Virtuoso Automated Device Placement and Routing Flow Guide

Product Version IC23.1
November 2023

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

Contents

1

Virtuoso Automated Device Placement and Routing

The Virtuoso automated device placement and routing flow comprises a series of tasks to generate automatically placed and routed layouts. The 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 Virtuoso automated device placement and routing flow uses the device-level router.

This device placement and routing flow helps circuit and layout designers using advanced node PDKs for their designs, as well as CAD teams that support such designs. The target designs for this flow are analog and mixed signal device-level designs.

In the Virtuoso automated device placement and routing flow, you use the Auto P&R assistant for device placement and the Routing assistant for device routing. The Auto P&R assistant is available in the Layout EXL and Layout MXL cockpits and the Routing assistant is available in the Layout MXL cockpit.

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.

Related Topics

Important Terms in the Virtuoso Placement Flow

Environment Setup for Automated Device Placement and Routing Flow

Virtuoso Automated Device Placement and Routing Flow

Auto Device Placement and Routing Workspace

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

Important Terms in the Virtuoso Placement Flow

The following table provides descriptions of a important terms used in the Virtuoso automated placement flow.

Term	Description
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.
Device-level Routing	A track (WSP) based high performance and high QoR router which generates fully routed LVS and DRC correct layout as part of a comprehensive automated solution for analog and custom device level designs.
Standard Cell Routing	The standard cell routing solution seamlessly integrates the Innovus NanoRoute™ router in the Virtuoso design environment. The flow starts from a schematic design from which a layout view is generated. You can then import IO pin and boundary information from an existing layout and step through the creation of Innovus-compatible row regions before completing power routing and the placement of tap cells and standard cells.
Chip Assembly Routing	The chip assembly routing flow lets you route a top-level design that has macro instances, I/O pads, and also contains devices and standard cells. It comprises of a series of tasks to quickly and automatically generate routed layouts that are constraint compliant and LVS correct, and follow DRCs as captured in the Virtuoso technology file.

Related Topics

Environment Setup for Automated Device Placement and Routing Flow

Virtuoso Automated Device Placement and Routing Flow

Auto Device Placement and Routing Workspace

Automated Device Placement and Routing Flow Environment Variables

Environment Setup for Automated Device Placement and Routing Flow

To run the automated device placement and routing flow, ensure that you have access to the following:

- **Virtuoso Release:**

  Device Placement: ICADVM18.1 ISR4 or later

  Device Routing: IC23.1 or later

- **License:**

Device Placement	Virtuoso Layout Suite EXL or higher tiers The Create Constraints step checks out the Virtuoso Schematic Editor XL (95115) license. The Transparent Modgen Array Editing and M0 trim insertion feature for certain technology nodes requires the Virtuoso Layout Suite MXL tier.
Device Routing	Virtuoso Layout Suite MXL Use Routing Assistant for assisted and auto device routing in the Virtuoso Layout Suite MXL tier.

- **PDK Settings:**

  Usually PDK settings are available either from the foundry or you can build one from the techLEF from the foundry.

  Key PDK settings required for the automated device placement and routing flow are:

  - Support for Virtuoso Placer and Virtuoso Space-based Router
  - A LAM file that defines the component types for row templates
  - Device registration for circuit finders to recognize Circuit Prospector structures and create constraints
  - (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

- **Common library for all layout and abstract views:**

  The layout and abstract views for device-level designs are in the same library.

- **Before running the device-level router, ensure that:**

  - Devices are placed inside the PR Boundary and needs to be snapped to Poly and Diffusion Grids/Row.
  - The placement is LVS correct and DRC compliant as specified in the technology file.
  - Layout design has rows or diffusion grids. One or more flow components malfunctions if rows or diffusion grids are not present. In addition, the router can run into pin escape issues due to absence of rows or diffusion grids.

Related Topics

Constraint Manager Assistant Customization SKILL Commands

Creating Rows

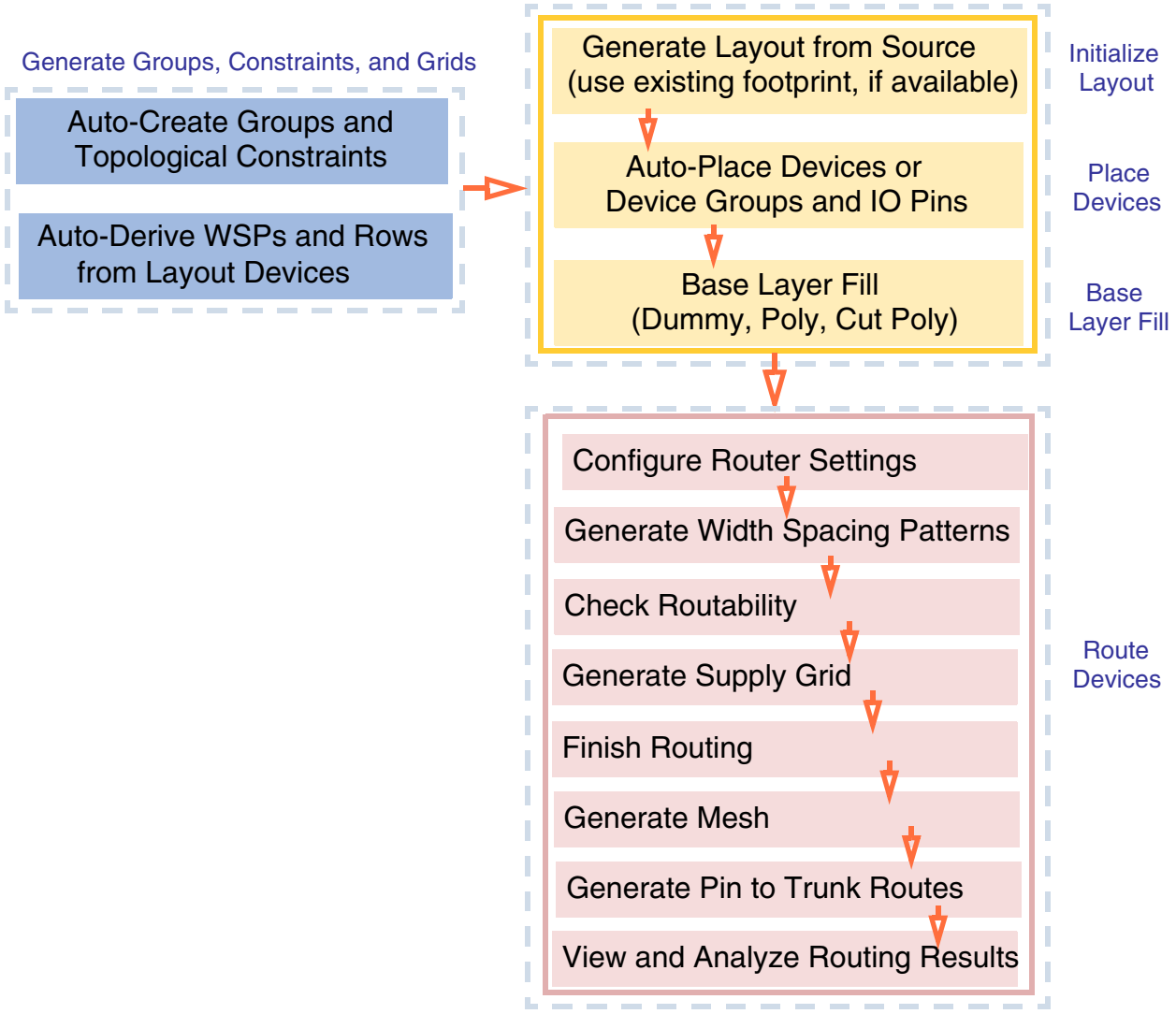
Virtuoso Space-based Router

Library and Attributes Mapping File Syntax

Virtuoso Automated Device Placement and Routing Flow

Virtuoso Automated Device Placement and Routing Flow

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



Flow steps:

Device-level placement:

- **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 Width Spacing Patterns (WSPs), row regions, and diffusion grids 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 and poly fill, where applicable. Inserting fill helps maintain continuity of instances within each row and ensures that the design is DRC-clean and meets density requirements.

Device-level routing:

- **Configure Router Settings:** The first step is to specify the routing options made available in the Routing assistant.
- **Generate Width Spacing Patterns:** Width spacing patterns can be generated, imported, or edited in the design for valid routing layers as required.
- **Check Routability:** In this step, routability checks are run to identify and flag potential issues with the design before running the router.
- **Generate Supply Grid:** Power rails are generated for power and ground nets.
- **Finish Routing:** Completes the routing for all/selected nets in the design using the Auto (for all/selected nets) or Assisted (selected nets only) routing options in the selected scope.
- **Generate Mesh:** Meshes on selected layers can be generated for selected critical nets. These meshes can then be used to establish connections with pins using second-level trunks.
- **Generate Pin to Trunk Routes:** Trunks are generated on a selected layer and the instance pins can then be connected to the generated trunks.
- **View and Analyze Routing Results:** Routing results are analyzed in a single table using the Routing Results Browser. It shows routing information such as the number of opens, shorts, wire lengths, DRC violations, and vias.

Related Topics

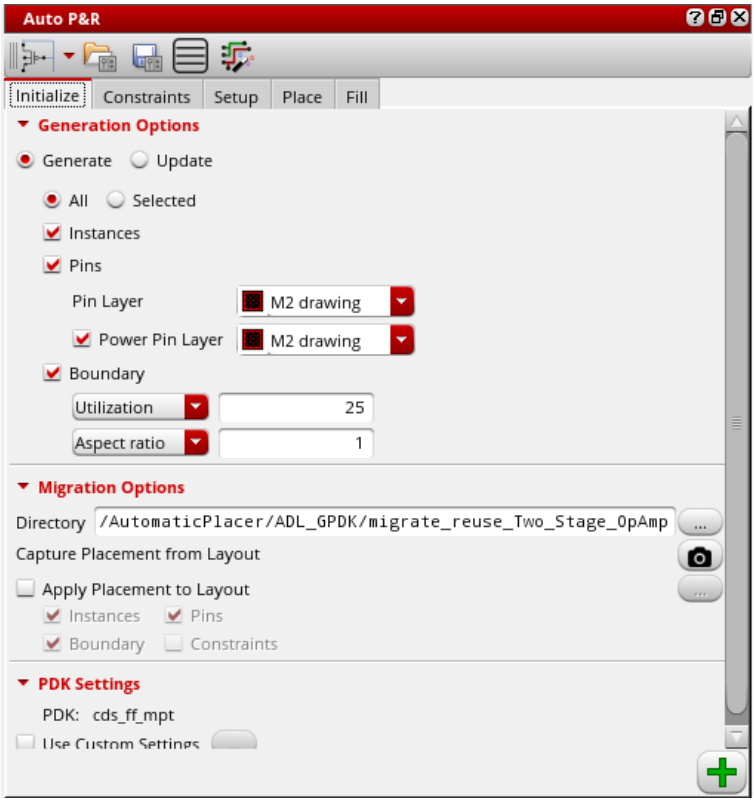
Initializing a Layout in the Automated Device Placement and Routing Flow

Auto Device Placement and Routing Workspace

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

- Choose *Window – Assistants – Auto P&R*.

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



The Auto P&R assistant is displayed. The Auto P&R assistant is the integrated, automatic placement solution available in Virtuoso.

Use the button or the aprAssistantMode environment variable to select one of the following automatic placement modes:

- Device: Runs the automated device placer. This is the default mode.
- Standard Cell: Runs the automated standard cell placer.

The tabs and options in the Auto P&R assistant differ based on the selected mode.

The Auto P&R assistant let you initialize, generate constraints, place, and fill the layout designs automatically, as per your requirements. Routing can be done through the Routing Assistant.

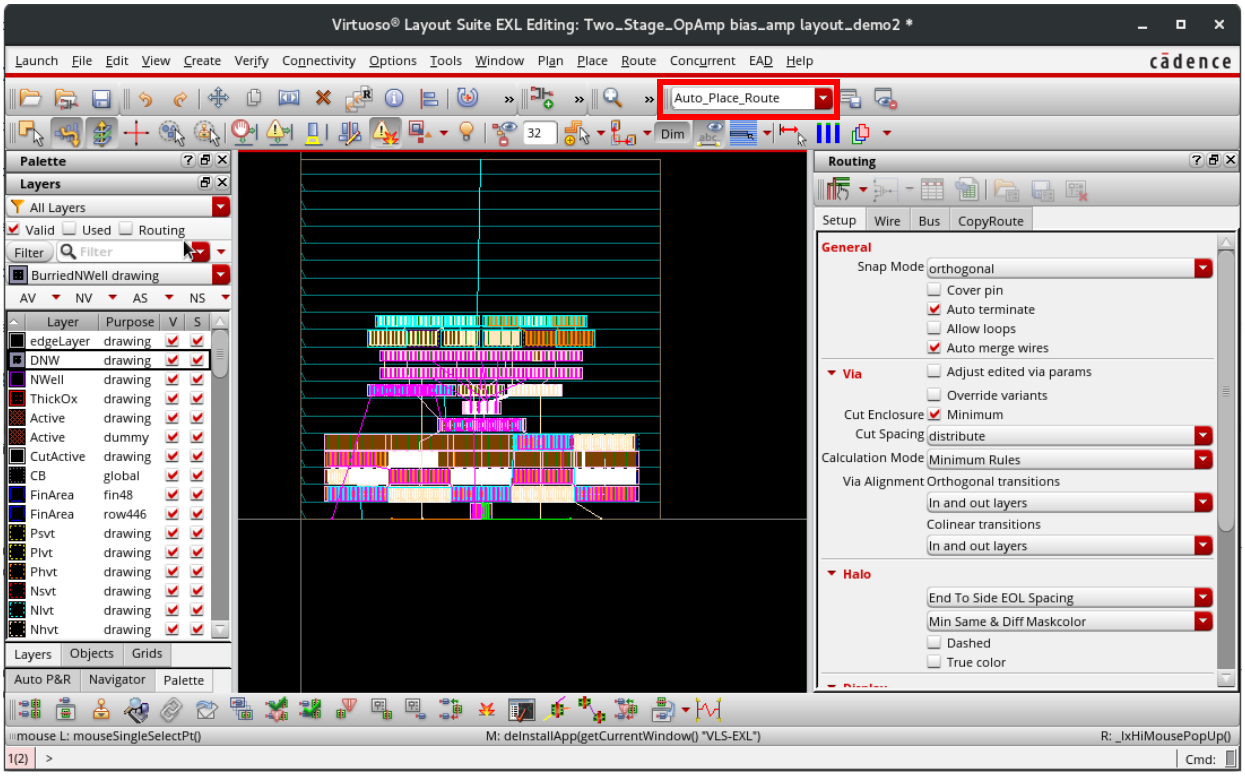
Each tab in the Auto P&R assistant represents a task in the automated device placement and routing flow. Proceed step-wise to generate an automatically placed and routed layout design.

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.

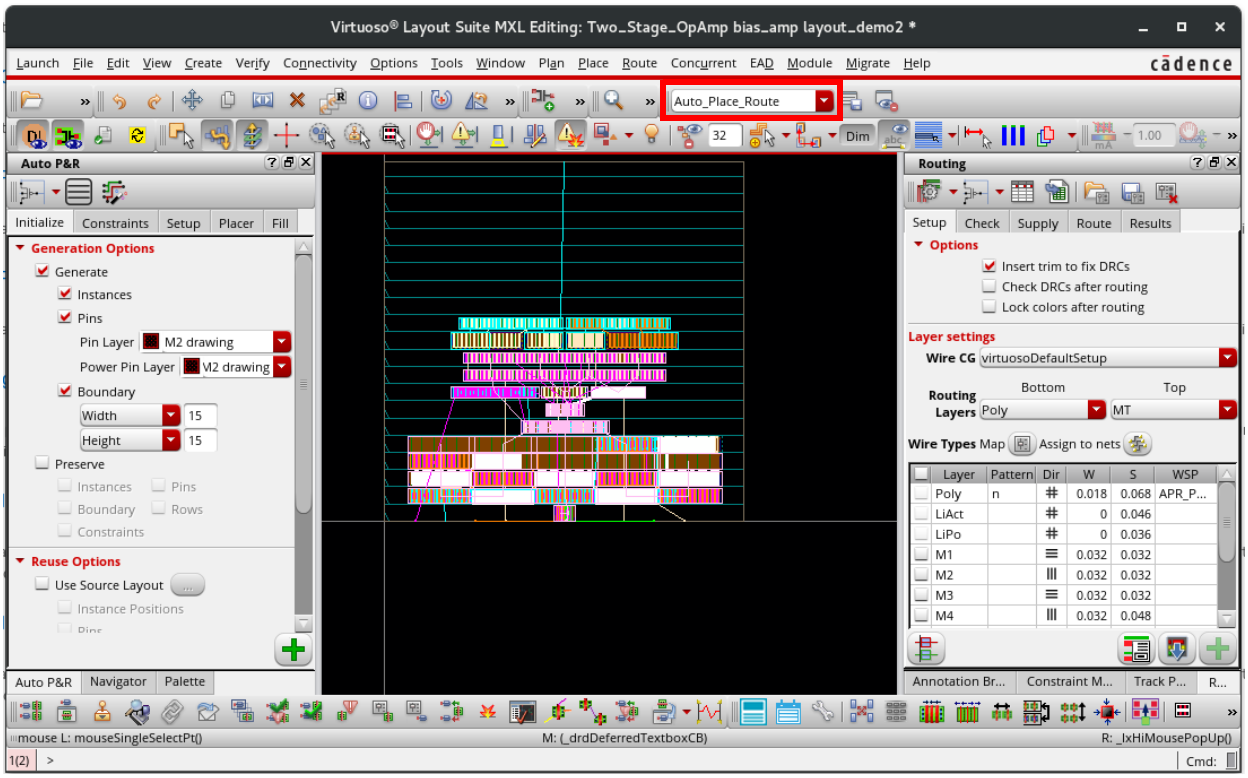
Auto_Place_Route Workspace

Layout EXL and MXL support the *Auto_Place_Route* workspace that provides an interface suitable for initializing, placing, and routing designs. The *Auto_Place_Route* workspace includes the Auto P&R assistant and the Routing assistant with the Interactive Routing option the EXL (or higher) tier or Interactive and Automatic Routing options in the MXL tier.

The following image shows the *Auto_Place_Route* workspace in Layout EXL:



The following image shows the *Auto_Place_Route* workspace in Layout MXL:



Related Topics

Virtuoso Automated Device Placement and Routing Flow

Customized Automated Device Placement and Routing Flow Steps

Tasks that you otherwise cannot perform directly using the Auto P&R assistant can be done by calling custom code, also called a hook.

There are two predefined hooks for each step in the automated device placement and routing flow. These hooks are automatically called before and after running the step.

The pre-hook and post-hook names are predefined, but the procedures do not exist by default. The following example shows a post-hook function that does not have any input argument:

```
procedure(aprInsertFillPostHook()
```

```
let(()
    <steps>
))
)
```

You can customize these hooks to extend the functionality of the automated device placement and routing flow steps. You must not change the predefined names of the hooks.

The following table lists the pre-hooks and post-hooks available in the Virtuoso automated device placement and routing flow:

Step	Pre-Hooks	Post-Hooks
Initialize	`aprInitializePreHook`	`aprInitializePostHook`
Constraints	`aprGenerateConstPreHook`	`aprGenerateConstPostHook`
Setup	`aprCreateRowGridPreHook`	`aprCreateRowGridPostHook`
Placer	`aprRunPlacePreHook`	`aprRunPlacePostHook`
Fill	`aprInsertFillPreHook`	`aprInsertFillPostHook`

Related Topics

Initializing a Layout in the Automated Device Placement and Routing Flow

Generating Constraints and Constraint Groups

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Generating and Deleting Base Layer Fill

2

Initialize and Plan a Layout

Initializing and planning a layout involves generating the devices to be placed, generating the constraints and constraint groups to be honored during placement, and defining the placement region.

Related Topics

Initializing a Layout in the Automated Device Placement and Routing Flow

Generating Constraints and Constraint Groups

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

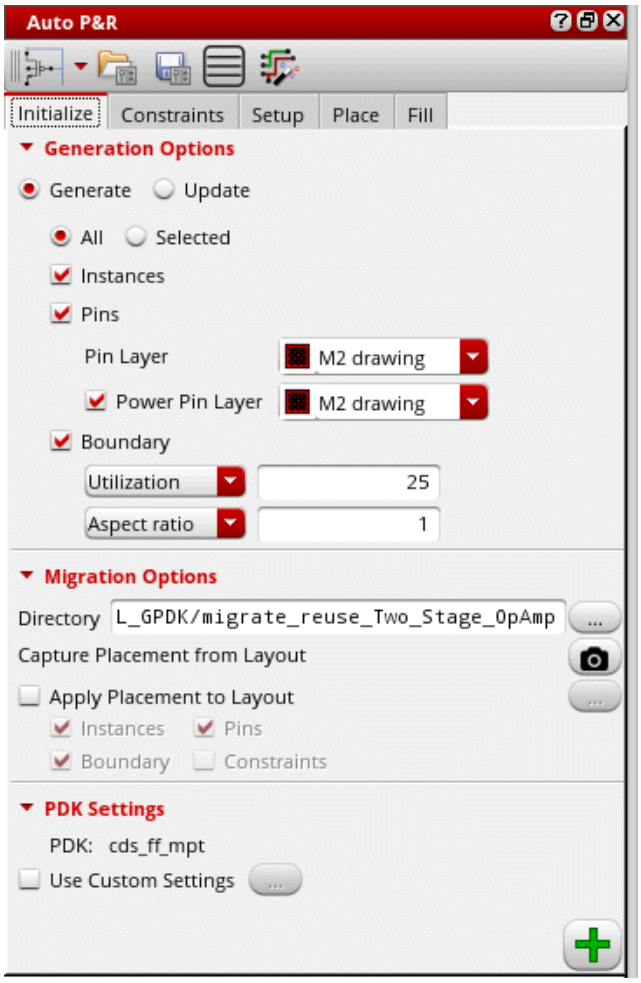
Placing Devices Automatically in the Automated Device Placement and Routing Flow

Initializing a Layout in the Automated Device Placement and Routing Flow

To initialize a layout view in the automated device placement flow:

1. Open the *Initialize* tab of the Auto P&R assistant.

2. Expand *Generation Options* and select *Generate* to generate new objects in the layout canvas or *Update* to update the existing objects.



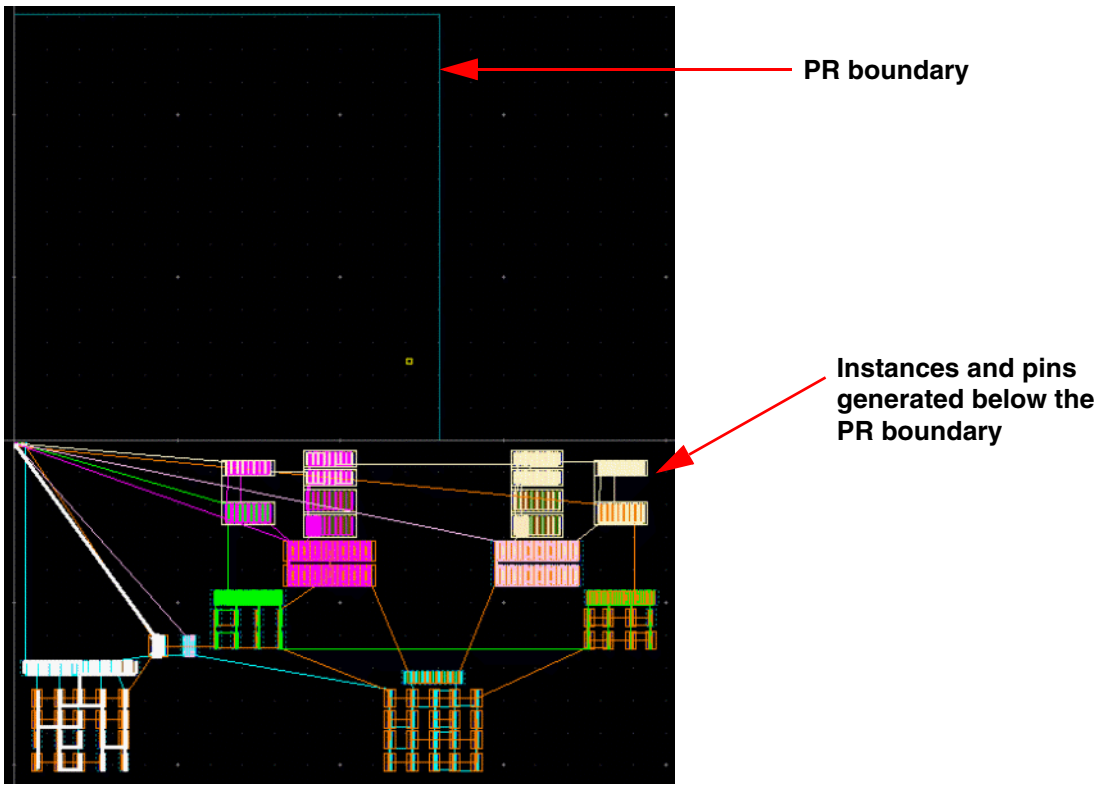
3. Set the scope of layout generation to either *All* or *Selected*.
4. Select *Instances* to generate all instances from the source schematic.
5. Select *Pins to* generate pins during initialization.
6. Select an LPP from the *Pin Layer* list that contains the device pins to be generated. The default value is the first metal layer in the layer stack.
7. Select *Power Pin Layer* to generate power pins.
8. Select an LPP from the *Power Pin Layer* list that contains the power and ground pins to be generated during initialization.
9. Select *Boundary* to generate a PR boundary as per the specified combination of *Utilization* and *Aspect Ratio* or *Width* and *Height*.

10. In Layout MXL, the *Migration Options* section is available, which lets you run the assisted flow of the Virtuoso® Custom Design Migration solution. For more information, see Setting Options for Custom Layout Design Migration.

11. Select *Use custom settings* to load PDK settings from a file. This option is useful for loading addition CDF parameters that are applicable only at lower nodes.

12. Select a file that contains the required PDK package API definition.

13. 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:

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" )`

Related Topics

Initialize

Constraints in the Automated Device Placement and Routing Flow

Device groups and constraints are central to the effective placement and routing of custom and analog layouts. Constraint generation is done after initializing the layout. 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 in the form of Modgens along with symmetry information for symmetric structures.

The Auto 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 *APR 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 P&R flow.

Note: For devices to be added to device groups, they need to be registered for the Circuit Prospector assistant.

For a list of circuit finders that the Auto P&R assistant supports, see APR Circuit Finders.

Use the *Constraints* tab of the Auto P&R assistant to find matching structures and generate constraints in the form of Modgens.

Modgens provide an easy way to manipulate device arrays for matching, for example by changing the aspect ratio and adding dummies.

By default, the Modgens and symmetric structures are listed in the Constraint Manager. The automatic device placer and router honor these constraints during design placement and routing.

The following environment variables control how Modgens are generated:

- aprCreateModgens: Generates constraints as Modgens. The default value is `t`. To create symmetry constraints instead of Modgens, set the value to `nil`:

  ```
  envSetVal("layoutXL.AP" "aprCreateModgens" 'boolean nil)
  ```

  The newly created symmetry constraints are grouped into figGroups and listed in the *View and Edit Constraints* section under the various categories.

- aprCreateModgenFromTemplate: Applies the default Modgen reuse template when creating the new Modgens. The default value is `t`.

