of the PR boundary. This option is available in the *Utilization* drop-down list.
  - **Height** specifies the height of the PR boundary. This option is available in the *Aspect Ration* drop-down list.
- **Pins** generates all the pins that are present on the selected **Pin Layer**.
- **PDK Setup for Auto P&R** section lets you specify the following PDK settings.
- **PDK** specifies the required PDK.
- **Prepare Cell for P&R** prepares Pcells to work in the placement and routing flow.

### *Related Topics*

Initializing a Layout

## Constraints

Use the *Constraints* tab to automatically generate constraints and constraint groups.

- **Circuit Finders** lists all available constraint finder groups. You can select the required finder groups to be run when you click *Create Constraints*.
- **View and Edit Constraints** lists all existing groups and constraints from the schematic view. You can edit the list as per your requirements.

### *Related Topics*

Generating Constraints and Constraint Groups

## WSP/Row

Use the *WSP/Row* tab to generate a row region and width spacing pattern (WSP), gate, and source/drain tracks as per your specifications. The rows are used for device placement. The tracks are used for device snapping and routing.

- **Create Row Region** creates a row region as per your specifications. The options to specify row height are enabled. Corresponding environment variable: createRowRegion
- **Auto Compute Row Height** specifies the row period. The option is selected by default. In this state, the row height is set to the maximum height of instances in the specified layers. If not selected, you can specify the required value.
- **Specify Row Height** specifies the row height.
- **Use Row Template** lets you select an existing row template.
- **Create Poly Pattern** generates a poly WSP grid.
- **Create Width Spacing Patterns** creates WSPs based on the following settings:
  - **Top Layer** and **Bottom Layer** specify the top and bottom layers for generating WSP tracks. The tracks are inserted in the specified top and bottom layers and in all intermediate layers. Corresponding environment variables: topWSPLayer, bottomWSPLayer
  - **#Gate Tracks** specifies the number of tracks to be used to connect to gate terminals in the poly layer. Corresponding environment variable: numTopGateTracks
  - **Gate Track Width** specifies the width of the gate tracks. Corresponding environment variable: gateTrackWidth
  - **Gate Track Spacing** specifies the spacing between the gate tracks. Corresponding environment variable: gateTrackSpacing
  - **S/D Track Width** specifies the width of the tracks to be used to connect to source and drain terminals. If specified, the **#Source/Drain Tracks** is calculated automatically. Corresponding environment variable: sdTrackWidth, sdTrackMode
  - **#S/D Tracks** specifies the number of source and drain tracks. If specified, the **#S/D Track Width** is calculated automatically. Corresponding environment variable: numSDTracks
  - **Import Width Spacing Patterns** opens the Select Patterns form, which lets you choose the required WSP Pattern file.

***Related Topics***

Placing Devices

## Placer

Use the *Placer* tab to customize and run the interactive placer to place devices and device groups in row regions. You can use the assisted placement options to refine the placement.

- **Objective** lets you choose the placement objective; whether the placement must be done to optimize **Area Only** (for compaction), **Wire Length Only** (for better routability), or **Area and Wire Length** (both compaction and better routability).
- **Floorplan** specifies the region for placement. You can select one of the following values:
  - **Aspect Ratio** specifies that the placement area must be calculated based on the specified aspect ratio. Corresponding environment variable: autoPlaceFixedAR
  - **PR Boundary** specifies that the placement must be within the PR boundary.
- **Run Auto Place** runs the placer.
- **Run ECO Place** runs the incremental placer on unplaced devices.
- **Interactive Placer** lets you set the following placement options:
  - Context Menu displays a shortcut menu that lets you perform context-sensitive operations on devices and device groups.
  - **Row Snapping** snaps devices to the nearest rows.
  - **WSP Snapping** snaps devices to the nearest WSP grids.
- **Information** displays information about the placement.

***Related Topics***

Placing Devices

## Fill

Use the *Fill* tab to add dummy and poly fill in gaps between the instances in the given design.