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

Related Topics

Constraint Manager Assistant Customization SKILL Commands

Applying a Default Modgen Reuse Template

Constraints

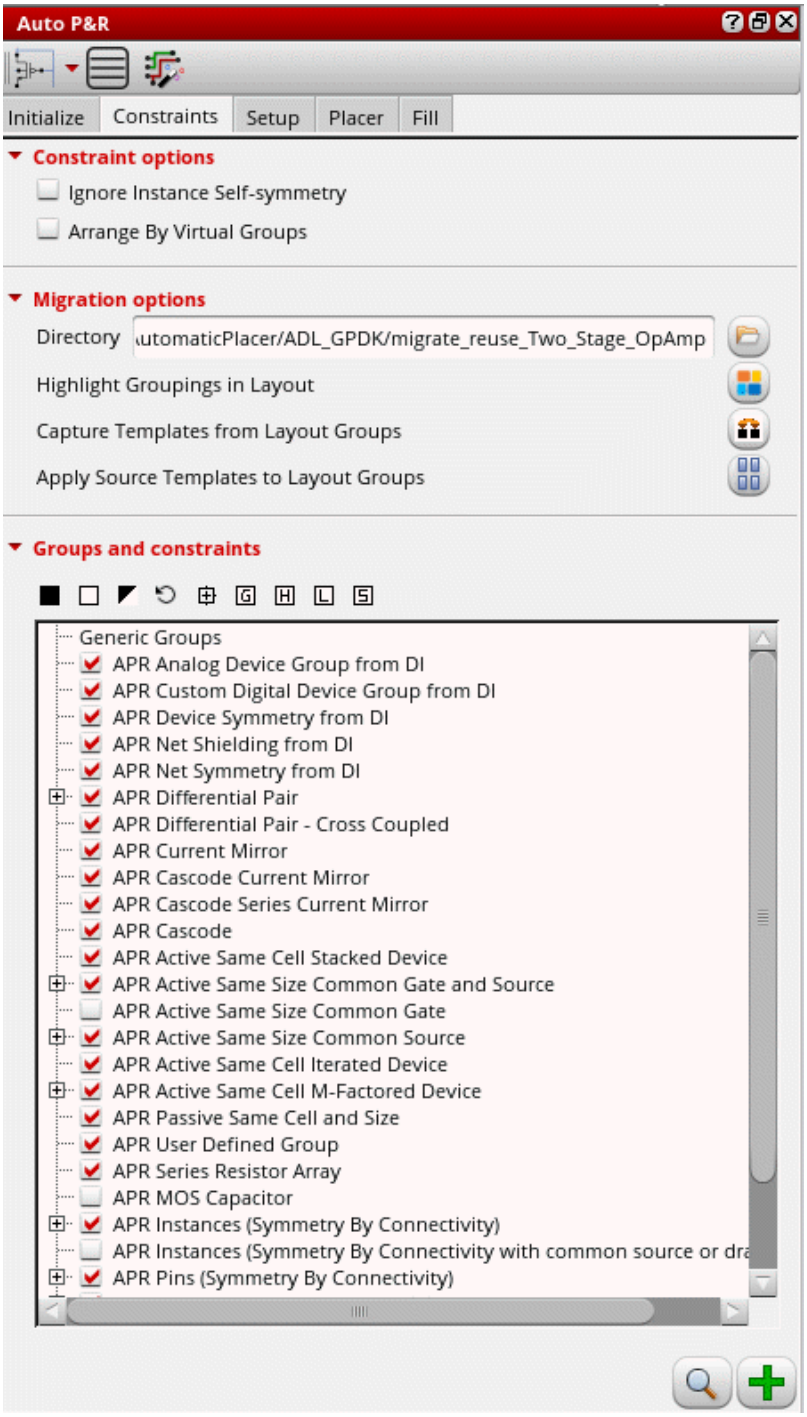
Generating Constraints and Constraint Groups

APR Circuit Finders

Generating Constraints and Constraint Groups

To generate constraints:

1. Open the *Constraints* tab of the Auto P&R assistant.



2. Select *Ignore Instance Self-symmetry* to specify that instance self-symmetry constraints are to be ignored.

Select this option to restrict propagation of self-symmetry of multiple grouped instances into a single symmetry constraint. This may be undesired when a compact placement is needed.

3. Select *Arrange by Virtual Groups* to display a preview of the virtual groups before running the placer. Devices are arranged vertically according to their virtual group, as shown below:

4. In the *Groups and constraints* pane, select the circuit finders to be run to identify matching devices and device groups in the design.

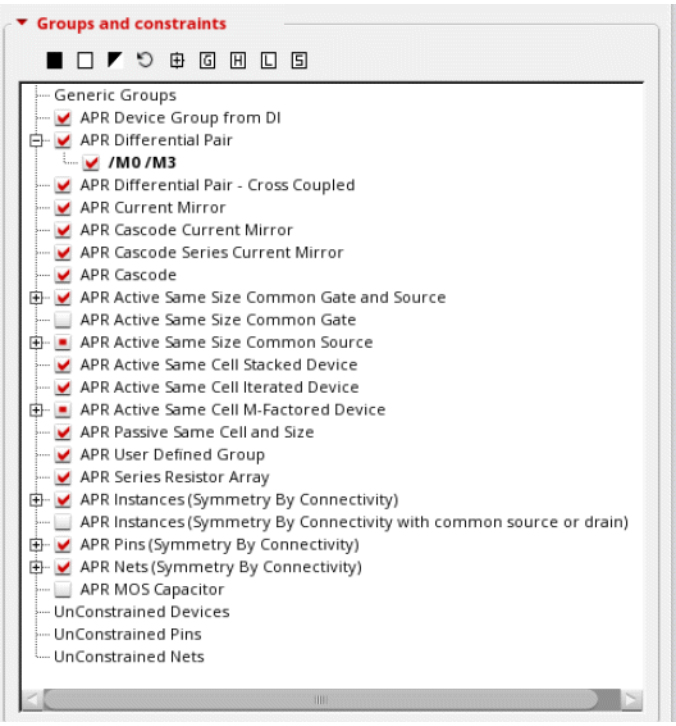
   By default, all applicable circuit finders are selected. Deselect the ones that you do not want to run. You can also use the selection options  . Click to restore default settings.

   For example, if a device pair is listed under two different constraint groups, you can select the device group for which they need to be included.

5. Click the find icon to search for all matching structures in the source cellview following the order in which they are listed. The `ciRunFinder` SKILL API is run with `hierScope` set to `allCellViews`. The finder searches for matching structures across all levels of the hierarchy.

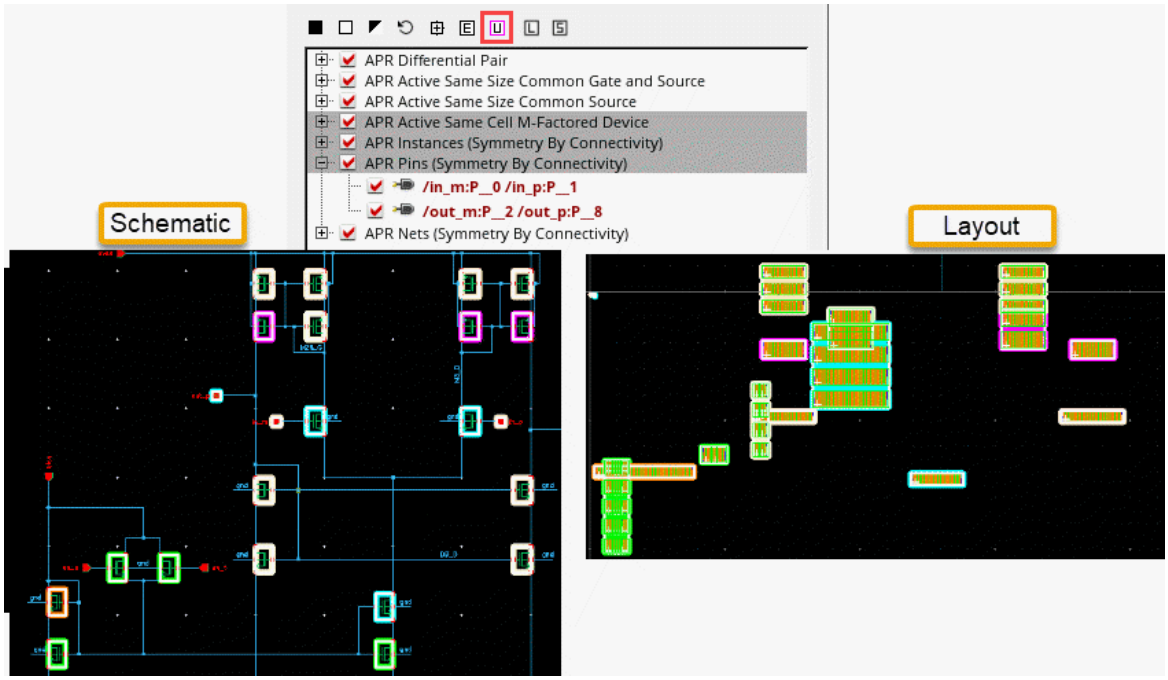
   Note: Structures with members in an already-found group are skipped.

All structures and device groups found are listed under their respective categories in the *Groups and Constraints* pane.



6. Click ⊞ to expand each group to view matching device groups. Alternatively, use the ⊞ icon above the table to expand all constraint group trees.
7. (Optional) Edit the device groups as per your requirements using the following options in the shortcut menu:
   - *Zoom*: Magnifies the selected device group in the layout canvas.
   - *Pan*: Pans the selected device group.
   - *Fix Group*: Adds a property to device groups so that they cannot be moved by the placer.
   - *Edit Group*: Invokes the Array Assistant to edit the design array constraint.
   - *Delete Constraint*: Deletes the device group.
8. Select the device groups for which constraints are to be created.
   - Use ■ □ to select all constant groups.
   - Use ⊞/⊟ to expand all constraint trees.
   - Use G/E to hide unused, empty constraint groups.

- Use H / U to highlight all instances of the selected constraint groups in the layout and schematic views. Each constraint group is highlighted in a different color.
- Use L to load the constraint order from a preset file.
- Use S to save the constraint order to a preset file.



9. Click *Create Constraints*.

New topological constraints corresponding to the matching devices groups are generated.

All identified structures are organized in Modgens along with symmetry information for symmetric structures. They are also listed in the Constraint Manager.

The device groups for which constraints are created are highlighted in red.

Note: The total number of constraints created might be lesser than the total number reported while running the finder. This is because the constraints are originally created on the individual instances and then propagated to the device groups (figGroups or Modgens). The individual instance constraints are deleted because the constraints on the device group supersede them.

The *Unconstrained Devices, Unconstrained Nets,* and *Unconstrained Pins* categories in Groups and Constraints pane list all unconstrained devices, nets, and pins 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 Topics

Constraints

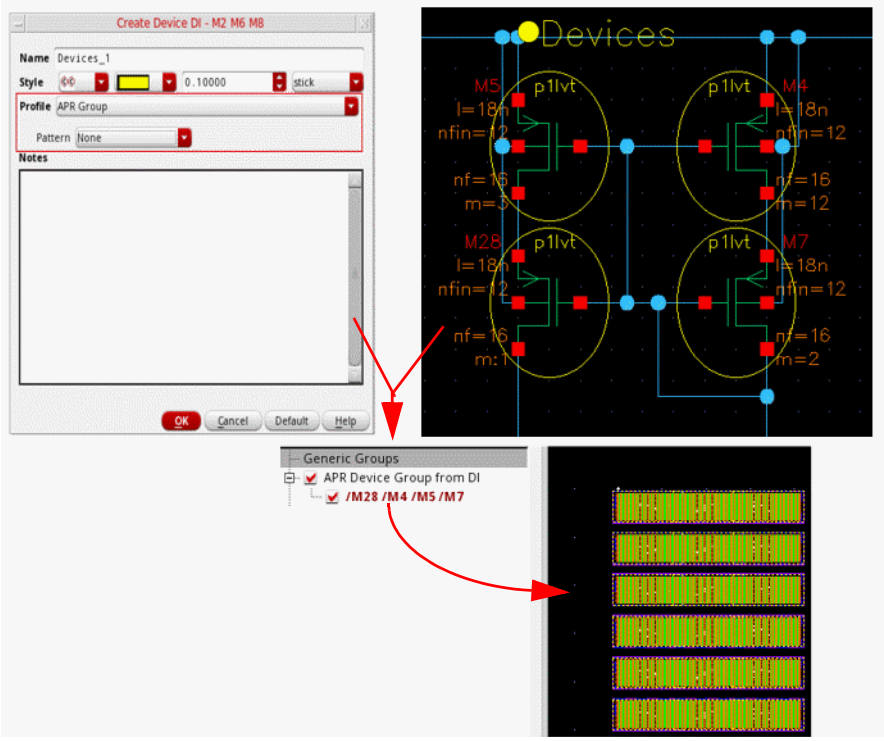
Constraints in the Automated Device Placement and Routing Flow

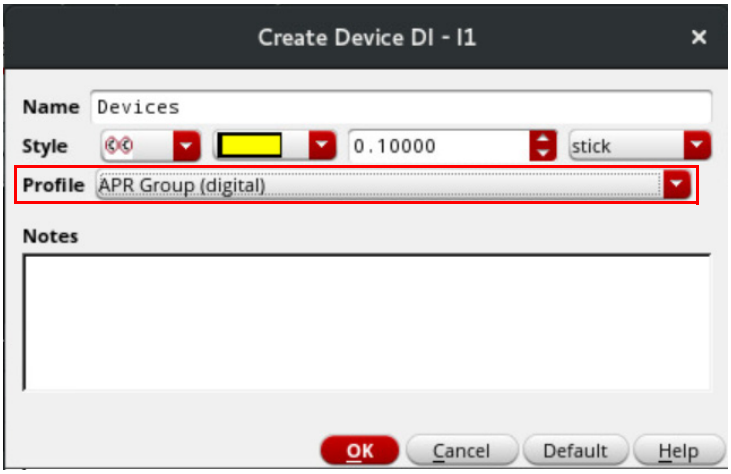
Editing a Modgen in the Automated Device Placement and Routing Flow

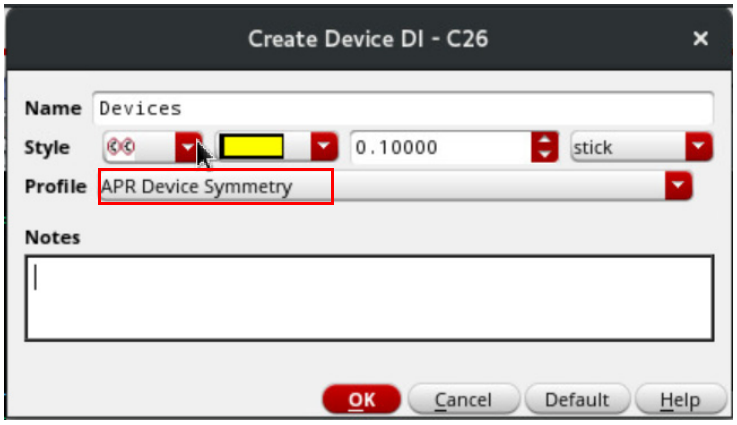
APR Circuit Finders

APR Circuit Finders

The following table describes the circuit finders that are the Auto P&R assistant supports.

APR Finders	Description
APR Analog Device Group from DI	Groups APR Group Design Intent structures that are set in schematic. In the schematic cellview, a Design Intent (DI) can be created for devices that have their `Profile` set to `APR Group`. The APR Device Group from DI constraint finder recognizes such devices and places them in a group. 

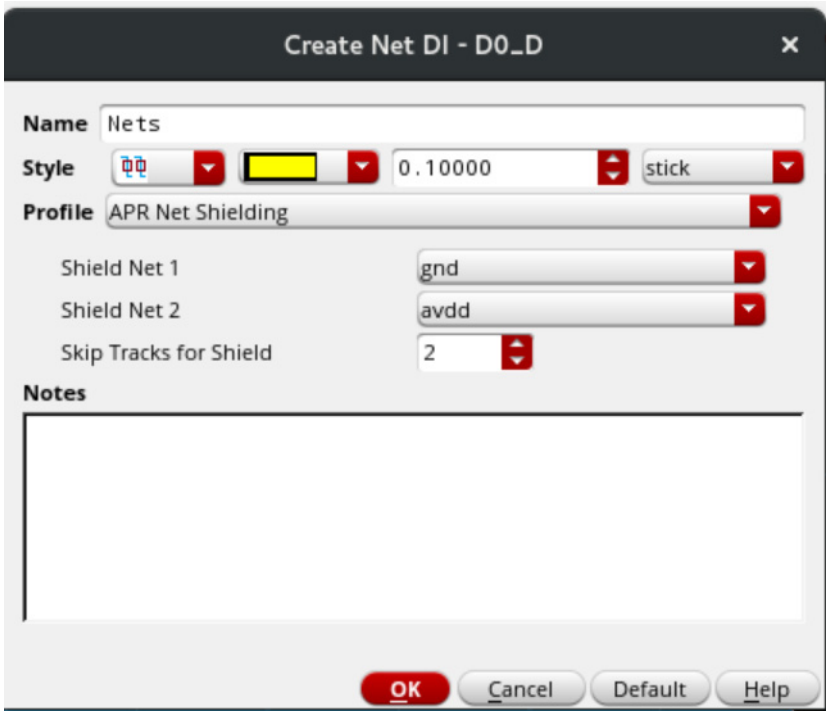
APR Custom Digital Device Group from DI	Groups APR Group (digital) Design Intent structures that are set in schematic. In the schematic cellview, a DI can be created for devices that have their `Profile` set to `APR Group (digital)`. The APR Custom Digital Device Group from DI constraint finder in the Auto P&R assistant recognizes such devices and places them in a group as complimentary pairs (P and N chained devices). 

APR Device Symmetry from DI	Groups APR Device Symmetry Design Intent structures that are set in schematic. In the schematic cellview, a DI can be created for devices that have their `Profile` set to `APR Device Symmetry`. The APR Device Symmetry from DI constraint finder in the Auto P&R assistant recognizes the devices on which placement and routing symmetry constraint design intents are defined in the schematic. These devices are grouped. 

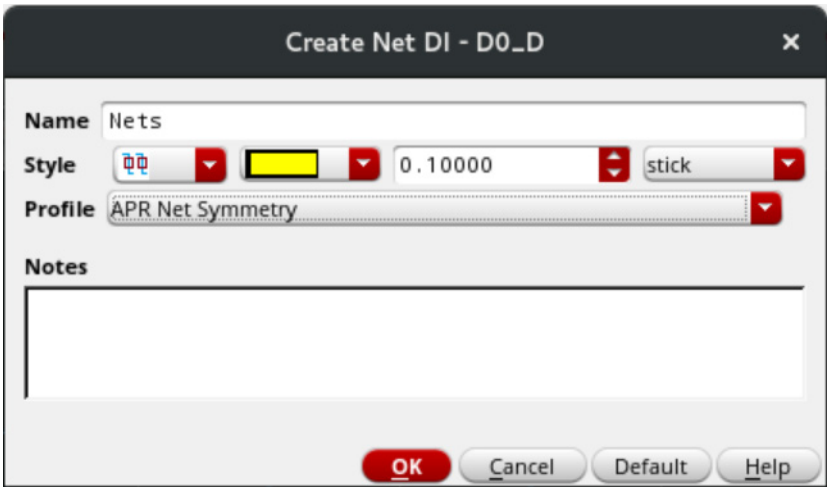
APR Net Shielding from DI

Groups APR Net Shielding Design Intent structures that are set in schematic.

In the schematic cellview, a DI can be created for nets that have their `Profile` set to `APR Net Shielding`. This DI supports only parallel shielding, which comprises a shielding wire on the left and right side of the selected net.



The APR Net Shielding from DI constraint finder in the Auto P&R assistant recognizes the nets on which net shielding design intents are defined in the schematic. These devices are grouped.

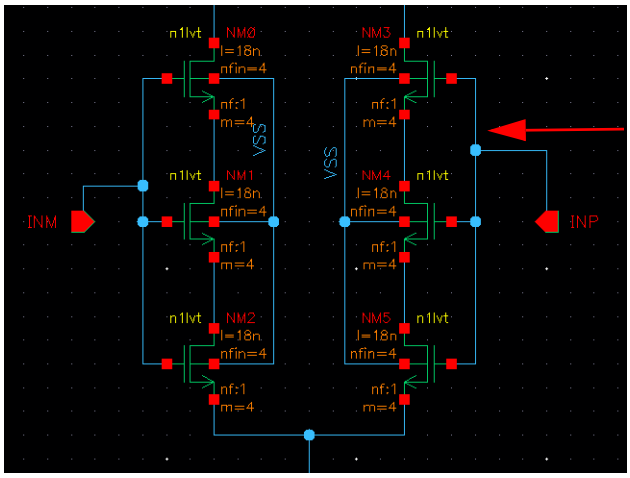
APR Net Symmetry from DI	Groups APR Net Symmetry Design Intent structures that are set in schematic. In the schematic cellview, a DI can be created for nets that have their `Profile` set to `APR Net Symmetry`.  The APR Net Symmetry from DI constraint finder in the Auto P&R assistant recognizes the nets on which net symmetry design intents are defined in the schematic. These devices are grouped.

APR Stack of Series Diffpair

Groups devices when there are more than one stacks connected in series forming a differential pair structure.

The devices must also have:

- A common bulk, if present.
- The same cell name.
- The same size (length and width) and fins.



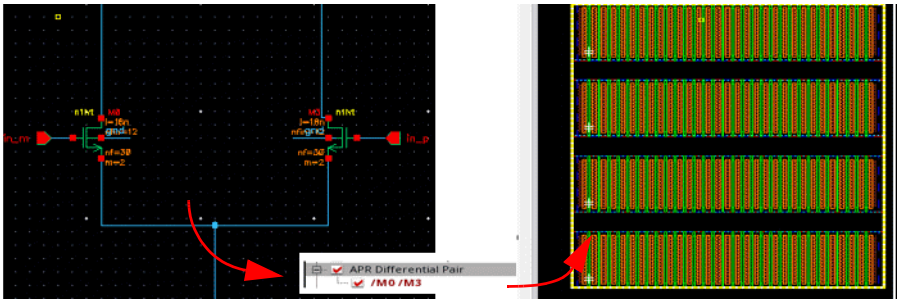
Two stacks are connected in series to form a differential pair

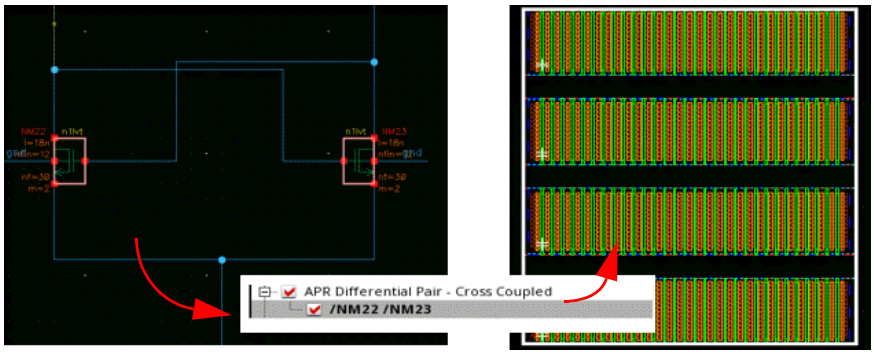
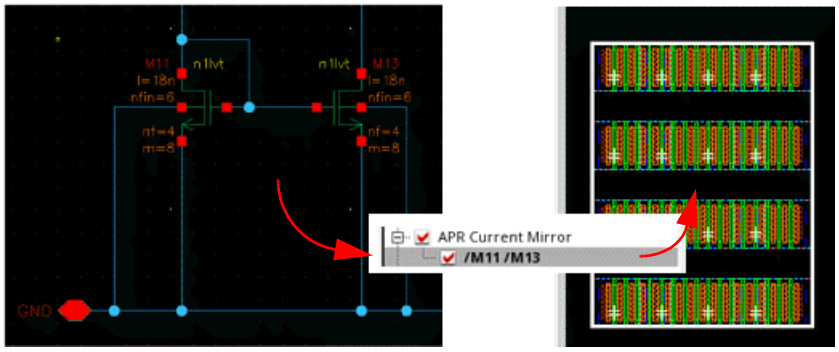
APR Differential Pair

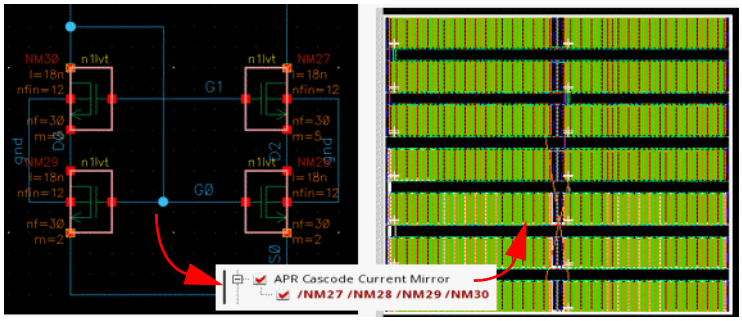
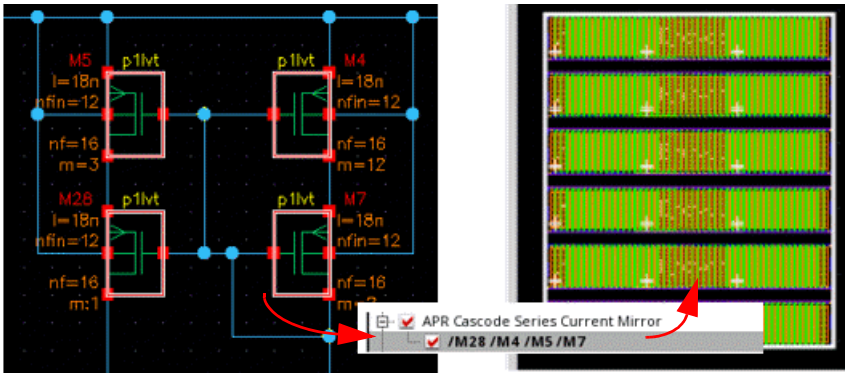
Groups devices that form a differential pair structure.

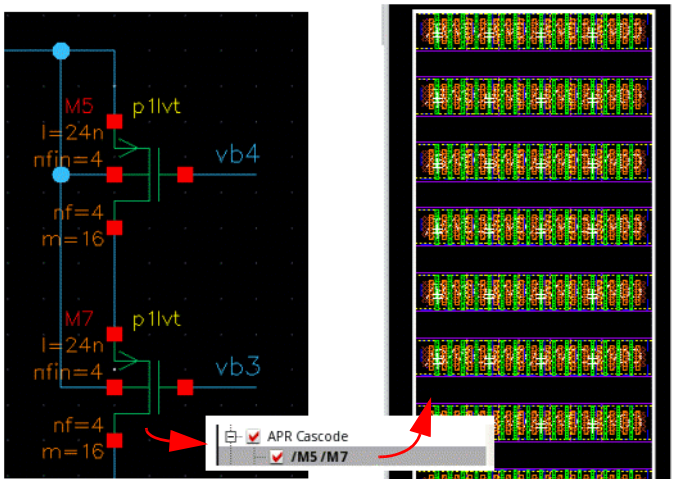
The devices must also have:

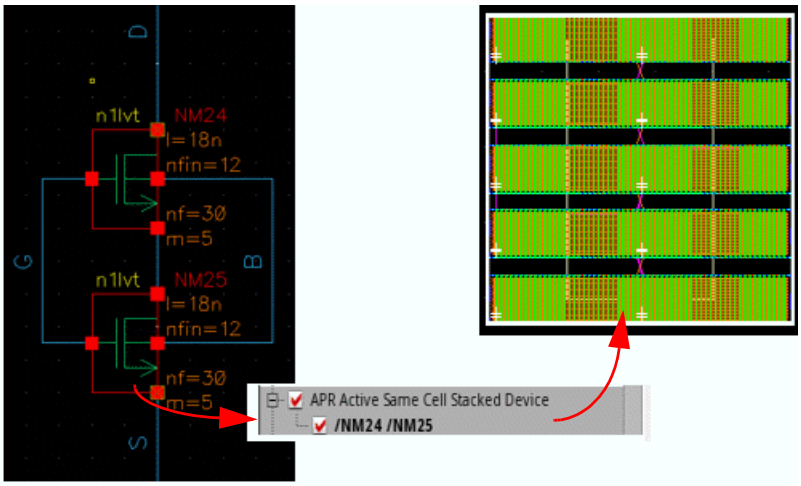
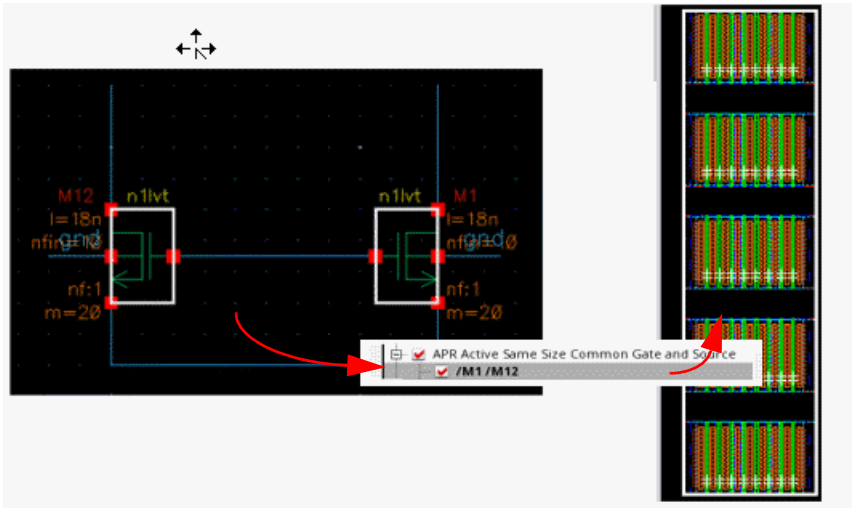
- A common bulk, if present.
- The same cell name.
- The same size (length and width), mfactor, fins, and fingers.



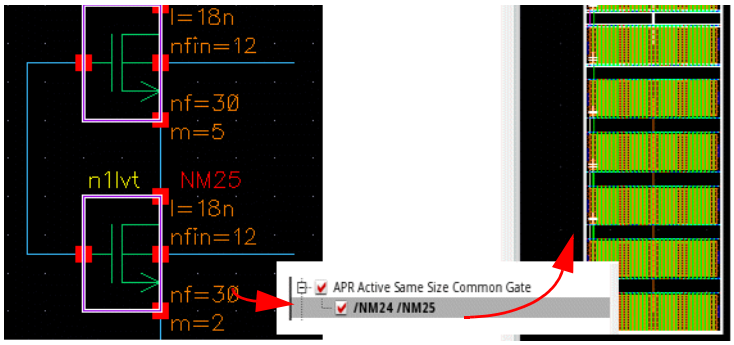
APR Differential Pair - Cross Coupled	Groups devices that form a cross-coupled differential pair structure. The devices must also have: ■ A common bulk, if present. ■ The same cell name. ■ The same size (length and width), mfactor, fins, and fingers. 
APR Current Mirror	Groups devices that form a current mirror structure. The devices must also have: ■ The same size (width and length), fins, and fingers. ■ The same cell name. 

APR Cascode Current Mirror	Groups devices that form a cascoded current mirror structure. The devices must also have: ■ The same cell name. ■ The same size (width and length), fins, and fingers. 
APR Cascode Series Current Mirror	Groups devices that form a cascoded series current mirror structure. The devices must also have: ■ A common gate. ■ The same cell name. ■ The same size (width and length), fins, and fingers. 

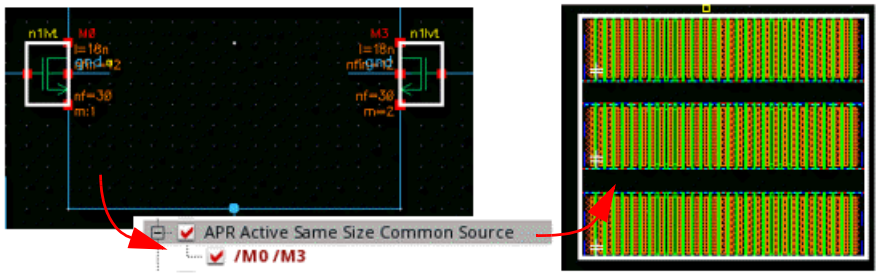
APR Cascode	Groups cascoded MOS transistor structures. The devices must also have: ■ A common gate. ■ The same cell name. ■ The same size (width and length), fins, and fingers. ■ A common bulk terminal, if any. 

APR Active Same Cell Stacked Device	Groups transistors in series stacks. The devices must have common gate and bulk terminals, if any. Also, the devices must be connected in series, where the drain of one device is connected to the source of the next device. Also, the devices must have the same size (length and width), number of fins, number of fingers, and mfactor. 
APR Active Same Size Common Gate and Source	Groups same-sized (same length and width) devices with the same gate and source connections. 

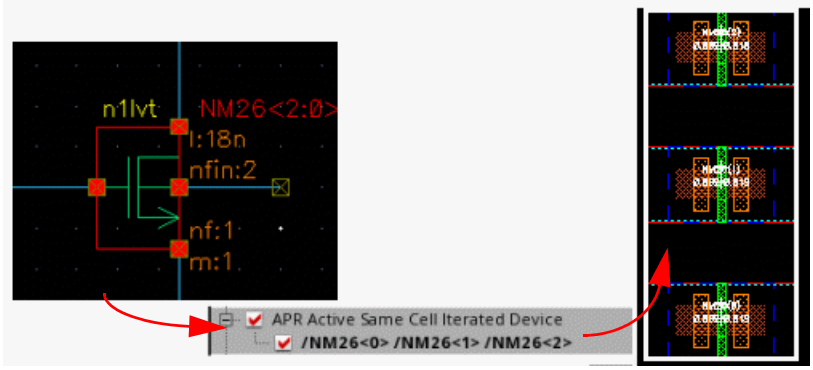
APR Active Same Size Common Gate	Groups same-sized devices (same length and width) that share the same gate connection.

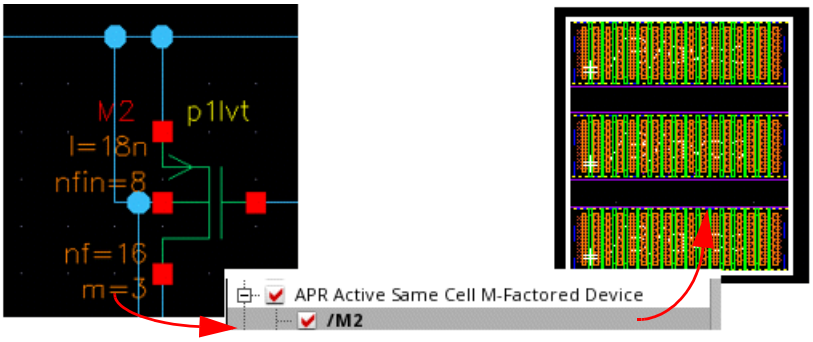
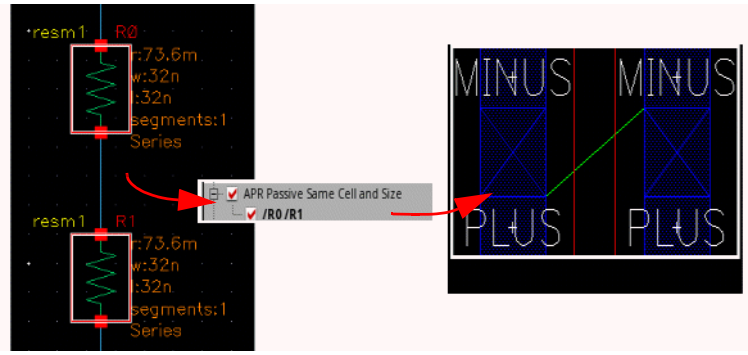


APR Active Same Size Common Source	Groups same-sized (same length and width) devices with the same source connection.



APR Active Same Cell Iterated Device	Groups iterated devices.



APR Active Same Cell M-Factored Device	Groups active devices with mfactor. 
APR Passive Same Cell and Size	Groups passive devices with same cell names or the same cell sizes and values. If more than one passive devices are found in a single group, the tool checks for devices with the same cell name, size, and value. 
APR User Defined Group	Groups device groups inside text boxes.
APR Series Resistor Array	Groups arrays of sequentially-arranged resistors that form the longest serial chain between two nets. The series chain terminates at a branch that connects to multiple resistors. Branches that connect to devices other than resistors are ignored.
APR MOS Capacitor	Groups MOS capacitors in the design. MOM capacitors are listed under the APR Passive Same Cell and Size finder, and have a higher priority in the constraints tree.

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 Pins (Symmetry By Connectivity)	Groups pins that are determined to be symmetric based on their connectivity.
APR Nets (Symmetry By Connectivity)	Groups nets that are determined to be symmetric based on their connectivity.

Related Topics

Finders

Constraints

Constraints in the Automated Device Placement and Routing Flow

Generating Constraints and Constraint Groups

Editing a Modgen in the Automated Device Placement and Routing Flow

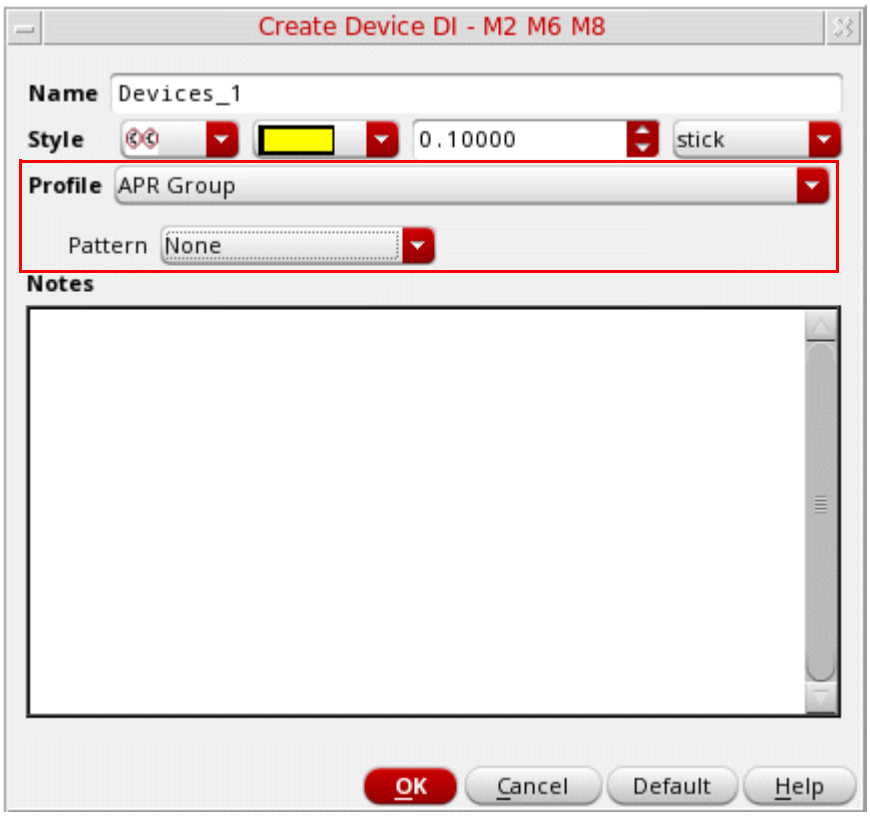
Generating Constraints for Design Intent Device Groups

The Auto P&R assistant honors the device groups defined in Virtuoso Design Intent in the schematic view. This flow involves interaction between Virtuoso Design Intent contained in Virtuoso Schematic XL and the automated device placement and routing flow contained in Virtuoso Layout XL.

To define a Design Intent:

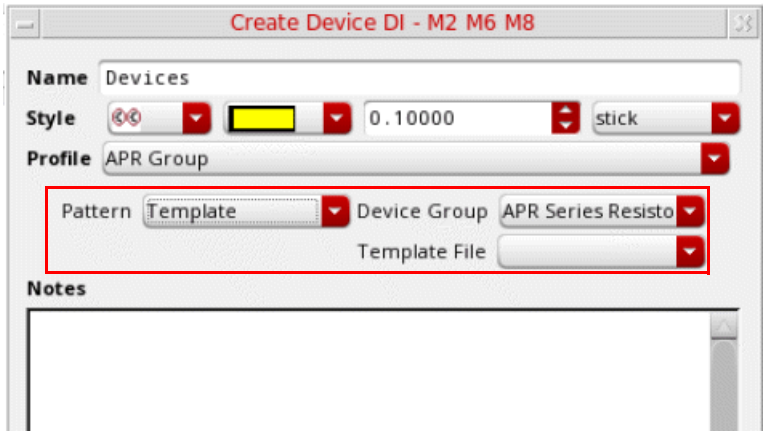
1. Select the required devices in the design canvas or Navigator assistant either in Schematic XL or Layout XL.

2. In Schematic XL, right-click and choose *Design Intent – Create Design Intent – Devices*. The Create Device DI form appears.



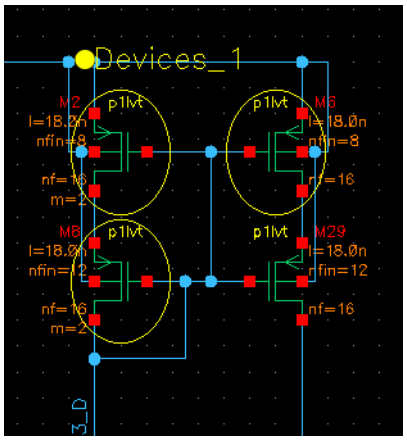
3. Specify a design intent name in the *Name* field.
4. Select a design intent annotation style from the *Style* drop-down list.
5. Select the annotation color, font size, and font from the corresponding drop-downs.
6. Set *Profile* to *APR Group*.
7. Select a device pattern from the *Pattern* drop-down. The available options are:
   - *None* (default): Does not apply any pattern preset to the selected devices.
   - *Preset*: Lets you set the pattern preset to *Uniform*, *Interdigitated*, *Compact*, or *Custom*.
   - *Template*: Lets you select the following:
     - *Device Group* for the selected devices

- *Template File* that contains the settings to be applied to the devices.



8. Click *OK* to create the Design Intent.

Design intent annotations are displayed in the design canvas to indicate the devices for which design intent is defined.

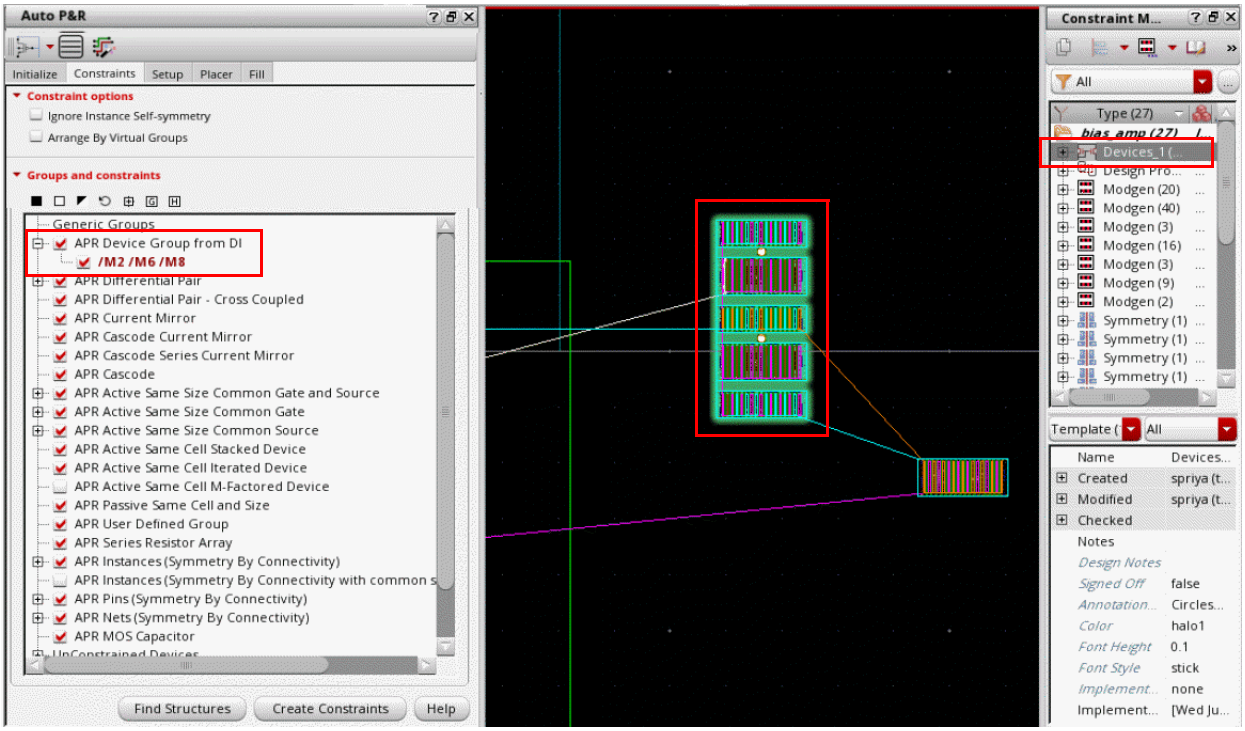


The Auto P&R assistant honors the Design Intent and creates a corresponding constraint as part of the *Constraints* step.

To generate the required constraints in the Auto P&R assistant:

1. Launch the Auto P&R assistant.
2. Generate devices in the layout canvas using options on the *Initialize* tab of the Auto P&R assistant.
3. On the *Constraints* tab, ensure that the *APR Device Group from DI* finder is selected to recognize Design Intent device groups.

4. Click *Find Structures*. The Design Intent device groups are identified and listed under the finder.
5. Click *Create Constraints* to create a constraint for each Design Intent device group.



The constraint is listed in the Constraints Manager.

Editing a Design Intent

You can edit a design intent using the Edit Design Intent form. To access the form:

1. Right-click the constraint that contains the required design intent.

2. Choose Edit DI from the shortcut menu to display the Edit Design Intent form.

Use the options in the Edit Design Intent form to update the design intent.

Related Topics

Generating Constraints and Constraint Groups

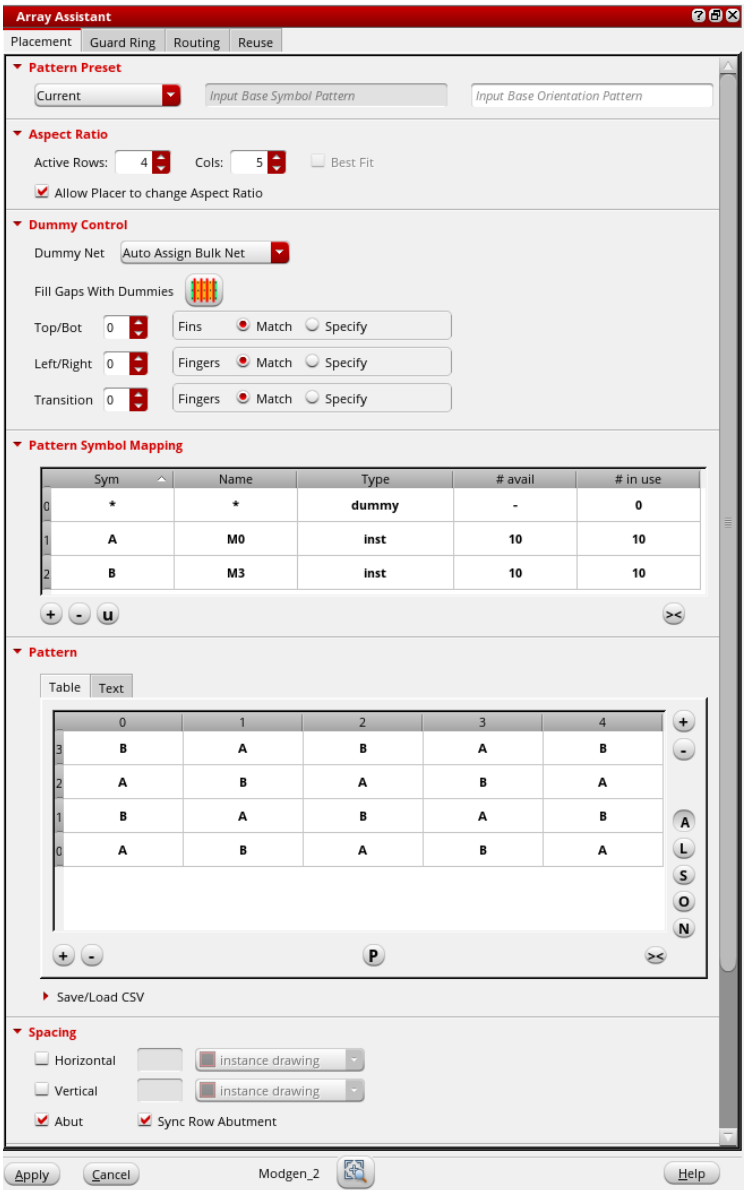
APR Circuit Finders

Edit Design Intent Form

Editing a Modgen in the Automated Device Placement and Routing Flow

To edit a Modgen from the Auto P&R assistant, after generating the constraints:

1. Expand the required category in the *Groups and Constraints* pane on the *Constraints* tab of the Auto P&R assistant.
2. Right-click the required Modgen.
3. Choose *Edit Group*. The Array Assistant is displayed.



The Array Assistant is a unified interface that lets you to quickly create and edit Modgens. The form integrates the key options from the Modgen Placement toolbar and Modgen Routing toolbars.

To delete a Modgen, right-click the Modgen and select *Delete Constraint* from the shortcut menu.

Related Topics

Automatic Generation of Modgens using the Array Assistant

Constraints

Constraints in the Automated Device Placement and Routing Flow

Generating Constraints and Constraint Groups

Editing a Modgen in the Automated Device Placement and Routing Flow

Setting Options for Custom Layout Design Migration

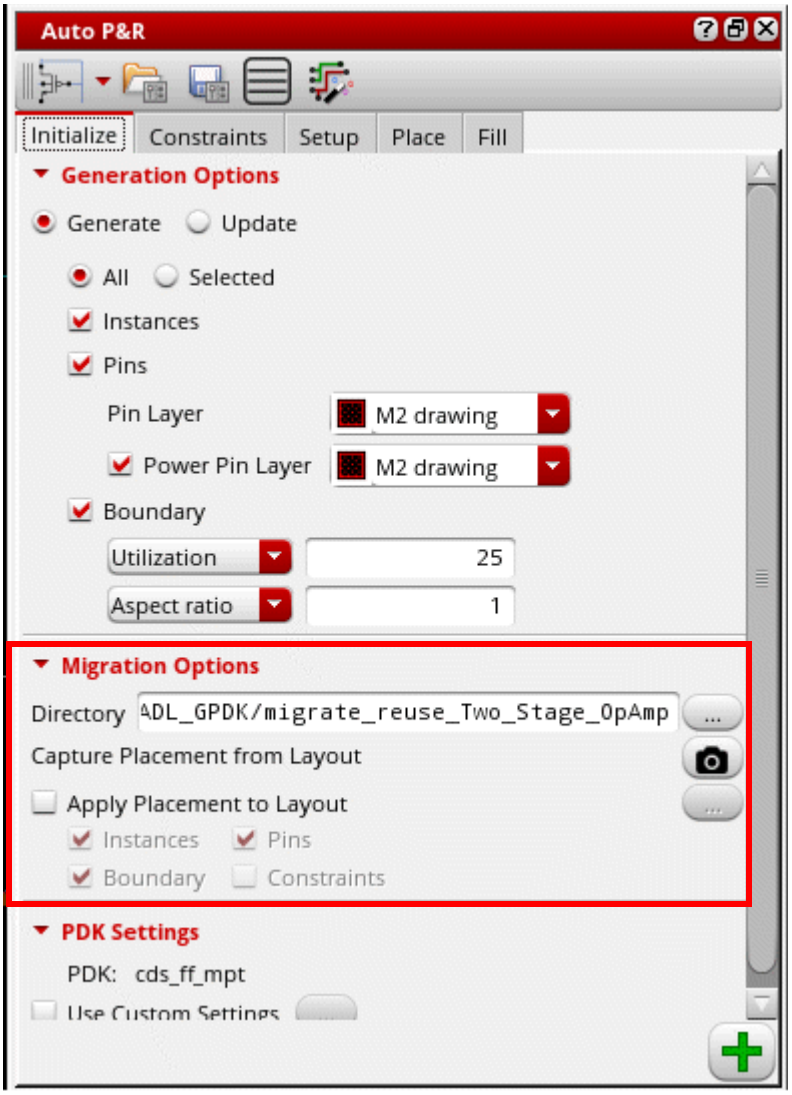
The Auto P&R assistant supports the assisted flow of the Virtuoso® Custom Design Migration solution. In Layout MXL, the following tabs of the Auto P&R assistant include an additional *Migration Options* section:

- *Initialize* tab: Captures design placement data, for example, the PR boundary, pins, instances, and routing shapes, from a source layout and applies the captured data to a target layout.
- *Constraints* tab: Captures groupings from a source layout and applies the captured data to a target layout.

To capture source data and apply it to a target layout:

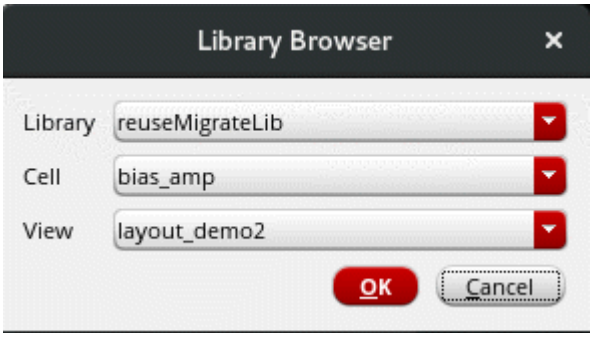
1. Open the source layout in Layout MXL.
2. Open the `Auto_Place_Route` workspace.
3. In the *Migration Options* section of the *Initialize* tab of the Auto P&R assistant, specify a *Migration Directory*. The captured source data is stored in this directory.

By default, the value is set to the current working directory in the format `"/migrate_reuse_<libName>"`.

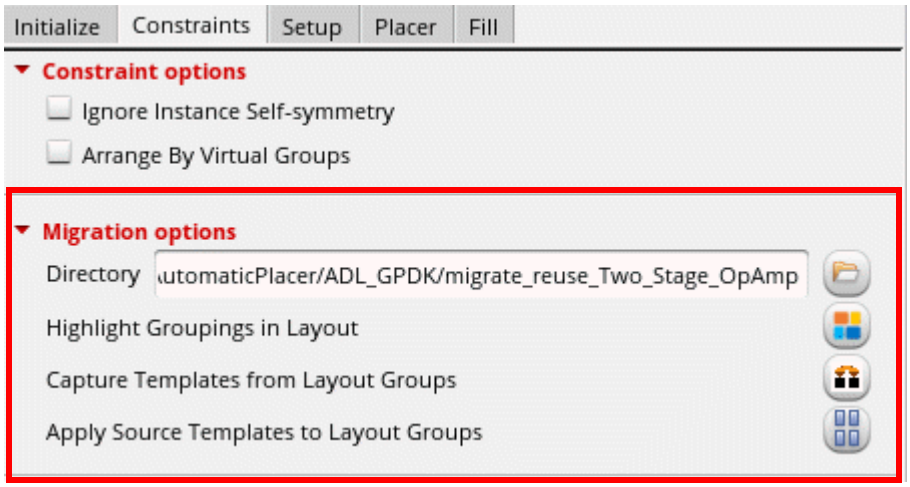


4. Click the *Capture Placement from Layout* button to capture data from the source layout.
5. Select *Apply Placement to Layout* to apply the captured placement data to a target layout.
6. Select the required settings to be applied to the target layout — *Instances*, *Pins*, *Boundary*, and *Constraints*.

7. Click the Ellipses button and select the target layout.

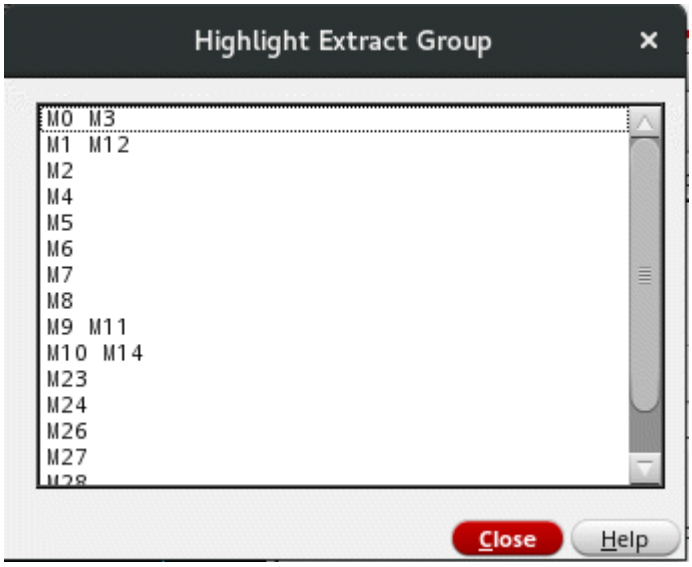


8. Click  to apply the captured data to the target layout and initialize the design.
9. Open the *Constraints* tab.



By default, *Directory* is set to the same as *Migration Directory* on the *Initialize* tab. If you make any changes, the value is automatically reset.