- **Enable Dummy Fill** inserts dummy fill in the gaps between instances.
  - **Neighbor** specifies that the `lib:cell:view` of the neighboring instance must be considered the master for creating dummy fill. A neighboring instance is considered as per the following sequence:
    - Instance to the left of the gap
    - Instance to the right
    - The first instance in the row below the gap
    - The first instance in the row above the gap is considered

      Corresponding environment variable: dummyFillNeighborMultipleMode
  - **Library**, **Cell**, and **View** specify the master cellview to be used for creating dummy fill when *Neighbor* is not selected. Corresponding environment variables: lobDummyFillCell, lobDummyFillLib, lobDummyFillView
  - **Net Name** specifies the net to which the dummy fill must be connected. If not specified, the dummy remains unconnected.
  - **Match Neighbor** specifies that the dummy fill must be connected to the same net as the neighboring instance. When *Match Net* is selected, *Net Name* is disabled.
  - **Create Fill As** inserts single instance or multiple instance dummy fill as per the setting of the dummyFillNeighborMultipleMode environment variable.
  - **Enable Hierarchical Fill** specifies whether dummy fill can be inserted in hierarchical designs.
- **Enable Poly Fill** inserts ploy fill in the gaps between instances.
  - **Poly Layer** specifies the layer-purpose pair to be used by the placer to derive cut-poly rails.
  - **Poly Fill Width** specifies the maximum width by which poly fill can be extended. Corresponding environment variable: lobPolyWidth
  - **Cut-Poly Layer Width** specifies the width of the cut-poly rails. The default value is 0.0u.
- **Select As** specifies the region to be filled. Select the required option:
  - **CellView** (default): Inserts device fill in all row regions in the current cellview.

- **Row Region**: Inserts device fill only in the row region you select from the drop-down list.
- **Custom Area**: Inserts device fill in the custom area drawn.

***Related Topics***

Adding Base Layer Fill

# B

# Automated Device Placement and Routing Flow Environment Variables

This appendix provides information on the names, descriptions, and graphical user interface equivalents for the environment variables used in the Automated Device-Level flow.

*Important*

Only the environment variables documented in this chapter are supported for public use. All other 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.

### Inherited Environment Variable Settings

Many of the environment variables honored by the placer are set in Layout XL. Information on these environment variables is not duplicated in this section. For more information, see Environment Variables in the Virtuoso Layout Suite documentation.

### Generating Constraints

- aprCreateModgens

### Generating Grids

- bottomWSPLayer
- createNonzeroWidthPoly
- createPolyPattern
- createRowRegion
- createPatternRegion

- finGridLayerPattern
- finGridMaterialType
- gateTrackSpacing
- gateTrackWidth
- multiPolyWSP
- numSDTracks
- numTopGateTracks
- sdTrackMode
- sdTrackWidth
- topWSPLayer
- trackWidth
- useHorizPeriod
- useRowHeight

### Running the Placer

- autoPlaceAdjustBoundary
- autoPlaceAdjustRowRegion
- autoPlaceAreaCost
- autoPlaceFixedAR
- autoPlaceFixedW
- autoPlaceGroupAbut
- autoPlaceLockFGAR
- autoPlacePOverN
- autoPlaceWireLenCost
- regenModgenPostProcess

### Adding Fill

- dummyFillNeighborMultipleMode
- lobDummyFillCell
- lobDummyFillLib
- lobDummyFillView
- lobPolyLPP
- lobPolyWidth

# Generating Constraints

## aprCreateModgens

```
layoutXL.AP aprCreateModgens boolean { t | nil }
```

### Description

Creates Modgens in the in the *View and Edit Constraints* section of the *Constraints* tab of the Auto Device P&R assistant.

The default value is `nil`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.AP" "aprCreateModgens")
envSetVal("layoutXL.AP" "aprCreateModgens" 'boolean t)
```

### *Related Topics*

Creating a Modgen

# Generating Grids

## bottomWSPLayer

```
layoutXL.autoWSPGen bottomWSPLayer boolean { t | nil }
```

### Description

Specifies the last layer for generating WSP tracks. Also specify topWSPLayer to specify the range of layers for which WSP tracks must be inserted.

The default is `""`, which implies that no layers are selected.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *WSP/Row* tab |
| Field | *Bottom Layer* |

### Examples

```
envGetVal("layoutXL.autoWSPGen" "bottomWSPLayer")
envSetVal("layoutXL.autoWSPGen" "bottomWSPLayer" 'string "Metal4")
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## createNonzeroWidthPoly

```
layoutXL.autoWSPGen createNonzeroWidthPoly boolean { t | nil }
```

### Description

Specifies whether poly snap patterns with zero width are allowed.

The default is `t`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.autoWSPGen" "createNonzeroWidthPoly")
envSetVal("layoutXL.autoWSPGen" "createNonzeroWidthPoly" 'boolean t)
envSetVal("layoutXL.autoWSPGen" "createNonzeroWidthPoly" 'boolean nil)
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## createPolyPattern

```
layoutXL.autoWSPGen createPolyPattern boolean { t | nil }
```

### Description

Specifies whether poly snap patterns must be generated.

The default is `t`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.autoWSPGen" "createPolyPattern")
envSetVal("layoutXL.autoWSPGen" "createPolyPattern" 'boolean t)
envSetVal("layoutXL.autoWSPGen" "createPolyPattern" 'boolean nil)
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## createRowRegion

```
layoutXL.autoWSPGen createPolyPattern boolean { t | nil }
```

### Description

Specifies whether row regions must be auto-generated along with WSPs.

The default is `t`.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *WSP/Row* tab |
| Field | *Create Row Region* |

### Examples

```
envGetVal("layoutXL.autoWSPGen" "createRowRegion")
envSetVal("layoutXL.autoWSPGen" "createRowRegion" 'boolean t)
envSetVal("layoutXL.autoWSPGen" "createRowRegion" 'boolean nil)
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## createPatternRegion

```
layoutXL.autoWSPGen createPatternRegion boolean { t | nil }
```

### Description

Controls the creation of a row snapping pattern and a pattern region for the current design.

The default is `nil`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.autoWSPGen" "createPatternRegion")
envSetVal("layoutXL.autoWSPGen" "createPatternRegion" 'boolean t)
envSetVal("layoutXL.autoWSPGen" "createPatternRegion" 'boolean nil)
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## finGridLayerPattern

```
layoutXL.autoWSPGen finGridLayerPattern string "layer_name"
```

### Description

Specifies the name of the layer that contains the required fin shapes. The fin grids on these fin shapes are snapped to the snap pattern of the design.

The default is `""`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.autoWSPGen" "finGridLayerPattern")
envSetVal("layoutXL.autoWSPGen" "finGridLayerPattern" 'string "Metal4")
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## finGridMaterialType

```
layoutXL.autoWSPGen finGridMaterialType string "layer_name"
```

### Description

Specifies the material type of the fin shapes. The fin grids on these fin shapes are snapped to the snap pattern of the design.

The default is `recognition`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.autoWSPGen" "finGridMaterialType")
envSetVal("layoutXL.autoWSPGen" "finGridMaterialType" 'string "recognition")
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## gateTrackSpacing

`layoutXL.autoWSPGen gateTrackSpacing float` *`float_number`*

### Description

Specifies the default spacing between the gate tracks.

The default is `0.0`.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *WSP/Row* tab |
| Field | *Gate Track Spacing* |

### Examples

```
envGetVal("layoutXL.autoWSPGen" "gateTrackSpacing")
envSetVal("layoutXL.autoWSPGen" "gateTrackSpacing" 'float 1.0)
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## gateTrackWidth

```
layoutXL.autoWSPGen gateTrackWidth float float_number
```

### Description

Specifies the default width of the gate tracks.

The default is `0.0`.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *WSP/Row* tab |
| Field | *Gate Track Width* |

### Examples

```
envGetVal("layoutXL.autoWSPGen" "gateTrackWidth")
envSetVal("layoutXL.autoWSPGen" "gateTrackWidth" 'float 1.0)
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## multiPolyWSP

```
layoutXL.AP multiPolyWSP boolean { t | nil }
```

### Description

Allows generation of multiple poly WSP definitions. This setting is useful in mixed mode placement of devices with different gate lengths or devices with the same gate length, but different poly pitches.

The default is `nil`. In this state, poly WSP definitions are not created even when multiple poly pitches are detected.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.AP" "multiPolyWSP")
envSetVal("layoutXL.AP" "multiPolyWSP" 'boolean t)
envSetVal("layoutXL.AP" "multiPolyWSP" 'boolean nil)
```