10. Click *Highlight Groupings In Layout* to display the Highlight Extract Group form.

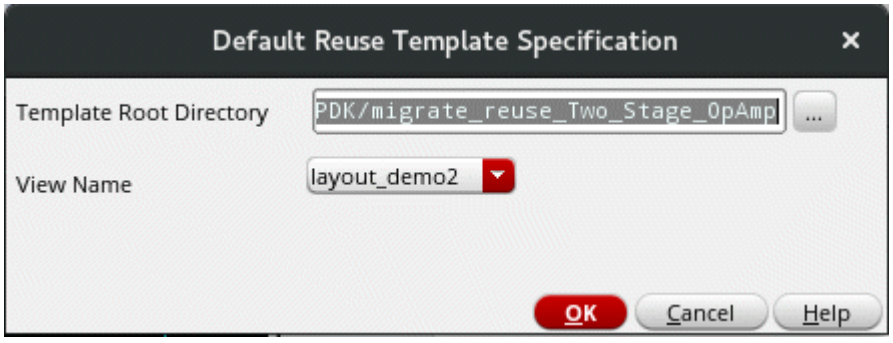


11. Select the groupings to be highlighted in the source layout. The groupings are highlighted in the design canvas.

12. Click *Close* to close the form.

13. Click the *Capture Templates from Layout Groups* button to capture groupings from source layout.

14. Click the *Apply Source Templates to Layout Groups* button. The Default Reuse Template Specification form appears.



15. By default, the *Template Root Directory* is the same as the directory in which the template was initially captured.

16. Select the view from which the groupings were captured.

17. Click *OK*.

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Generating row regions and grids is an optional task, but is highly recommended to obtain better results in the automated device placement and routing flow. During placement, devices are snapped to their nearest row regions or grids. In the absence of row regions and grids, coarse placement is done, where approximate placement locations are determined based on the design constraints and congestion.

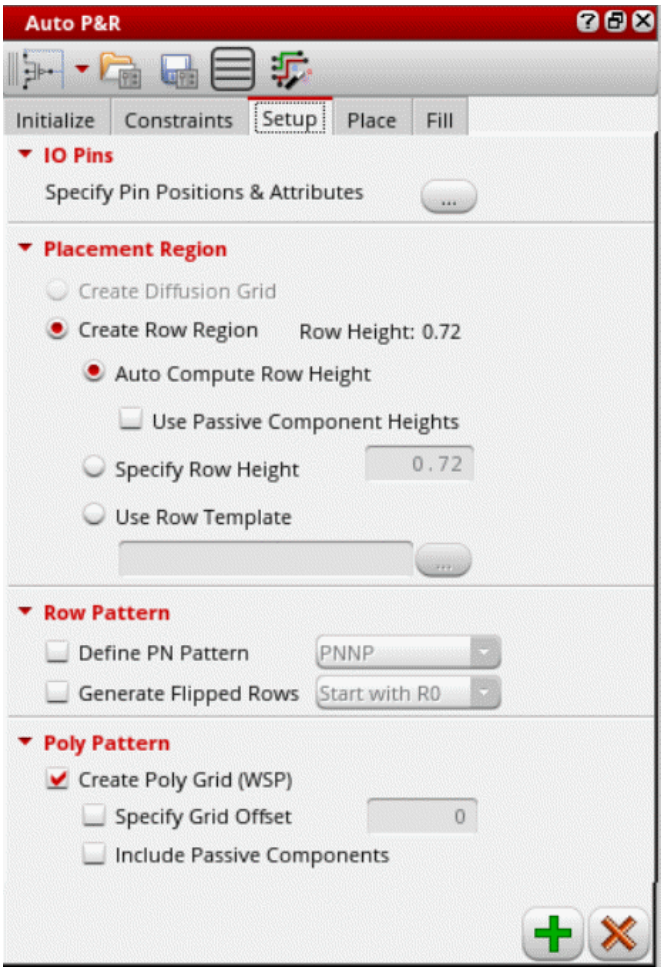
At advanced nodes, grids—including fins, WSPs, poly snap patterns, and rows—are important for the optimal placement and routing of custom and analog layouts. Row regions or diffusion grids are used to facilitate device placement and derive WSP information, which used for device routing and snapping.

The automated device placement and routing flow lets you derive poly and diffusion grids or row regions automatically based on the device footprint, layers, and DRCs, with minimal user input. Row regions are created in the layout based on the row templates.

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

To derive grids and row regions:

1. Open the *Setup* tab of the Auto P&R assistant.



2. Click the button beside *Specify Pin Positions & Attributes* to open the *Pin Planner* tab of the Pin Placement form.

   Use the options on this form to set pin constraints and plan the placement of pins in the design.

3. Complete the required pin settings and close the Pin Placement form.

4. Select a mode to specify the placement region. The *Create Diffusion Grid* option is available only if the enablePlaceWithWsp environment variable to `t`.

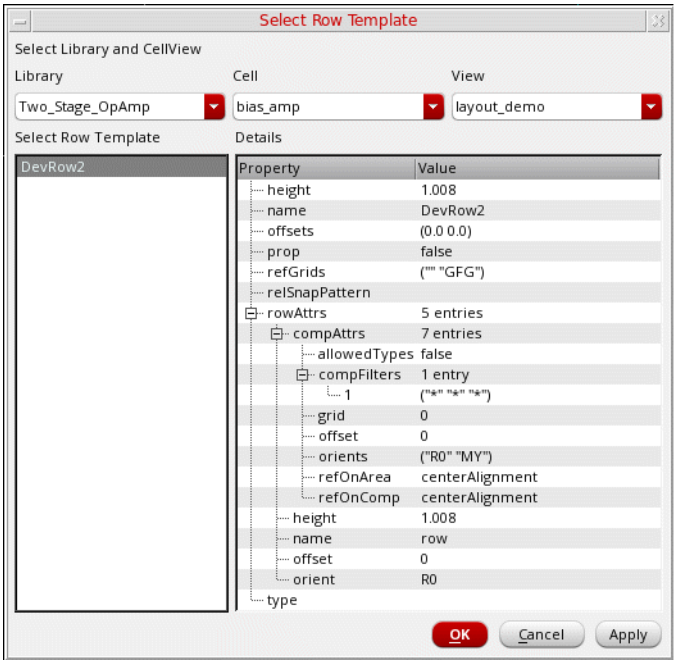
   For more information, see Placing Multi-Height Devices Using Automatic Device Placer.

5. Select *Create Row Region* to specify whether rows are to be created in the row region.

6. Select one of the following options:

- ❑ *Auto Compute Row Height* to automatically calculate the row height
- ❑ *Specify Row Height* to input the row height
- ❑ *Use Row Template* to import row templates

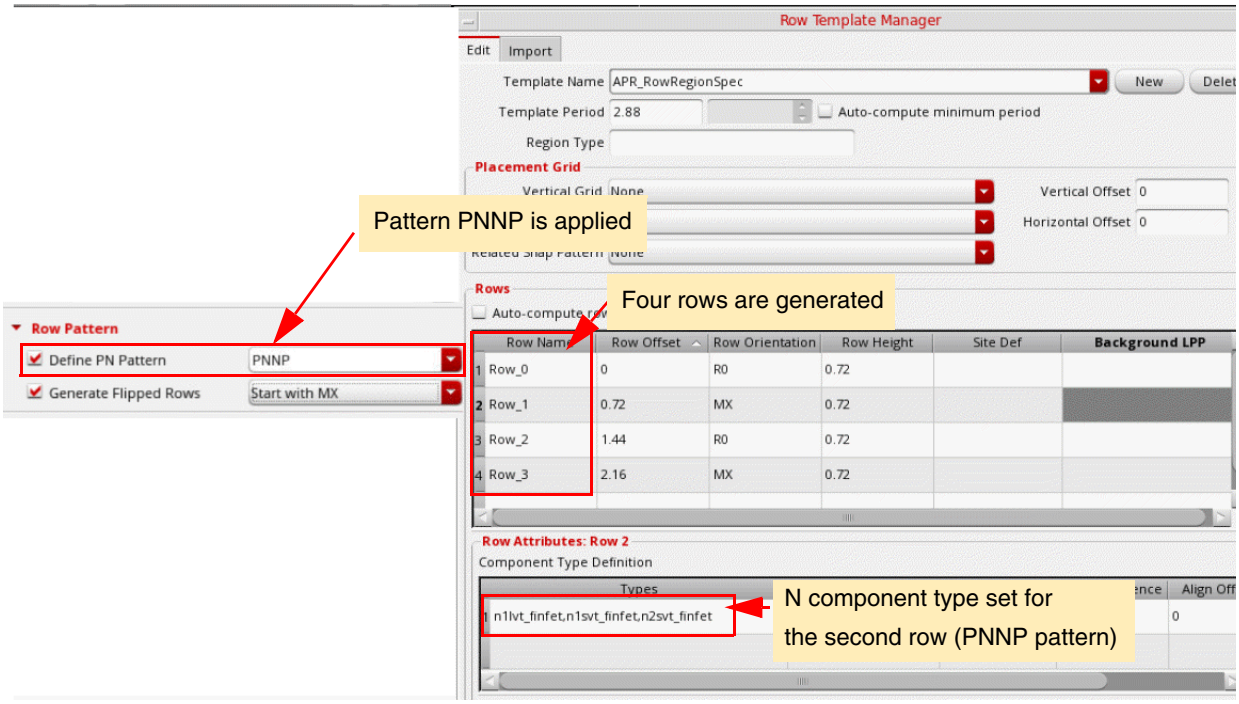
7. If *Auto Compute Row Height* is selected, select *Use Passive Component Heights* to consider passive components for row height calculations.
8. If *Use Row Template* is selected:
   a. Click *Browse* to display the Select Row Template form.



   b. Select the required *Library*, *Cell*, and *View*. All templates stored in the selected cellview are listed in the *Select Row Template* box.
   c. Select the required row template.
   d. Check the row template properties in the *Details* panel to ensure that they meet your requirements.
   e. Click *OK*.

   The name of the selected row template is displayed in the *Use row template* field.
9. Select *Define PN Pattern* to control the distribution of P and N devices in rows.
10. Either type the required PN pattern or select a PN pattern from the drop-down list.

In the following example, the `PNNP` pattern is selected. Multiple row attributes with their component types set are created.



11. Select *Generate Flipped Rows* to allow generation of flipped rows, as shown in the above example.

    Select an orientation with which the first (bottom-most) row must start: *Start with R0* or *Start with MX*.

12. Select *Create Poly Grid (WSP)* to create poly layer patterns as WSPs. If the devices have different poly pitches, multiple poly patterns are created.

13. Select *Specify Grid Offset* to specify an offset for the poly grid.

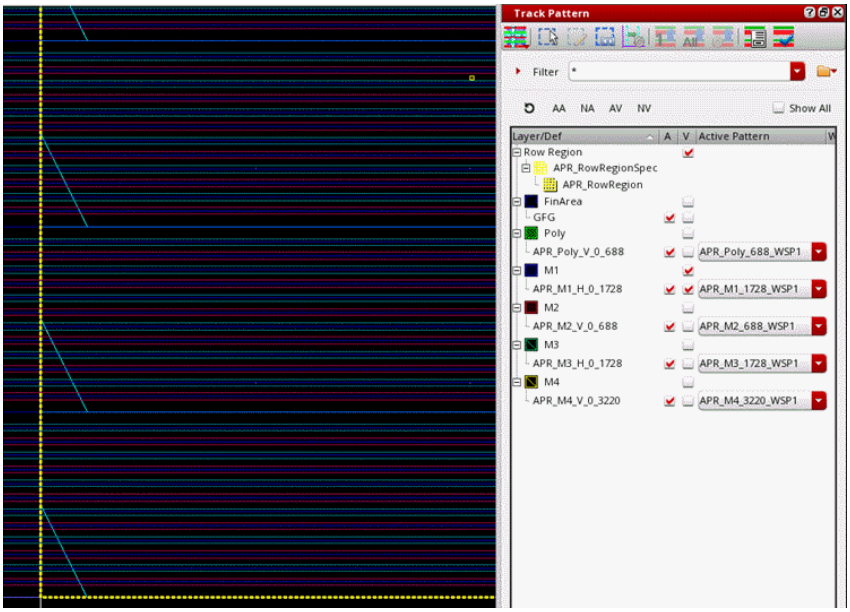
    Specify an offset value in the adjoining text box.

14. Select *Include Passive Components* to recognize the poly layer patterns of passive devices while generating WSPs.

15. Click *Apply* to create a row template, a row region, rows, and poly patterns, as specified. The P/N pattern selection controls how rows are created.

    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.

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

3

Device Placement Using the Auto P&R Assistant

In the Virtuoso automated device placement and routing flow, device placement is done after initializing the layout and generating the required constraints and tracks. The Virtuoso automated device placement and routing flow supports two placement modes:

- **Automatic Placement:** Devices are placed 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.

Related Topics

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Placing Devices Interactively in the Automated Device Placement and Routing Flow

Placement of Non-Uniform Devices and Passive Devices

In addition to uniform devices, designs may contain devices that are non-uniform and passive.

Non-uniform devices have certain parameters that do not match those of the other devices in the design. For example, they may have different heights, gate lengths, pitches, bulks, or voltage thresholds (VTs).

Passive devices include resistors, capacitors, inductors, transformers, and transmission lines that might occupy most of the design area.

Honoring non-uniform and passive devices might adversely impact the quality of placement of active components, and thereby the placement results.

To overcome this and to ensure correct placement, the device placer automatically creates virtual groups (VGs) of all non-uniform devices in a design. Virtual groups segregate devices into groups of uniform device types.

The *Constraints*, *Setup*, *Placer*, and *Fill* tabs of the Auto P&R assistant include options that let you view and control certain aspects of virtual groups and passive devices.

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Base Layer Fill in the Automated Device Placement and Routing Flow

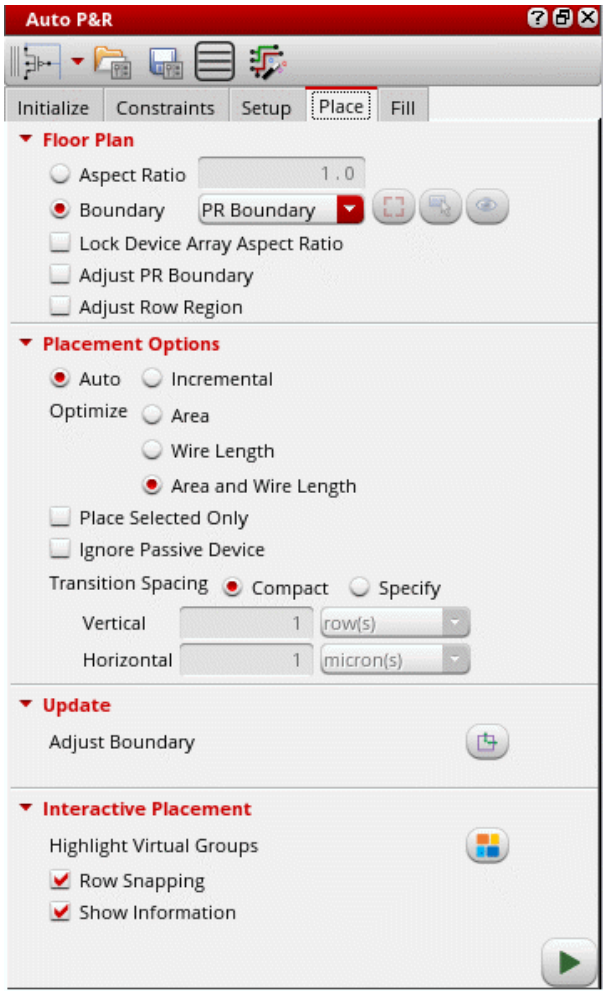
Placing Multi-Height Devices Using Automatic Device Placer

Placing Devices Automatically in the Automated Device Placement and Routing Flow

The Virtuoso device-level automatic 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. The placer reshapes Modgens, unless they are locked, following the specified transition spacing.

To run the Virtuoso device-level automatic placer:

1. Open the *Place* tab of the Auto P&R assistant.



2. Select the placement region in the *Floor Plan* section by either specifying an *Aspect Ratio* or by selecting *PR Boundary* or *Custom Area* in *Boundary* mode.

3. If *Custom Area* is selected, click the Draw icon and draw the required area boundary in the layout canvas. When you draw a new custom area, the previous one is automatically deleted.

4. Select *Lock Device Array Aspect Ratio* to lock the aspect ratio of all Modgens when running the placer.

5. Select *Adjust PR Boundary* to run the placer unconstrained by the current PR boundary.

   Devices are placed so as to achieve better QoR and wire length. After placement, the PR boundary is adjusted accordingly.

6. Set the placement mode to *Auto* to run the placer on the entire design.

   To incrementally place non-optimally placed devices, select *Incremental*.

7. Select one of the following settings in the *Optimize* section:

   - *Area:* Place devices to optimize compactness.
   - *Wire Length* (*Auto* mode)*:* Minimizes the wire length for better routability.
   - *Area and Wire Length* achieves a balance between compact placement and reduced wire length.

   In *Incremental* mode, the following options are available:

   - *Similarity* considers the placed devices as a reference to achieve similar placement of additional devices.
   - *Area and Wire Length* achieves a balance between compact placement and reduced wire length.

8. Select *Place Selected Only* to run the placer only on those the objects that are selected in the layout canvas. These objects are placed inside the selected placement *Boundary*.

9. Select *Ignore Passive Device* to ignore passive components while running the placer.

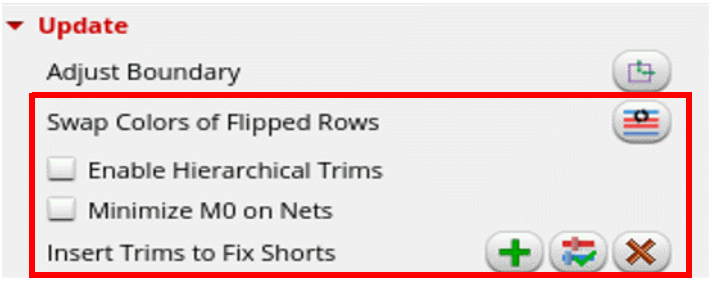
10. If your process node supports it, use the *Insert Trim to Fix Shorts* option to add trims during the placement run in order to avoid shorts.

11. Set *Transition Spacing* to either *Compact* or *Specify.* Use the fillRegionHonorTransitionSpacingenvironment variable to control the direction in which the transition spacing is to be applied when inserting device fill.

12. When set to *Specify*, add absolute vertical and horizontal spacing values. Also set the unit for vertical spacing to *microns* or *rows* when row regions are created and *microns* if diffusion grids are created.

13. Click the *Adjust Boundary* icon in the *Update* section to resize the PR boundary such that it covers any devices that were earlier placed outside the PR boundary.

14. You can perform the following additional tasks in certain advanced node flows:



a. Select *Swap Colors of Flipped Rows* to automatically update the color swap parameters in the active cells such that the same-masked colors of the lowest metal layer do not face each other.

b. Select *Enable Hierarchical Trims* to enable trim insertion in hierarchical designs.

c. Select *Minimize M0 on Nets* to minimize the capacitance on M0 nets in addition to the regular trim insertion. The length of the M0 wires is trimmed using multiple trims to minimize the capacitance.

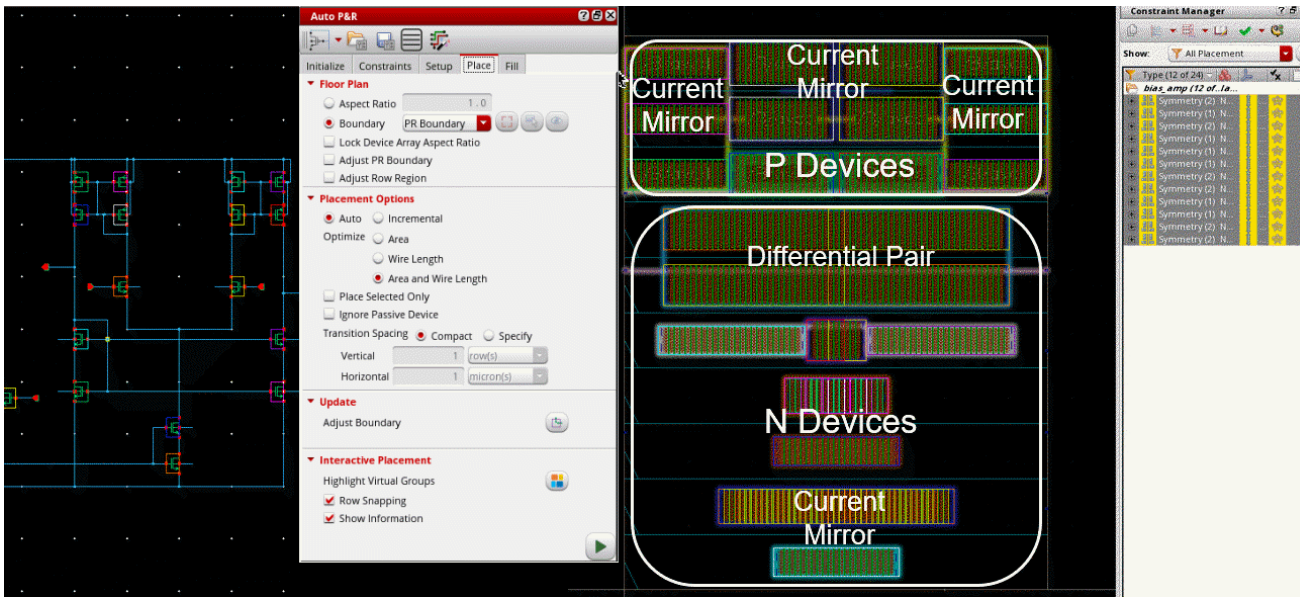
d. In *Insert Trims to Fix Shorts*, you can add trims to devices in the current cellview or PR boundary, validate trims, and delete trims

15. Click the  icon to 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.

After running the placer, the devices are placed as per your specifications and a placement report is displayed in the CIW.

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

The placer cannot run on a design with base layer fill. If base layer fill is detected, a message that provides the option to delete the fill is displayed. Select one of the following options:

- *Yes*: Deletes top-level fill and runs the placer.
- *No*: Stops the placer and retains the top-level fill.

Related Topics

Placement of Non-Uniform Devices and Passive Devices

Placing Devices Interactively in the Automated Device Placement and Routing Flow

Placing Multi-Height Devices Using Automatic Device Placer

Automatic Placement Report

After running the Virtuoso device-level automatic placer, a placement report is displayed in the CIW.

```
INFO (AP-10003): Completed runtime 9.16 seconds
                    Area    Width   Height Aspect   RowUtil   CoreArea CoreWidth CoreHeight WireLength
                (micron^2) (micron) (micron)  Ratio (percent) (micron^2)  (micron)   (micron)   (micron)
    solution       130.22    16.75     7.78   2.15        24     129.13     16.75       7.71     264.00

                Violations         Spacing        Overlap     Constraint
    solution                             0              0              0
```

The placement report includes the following placement information:

Title	Description
`Area`	Product of `Width` and `Height` values. The area is calculated in `dbu`, converted to user units, and then rounded off to two decimal places.
`Width`	Width of the PR boundary after running the placer.
`Height`	Height of the PR boundary after running the placer. The width and height values are measured from pins and instances and could be smaller or larger than the PR boundary.
`Aspect Ratio`	Width-to-height ratio of the PR boundary.

`RowUtil`	Percentage of row area that is filled with top-level instances, Modgens, and other figGroups. If a Modgen has a gap, the entire Modgen width is considered as occupied.
`CoreArea`	Area occupied only by instances. The value excludes pins.
`CoreWidth`	Width of the area occupied only by instances.
`CoreHeight`	Height of the area occupied only by instances.
`WireLength`	Total wire length achieved after running the placer. The value is measured using the half-perimeter distance model in which half of the perimeter of the bbox is considered the approximate wire length.
`Violations`	Number of violations reported during placement.
`Spacing`	Number of spacing violations that have caused DRC errors.
`Overlap`	Number of overlaps that have caused DRC errors.
`Constraint`	Number of constraint violations.

Caution

The lower left of the PR boundary is always included when measuring the width, height, area, coreWidth, coreHeight, and coreArea values.

Related Topics

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Place (Auto P&R assistant, Device mode)

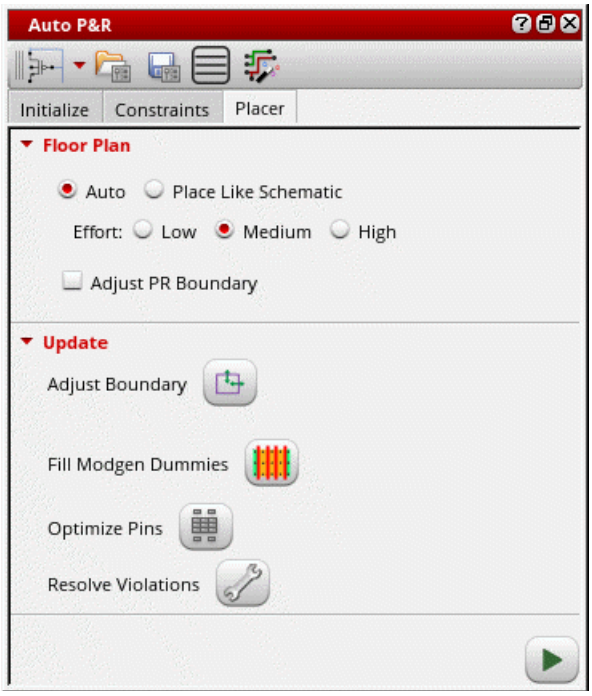
Placing Devices in Mature Node Designs

The Virtuoso device-level automatic placer is not restricted only to advanced node designs. You can run the placer on mature nodes designs as well after making a few initial settings. The placement options for mature nodes are different from those for advanced nodes.

To run the placer on a mature node design:

1. Open the required layout cellview.
2. Set the matureOrAdvancedNode environment to `mature`. When set to `auto`, the tool reads the virtuoso techfile constraints to automatically determine an appropriate process node.
3. Open the Auto P&R assistant.
4. Generate the design using the options on the *Initialize* tab. See Initializing a Layout in the Automated Device Placement and Routing Flow.
5. Generate the required constraints using the options on the *Constraints* tab. See Generating Constraints and Constraint Groups.
6. Open the *Place tab*.
7. In the *Floor Plan* section, set the placement mode to one of the following:

- *Auto*: Lets you specify the placement settings before running the placer. The placer optimizes for area or wire length and avoids design rule violations.



- *Place Like Schematic*: Places objects in locations similar to those in the corresponding schematic.

8. With *Floor Plan* set to *Auto*, select a placement effort level — *Low*, *Medium* or *High*.

   The runtime of the placer depends on the effort chosen. The time for each effort level increases in roughly linear increments. For example, if a design takes five minutes to place on *Low*, then it would take ten minutes on *Medium*, and fifteen minutes on *High* effort levels.

   Try the *Low* effort level first, because many designs perform well at this level. *Low* effort level is best suited for regular, structured designs with no non-conformal group boundaries. If your design is less structured or includes a large number of constraints, run the design at the *Medium* effort level. You may also consider this effort level if you have a few conformal group boundaries under placer control. If a design fails to meet your expectations at *Medium* effort, then increase the effort level to *High*. *High* effort utilizes the same algorithms as *Medium*, but exerts itself more fully on each step of the process.

9. Click  to run the placer.

10. Click *Adjust Boundary* icon in the *Update* section to resize the PR boundary such that it covers any devices that were earlier placed outside the PR boundary.

11. Click *Fill Modgen Dummies* to fill the gaps between Modgens selected on the layout canvas with Modgen dummies. If there are no Modgens selected, then dummies are added to fill the gaps of all the Modgens inside the PR Boundary on the layout canvas.

12. Click *Optimize Pins* to run the pin optimizer to position pins in a manner that helps obtain the shortest possible net length.

13. Click *Resolve Violations* to resolve DRC violations and any overlaps happened during placement.

Related Topics

Initializing a Layout in the Automated Device Placement and Routing Flow

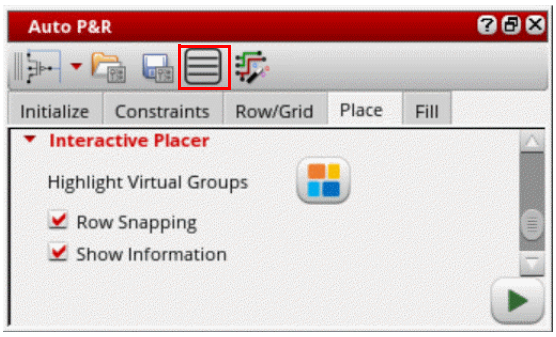
Generating Constraints and Constraint Groups

matureOrAdvancedNode

Placing Devices Interactively in the Automated Device Placement and Routing Flow

To run the device-level interactive placer:

1. Open the *Place tab* of the Auto P&R assistant.



2. Select the Context Menu icon in the toolbar to display a shortcut menu whenever one or more devices are selected in the layout canvas. You can perform context-sensitive operations on devices and device groups directly in the layout canvas.
3. Select *Highlight Virtual Groups* to highlight all virtual groups in the layout canvas. The icon automatically changes to *Dehighlight Virtual Group*, which lets you remove the highlights.
4. Select *Row Snapping* to display row snapping-related details in the *Information* pane during interactive placement of devices.
5. Hover over devices to display a context-sensitive menu that provides interactive placement commands to refine placement of devices and device groups.
6. Select the required interactive placement commands.