### *Related Topics*

Placing Devices

## numSDTracks

```
layoutXL.autoWSPGen numSDTracks int integer
```

### Description

Specifies the number of source and drain tracks to be accommodated in each WSP track. This value is applicable only if sdTrackMode is set to `byCount`. The default is `8`. The track width is calculated automatically based on the gate, source, and drain (S/D) track settings.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *WSP/Row* tab |
| Field | *#S/D Tracks* |

### Examples

```
envGetVal("layoutXL.autoWSPGen" "numSDTracks")
envSetVal("layoutXL.autoWSPGen" "numSDTracks" 'int 10)
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## numTopGateTracks

```
layoutXL.autoWSPGen numTopGateTracks int integer
```

### Description

Specifies the number of gate tracks to be accommodated within a WSP track. The default is `2`, the minimum allowable.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *WSP/Row* tab |
| Field | *#Gate Tracks* |

### Examples

```
envGetVal("layoutXL.autoWSPGen" "numTopGateTracks")
envSetVal("layoutXL.autoWSPGen" "numTopGateTracks" 'int 4)
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## sdTrackMode

```
layoutXL.autoWSPGen sdTrackMode cyclic { "byCount" | "byWidth" }
```

### Description

Specifies whether the number of S/D tracks is based on the absolute count (`byCount`) or must be calculated based on the specified S/D track width (`byWidth`).

The default is `"byCount"`. In this mode, use numSDTracks to specify the absolute count.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *WSP/Row* tab |
| Field | *#S/D Tracks, S/D Track Width* |

### Examples

```
envGetVal("layoutXL.autoWSPGen" "sdTrackMode")
envSetVal("layoutXL.autoWSPGen" "sdTrackMode" 'cyclic "byWidth")
envSetVal("layoutXL.autoWSPGen" "sdTrackMode" 'cyclic "byCount")
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## sdTrackWidth

```
layoutXL.autoWSPGen sdTrackWidth float float_number
```

### Description

Specifies the default width of the source and drain tracks.

The default is `0.0`.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *WSP/Row* tab |
| Field | *S/D Track Width* |

### Examples

```
envGetVal("layoutXL.autoWSPGen" "sdTrackWidth")
envSetVal("layoutXL.autoWSPGen" "sdTrackWidth" 'float 1.0)
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## topWSPLayer

```
layoutXL.autoWSPGen topWSPLayer string "layer_name"
```

### Description

Specifies the topmost layer for generating WSP tracks. Also specify bottomWSPLayer to indicate the routing layer range for which WSPs must be created. WSP tracks are inserted in these and all intermediate layers.

The default is `""`.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *WSP/Row* tab |
| Field | *Top Layer* |

### Examples

```
envGetVal("layoutXL.autoWSPGen" "topWSPLayer")
envSetVal("layoutXL.autoWSPGen" "topWSPLayer" 'string "Metal4")
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## trackWidth

```
layoutXL.autoWSPGen trackWidth float float_number
```

### Description

Specifies the default width of each track.

The default is `0.0`.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *WSP/Row* tab |
| Field | *Compute Pattern Width* |

### Examples

```
envGetVal("layoutXL.autoWSPGen" "trackWidth")
envSetVal("layoutXL.autoWSPGen" "trackWidth" 'float 1.0)
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## useHorizPeriod

```
layoutXL.autoWSPGen useHorizPeriod float float_number
```

### Description

Specifies the horizontal period of the rows to be generated.

The default is `0.0`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.autoWSPGen" "useHorizPeriod")
envSetVal("layoutXL.autoWSPGen" "useHorizPeriod" 'float 1.0)
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

## useRowHeight

```
layoutXL.autoWSPGen useRowHeight float float_number
```

### Description

Specifies the horizontal height.

The default is `0.0`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.autoWSPGen" "useRowHeight")
envSetVal("layoutXL.autoWSPGen" "useRowHeight" 'float 1.0)
```

### *Related Topics*

Deriving WSPs and Row Regions

WSP/Row

# Running the Placer

## autoPlaceAdjustBoundary

```
layoutXL.AP autoPlaceAdjustBoundary boolean { t | nil }
```

### Description

Automatically adjusts the PR boundary where necessary during placement so that it encloses all the components in the design.

The default is `nil`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.AP" "autoPlaceAdjustBoundary")
envSetVal("layoutXL.AP" "autoPlaceAdjustBoundary" 'boolean t)
envSetVal("layoutXL.AP" "autoPlaceAdjustBoundary" 'boolean nil)
```

### *Related Topics*

Placing Devices

Placer

## autoPlaceAdjustRowRegion

```
layoutXL.AP autoPlaceAdjustRowRegion boolean { t | nil }
```

### Description

Automatically adjusts the row region while running the Virtuoso device-level placer.

The default is `nil`, in which state row regions are nor modified during device placement, and therefore there are no row region overlaps.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.AP" "autoPlaceAdjustRowRegion")
envSetVal("layoutXL.AP" "autoPlaceAdjustRowRegion" 'boolean t)
envSetVal("layoutXL.AP" "autoPlaceAdjustRowRegion" 'boolean nil)
```

### *Related Topics*

Placing Devices

Placer

## autoPlaceAreaCost

`layoutXL.AP autoPlaceAreaCost float` *`float_number`*

### Description

Specifies the area cost, depending on which the placer attempts to achieve a compact placement.

The default is `1.0`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.AP" "autoPlaceAreaCost")
envSetVal("layoutXL.AP" "autoPlaceAreaCost" 'float 2.0)
```

### *Related Topics*

Placing Devices

Placer

## autoPlaceFixedAR

```
layoutXL.AP autoPlaceFixedAR float float_number
```

### Description

Specifies the aspect ratio for generating the PR boundary.

The default is `-1.0`.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *Placer* tab |
| Field | *Aspect Ratio* |

### Examples

```
envGetVal("layoutXL.AP" "autoPlaceFixedAR")
envSetVal("layoutXL.AP" "autoPlaceFixedAR" 'float 0.5)
```

### *Related Topics*

Placing Devices

Placer

## autoPlaceFixedW

```
layoutXL.AP autoPlaceFixedW float float_number
```

### Description

Specifies the width of the PR boundary.

The default is `-1.0`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.AP" "autoPlaceFixedW")
envSetVal("layoutXL.AP" "autoPlaceFixedW" 'float 0.5)
```

### *Related Topics*

Placing Devices

Placer

## autoPlaceGroupAbut

```
layoutXL.AP autoPlaceGroupAbut boolean { t | nil }
```

### Description

Specifies whether devices are to be automatically abutted after placement.

The default is `t`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.AP" "autoPlaceGroupAbut")
envSetVal("layoutXL.AP" "autoPlaceGroupAbut" 'boolean t)
envSetVal("layoutXL.AP" "autoPlaceGroupAbut" 'boolean nil)
```

### Related Topics

Placing Devices

Placer

## autoPlaceLockFGAR

```
layoutXL.AP autoPlaceLockFGAR boolean { t | nil }
```

### Description

Locks the aspect ratio of all Modgens when running the Virtuoso device-level placer.

The default is `nil`, in which state the user preferences set in the Auto Device Array form are honored.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.AP" "autoPlaceLockFGAR")
envSetVal("layoutXL.AP" "autoPlaceLockFGAR" 'boolean t)
envSetVal("layoutXL.AP" "autoPlaceLockFGAR" 'boolean nil)
```

### Related Topics

Placing Devices

Placer

## autoPlacePOverN

```
layoutXL.AP autoPlacePOverN boolean { t | nil }
```

### Description

Specifies whether p-devices can be placed over n-devices.

The default is `t`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.AP" "autoPlacePOverN")
envSetVal("layoutXL.AP" "autoPlacePOverN" 'boolean t)
envSetVal("layoutXL.AP" "autoPlacePOverN" 'boolean nil)
```

### Related Topics

Placing Devices

Placer

## autoPlaceWireLenCost

```
layoutXL.AP autoPlaceWireLenCost float float_number
```

### Description

Specifies the wire length cost, depending on which the placer attempts to achieve an optimized placement.