Related Topics

Interactive Placement Commands

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Placing Multi-Height Devices Using Automatic Device Placer

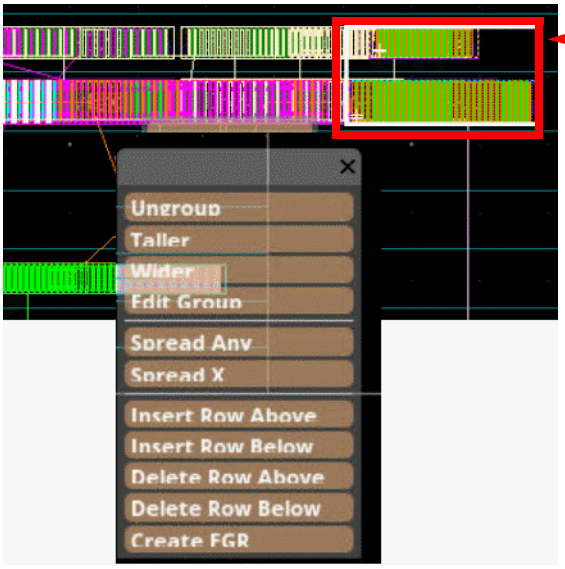
Interactive Placement Commands

Select the Context Menu button in the toolbar of the Auto P&R assistant to display a shortcut menu when you hover over devices and device groups.

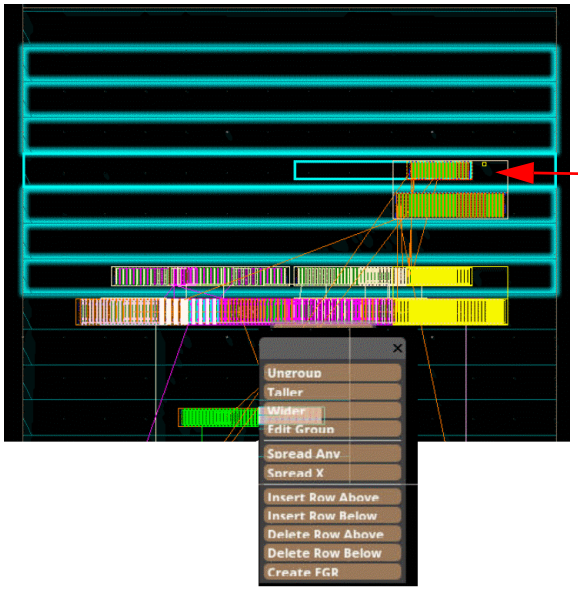
Use the following interactive placement commands to refine placement of these devices and device groups.

Group and Ungroup Devices

Groups and ungroups 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.



Vertical grouping of devices across two different rows



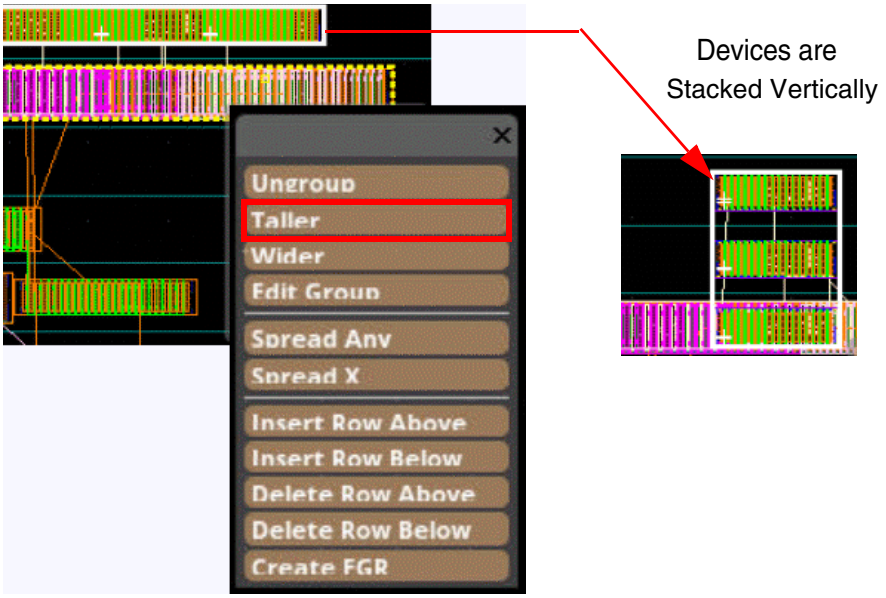
Device is moved to a new location; other devices in the group are also moved to maintain their relative positions.

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

Taller

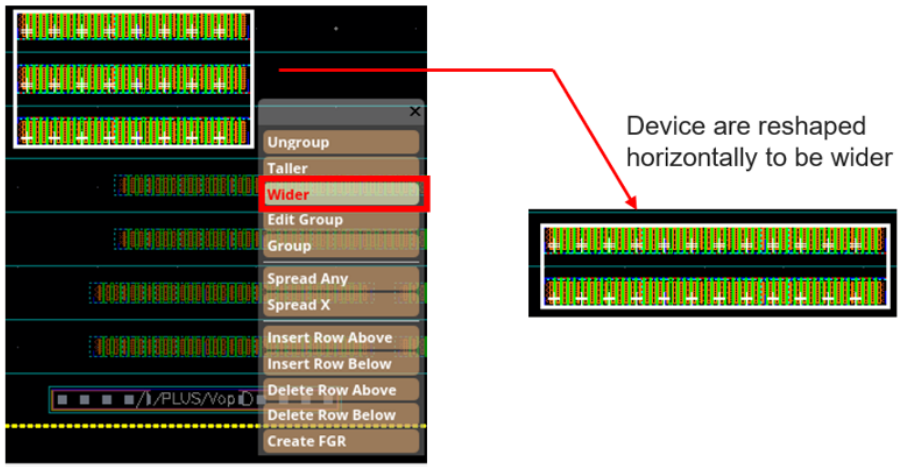
Stacks the selected devices vertically. This option is applicable only to grouped devices.

The following example shows the effect of applying *Taller* to a horizontal group of devices.



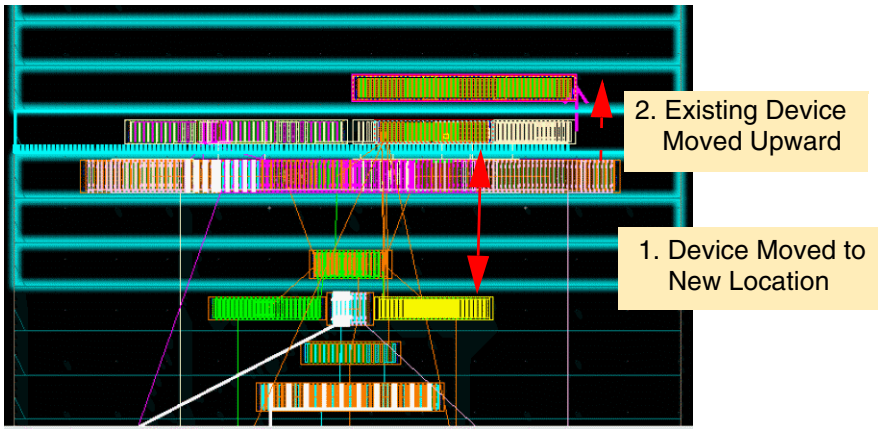
Wider

Reshapes the selected devices horizontally to be wider. This option is applicable only to grouped devices.



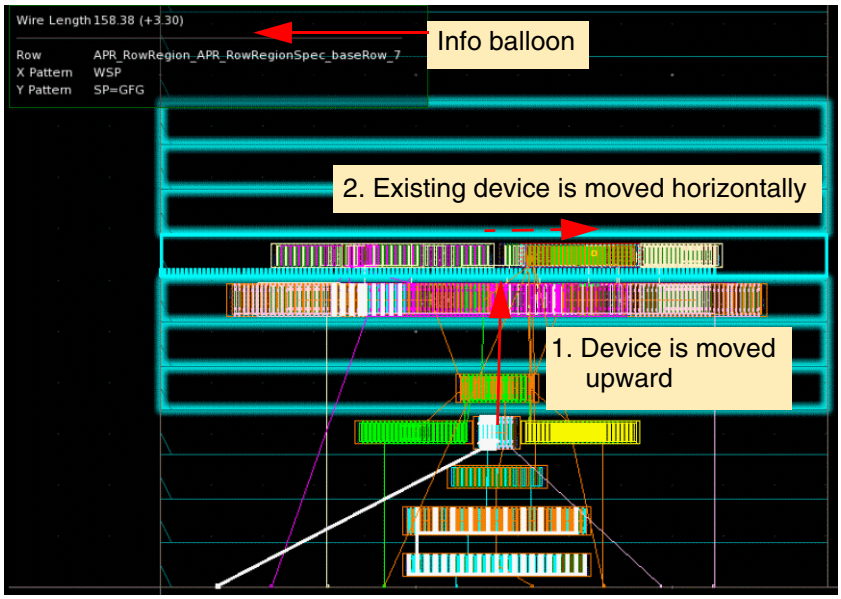
Spread Any

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 overlap. 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 Array Assistant. Use the options in the form to create and edit Modgens. For more information, see Automatic Generation of Modgens using the Array Assistant.

Insert Row Above

Inserts a blank row above the selection. This option is available only when row regions are generated and not when a diffusion grid is generated.

Insert Row Below

Inserts a blank row below the selection. This option is available only when row regions are generated and not when a diffusion grid is generated.

Delete Row Above	Deletes the row above the selection. The delete row commands do not check whether the rows to be deleted are non-empty rows or whether the deletion would result in any device overlaps.
Delete Row Below	Deletes the row below the selection.
Create FGR	Opens the Create Guard Ring form that lets you create a fluid guard ring (FGR).
Create Fill and AGR	Opens the Fill and GR form, which lets you add fill and guard rings at the same time. This option is available only in certain advanced node flows.

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.

Related Topics

Placing Devices Interactively in the Automated Device Placement and Routing Flow

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Base Layer Fill in the Automated Device Placement and Routing Flow

Placing Multi-Height Devices Using Automatic Device Placer

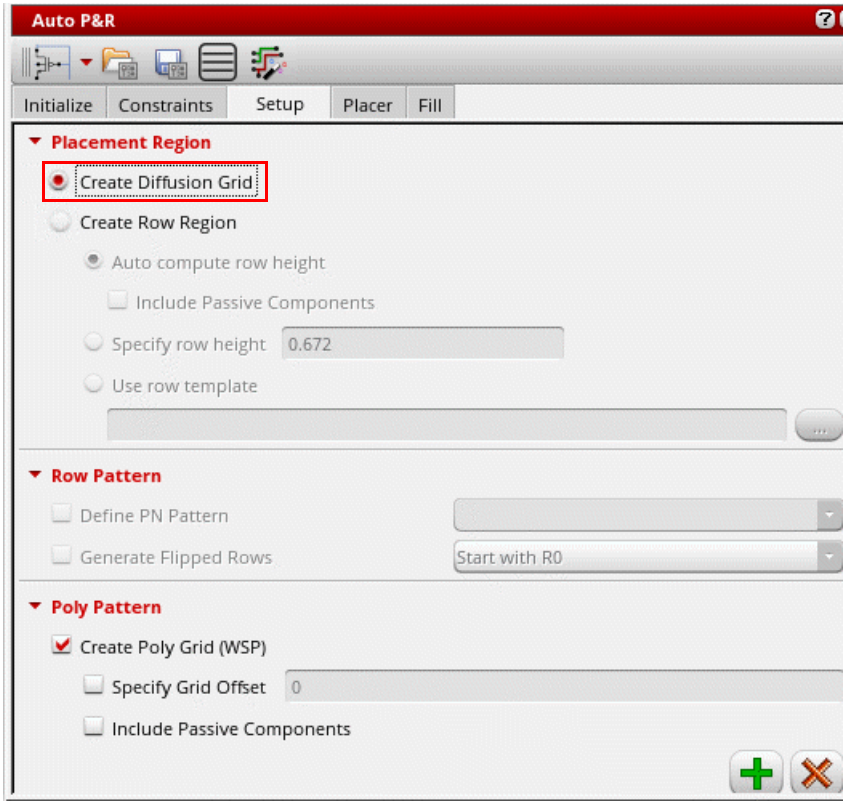
The Virtuoso automated device placement and routing flow supports diffusion grid-based placement, which enables rowless placement of multi-height devices. This technology creates a specialized grid based on the heights of the diffusion layers of different devices in a design. During placement, the devices in the design are snapped to the grid according to the height of their diffusion layers.

To generate a diffusion grid and place multi-height devices in it:

1. Set the enablePlaceWithWsp environment variable to `t` to enable the flow.
2. Open the required layout view in Layout EXL or higher tiers.

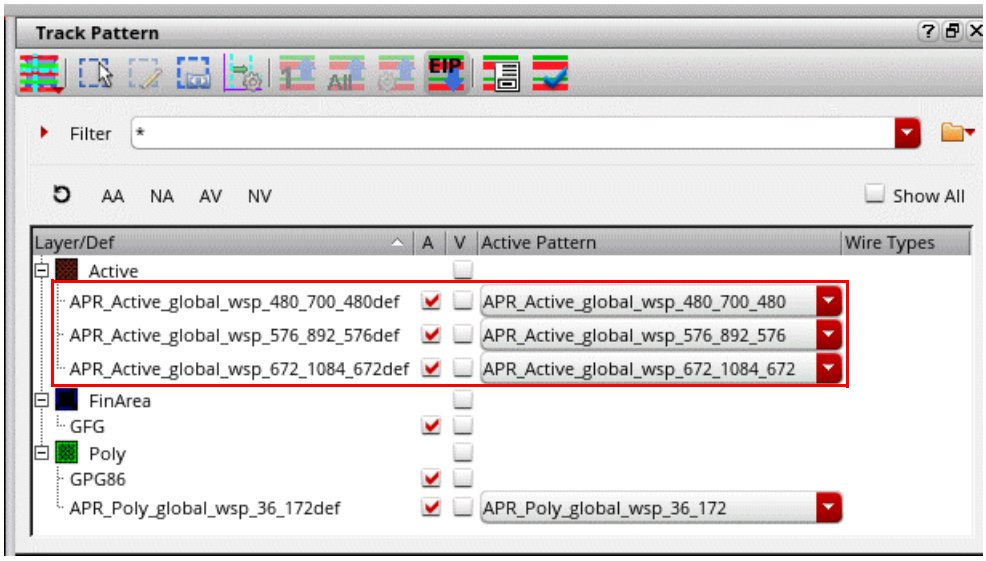
3. Display the *Setup* tab of the Auto P&R assistant.

   The *Create Diffusion Grid* check box appears on this tab only if `enablePlaceWithWsp` is set to `t` before launching the session.



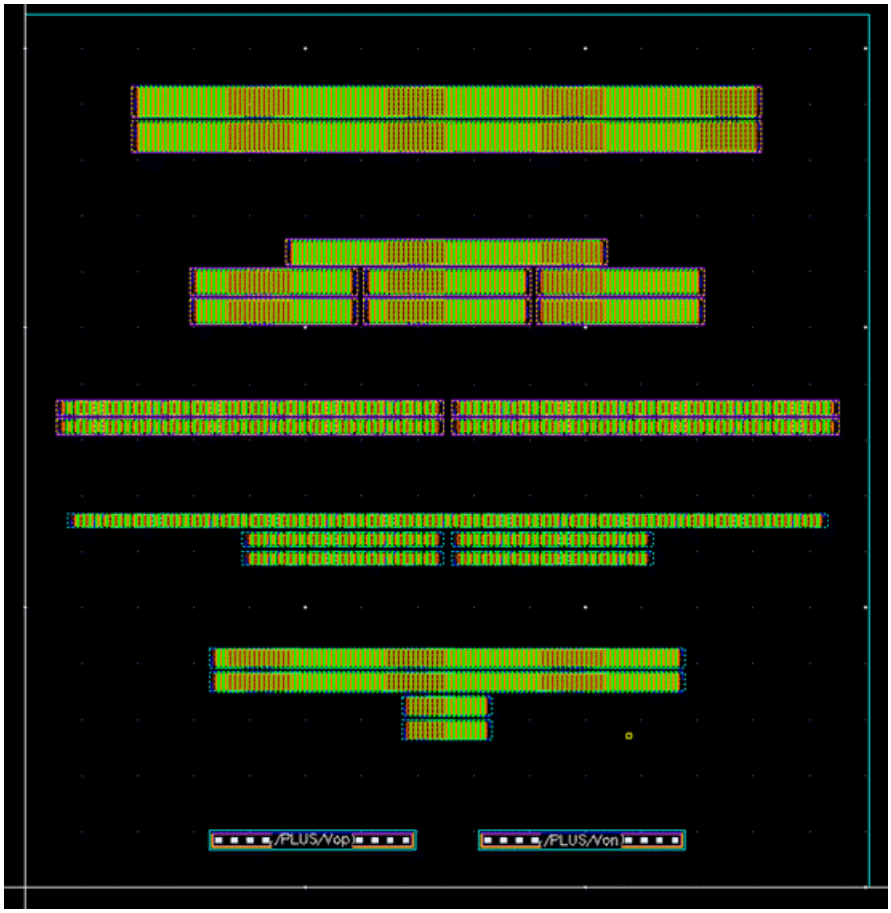
4. Select *Create Diffusion Grid*. All row region-related options are automatically inactivated.
5. Click  to generate the diffusion grid.
6. Open the Track Pattern assistant to view the diffusion grids.

WSPs of different widths are generated to match the height of devices in the design.



7. Open the *Place tab* of the Auto P&R assistant.
8. Click ▶ to run the placer.

Diffusion height-based placement is done. Devices are snapped to grids according to their height of diffusion (Active).



Related Topics

Placement of Non-Uniform Devices and Passive Devices

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

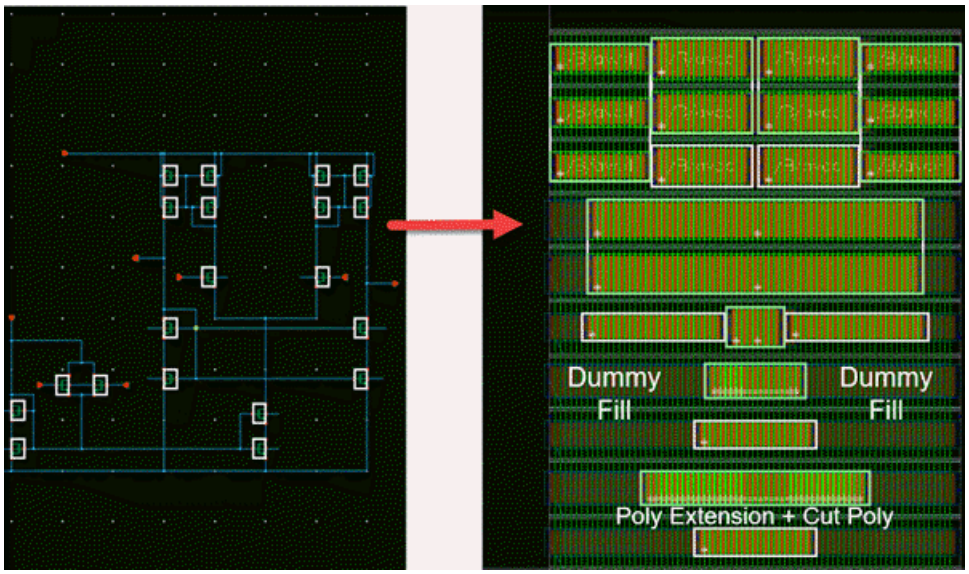
Base Layer Fill in the Automated Device Placement and Routing Flow

After device placement, there might be some gaps or unfilled areas in layout designs. Unfilled areas might also result if the *Utilization* setting was selected on the *Initialize* tab when generating devices.

Running a DRC check on such designs 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.

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.



Related Topics

Generating and Deleting Base Layer Fill

Troubleshoot Poly Fill Generation

Generating and Deleting Base Layer Fill

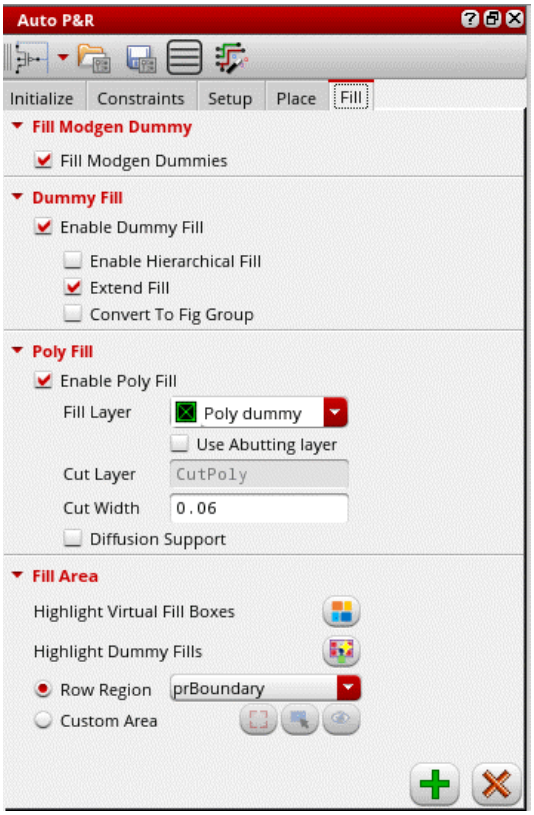
Before generating base layer fill in the Virtuoso automated device placement and routing flow, ensure the following:

- The N- and P- cells have their component class set correctly to one of `PFIN`, `NFIN`, `NMOS`, or `PMOS`.
- The connections in the design are correct.
- Signal type is correctly defined for power and ground nets.
- The same type of devices with different bulk nets are not placed in the same row. If present, an extra space is added while inserting dummy fill to avoid shorts.

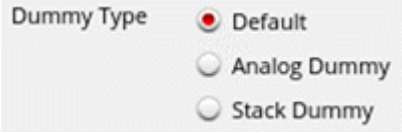
The fill command adds a single instance of multi-fingered fill in gaps between active devices.

To generate base layer fill:

1. Open the *Fill* tab of the Auto P&R assistant.



2. Select *Fill Modgen Dummies* to fill the gaps between Modgens selected on the layout canvas with Modgen dummies. If there are no Modgens selected, then dummies are added to fill the gaps of all the Modgens inside the PR Boundary on the layout canvas.
3. Specify the type of Modgen dummies to be inserted when the *Fill Modgen Dummies* icon is clicked in the *Fill* tab—*Default*, *Analog Dummy*, or *Stack Dummy*. This option is available only in certain advanced nodes designs.



4. Select *Enable Dummy Fill* to enable the dummy fill options.
5. Select *Enable Hierarchical Fill* to search through the hierarchy to identify gaps to be filled.
6. Select *Extend Fill* to extend cell fill in all directions. This option takes into consideration the transition spacing specified on the *Place tab*.
7. Select *Convert to Fig Group* to convert fill to a figGroup when the fill is executed on a draw region or virtual group.
8. Select *Enable Poly Fill* to insert poly fill.
9. Select a *Fill Layer* to derive cut-poly rails. The default value is the first layer-purpose pair that has its layer function set to `Poly`.
10. Select *Use Abutting layer* to use the abutting layer of the active or passive region as the fill layer.
11. *Cut Layer* is set to `CutPoly`. You cannot edit this value.
12. Specify the *Cut Width*. The default value is set to the minimum width value for the cut layer as defined in the technology file.
13. Select *Diffusion Support* to allow poly fill to be inserted around diffusion shapes.
14. Click *Highlight Virtual Fill Boxes* to highlight virtual fill boxes in which fill cells are to be inserted.
15. Select *Highlight Dummy Fills* to highlight dummies in the layout. You can visually differentiate between active devices and dummies.
16. Use one of the following methods to specify the area to be filled:

- *Row Region* (default): Select a row region from the drop-down list Inserts fill only in the specified row region. You can select either *All* (default), which refers to all row regions in the current cellview, or a specific row region from the drop-down list.
- *Custom Area*: Inserts fill in the custom area drawn. Click the icon and draw the required area in the canvas. When you draw a new custom area, the previous one is automatically deleted.

  Note: Fill is not inserted in areas outside row regions.

17. In certain advanced node flows, you can use the following options to create or delete guard rings after inserting fill:

Insert Guard Ring | Delete Guard Ring

The Create Guard Ring and Delete Guard Ring buttons work for both row regions and custom area. When there is no a valid row region, these buttons are disabled.

Add fill before creating a guard ring, especially if the existing virtual groups are not rectangular or contain gaps.

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

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.

Poly fill are automatically snapped to the nearest snap patterns. In the absence of active snap pattern definitions, tracks are derived based on existing poly shapes. This may result in empty areas in regions where there is no active snap pattern definitions or poly shapes.

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*.

Related Topics

Defining a Component Type

Base Layer Fill in the Automated Device Placement and Routing Flow

Troubleshoot Poly Fill Generation

Troubleshoot Poly Fill Generation

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.

Related Topics

Base Layer Fill in the Automated Device Placement and Routing Flow

Generating and Deleting Base Layer Fill

4

Virtuoso Automated Device-Level

In the Virtuoso automated device placement and routing flow, routing is done after device placement in a design.

The flow supports the following routing modes using the Routing Assistant available in Layout MXL.

- **Automatic Routing**: Enables high-speed shape-based routing, allowing gridded or grid-less, and track-based routing of regular and power signals for physical designs. For Automatic routing, you can use the Routing assistant available in the Layout MXL tier. The following automatic routing types are supported:
  - Device-level
  - Standard cell
  - Chip assembly

  For Automatic routing, use the Routing assistant available in the Layout MXL tiers.
- **Interactive Routing**: Enables you to route connections interactively within the Virtuoso environment. These capabilities provide efficient ways to interactively and automatically route connections in order to meet critical design constraints and rules. The interactive routing capabilities are fully enabled on all process nodes including the most advanced process technologies. For interactive routing, use the Routing assistant available in the Layout EXL and higher tiers.

The device-level router lets you:

- Specify valid routing layers as per a given wire editing constraint group
- Generate, import, and edit Width Spacing Patterns (WSP) for valid routing layers as required for advanced node processes.
  - WSPs can be automatically generated or imported for an existing layout
  - WSP Manager in Virtuoso can be opened to edit the WSPs
  - Supply WSPs can be created with specified patterns to generate the supply grid

- WSP tracks can be assigned specific wire types and nets mapped to the wire types
- Existing I/O pins in the layout can be snapped to imported WSPs

- Specify global options to insert trims and run DRD checks and routing

The device-level routing supports Same Mask and Mask Name settings in the Constraint Manager for symmetry and net class constraint in all types of device routing (P2T, Mesh, and Finish, and Automatic routing).

Related Topics

Generating Width Spacing Patterns for Device-level Routing

Checking Layout Routability after Generating Grids and Running Device Placer

Generating a Supply Grid

Finishing Routing for Signal Nets

Generating Mesh for Selected Nets

Generating Pin to Trunk Routing for Selected Nets

Viewing and Analyzing Device-Level Routing Results

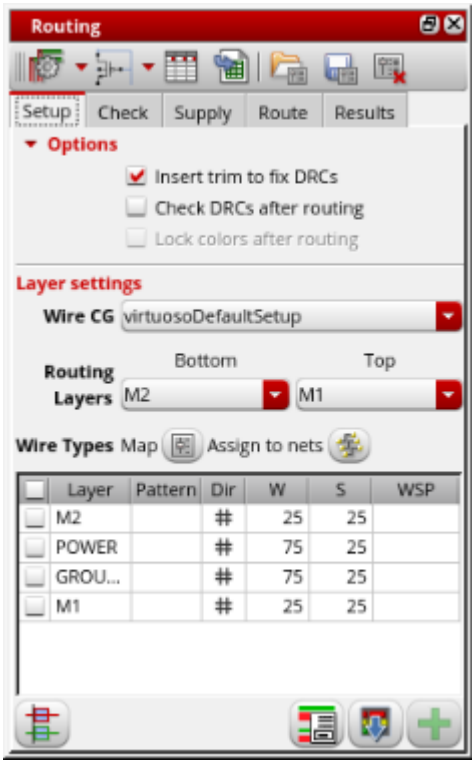
Configuring Device-Level Router Settings

Before routing a design, you need to configure the routing options in the Routing assistant for device-level routing. To configure router settings:

1. Open a design in Layout MXL.
2. Choose *Window – Assistants – Routing*.

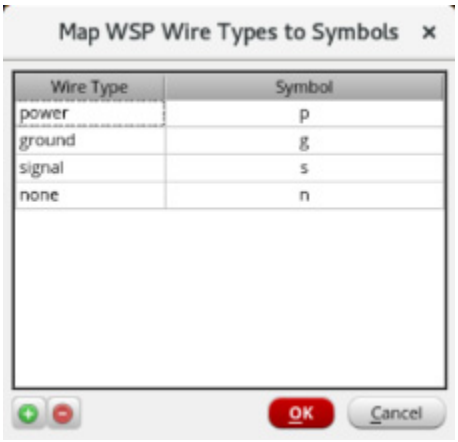
   Alternatively, right-click anywhere on the layout window menu bar and choose *Assistants – Routing*.