The default is `1.0`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.AP" "autoPlaceWireLenCost")
envSetVal("layoutXL.AP" "autoPlaceWireLenCost" 'float 2.0)
```

### *Related Topics*

Placing Devices

Placer

## regenModgenPostProcess

```
layoutXL.AP regenModgenPostProcess boolean { t | nil }
```

### Description

Specifies whether Modgens must be regenerated during placement to ensure that the Modgen parameters match those of the row region in which they are placed.

The default is `t`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL.AP" "regenModgenPostProcess")
envSetVal("layoutXL.AP" "regenModgenPostProcess" 'boolean t)
envSetVal("layoutXL.AP" "regenModgenPostProcess" 'boolean nil)
```

### Related Topics

Placing Devices

Placer

# Adding Fill

## dummyFillNeighborMultipleMode

```
layoutXL.fillEngine dummyFillNeighborMultipleMode cyclic { "singleFinger" |
     "exactCopy" }
```

### Description

Specifies the number of fingers in the dummy fill.

- `singleFinger`: Dummy has a single finger.
- `exactCopy`: Dummy fill has the same number of fingers as the neighboring device.

The default is `singleFinger`.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *Fill* tab |
| Field | *Match Neighbor* |

### Examples

```
envGetVal("layoutXL.fillEngine" "dummyFillNeighborMultipleMode")
envSetVal("layoutXL.fillEngine" "dummyFillNeighborMultipleMode" 'cyclic
"exactCopy")
```

### *Related Topics*

Adding Base Layer Fill

Fill

## lobDummyFillCell

```
layoutXL lobDummyFillCell string "cell_name"
```

### Description

Specifies the name of the master cellview for creating dummy fill.

The default is `""`, which indicates that no dummy fill will be inserted.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *Fill* tab |
| Field | *Dummy Fill – Cell* |

### Examples

```
envGetVal("layoutXL" "lobDummyFillCell")
envSetVal("layoutXL" "lobDummyFillCell" 'string "myCell")
```

### *Related Topics*

Adding Base Layer Fill

Fill

## lobDummyFillLib

```
layoutXL lobDummyFillLib string "library_name"
```

### Description

Specifies the library containing the master cellview to be used for creating dummy fill.

The default is `""`, which indicates that no dummy fill will be inserted.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *Fill* tab |
| Field | *Dummy Fill – Library* |

### Examples

```
envGetVal("layoutXL" "lobDummyFillLib")
envSetVal("layoutXL" "lobDummyFillLib" 'string "myLib")
```

### *Related Topics*

Adding Base Layer Fill

Fill

## lobDummyFillView

```
layoutXL lobDummyFillView string "library_name"
```

### Description

Specifies the view name of the master cellview for creating dummy fill. The default is `""`, which indicates that no dummy fill will be inserted.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *Fill* tab |
| Field | *Dummy Fill – View* |

### Examples

```
envGetVal("layoutXL" "lobDummyFillView")
envSetVal("layoutXL" "lobDummyFillView" 'string "myView")
```

### *Related Topics*

Adding Base Layer Fill

Fill

## lobPolyLPP

```
layoutXL lobPolyLPP string "library_name"
```

### Description

Specifies the library containing the master cellview to be used for creating poly fill.

The default is `""`.

### GUI Equivalent

None

### Examples

```
envGetVal("layoutXL" "lobPolyLPP")
envSetVal("layoutXL" "lobPolyLPP" 'string "myLib drawing")
```

### *Related Topics*

Adding Base Layer Fill

Fill

## lobPolyWidth

```
layoutXL lobPolyWidth string "library_name"
```

### Description

Specifies the width of the poly fill.

The default is `""`.

### GUI Equivalent

| | |
|---|---|
| Command | Auto Device P&R assistant – *Fill* tab |
| Field | *Poly Fill – Poly Fill Width* |

### Examples

```
envGetVal("layoutXL" "lobPolyWidth")
envSetVal("layoutXL" "lobPolyWidth" 'string "3")
```

### *Related Topics*

Adding Base Layer Fill

Fill

C

# Automated Device Placement and Routing Flow SKILL Functions

This section provides syntax, descriptions, and examples for the Cadence® SKILL functions associated with the Virtuoso automated device placement and routing flow.

*Important*

Only the functions documented in this chapter are supported for public use. All other functions, regardless of their name or prefix, and undocumented aspects of the functions described below, are private and are subject to change at any time.

## List of Virtuoso Automated Device Placement and Routing Flow SKILL Functions

- apPlaceAuto
- apSnapInsts
- lobAddCopyFill
- lobBaseLayerDummyFillCB
- lobBaseLayerDummyFillWrapperCB
- lobIsCopyFill
- lobRemoveCopyFill

## apPlaceAuto

```
apPlaceAuto(
     d_cellviewID
     [ ?fixedAR f_fixedAR ]
     [ ?fixedW f_fixedW ]
     [ ?areaCost f_areaCost ]
     [ ?wireLenCost f_wireLenCost ]
     [ ?pOverN g_pOverN ]
     [ ?groupAbut g_groupAbut ]
     [ ?adjustBoundary g_adjustBoundary ]
     [ ?incr g_incr ]
     [ ?fillPopup g_fillPopup ]
     )
     => t / nil
```

### Description

Runs the Virtuoso device-level automatic placer with the specified parameters.

### Arguments

*d_cellviewID* Database ID of the cellview.

?fixedAR *f_fixedAR*

Specifies the desired aspect ratio for the generated boundary.

?fixedW *f_fixedW*

Specifies the width of the boundary.

?areaCost *f_areaCost*

Specifies the area cost, depending on which the placer attempts to achieve a compact placement.

?wireLenCost *f_wireLenCost*

Specifies the wire length cost, depending on which the placer attempts to achieve an optimized placement.

?pOverN *g_pOverN*

Specifies whether P-devices can be placed over N-devices.

?groupAbut *g_groupAbut*

Specifies whether devices are to be automatically abutted after placement.

?adjustBoundary *g_adjustBoundary*

Specifies whether the PR boundary can be adjusted during device placement.

?incr *g_incr*

Specifies whether the placer must be run in the incremental mode. When set to `t`, the incremental placer runs without changing the aspect ratio of the figGroups and Modgens in the design.

?fillPopup *g_fillPopup*

Specifies whether a pop-up window, which prompts whether existing fill are to be removed, must be displayed. The default value is `t`.

### Value Returned

| | |
|---|---|
| `t` | The Virtuoso device-level automatic placer was run. |
| `nil` | The command was unsuccessful. |

### Examples

Runs the Virtuoso device-level automatic placer on the specified cellview with default placement settings.

```
; default run
apPlaceAuto(cv)
```

Runs the Virtuoso device-level automatic placer on the specified cellview. A placement boundary as per the specified aspect ratio is generated.

```
; specify aspect ratio
apPlaceAuto(cv ?fixedAR 2.0)
```

Runs the Virtuoso device-level automatic placer on the specified cellview. A placement boundary as per the specified width and wire length is generated. P-devices are not placed over N-devices.

```
; specify width, wireLenCost and turn off pOverN
apPlaceAuto(cv ?fixedW 20 ?wireLenCost 0 ?pOverN nil)
```

## apSnapInsts

```
apSnapInsts(
     ?cellView d_cellviewID
     ?snapObjSet l_snapObj
     ?snapToDirX { low | center | high }
     ?snapToDirY { low | center | high }
     [ ?disableRowSnap g_disableRowSnap ]
     [ ?disableSPSnap g_disableSPSNap ]
     [ ?enableWSPSnap g_enableWSPSnap ]
     )
     => t / nil
```

### Description

Snaps the specified instances to rows and snap grids.

### Arguments

`?cellView` *`d_cellviewID`*

Database ID of the cellView that contains the instances to be snapped. The defaults is the current cellview.

`?snapObjSet` *`l_snapObj`*

A list of instances to snap. If the set is not specified, all instances in the design are subject to snap.

`?snapToDirX` { `low` | `center` | `high` }

Snap location for the X direction. Valid values are: `low`, `center`, and `high`. The default value is `center`, which is the closet to the grid.

`?snapToDirY` { `low` | `center` | `high` }

Snap location for the Y direction. Valid values are: `low`, `center`, and `high`. The default value is `center`, which is the closet to the grid.

`?disableRowSnap` *`g_disableRowSnap`*

Disable snapping to rows. The default value is `nil`.

`?disableSPSnap` *`g_disableSPSNap`*

Disable snapping to snap grids. The default value is `nil`.

`?enableWSPSnap` *`g_enableWSPSnap`*

Enable snapping to WSP grid. The default value is `nil`.