The *Setup* tab of the Routing assistant is displayed.

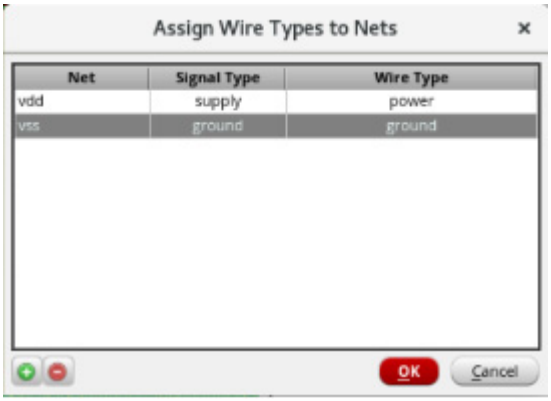


3. Select *Insert trim to fix DRCs* to insert trims to devices automatically to fix DRC errors.
4. Select *Check DRCs after routing* to automatically run DRC checks after routing.
5. Select a wire constraint group from the *Wire CG* drop-down list.
6. Choose the bottom and top routing layer from the *Bottom* and *Top* drop-down lists. Specifying valid routing layers updates the WSPs visible in the table and specifies which layers the router should use for routing.

7. Click *Wire Types Map* . The Map WSP Wire Types to Symbols form is displayed. This form is used to display the wire type and their symbols and also lets you create your own custom wire type and their symbol.



8. Click *Assign to Nets* . It opens the Assign Wire Types to Nets form with a table that assigns wireTypes to nets.



9. Select the layers for which you want to configure the router settings. You can either select all layers or one or more of them.

10. Once the router settings are configured, you can generate width spacing patterns (WSPs).

Related Topics

Configuring Device-Level Router Settings

Generating Width Spacing Patterns for Device-level Routing

Generating Width Spacing Patterns for Device-level Routing

In automated routing technology, you can use one of the following methods to generate WSPs:

- Using the WSP Manager
- Importing WSPs
- Generating WSPs automatically

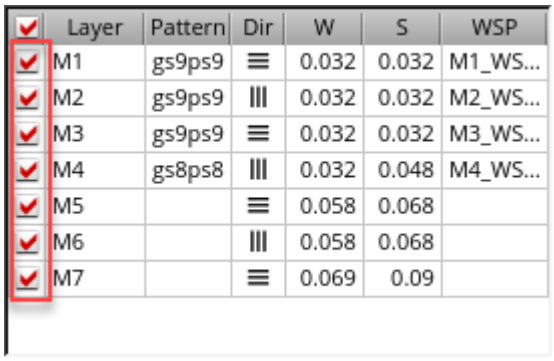
This topic covers the method of generating WSPs automatically. For information on the other methods, see Using the WSP Manager.

To automatically generate WSPs:

1. Open a design in Layout MXL.
2. Choose *Window – Assistants – Routing*.

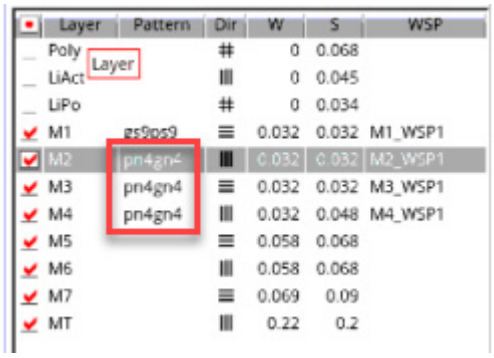
   Alternatively, right-click anywhere on the layout window menu bar and choose *Assistants – Routing*.
3. In the *Setup* tab, select the bottom and top routing layers from the *Bottom* and *Top* drop-down list.

   Specifying valid routing layers updates the WSPs visible in the table and specifies which layers the router should use for routing.
4. Either select all layers or the required layers on which you want to generate WSPs.



Layer	Pattern	Dir	W	S	WSP
M1	gs9ps9	≡	0.032	0.032	M1_WS...
M2	gs9ps9	|||	0.032	0.032	M2_WS...
M3	gs9ps9	≡	0.032	0.032	M3_WS...
M4	gs8ps8	|||	0.032	0.048	M4_WS...
M5		≡	0.058	0.068	
M6		|||	0.058	0.068	
M7		≡	0.069	0.09	

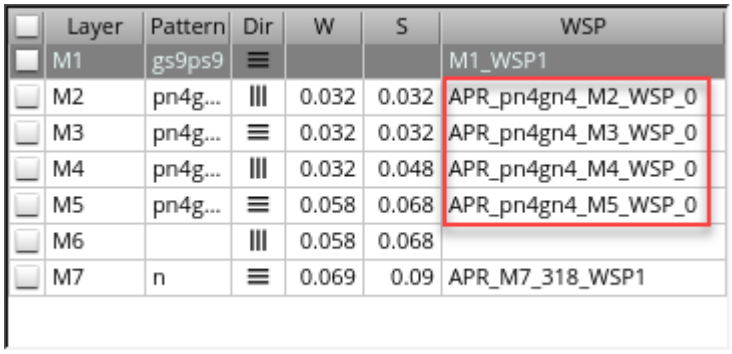
5. Specify a pattern in the *Pattern* column of the layer table. This is to allocate tracks for supply wire types to generate the supply grid on selected layers.



The Map WSP Wire Types to Symbols Form shows the character or symbol – `wireType` mapping in the WSP to be created. These characters or symbols are used in the pattern string.

6. Click *Auto-generate WSPs* at the bottom of the Routing assistant.

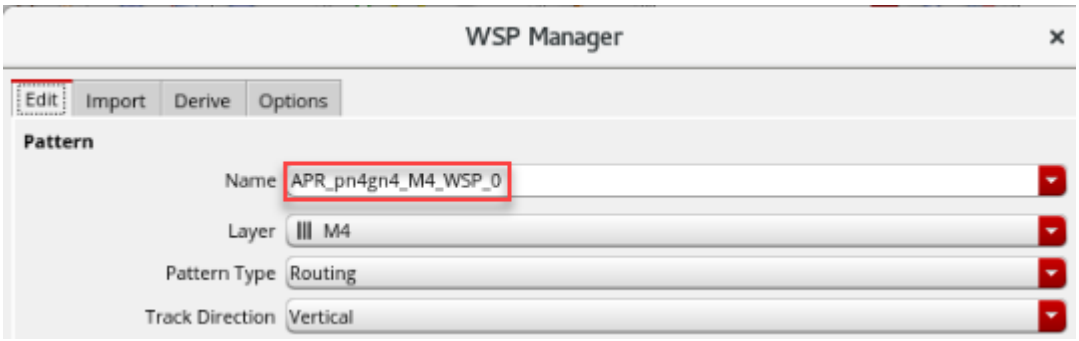
WSPs are automatically generated for the selected metal layers.



In certain nodes, local grids are generated for the lowest metal layer when the design is a non-uniform design. This makes the WSP column against the lowest metal layer for which the local grids are created go blank. These local grids can be seen in the track pattern assistant by selecting the *Show All* option.

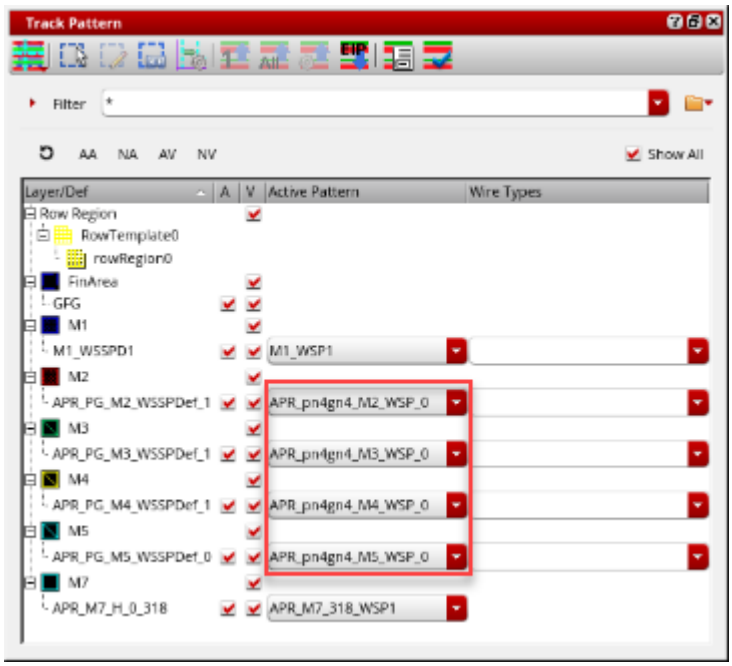
7. Select any layer in the layer table to view the attributes of the automatically created WSP.
8. Click the *Show WSP Manager* button at the bottom of the Routing assistant.

The WSP attributes of the selected layer are displayed.



9. Choose *Window – Assistant – Track Pattern* from the layout window menu bar to open the Track Pattern assistant.
10. Select *Show All*.

    You can see the pattern-based auto-generated WSPs prefixed with `APR_`.



11. In the Routing Assistant, click *Snap pins to WSPs* and see that the IO pins can snap to the generated WSPs.

Related Topics

Configuring Device-Level Router Settings

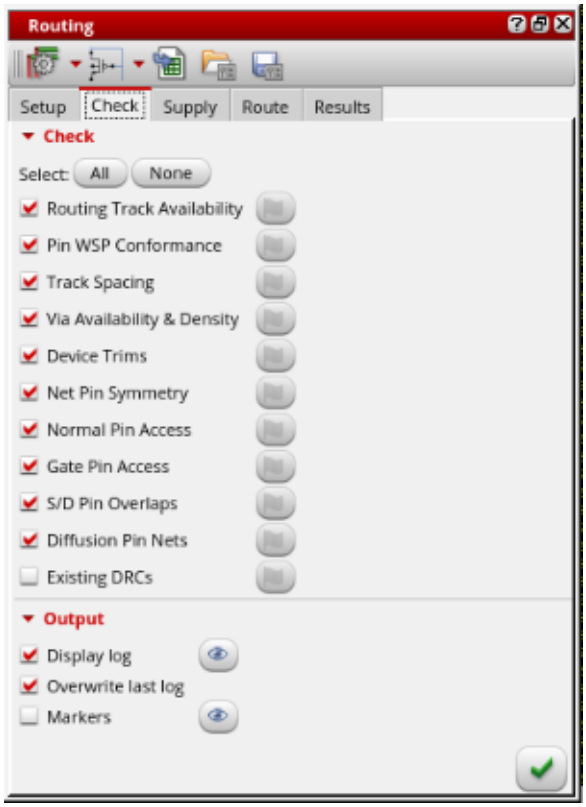
Checking Layout Routability after Generating Grids and Running Device Placer

You can run pre-routing checks to detect design issues before routing a design. Running the checks lets you identify potential situations or objects that may cause trouble for the router later in the flow.

To check the routability of a design:

1. Open a design in Layout MXL.
2. Choose *Window – Assistants – Routing*.

   Alternatively, right-click anywhere on the layout window menu bar and choose *Assistants – Routing*.
3. In the Routing assistant, choose the *Check* tab.

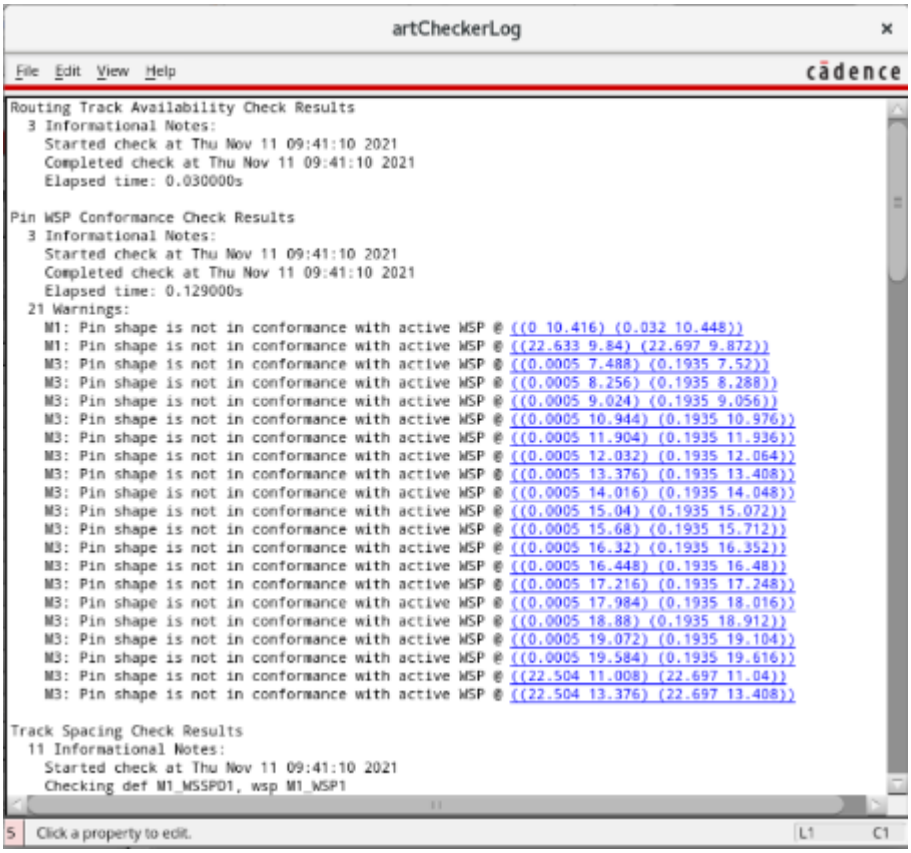


4. Click *All* to select all pre-routing checks or select the required checks by clicking the check box next to the pre-route check name.
5. Click *Run pre-route checks* .

   After running the checks, green, orange, and red status flags appear that indicate whether the check was passed, there was a warning, or there was an error. No red flags indicate that the design is correctly set up.

   Clicking the button against each flag takes you to the location in the log file where that check result was logged.

6. Click the *Display existing log file* button next to the *Display log* option to view the log file and check for any issues.



The log file provides additional information with clickable links to show the issue in the layout. You can also view the issues in the Annotation Browser using the *Markers* option.

Related Topics

Generating Width Spacing Patterns for Device-level Routing

Configuring Device-Level Router Settings

Generating a Supply Grid

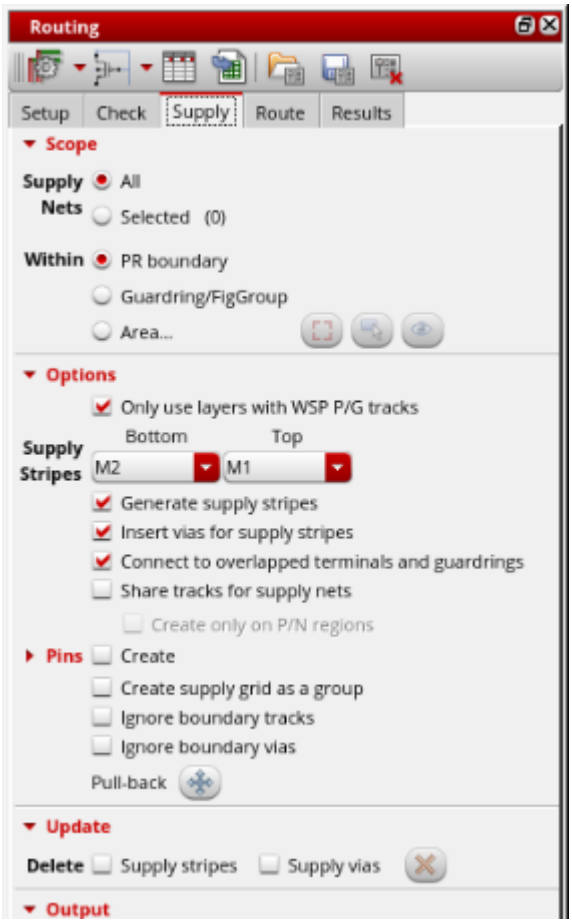
In certain device-level libraries, boundary cells have blockages. In such cases, the flow must be modified to not use rails in the row region and to place boundary cells before routing the design. Generally a supply grid should be created before device-level placement so that the placement can prevent routing from interfering with access to device-level pins.

To generate a supply grid:

1. Open a design in Layout MXL.
2. Choose *Window – Assistants – Routing*.

   Alternatively, right-click anywhere on the layout window menu bar and choose *Assistants – Routing*.

   The Routing assistant appears.
3. Choose the *Supply* tab.



4. Click *Selected* to define the scope of supply nets to route. You can also choose to route *All* supply nets.
5. Select *Area* under *Within* to generate supply grid in a user-defined area. You can select *PR boundary* to generate supply grid in the entire PR Boundary or select *Guardring/FigGroup* to generate supply grid inside the user selected FigGroup or Guradring group.
6. If *Area* is selected, click *Draw the area bbox* to draw the area for the supply grid.

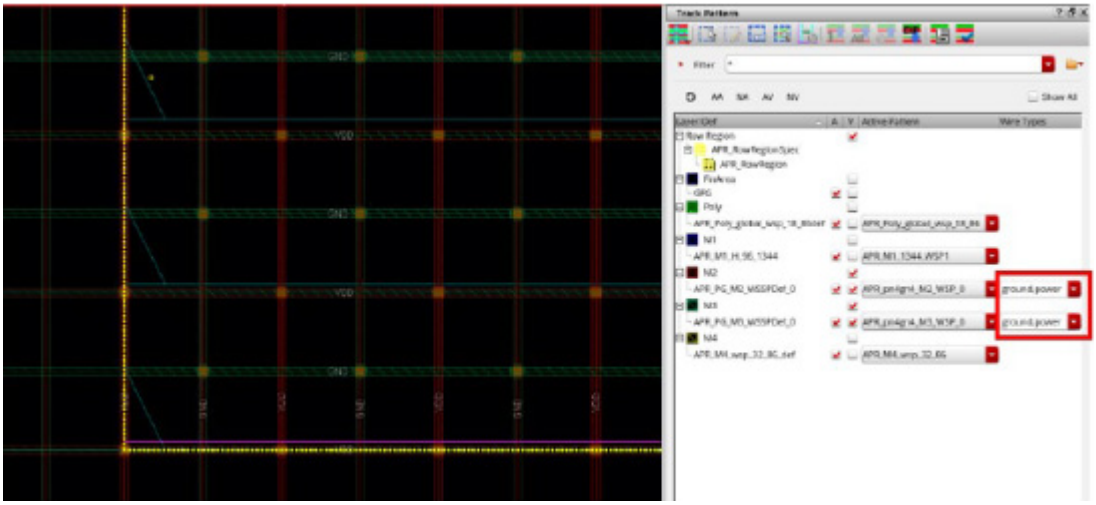
7. Select Only use layers with WSP P/G tracks if you want to restricts supply routing to layers with existing power or ground tracks.
8. Select the bottom and top layers for supply stripes from the *Supply Stripes Bottom* and *Top* drop-down lists. The layers are auto filled based on the pattern entered during WSP generation and if there are PG tracks assigned to the layers.

   This generates stripes only on tracks with power and ground wireTypes that are assigned to nets that have `sigTypes` of supply and ground.
9. Ensure that the following options are selected.
   - *Generate supply stripes* to generate the horizontal and vertical stripes on the selected layers,
   - *Insert vias for supply stripes* to insert vias in the supply grid created.
   - *Connect to overlapped terminals and guardrings* to connect the supply grid to any instance terminals and guarding that are overlapping the supply grid.
10. Select *Share tracks for supply nets* for supply nets to share power ground tracks as needed in the appropriate P and N regions.
11. Select *Create* next to the *Pins* option to generate pins instead of path segs.

    You can expand the *Pins* option to find options for creating labels against the pins, create pins instead of pathSegs for all supply layers or selected supply layers and to create pins on ends of the pathSegs only.
12. Select *Create Supply Grid as a group* to create the supply grid as a figGroup.
13. Select *Ignore boundary tracks* and *Ignore boundary Vias* to avoid creating tracks and vias overlapping the PR Boundary edge.
14. Click *Run power route*  at the lower-right corner of the *Supply* tab.
15. Once supply grid generation is complete as indicated by the progress bar, remove the drawn area by clicking *show/hide the area bbox*  next to the *Draw* button.
16. Select the power and ground nets in the Navigator assistant to see the supply grid on the selected layers.

    Note: You can set the highlight option to true color and select the shapes on supply nets for better visualization.

17. To clearly visualize the supply grid, select the power and ground wire type from the *Wire Types* drop down next to the metal layers in the Track Pattern assistant."



You can zoom into the layout to see that the supply grid follows the supply (power and ground) wire types and skips tracks that are not assigned a supply wire type.

Related Topics

Generating Width Spacing Patterns for Device-level Routing

Configuring Device-Level Router Settings

Checking Layout Routability after Generating Grids and Running Device Placer

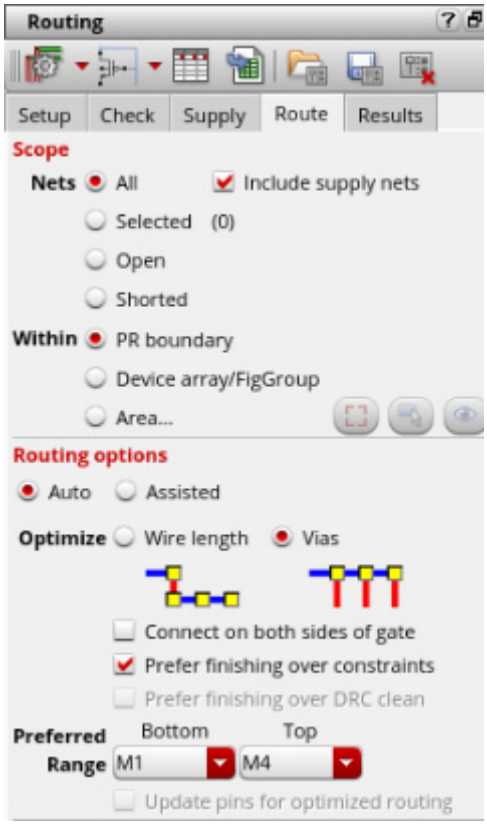
Routing in Automatic Mode

When routing noncritical nets, use Auto Routing option for selected or all nets as a quick routing solution. To do this:

1. Open a design in Layout MXL.
2. Choose *Window – Assistants – Routing*.

   Alternatively, right-click anywhere on the layout window menu bar and choose *Assistants – Routing*.

3. Click the *Route* tab in the routing Assistant and in the *Routing Options* section select *Auto.*



4. Select PR Boundary or *Area or Device Array/FigGroup* option in the *Scope* section of the *Route* tab.
5. Select either *Wire Length* or *Vias* from the *Optimize* field in the *Routing Options* section. By default, *Vias* is selected.
6. Select *Connect on both sides of gate* only if needed and if there are two poly connections in the device.
7. Ensure *Prefer finishing over constraints* is selected for better convergence. If unselected, the constraints are preferred over convergence.
8. Specify the preferred routing layer range from the *Bottom* and *Top* drop-drown list.
9. Click *Run signal router*  .

   The router completes routing the selected or all nets within the specified scope. Routing errors are reported in the CIW, which might be hidden underneath other windows. The CIW can be raised automatically by setting the following environment variable:

```
envSetVal("ui" "raiseCIWonError" 'boolean t)
```

Related Topics

Configuring Device-Level Router Settings

Checking Layout Routability after Generating Grids and Running Device Placer

Generating a Supply Grid

Finishing Routing for Signal Nets

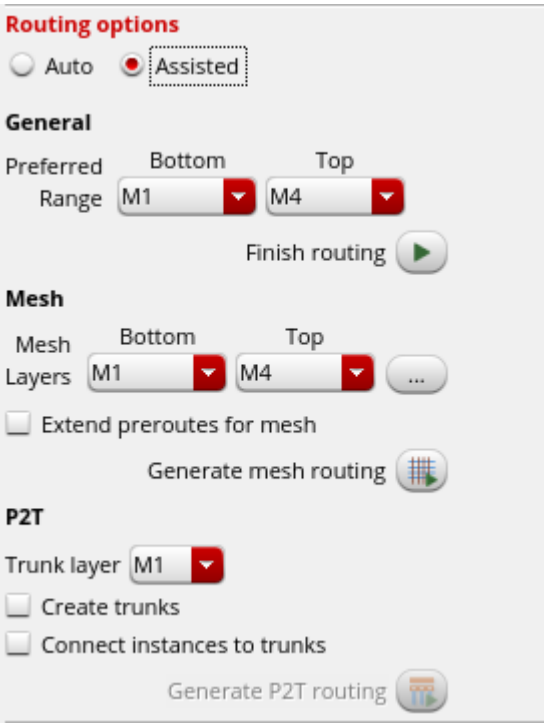
Finishing Routing for Signal Nets

When routing a net that is not complex or when it is not necessary to perform detailed routing patterns, using the *Finish routing* feature is a quick solution. To do this:

1. Open a design in Layout MXL.
2. Open the Navigator assistant and select *Objects – Nets*.
3. Select a net in the Navigator assistant.
4. Choose *Window – Assistants – Routing*.

   Alternatively, right-click anywhere on the layout window menu bar and choose *Assistants – Routing*.

5. Click the *Route* tab in the Routing Assistant.



6. Select *Assisted as* the routing option from the *Routing Options* section in the Route tab of the Routing Assistant.
7. Ensure that the scope of the nets is always set as *Selected* since assisted routing routes only one selected net at a time.
8. Click *Selected* to define the scope of nets for routing.
9. Specify the preferred routing layer range from the *Bottom* and *Top* drop-down list.
10. Click *Finish routing* .

    The router completes the routing of the selected net.

    Routing errors are reported in the CIW, which might be hidden underneath other windows. The CIW can be raised automatically by setting the following environment variable:

```
envSetVal("ui" "raiseCIWonError" 'boolean t)
```

Related Topics

Generating Width Spacing Patterns for Device-level Routing

Configuring Device-Level Router Settings

Checking Layout Routability after Generating Grids and Running Device Placer

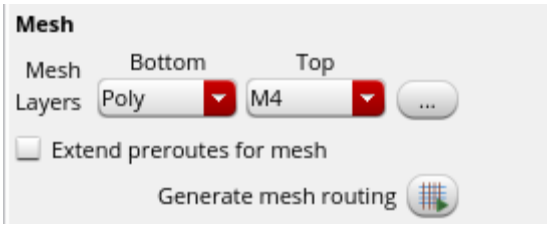
Generating a Supply Grid

Generating Mesh for Selected Nets

When routing a complex design with many nets, it is sometimes necessary to generate mesh on certain critical nets to distribute current and reduce resistance. The steps to complete mesh routing are:

1. Open a design in Layout MXL.
2. Open the Navigator assistant and select *Objects – Nets*.
3. Select a net in the Navigator assistant.
4. Choose *Window – Assistants – Routing*.

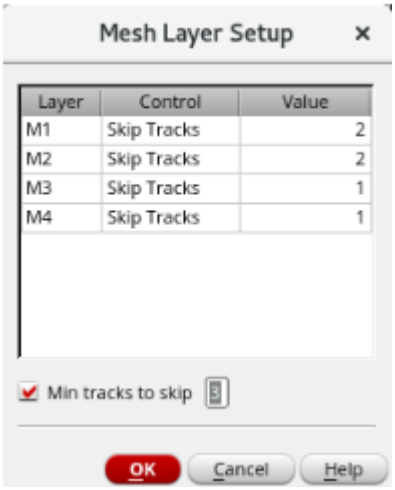
   Alternatively, right-click anywhere on the layout window menu bar and choose *Assistants – Routing*.
5. Select the scope of the mesh to be generated as either *PR Boundary*, *Device Array/ Fig Group* or *Area*.
6. In the *Mesh* section, select the bottom and top mesh layers from the *Bottom* and *Top* drop-down lists.