### Value Returned

| | |
|---|---|
| *`t`* | Instances were snapped to the specified rows and snap grids. |
| `nil` | The command was unsuccessful. |

### Example

Snaps the selected instances in the current cellview to the specified grids in the X and Y directions.

```
apSnapInsts(
    ?cellView geGetEditCellView()
```

```
    ?snapObjSet geGetSelSet()
    ?snapToDirX "low"
    ?snapToDirY "center"
)
```

## lobAddCopyFill

```
lobAddCopyFill(
     d_cellviewID
     l_source
     t_targetArea
     l_netName
     t_createAsMode
     g_hierarchy
     )
     => l_fillFigIDs / nil
```

### Description

Identifies gaps in the row regions in the given target area and fills them with the specified target fill cells.

### Arguments

| | |
|---|---|
| *d_cellviewID* | Database ID of the cellview. |
| *l_source* | Specifies the instance to be considered as the master for creating fill. Valid values are `Neighbor`, `dbInstId`, `dbCellViewId`, `LCVSpec`, list of `dbInstIds`, list `dbCellViewIds`, list of `LCVSpecs`.<br>Format for LCVSpec:<br>`LCVSpec = list(libName cellname viewName list(list(paramName1 paramValue1) .... list(paramNameN paramValueN)))`<br>Lists are not valid when `createAsMode` is set to `Single`. |
| *t_targetArea* | Specifies the target area to be filled. Valid values are cellview ID, name of row region name, or coordinates defining a custom area. |
| *l_netName* | Specifies the net names to which the fill must be connected. Default value is "", which indicates no connection. |
| *t_createAsMode* | Specifies whether fill must be inserted as a multi-fingered single instance (`Single`) or as single-fingered multiple instances (`Multiple`). |
| *g_hierarchy* | Specifies whether copy fill must look through the hierarchy to identify gaps that need to be filled. |

### Value Returned

| | |
|---|---|
| *l_fillFigs* | List of fill IDs that were created. |
| `nil` | The command was unsuccessful. |

### Example

Fills all the rows within a cellview using the devices adjacent to the identified gap.

```
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
    fillDevices=lobAddCopyFill(cellView "Neighbor" cellView~>bBox "" "Single" nil)
  )
)
```

## lobBaseLayerDummyFillCB

```
lobBaseLayerDummyFillCB(
     )
     => t / nil
```

### Description

Invokes the *Fill* tab of the Auto Device P&R assistant.

### Arguments

None

### Value Returned

| | |
|---|---|
| `t` | The *Fill* tab of the Auto Device P&R assistant has been displayed. |
| `nil` | The command was unsuccessful. |

### Example

Invokes the *Fill* tab of the Auto Device P&R assistant.

```
lobBaseLayerDummyFillCB()
```

## lobBaseLayerDummyFillWrapperCB

```
lobBaseLayerDummyFillWrapperCB(
     )
     => t / nil
```

### Description

Checks for license availability before invoking the *Fill* tab of the Auto Device P&R assistant.

### Arguments

None

### Value Returned

| | |
|---|---|
| `t` | The *Fill* tab of the Auto Device P&R assistant has been displayed. |
| `nil` | The command was unsuccessful. |

### Example

Checks for the required licenses and then invokes the *Fill* tab of the Auto Device P&R assistant.

```
lobBaseLayerDummyFillWrapperCB()
```

## lobIsCopyFill

```
lobIsCopyFill(
     d_dbID
     )
     => t / nil
```

### Description

Checks whether the specified object has been created using the copy fill functions.

### Arguments

| | |
|---|---|
| *d_dbID* | Database ID of the object. |

### Value Returned

| | |
|---|---|
| `t` | The object was created using copy fill functions. |
| `nil` | The object was not created using copy fill functions. |

### Example

Selects all copy fill instances that exist within a cellview and checks whether they were created using the copy fill functions.

```
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
     geSelectFigs(setof(inst cellView~>instances lobIsCopyFill(inst))
  )
)
```

## lobRemoveCopyFill

```
lobRemoveCopyFill(
     d_cellviewID
     )
     => t / nil
```

### Description

Removes all objects copy that were created using the copy fill functions from the specified cellview.

### Arguments

| | |
|---|---|
| *d_cellviewID* | Database ID of the cellview. |

### Value Returned

| | |
|---|---|
| t | One or more objects were deleted. |
| nil | There are no objects created using copy fill functions in the specified cellview. |

### Example

Removes all copy fill objects from the current cellview.

```
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
    lobRemoveCopyFill(cellView)
  )
```