7. Click the ellipses (...) button next to the *Mesh layers* field.

   The Mesh Layer Setup Form is displayed.

8. In the Mesh Layer Setup form, select either *Skip Tracks*, *Mesh Count*, or *Term Align* against each metal layer, as required. You can also select and modify the *Min tracks to skip*, if required.



9. Click *OK*.
10. Ensure that *Extend preroutes for mesh* is unselected. The *Extend preroutes for mesh* option can be selected or unselected, as required. If selected, the mesh is extended to be drawn all the way to the edge of the longest pre routes that exists before generating the mesh.
11. Click *Generate mesh routing* to generate meshes on mesh layers.

    Mesh is formed to connect two horizontal trunks.

Related Topics

Generating Width Spacing Patterns for Device-level Routing

Configuring Device-Level Router Settings

Checking Layout Routability after Generating Grids and Running Device Placer

Finishing Routing for Signal Nets

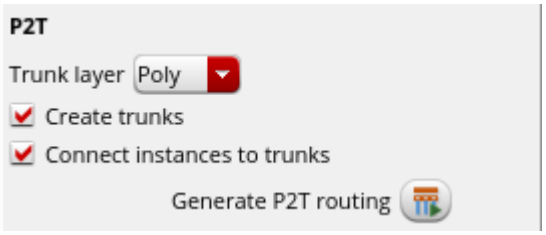
Generating Pin to Trunk Routing for Selected Nets

When routing a complex design with many nets, it is ideal to use mesh and Pin to Trunk (P2T) routing to form clean routing structures effectively. For certain nets, Pin to Trunk (P2T) routing

can be used, as required, where a trunk is first created on a certain layer and then the instance terminal pins is connected to these trunks. To do this:

1. Open a design in Layout MXL.
2. Open the Navigator assistant and select *Objects – Nets*.
3. Select a net in the Navigator assistant.
4. Choose *Window – Assistants – Routing*.

   Alternatively, right-click anywhere on the layout window menu bar and choose *Assistants – Routing*.
5. (Optional) Click *Delete* to delete any routing on the selected net.
6. Select the scope of the mesh to be generated as either *PR Boundary*, *Device Array/ Fig Group* or *Area*.
7. Select a layer from the *Trunk layer* drop-down list in the *P2T* section.
8. Click to select the *Create trunks* and *Connect instances to trunks* options to first generate the trunks and then connect the instance terminal pins to these trunks.

   The router creates trunks and connects the trunks to the terminal pins of the instances on the selected net.
9. Click *Generate P2T routing*  .

   The router creates trunks and connects the trunks to the instances on the selected net. The two horizontal trunks are made and all pins on the devices are connected to the two trunks.

If there are two or more trunks created on the net, than to complete routing the net, the two trunks need to be connected. For this, use the mesh routing or finish routing to connect the multiple horizontal trunks. For more information, see Generating Mesh for Selected Nets.

Related Topics

Configuring Device-Level Router Settings

Viewing and Analyzing Device-Level Routing Results

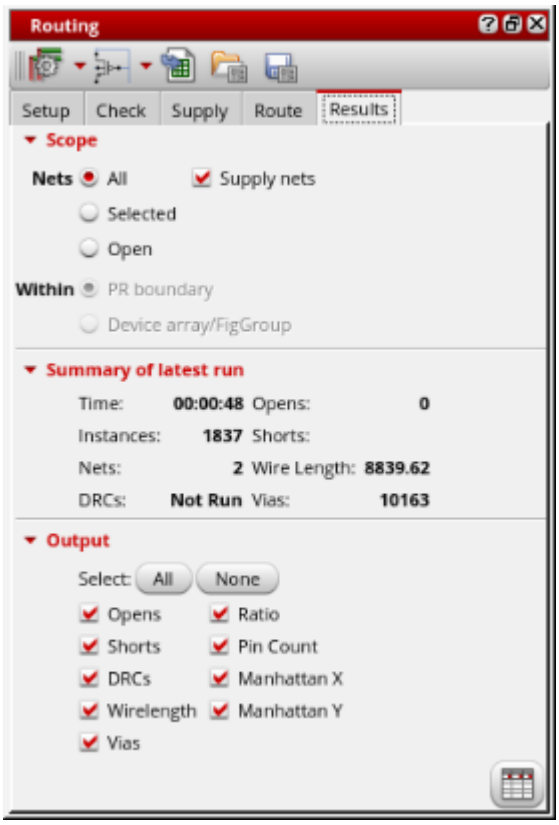
The Routing Results Browser includes a table that displays routing results, for example the number of opens, shorts, wire length, DRC, and vias. Use this table to analyze the routing results,

To view and analyze routing results:

1. Open a design in Layout MXL.
2. Choose *Window – Assistants – Routing*.

   Alternatively, right-click anywhere on the layout window menu bar and choose *Assistants – Routing*.

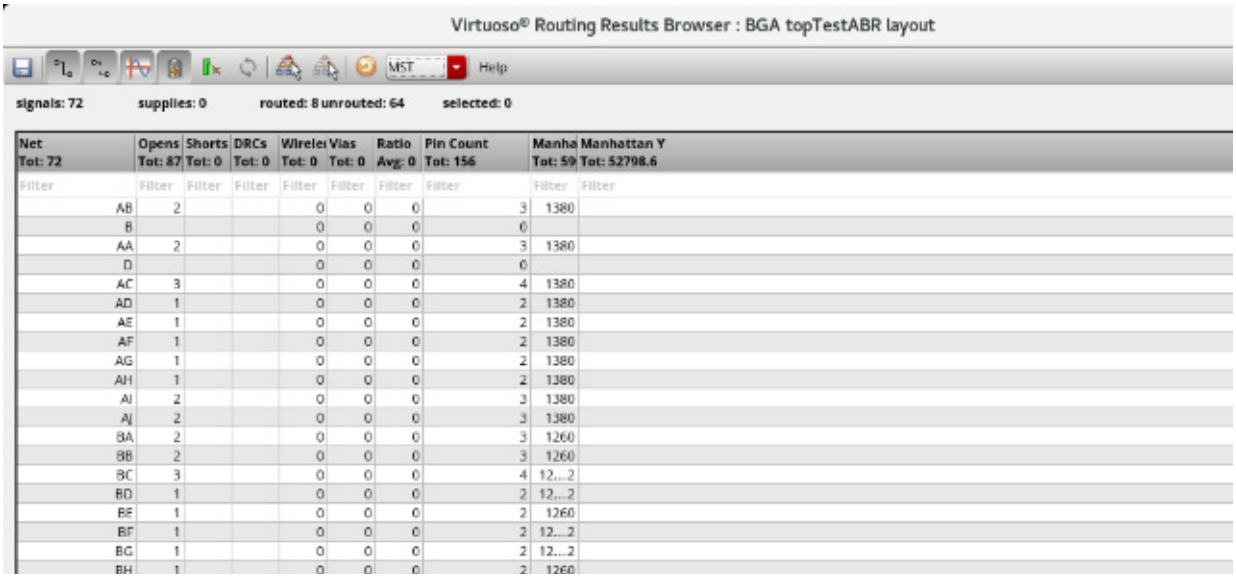
3. Click the *Results* tab in the Routing Assistant.



4. Select the scope of nets to analyze the routing result. You can either select *All*, *Selected*, or *Open* nets.
5. Select *Supply Nets* to see the results of the power and ground nets.
6. Select *All* in the *Output* section to display all the output columns in the Routing Results Browser.

   You can also select specific outputs for which you want the column to be displayed in the Routing Results Browser.
7. Click *Show results browser* at the bottom right corner.

The Routing Results Browser appears. You can see the total number of routed nets, opens, shorts, and the details of the violations.



Related Topics

Configuring Device-Level Router Settings

Checking Layout Routability after Generating Grids and Running Device Placer

A

Automated Device Placement and Routing User Interface

In the Virtuoso device placement and routing flow, you use the Auto P&R assistant for device placement and the Routing assistant for device routing. The Auto P&R assistant is available in the Layout EXL (and higher) cockpit and the Routing assistant is available in the Layout MXL cockpit.

The Auto P&R assistant is the integrated, automatic placement solution that lets you initialize, set up, place, and add device fill to the layout design automatically as per your requirements.

The Routing assistant has the same look-and-feel for device, standard cell, and chip assembly routing with a common toolbar for all routing types. The Routing assistant is a dockable assistant pane that provides various options to let you perform tasks related to various routing types.

This topic lists the device placement and routing assistant and forms.

- Auto P&R Assistant User Interface for Device-Level Placement
- Default Reuse Template Specification Form
- Highlight Extract Group Form
- Routing Assistant User Interface for Device-Level Routing
- Mesh Layer Setup Form

Auto P&R Assistant User Interface for Device-Level Placement

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

Tab	Description
Initialize	Generates layout representations of the schematic design components. Any existing components in the layout view are deleted and are regenerated from scratch.
Constraints	Automatically generates constraints and constraint groups.
Setup	Generates a row region or diffusion grids and poly width spacing pattern (WSP). The rows or grids, and WSPs are used for device placement, snapping, and routing.
Place	Lets you customize placement settings and run the placer.
Fill	Adds dummy and poly fill in gaps between the instances in a design.

Initialize

The following table describes the fields available on the *Initialize* tab of the Auto P&R assistant.

Field	Description
Generation Options	Lets you select the objects to be generated in the layout.
Mode	Specifies a device generation mode: ■ *Generate*: Generates new devices in the layout design.. ■ *Update*: Updates existing objects in the layout design.
Generate	Specifies the scope of layout generation. ■ *All*: Generates all objects present in the source. ■ *Selected*: Generates only the selected objects.

Field	Description
Instances	Generates all the instances in the schematic that do not have any ignore properties attached to them.
Pins	Generates devices and power pins.
Pin Layer	Specifies the layer in the source schematic that contains the device pins to be generated in the target layout.
Power Pin Layer	Specifies whether power pins are to be generated. You can select the layer in the source schematic that contains the power and ground pins that are to be generated in the target layout. By default, power and ground pins are not generated. The first metal layer in the layer stack is set as the default value.
Boundary	Generates a PR boundary. Environment variable: init_boundaryUtilizationOrWidth
Utilization	Specifies the percentage of area within the cell boundary that can be filled with objects. The default is 25%. Environment variable: init_boundaryUtilizationVal
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	Specifies the exact width of the PR boundary. This option is available in the *Utilization* drop-down list. Environment variable: init_boundaryWidthVal
Height	Specifies the exact height of the PR boundary. This option is available in the *Aspect Ratio* drop-down list.
Migration Options	This section is available only in Layout MXL and is part of the assisted custom layout design migration flow. Use the options in this section to capture source data and apply it to a target layout.
Migration Directory	Specifies the path to the directory in which the captured source data is stored. The target layout references this location for applying the captured source data.

Field	Description
Capture Placement from Layout 	Captures data from the source layout and stores it in the specified *Migration Directory.*
Apply Placement to Layout	Specifies the target layout to which the captured reuse information must be applied.
Layout Objects	Specifies the objects in the target layout to which reuse settings are to be applied. The available options are: ■ *Instances* ■ *Pins* ■ *Boundary* ■ *Constraints*
PDK Settings	Specifies options to be used in layout reuse flow.
Use custom settings	Loads PDK settings from a specified file. This option is useful for loading additional parameters and values, for example fill overrides, environment variables, and device registration that are applicable only at lower nodes. In the custom settings file, the apCustomPDKSettings API is defined to specify the required PDK and its settings. Environment variables: apUseCustomSettings, pdkMapping, init_useCustomSettings
Generate	Generates the selected objects in the layout canvas.

Constraints

The following table describes the fields available on the *Constraints* tab of the Auto P&R assistant.

Field	Description
Constraint Options	Specifies settings to apply when generating constraints.
Ignore Instance Self-symmetry	Restricts propagation of self-symmetry of multiple grouped instances into a single symmetry constraint. Environment variable: ignoreInstSelfSymmetry

Field	Description
Arrange By Virtual Groups	After creating constraints, displays a preview of the virtual groups before placement. All devices are arranged in columns according to their virtual groups. Environment variable: arrangeByVGs
Migration options	This section is available only in Layout MXL and is part of the assisted custom layout design migration flow. Use the options in this section to capture source data and apply it to a target layout.
Directory	Path to the directory in which the captured data is stored.
Highlight Groupings in Layout	Displays the Highlight Extract Group form, where you can select the groupings to be highlighted.
Capture Templates from Layout Groups	Captures groupings from the source layout. This information is stored in the specified *Directory*.
Apply Source Templates to Layout Groups	Applies the captured groupings to the target layout.
Groups and Constraints	Specifies settings to apply when generating constraints.
Select or Deselect all Constraint Groups 	Selects or deselects all constraint groups.
Invert Selection	Inverts selection.
	Reverts selection to previous state.
Expand or Collapse Trees 	Expands or collapses all constraint group trees.

Field	Description
Hide Unused Categories / Show All Categories 	Hides or shows unused, empty constraint groups.
Highlight and Unhighlight 	Highlights all instances of the selected constraint groups in the layout and schematic views. Each constraint group is highlighted in a different color.
Load 	Loads the constraint order from a preset file `constPresets.device.array`, which is located in the `.cadence/dfII/APR/<num>/presets` directory.
Save 	Saves the current constraint order to a preset file `constPresets.device.array`, which is located in the `.cadence/dfII/APR/<num>/presets` directory.
Table	Lists all available constraint finder groups. You can select the required finder groups to be run when you click *Find Structures*.
Find Structures	Searches for matching structures in the source cellview and lists them under respective categories in the *Groups and Constraints* pane.
Create Constraints	Generates the constraints in the form of Modgens and symmetry constraints in the Constraint Manager.

Setup

The following table describes the fields available on the *Setup* tab of the Auto P&R assistant.

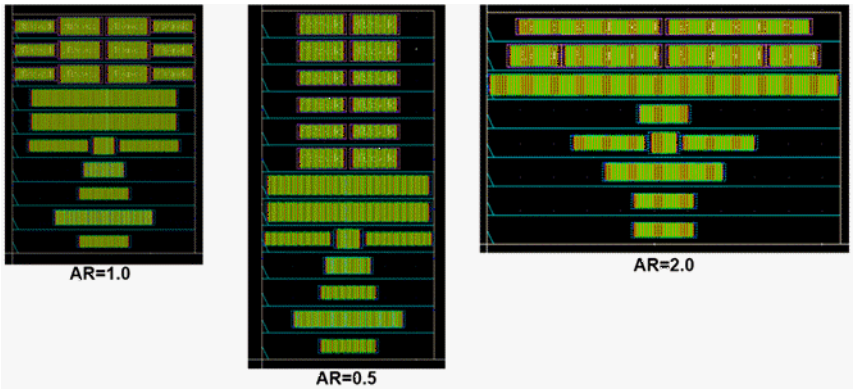
Field	Description
Specify Pin Positions & Attributes	Opens the *Pin Planner* tab of the Pin Placement form. Use the options in this form to set constraints and plan the placement of the pins in the design. The form also includes the *Pin Optimization* tab that lets you position the pins of blocks in a manner that helps obtain the shortest possible net length at a particular level in the design.
Row Region	Defines row region parameters.
Create Diffusion Grid	Creates a diffusion grid based on the heights of the diffusion layers of different devices in the design. All row region-related options are automatically inactivated. The *Create Diffusion Grid* check box appears on this tab only if enablePlaceWithWsp is set to `t` before launching the session. See Placing Multi-Height Devices Using Automatic Device Placer.
Create Row Region	Creates a row region as per your specifications. The options to specify row height are enabled. Environment variable: createPatternRegion By default, the name of the row region does not include the cell name. Set includeCellNameInRRName to `t` to include the cell name as part of the row region name.
Auto Compute Row Height	Specifies the row period. The option is selected by default. In this state, the tool calculates the row height based on the devices in the design, for example the maximum instance height and the heights of the gate, source, and drain tracks are considered. The calculated value is displayed in the *Row Height* field. If not selected, you can specify the required value. Environment variable: rowRegionMode
Use Passive Component Heights	Considers passive components, such as inductors, transformers, and transmission lines, for row height calculations. By default, passive components are ignored during row region and poly grid generation. Environment variable: usePassiveComponentHeights

Field	Description
Specify Row Height	Lets you specify the row height. Environment variable: rowHeight
Use Row Template	Lets you to import existing row templates. Environment variables: useRowTemplate, importFromCell, importFromLib, importFromView
Row Pattern	Defines row parameters.
Define PN Pattern	Controls the distribution of P and N devices in rows. The default value is */*/*, in which case the row region specification has a single row attribute. The available patterns are `PNNP` and `NPPN`. You can either select one of these patterns or specify a custom pattern, such as `NPNP`. You can apply the pattern and verify it in the Row Template Manager. Environment variable: definePNPattern
Generate Flipped Rows	Generates flipped rows and lets you select the orientation of the first (bottom-most) row. The available options are *Start with R0* and *Start with MX*. Environment variables: genFlippedRows, flipFirstRow
Poly Pattern	Defines poly layer patterns.
Create Poly Grid (WSP)	Creates poly layer patterns as WSPs. Environment variable: rowHeight
Specify Grid Offset	Specifies the offset of poly grids. Environment variables: specifyGridOffset, gridOffset
Include Passive Components	Recognizes the poly layer patterns of passive devices while generating WSPs. **Note:** Only physical layers are considered for height calculation. Environment variable: includePassiveComponentPoly
Generate button	Creates a row template, a row region, rows, and poly patterns, as specified.
Clear button	Clears the row region settings and resets them to their default values.

Place

The following table describes the fields available on the *Place tab* of the Auto P&R assistant.

Field	Description
Floor Plan	Specifies the region for placement. **Note:** The options in the *Floor Plan* section are not available when the placement mode is set to *Incremental* in the *Placement Options* section.

Field	Description
Aspect Ratio	Specifies that the placement area must be calculated based on the specified aspect ratio. Environment variable: autoPlaceFixedAR The default value is `1.0`, 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.  You can specify either a single value or multiple values in the *Aspect Ratio* field. When a single value is specified, the result is displayed in the same view. When multiple values are specified, a *_number* suffix is added at the view name. For example, for a view named `layout`, when the aspect ratio is set to `1`, the result is saved as `layout_1.00`. If a view with that name exists, `layout_1.00_0` is used. If that exists, then `layout_1.00_1` is used, and so on. If there is a comma after a single value, for example `1.0,`, the input is treated as multiple values and a new view is created. The valid formats for specifying multiple values are: ■ *`Aspect Ratio 1`*`,`*`Aspect Ratio 2`*`,...`: For example, `1.0,1.5,2.0` specifies three aspect ratios `1.0`, `1.5`, and `2.0`. ■ *`Start`*`:`*`End`*`:`*`Step`*: For example, `1.0:2.0:0.5` specifies aspect ratios `1.0`, `1.5`, and `2.0`.

Field	Description
	■ `[start:end]`: You can also specify the aspect ratio range, for example `[0.1, 5]`. The result has an aspect ratio that is larger than `0.1` and less than `5`. In *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. 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. To lock the aspect ratio of all Modgens while running the placer, set autoPlaceLockFGAR to `t`. The aspect ratio-related settings defined in the Array Assistant are ignored. When set to `nil` (default), the aspect ratio settings in the Array Assistant are honored while running the placer.
Boundary	Fits all the objects inside the specified boundary. ■ *Boundary*: Places all objects within the selected placement boundary. The default value is *PR Boundary*. Other placement boundaries that are available in the design are listed in the drop-down list. ■ *Custom Area*: Lets you either draw a boundary around the visible area or draw a custom area boundary in the design canvas. All objects are placed inside the area boundary. You can show or hide the boundary. In this mode, *Adjust PR Boundary* is not available. 
Lock Device Array Aspect Ratio	Locks the aspect ratio of all Modgens when running the Virtuoso device-level placer.

Field	Description
Adjust PR Boundary	Adjusts the PR boundary while running the placer according to the placement results. This option is available only when *Boundary* is set to *PR Boundary*. Environment variable: autoPlaceAdjustBoundary
Adjust Row Region	Automatically adjusts the row region while running the Virtuoso device-level placer according to the placement results. Environment variable: autoPlaceAdjustRowRegion
Placement Options	Customizes the automatic placer settings.
Auto	Runs the Virtuoso device-level automatic placer on the entire design.
Incremental	Runs the placer on unplaced devices such as 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, and therefore the options in the *Floor Plan* section are not available. However, the PR boundary and row regions are automatically adjusted. **Note:** The incremental placer does not run on objects that are located on the lower left of PR boundary. The incremental place run results depend on the starting layout.
Optimize	Lets you choose the following optimization options: When *Mode* is set to *Auto*, the available placement options are: *Area* (for compaction), *Wire Length* (for better routability), and *Area and Wire Length* (both compaction and better routability). When *Mode* is set to *Incremental*, the available placement options are: *Area* (for compaction), *Similarity* (for placement similar to the placed devices), and *Area and Similarity* (both compaction and similar placement).
Place Selected Only	Runs the placer only on those the objects that are selected in the layout canvas.

Field	Description
Ignore Passive Device	Ignores passive devices while running the placer. This option is useful when the design contains large passive devices that occupy most of the design aspect ratio. Honoring such passive devices might impact the placement quality of active components. With this option selected, the active devices are given a higher priority during placement.
Insert Trims to Fix Shorts	Inserts trims to devices automatically while running the automatic or incremental placer. All existing trims are deleted before inserting new ones. To avoid shorts, add transition spacing between virtual groups so that the trims in one virtual group do not overlap the bbox of any other virtual group. Also ensure that adjacent trim layers are at least two MD pins apart. If not, add dummies or change the placement to increase the spacing between them. **Note:** This option is available only in certain advanced node flows.
Transition Spacing	Adds transition rows or spacing above, below, or between the row regions. This space can be used for inserting guard rings, tap cells, or transition cells. The available options are: ■ *Compact*: The row vertical gap is set to `0` and the horizontal gap is set to the minimum value, following DRC rules. ■ *Specify*: Lets you specify absolute vertical and horizontal spacing values. You can also set the unit for vertical spacing to *microns* or *rows* to specify transition spacing in terms of the number of microns or the number of rows. Environment variables: vgSpacingMode, vgHorizontalSpaceSpecify, vgVerticalSpaceSpecify
Update	Updates the following settings after running the placer.
Adjust Boundary	Resizes the PR boundary such that it covers the devices placed outside the PR boundary.

Field	Description
Swap Colors of Flipped Rows	Swaps colors of flipped rows automatically while running the placer and updates the color swap parameters such that same-masked colors do not face each other. When selected, the `Color Swap` parameter for `MX` and `R180` devices (Devices in `MX` Rows) is turned on. When not selected, the `Color Swap` parameter for `R0` and `MY` devices (Devices in `R0` rows) is turned off. This is to ensure that same M0 mask color is not applied to adjacent tracks. **Note:** This option is available only in certain advanced node flows.
Enable Hierarchical Trims	Enables trim insertion in hierarchical designs. **Note:** This option is available only in certain advanced node flows.
Minimize M0 on Nets	Minimizes the capacitance on M0 nets in addition to the regular trim insertion. **Note:** This option is available only in certain advanced node flows.
Insert Trims to Fix Shorts	Lets you run the following tasks: ■ Insert trims to fix shorts automatically while running the placer. Trims are added to devices in the current cellview and its PR boundary. ■ Run checks to identify potential trim violations. ■ Delete trims to remove all shorts in the cellview. **Note:** This option is available only in certain advanced node flows.
Interactive Placer	Refines the interactive placement settings.
Highlight Virtual Groups	Highlights all virtual groups in the layout canvas. The icon automatically changes to *Dehighlight Virtual Group*, which lets you remove the highlights.
Row Snapping	Snaps devices to the nearest rows.
Show Information	Displays information about the placement.

Fill

The following table describes the fields available on the *Fill* tab of the Auto P&R assistant.

Field	Description
Fill Modgen Dummies	Inserts dummies to fill gaps between Modgens in virtual groups. You can select the required Modgens before clicking this icon. If no Modgens are selected, the command is run on all active device Modgens in the current cellview. If there are unabuttable devices in the Modgen, the tool checks cell fill to identify the fill object information for the gap and inserts identical dummies. Gaps caused by unabuttable devices are filled with the least number of dummies.
Dummy Type	Specifies the type of Modgen dummies to be inserted when the *Fill Modgen Dummies* button is clicked. The available dummy types are: ■ *Default*: Inserts dummies that match the devices, analog or stack gate analog cell. ■ *Analog Dummy*: Inserts dummies of type analog cell. ■ *Stack Dummy*: Inserts dummies of type stack gate analog cell. **Note:** This option is available only in certain advanced node flows. Environment variables: modgenDummyTypeAnalogLCV, modgenDummyTypeStackLCV, modgenDummyTypeStackNum
Dummy Fill	Specifies dummy fill settings.
Enable Dummy Fill	Inserts dummy fill in the gaps between instances. Environment variable: enableDummyFill,
Enable Hierarchical Fill	Specifies whether dummy fill can be inserted in hierarchical designs.

Field	Description
Extend Fill	Specifies whether the cell fill must be extended in all directions to cover the gaps within the PR boundary. With this option selected, the fill inserted in the empty spaces are further extended in the horizontal and vertical directions to cover all gaps within the PR boundary. Use environment variables vgFillSpaceSpecifyUnit and vgFillSpacePitchesOrUserUnit to specify the values by which fill are to be extended. Environment variables: dummyFillEnableFillEmptyRow,
Convert to Fig Group	Converts fill to a figGroup when the fill is executed on a draw region.
Poly Fill	Inserts poly fill in the gaps between instances. Poly fill extends the gate poly ands add cut poly rails as per the DRC rules. 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. Poly fill does not fill diffusion areas and areas with placement blockages.
Enable Poly Fill	Enables options to insert poly fill.
Fill Layer	Specifies the layer-purpose pair to be used by the placer to derive cut-poly fill.
Use Abutting layer	Uses the abutting layer as the fill layer instead of the poly fill layer. Select this option when there are instances with dummy poly to fill active gates with `Poly Drawing` layer and dummy poly with poly `dummy` layer.
Cut Layer	Sets the cut layer to `Cut Poly`.
Cut Width	Specifies the width of the cut-poly rails.
Diffusion Support	Allows poly fill to be inserted around diffusion shapes. When this option is not selected, poly fill are kept away from diffusion shapes.
Fill Area	Specifies the region to be filled.
Highlight Virtual Fill Boxes	Highlights the boxes in which fill is to be inserted.

Field	Description
Highlight Dummy Fills	Highlights dummies in the layout. Click this button after adding fill devices to differentiate between active devices and dummies.
Row Region	Specifies whether fill is inserted in all row regions or only in the row region selected from the drop-down list.
Custom Area	Inserts device fill in the custom area drawn.
Insert Fill	Generates the required dummy and poly fill as per your specifications.
Delete Fill	Deletes the selected device fill.
Create Guard Ring and *Delete Guard Ring*	These buttons are available only for certain advanced node flows. The Create Guard Ring and Delete Guard Ring buttons work for both row regions and custom area. When there is no a valid row region, these buttons are disabled. Add fill before creating a guard ring, especially if the existing virtual groups are not rectangular or contain gaps.

Related Topics

Initializing a Layout in the Automated Device Placement and Routing Flow

Generating Constraints and Constraint Groups

Virtuoso Automated Device Placement and Routing Flow

Generating and Deleting Base Layer Fill

Default Reuse Template Specification Form

(Layout MXL) Use the Default Reuse Template Specification form to apply the default Modgen reuse template to either the selected structures or to all structures in the selected layout.

Field	Description
Template Root Directory	Specifies the path to the reuse template file.

Field	Description
View Name	Specifies the name of the layout to which the reuse template must be applied.

Related Topics

Generating Constraints and Constraint Groups

Setting Options for Custom Layout Design Migration

Highlight Extract Group Form

Use the Highlight Extract Group form to select the constraint groups that are to be highlighted in the layout canvas.

Field	Description
List of Groups	Displays a list of constraint groups that are available in the active layout design. You can select the constraint groups that you want to highlight. These constraint groups are selected for custom design layout migration flow.

Related Topics

Generating Constraints and Constraint Groups

Setting Options for Custom Layout Design Migration

Routing Assistant User Interface for Device-Level Routing

The Routing assistant is a simplified, consistent, flow-based assistant for all design types. It has the same look and feel for device, standard cell and chip assembly routing with a common toolbar for all routing types. The Routing assistant is a dockable assistant pane that provides various options to let you perform tasks related to advanced routing technology.

The Routing assistant lets you:

- Select the routing mode and design type
- Load and save routing options
- Manage the routing flow through dedicated tabs
  - Lets you define the scope and options for each task and specifying the output
  - Lets you set up, run pre-routing checks, power grid generation, routing, and view results
  - Provides access to a Routing Results Browser
- Display alerts and other information about the width spacing patterns you are creating and updating
- Access the unified Routing Constraint Manager

The Routing Assistant has the following components:

Routing Assistant Toolbar	Lets you access the buttons to complete the steps of the routing flow.
Routing Assistant Tabs	Lets you specify the options for running the selected routing type.
Routing Assistant Command Buttons	Lets you access the buttons on each tab to compete a routing task.

Routing Assistant Toolbar

The following table lists the functions of the different buttons on the Routing assistant toolbar:

Icon	Command	Description
	*Change routing mode*	Selects the routing mode. The two routing modes are *Automatic* and *Interactive*.
	*Change routing type*	Changes the type of routing you want to run on the design. The three routing types are: *Device*, *StdCell*, *Chip Assembly*.
	*Raise the pre-routing browser*	Opens the pre-routing browser and provides net information prior to routing.
	*Set router constraints*	Opens the Routing Constraint Manager

Icon	Command	Description
	*Load preset options*	Lets you load options from an existing preset file.
	*Save preset options*	Lets you save the preset options to a file.
	*Delete preset options*	Lets you delete any existing preset options.

Routing Assistant Tabs

The following table lists the functions of the different tabs in the Routing assistant:

Tab	Description
Setup	Lets you specify the settings and generate or import WSPs for running the router.
Check	Lets you select the checks that you want to run before routing the design.
Supply	Lets you run the power router. You need to have power and ground wireType tracks in your width spacing patterns. Usually, you run this routing before placement.
Route	Lets you specify the scope and run routing on signal nets.
Results	Lets you select the scope and the results columns that should be displayed in Routing Results Browser.

The options in the Routing Assistant depend on the routing type in which the design is open. These routing types are:

- **Device-level** This routing flow help users to get familiar with the device-level placement and routing solution in Virtuoso for advanced-node, with the focus on uniform designs.
- **Standard Cell** This routing technology seamlessly integrates the NanoRoute router in the Virtuoso environment. It provides different ways for you to generate WSPs as well as route without them, relying on Innovus created tracks.
- **Chip Assembly** This routing technology is meant for routing a top-level design that has macro instances, I/O pads, and can also contain stdcell areas. It also addresses memory type designs using Spine routing.

Setup

The following table describes the fields available on the *Setup* tab of the Routing assistant for device-level routing type.

Field	Description
Options	
Insert trim to fix DRCs	Inserts trim shapes in the design to fix DRC errors. Environment variable: setup_insertTrim
Check DRCs after routing	Controls whether to automatically runs DRD design rule checks after routing. Environment variable: setup_checkDRCsAfterRouting
Lock Colors after routing	Controls whether to automatically lock colored shapes after routing. This option is currently grayed and unavailable for use. Environment variable: setup_lockColorsAfterRouting
Extraction Options	Lets you specify the layers for which to generate WSPs.
Extracted Pin Style	Lets you specify the pin extraction style before routing. The available options are: ■ Labeled shapes only ■ Connected shapes on same layer ■ Whole net on routing layers
Layer Settings	Lets you specify the layers for which to generate WSPs.
Wire CG	Specifies the constraint group used to supply a default set of constraints for routing modes. The default constraint group is `virtuosoDefaultSetup`.
Routing Layers	Specifies valid bottom and top layers for routing.
Wire Types Map	Opens a form with a table that maps the wireType to the symbol to be used in the pattern specification in the table.
Wire Types Assign to nets	Opens a form with a table that assigns wireTypes to nets.
Layer table	Provides a list of layers on which WSPs are generated.
Layer	The name of the layer.

Field	Description
Pattern	Generates the pattern of WSP as per your requirement. For example, `gs3ns2g3` where `g` stands for ground, `s` for signal, `n` for Null, and `p` for power.
Dir	The routing direction of the layer. The supported routing directions include *Horizontal*, *Vertical*, *Orthogonal*, and *Forbid*. *Forbid* implies that no direction is considered.
W	The width specified for the layer.
S	The spacing specified for the layer.
WSP	The width spacing patterns defined for the layer. All available WSPs are displayed in the *WSP* drop-down list. If any WSP pattern for a layer is not available in the Track Pattern Assistant, then the *WSP* drop-down appears blank. When you select a WSP from the drop-down list in the *Setup* tab of the Routing assistant, the WSP is active in the Track Pattern Assistant. If you make any WSP pattern active in the Track Pattern Assistant, it is displayed as an active option in the *WSP* drop-down list in the Setup tab of the Routing assistant.

Check

The following table describes the fields available on the *Check* tab of the Routing assistant for device-level routing type.

Field	Description
Check	Provides a list of checks that can be run for the routing type.
Select	Lets you specify the checks to be run. Click *All* or *None* to select or deselect all checks with a single click.
Routing Track Availability	Checks for the availability of routing tracks on a metal layer.
Pin WSP Conformance	Checks for the misalignment between metal layer pins and WSP tracks in terms of color, width, and alignment.
Track Spacing	Checks if the track spacing is DRC correct.

Field	Description
Via Availability & Density	Checks for congestion or limitations for via placement and track density.
Device Trims	Checks for insufficient space for DRC clean trim insertion.
Net Pin Symmetry	Checks the symmetrical nets pin placement.
Normal Pin Access	Checks if the metal layer pins are accessible and not dead or blocked.
Gate Pin Access	Checks for misalignment between WSP tracks and gate pins.
S/D Pin Overlaps	Checks for the misalignment of WSP tracks with source and drain pins.
Diffusion Pin Nets	Checks the mismatch between diffusion pin nets on device and top-level nets.
Trim Insertion Validity	Reruns the trim engine deleting and replacing the existing trims such that there are no trim related violations. **Note:** This option is available only in certain advanced node flows.
Existing DRCs	Specifies whether to run design rule checks for DRD. Environment variable: check_existingDRCs
Output	Settings to display the output of routing checks.
Display Log	Controls the display of the checker log in the CIW once the checks are run. Environment variable: check_displayLog
Overwrite last log	Controls the overwriting of the last log file. When the option is deselected, the existing log file is retained. Environment variable: check_overwriteLog
Markers	Controls the generation of markers for errors. These error markers can be viewed in the *Misc* tab of the Annotation Browser. Environment variable: check_generateMarkers

Supply

The following table describes the fields available on the *Supply* tab of the Routing assistant for device-level routing type.

Field	Description
Scope	Defines the scope of supply routing.
Supply Nets	Specifies whether to route all or selected supply nets. Environment variable: supply_nets
Within	Specifies whether to route everything inside the PR boundary or only within a guard ring or figGroup, or specify an area. You can also create supply stripes inside a row region or wsp region. The available options are: *PR boundary*, *Guardring/FigGroup*, *Area*, and *WSP/Row*. Environment variable: supply_netsWithin
Options	
Only use layers with WSP P/G tracks	Restricts supply routing to layers with existing power or ground tracks. However, when unselected, allows all routing layers to be used.
Supply stripes	Lets you select the bottom and the top metal layers for routing. Environment variable: supply_deleteStripes
Generate supply stripes	Generates stripes when supply routing is run. Environment variable: supply_genSupplyStripes
Insert vias for supply stripes	Inserts the vias between the intersection of the layer above and below the via. Environment variable: supply_insertVias
Connect to overlapped terminals and guardrings	Specifies whether or not the supply router should connect to instance terminal pins and guard rings. Environment variable: supply_connectToTermAndGR
Share tracks for supply nets	Determines whether or not the supply nets should share power and ground tracks as needed in the appropriate P and N regions. Environment variable: supply_shareTracks

Field	Description
Create only on P/N regions	Determines whether or not to only route inside Pch or Nch region or not under shared track mode. Environment variable: supply_createOnPNRegions
Create supply grid as a group	Specifies to create the supply grid as a figGroup. Environment variable: supply_createGridAsGroup
Ignore boundary tracks	Creates power stripes on tracks that are coincident with the PR boundary. When set to `t`, no power stripes (or power vias) are created coincident to the PR Boundary edge.
Ignore boundary vias	Creates power vias on tracks that are coincident with the PR boundary. When set to `t`, no power stripes (or power vias) are created coincident to the PR Boundary edge.
Assign to nets	Opens a form with a table that assigns wireTypes to nets.
Pull-back	Controls the spacing of power nets per layer from the PR boundary to avoid any DRC.
Pins	Lets you specify the pin related options for routing.
Create	Specifies to create a pin instead of a pathSeg for the stripe. Environment variable: supply_createPins
Create label	Creates labels on the pins.
Use all supply stripes layer	Creates pins on all supply stripe layers.
Use selected layers	Creates pins on selected supply stripe layers.
Create on ends	Creates pins only on the ends of the supply stripes.
Create on pin purpose	Enables to use pin as the layer purpose for the created pin. Environment variable: supply_createPinsOnPinPurpose
Update	
Delete Supply stripes	Deletes the stripes generated by the supply router before power routing. Environment variable: supply_deleteStripes
Delete Supply vias	Deletes the vias generated by the supply router before power routing. Environment variable: supply_deleteVias

Field	Description
Output	
Display log	Controls the display of the log in the CIW when supply routing is run. Environment variable: supply_displayLog
Overwrite last log	Specifies whether to overwrite the last log file or keep the existing one. When the option is selected, the existing log is overwritten. Environment variable: supply_overwriteLog
Current cellview	Write the output of the supply routing to the current cellview. Environment variable: supply_routedLoc
Other cellview	Lets you select a view name to write the output of the supply routing to another cellview. You can also specify a non-database existing name. The name of the cell is `$cell_proute` with `_proute` as the postfix to the cell name. The view name is always layout. Environment variable: supply_defaultRoutedView, supply_routedLoc, supply_defaultRoutedCellExpression
Save routing only	Specifies whether to copy only the supply grid or all initial data and the supply grid to the new cellview. Environment variable: supply_saveRoutingOnly

Route

The following table describes the fields available on the *Route* tab of the Routing assistant for device-level routing type.

Field	Description
Scope	Defines the scope of signal routing.
Nets	Specifies whether you want to route all nets, selected nets, or only nets with opens or shorts when the routing option is selected as *Auto*. When the routing options is selected as *Assisted*, the scope is *Selected* only, which allows routing of a single net at a time. Environment variable: route_nets

Field	Description
Include supply nets	Includes the power and ground nets for routing. This option is only available when the routing option is selected as *Auto*.
Within	Specifies whether to route everything inside the PR boundary, within a specified area, or within a device array or figGroup. Environment variable: route_netsWithin
Routing Options	Lets you specify the routing options for automatic and assisted routing.
Auto	When selected, lets you specify the options for automatic routing.
Optimize Wire length	Optimizes the wire length during routing.
Optimize Vias	Optimizes the via count during routing.
Connect on both sides of gate	Connects to both vias if there are two poly connections (vias).
Prefer finishing over constraints	Routes the larger sized nets first over constraint nets so that convergence is given preference over constraints.
Prefer finishing over DRC clean	
Preferred Range	Lets you specify the preferred bottom and top layers to be routed.
Update pins for optimized routing	
Assisted	Lets you specify the routing options for assisted routing.
Enable hierarchical trims	Enables trim insertion in hierarchical designs. **Note:** This option is available only in certain advanced node flows. Environment variable: route_enableHierarchicalTrims

Field	Description
Minimize M0 on nets	Minimizes the capacitance on M0 nets in addition to the regular trim insertion. **Note:** This option is available only in certain advanced node flows. Environment variable: route_minimizeM0OnNets
Insert M0 trims	Inserts M0 trim shapes to fix shorts. **Note:** This option is available only in certain advanced node flows.
Delete M0 trims	Deletes all M0 trim shapes. **Note:** This option is available only in certain advanced node flows.
General	
Preferred range	Lets you specify the bottom and top layers to be routed.
Finish routing	Lets you run finish routing on the preferred range of layers.
Mesh	
Mesh layers	Lets you specify the bottom and top layers to be routed in the mesh.
Extend preroutes for mesh	Lets you select whether or not to extend pre-routes to active meshing layer. Environment variable: route_meshExtendPreroutes
Generate mesh routing	Lets you create mesh on the specified layers of the selected nets within the specified scope.
P2T	
Trunk layer	Lets you specify the bottom and top layers to be routed in pin to trunk routing.
Create trunks	Creates trunks for the selected or all nets in the design. Environment variable: route_createTrunks
Connect instances to trunks	Connects pins of devices and macro instances to existing trunks. Environment variable: route_connectInstsToTrunks
Generate P2T routing	Lets you run the Pin to Trunk routing on the specified scope either on all the nets or selected nets.
Update	

Field	Description
Delete Select	Lets you select either all or no options for deletion. You can also select specific options.
Wires and vias	Deletes the wires and vias created by the router. Environment variable: route_deleteWiresAndVias
Trunks	Deletes the routed trunks. Environment variable: route_deleteTrunks
Manual routing	Deletes the manually routed wires and vias. Environment variable: route_deleteManualRouting
Output	
Display log	Controls the display of the log in the CIW when signal routing is run. Environment variable: route_displayLog
Overview last log	Specifies whether to overwrite the last log file or keep the existing one. When the option is selected, the existing log is overwritten. Environment variable: route_overwriteLog
Current cellview	Writes the output of the supply routing to the current cellview. Environment variable: route_defaultRoutedView
Other cellview	Lets you specify a view name to write the output of the signal routing to another cellview. Environment variable: route_defaultRoutedView
Save routing only	Specifies whether to copy only the supply grid or all initial data and the supply grid to the new cellview. Environment variable: route_saveRoutingOnly
Create routing as a group	Specifies whether or not to create routing as a figGroup. Environment variable: route_createRoutingAsAGroup

Results

The following table describes the fields available on the *Results* tab of the Routing assistant for device-level routing type.

Field	Description
Scope	Defines the scope of the routing results.
Nets	Specifies whether you want to show the routing result of all nets, selected nets, or only nets with opens. Environment variable: results_nets
Supply Nets	Shows the routing result of power and ground nets. Environment variable: results_supplyNets
Within	Specifies whether you want to show the routing result of nets inside the PR boundary or within a device array or figGroup. Environment variable: route_netsWithin
Summary of latest run	Displays the summary of the routing results for various parameters, such as *Time*, *Instances*, *Nets*, *DRCs*, *Opens*, *Shorts*, *Wire Length*, *Vias*.
Output	Specifies the output columns to be displayed in the Routing Results Browser
Select	Lets you select either all or none parameters for which output should be displayed. The parameters are: *Rule Violations*, *Symmetry Violations*, *Matched Length Violations*, *Shield Violations, Opens*, *Shorts*, *DRCs*, *Wirelength*, *Vias*, *Ratio*, *Pin count*, *Manhattan X*, and *Manhattan Y*.

Routing Assistant Command Buttons

The following table lists the functions of the different command buttons in the Routing assistant:

Icon	Command	Description
	*Snap pins to WSPs*	Snaps IO pins to width spacing patterns.
	*Show WSP Manager*	Displays WSP Manager.

Icon	Command	Description
	*Import WSPs*	Imports width spacing patterns to the current cellview.
	*Auto-generate WSPs*	Generates width spacing patterns automatically on the selected layers.
	*Run pre-route checks*	Runs pre-routing checks and saves the result to a log file.
	*Run Power route*	Runs power routing.
	*Run Signal router/Finish Routing*	Runs signal routing.
	*Generate mesh routing*	Runs mesh routing to let you create mesh on the specified layers.
	*Generate P2T routing*	Runs signal routing.
	*Show results browser*	Displays the Virtuoso Routing Results Browser.

Related Topics

Mesh Layer Setup Form

Accessing the Routing Assistant

Routing Assistant

Virtuoso Pre-Route Browser

Virtuoso Routing Results Browser

Virtuoso Routing Constraint Manager

Routing Constraint Manager User Interface

Mesh Layer Setup Form

The Mesh Layer Setup form is used to control the pitch of the metal layer mesh.

Column	Description
Layer	Displays the layer name on which mesh wire is to be created.
Control	Displays the selected control. The options are *Skip Tracks*, *Mesh Count*, or *Term Align*.
Value	The value depends on the selected control. ■ If *Skip Tracks* is selected, this is the value of the number of tracks that should be skipped between mesh wires. ■ If *Mesh Count* is selected, this value is the number of mesh wires that are created. ■ If *Term Align* is selected, it ensures that vertical mesh wires align with the source and drain pins of the devices.
Min tracks to skip	Specifies how many tracks should be skipped between mesh wires. Environment Variable: route_meshMinSkipTracks, route_meshMinSkipTracksCount

Related Topics

Routing Assistant User Interface for Device-Level Routing

B

Automated Device Placement and Routing Flow Environment Variables

The following list provides the names of the Automated Device-Level Placement and Routing Flow environment variables.

Device Initialization Settings

- apStartR0EvenRows
- apUseCustomSettings
- aprEnableSetPPConn
- pdkMapping

Device Constraints Settings

- abutAutoGroups
- aprCreateModgens
- aprCreateModgenFromTemplate
- arrangeByVGs
- groupPattern
- ignoreInstSelfSymmetry

Device Grid Settings

- createAlternateColoredWSP
- createDiffusionGrid
- createPolyPattern
- createRowRegion
- definePNPattern

- diffLayerNames
- enableDRCCompliantGridGen
- enableGFGRowRegionSnapping
- flipFirstRow
- flippedCellTypes
- genFlippedRows
- globalRowHeightSnappingMode
- gridOffset
- importFromCell
- importFromLib
- importFromView
- includeCellNameInRRName
- includePassiveComponentPoly
- metal_Spacing
- metal_Width
- metal_WSPMode
- multiPolyWSP
- patternName
- polyPurposes
- rowHeight
- rowRegionMode
- specifyGridOffset
- usePassiveComponentHeights
- useRowTemplate
- wireTypeAbbrev

Automatic WSP Generation Settings

- bottomWSPLayer
- createNonzeroWidthPoly
- createPatternRegion
- createRowRegion
- finGridLayerPattern
- finGridMaterialType
- gateTrackSpacing
- gateTrackWidth
- importFromCell
- importFromLib
- importFromView
- numSDTracks
- numTopGateTracks
- sdTrackMode
- sdTrackWidth
- topWSPLayer
- trackWidth
- useRowHeight
- useRowTemplate
- wireTypeAbbrev

Device Placement Settings

- apAwareMoveThreshold
- apAwareMoveThreshold
- autoPlaceAbutInstances
- autoPlaceAbutmentSymAxis
- autoPlaceAdjustBdryToUnplaceable

- autoPlaceAdjustBoundary
- autoPlaceAdjustRowRegion
- autoPlaceAreaCost
- autoPlaceAreaCost
- autoPlaceCustomArea
- autoPlaceExcludeList
- autoPlaceFixedAR
- autoPlaceFixedW
- autoPlaceIgnorePassive
- autoPlaceInsertTrims
- autoPlaceLockFGAR
- autoPlaceRegion
- autoPlaceSelectedOnly
- autoPlaceSimilarityCost
- autoPlaceSymAxis
- autoPlaceWireLenCost
- calcBBoxIgnoreMaterialTypes
- calcFGBoxFromMember
- enablePlaceWithWsp
- enablePlaceWithWsp
- ignoreOverlapCheckProp
- init_boundaryAspectRatioOrHeight
- init_boundaryAspectRatioVal
- init_boundaryHeightVal
- init_boundaryUtilizationOrWidth
- init_boundaryUtilizationVal
- init_boundaryWidthVal

- init_createPowerPins
- init_generateBoundary
- init_generateInstances
- init_generatePins
- init_mode
- init_scope
- init_scope
- init_useCustomSettings
- init_useSourceLayout
- init_useSourceLayoutBoundary
- init_useSourceLayoutConstraints
- init_useSourceLayoutInstances
- init_useSourceLayoutPins
- matureOrAdvancedNode
- optimization
- placePOverN
- regenModgenPostProcess
- setup_createPolyPattern
- setup_definePNPattern
- setup_flippedRowsStartWith
- setup_genFlippedRows
- setup_gridOffset
- setup_includePassiveComponentPoly
- setup_patternName
- setup_placementRegion
- setup_rowRegionMode
- setup_specifyGridOffset

- setup_specifyRowHeight
- setup_usePassiveComponentHeights
- setup_useRowTemplate
- **vgFillSpacePitchesOrUserUnit**
- vgFillSpaceSpecifyUnit
- vgHorizontalIntraGroupSpaceInPitches
- vgHorizontalSpaceSpecify
- vgPriority
- vgSpacingMode
- `vgVerticalSpaceSpecify`
- `vgVerticalSpaceSpecify`

Device Fill Settings

- activeToActiveMinFingers
- activeToPRBoundaryMinFingers
- allowBackAnnotateForAllCells
- fillRegionHonorTransitionSpacing
- forceAbutM0Regen
- ftiToActiveMinFingers
- ftiToFtiMinFingers
- ftiToPRBoundaryMinFingers
- ignoreColorCheck
- modgenDummyFillType
- modgenDummyStacksMinSize
- multiFingerTransFill
- stackDummyFingerCount
- transitionFillFingerCount

- transitionUseDummyCell

Device Infrastructure Settings

- allowM0OnUIControlsWhenApplicable
- createBidirectionalHalos
- createViaKeepoutMetalHalos
- createViaKeepoutZoneHalos
- echoViaHalos
- echoViaVariants
- enableAlternateVias
- enablePolyViaHalos
- mergeWeakPins
- nullViaVariantAnError
- pinPurposes
- preferSmallestCutClass
- preferStandardVias
- strictShieldHalos
- useHaloFile
- viaParamOverrides

Auto P&R Assistant Settings

- **aprAssistantMode**

layoutXL.APassist

- backAnnotateAll
- dummyFillEnableFillEmptyRow
- dummyFillLib

layoutXL.AP

- useLayoutAsSeed

- dummyFillCell
- dummyFillEnableFillEmptyRow
- dummyFillLib
- dummyFillNeighbor
- dummyFillNeighborMultipleMode
- dummyFillNetMatchNeighbor
- dummyFillNetName
- dummyFillView
- enableDummyFill

Auto Device Array

- adaDisableTab
- aprEnableSetPPConn
- aprCreateModgens
- aprCreateModgenFromTemplate
- arrayTemplateFileDir
- modgenDummyTypeAnalogLCV
- modgenDummyTypeStackLCV
- modgenDummyTypeStackNum

Device Router Settings

- addStackedMetalsOnNets
- check_displayLog
- check_existingDRCs
- check_generateMarkers
- check_logDir
- check_logFile
- check_overwriteLog

- colorToStackedMetals
- connectAllMetalizedPins
- controlDeviceLayers
- extendPaths
- extractDepth
- extractStopLevel
- meshViaStackLimit
- multiConn
- multiCutsOnPin
- postRouteTrigger
- preferredLayerStrength
- preRouteTrigger
- prouteGroupByLayer
- prouteUsePinPurposeForCreatedPin
- prouteWaivedMarkerLayer
- results_nets
- results_netsWithin
- results_supplyNets
- route_connectBothSidesOfGate
- route_connectInstsToTrunks
- route_createRoutingAsAGroup
- route_defaultRoutedView
- route_deleteManualRouting
- route_deleteTrunks
- route_deleteWiresAndVias
- route_displayLog
- route_enableHierarchicalTrims

- route_finishOverConstraints
- route_finishOverDRCClean
- route_meshExtendPreroutes
- route_meshMinSkipTracks
- route_meshMinSkipTracksCount
- route_meshNets
- route_meshNetsWithin
- route_minimizeM0OnNets
- route_nets
- route_netsWithin
- route_objective
- route_overwriteLog
- route_routedLoc
- route_routingMode
- route_saveRoutingOnly
- route_supplyNets
- route_updatePins
- route_weightedSumDistance
- setup_checkDRCsAfterRouting
- setup_insertTrim
- setup_lockColorsAfterRouting
- shieldInsertBlockageInsideShieldGap
- showOpensTreeBoxes
- supply_connectToTermAndGR
- supply_createGridAsGroup
- supply_createOnPNRegions
- supply_createPinLabel

- supply_createPins
- supply_createPinsOnEnds
- supply_createPinsOnPinPurpose
- supply_defaultRoutedCellExpression
- supply_defaultRoutedView
- supply_deleteStripes
- supply_deleteVias
- supply_displayLog
- supply_genSupplyStripes
- supply_generateStaples
- supply_IgnoreBoundaryTracks
- supply_IgnoreBoundaryVias
- supply_insertVias
- supply_nets
- supply_netsWithin
- supply_overwriteLog
- supply_pinLayers
- supply_routedLoc
- supply_pinLayerSet
- supply_saveRoutingOnly
- supply_shareTracks
- supply_useExisitingPGTracks
- tieShieldTieCellList

Device Infrastructure Settings

- addStackedMetalsOnNets
- check_displayLog

- check_existingDRCs
- check_generateMarkers
- check_logDir
- check_logFile
- check_overwriteLog
- colorToStackedMetals
- connectAllMetalizedPins

abutAutoGroups

```
APR.device.constraint abutAutoGroups boolean { t | nil }
```

Description

Specifies whether the device groups generated on the *Constraints* tab of the Auto P&R assistant are to be abutted automatically.

The default value is `t`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.constraint" "abutAutoGroups")
envSetVal("APR.device.constraint" "abutAutoGroups" 'boolean nil)
```

Related Topics

Generating Constraints and Constraint Groups

Auto P&R Assistant User Interface for Device-Level Placement

activeToActiveMinFingers

```
APR.device.fill activeToActiveMinFingers int minFingers
```

Description

Specifies the minimum gap between active instances in terms of the minimum number of fingers of the dummy fills.

The default value is `5`, which indicates that the gap must be at least 5 fingers wide.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.fill" "activeToActiveMinFingers")
envSetVal("APR.device.fill" "activeToActiveMinFingers" 'int 7)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Auto P&R Assistant User Interface for Device-Level Placement

activeToPRBoundaryMinFingers

ftiToActiveMinFingers

ftiToFtiMinFingers

ftiToPRBoundaryMinFingers

activeToPRBoundaryMinFingers

```
APR.device.fill activeToPRBoundaryMinFingers int int_minFingers
```

Description

Specifies the minimum gap between active instances and the PR boundary in terms of the minimum number of fingers of the dummy fills.

The default value is `6`, which indicates that the gap must be at least 6 fingers wide.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.fill" "activeToPRBoundaryMinFingers")
envSetVal("APR.device.fill" "activeToPRBoundaryMinFingers" 'int 8)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

activeToActiveMinFingers

ftiToActiveMinFingers

ftiToFtiMinFingers

ftiToPRBoundaryMinFingers

adaDisableTab

```
gpe adaDisableTab string "tabName"
```

Description

Makes the specified tab unavailable in the Array Assistant. You can specify one or more the following tabs: `"Placement" "GuardRing" "Routing"` and `"Reuse"`.

The default is `""`, where all tabs are available.

GUI Equivalent

None

Examples

```
envGetVal("gpe" "adaDisableTab")
envSetVal("gpe" "adaDisableTab" 'string "GuardRing Routing")
```

Related Topics

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Automatic Generation of Modgens using the Array Assistant

addStackedMetalsOnNets

```
APR.device.route addStackedMetalsOnNets string "{netName metalLayer1
     metalLayer2}"
```

Description

Add additional stacked metals and vias on nets.

The default value is `""`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "addStackedMetalsOnNets")
```

The following example creates additional stacked metals and vias from `M2` to `M4` on net `tail` and net `tail_b` pathSegs:

```
envSetVal("APR.device.route" "addStackedMetalsOnNets" 'string "{tail M2 M4}
{tail_b M2 M4}")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

allowBackAnnotateForAllCells

```
APR.device.fill allowBackAnnotateForAllCells boolean { t | nil }
```

Description

Backannotates transition and non-transition fills from the layout to the schematic view.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.fill" "allowBackAnnotateForAllCells")
envSetVal("APR.device.fill" "allowBackAnnotateForAllCells" 'boolean nil)
```

Related Topics

Base Layer Fill in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

allowM0OnUIControlsWhenApplicable

```
APR.device.infra allowM0OnUIControlsWhenApplicable boolean { t | nil }
```

Description

Enables M0 on flow specific features and options for applicable PDKs.

The default value is `t`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.constraint" "allowM0OnUIControlsWhenApplicable")
envSetVal("APR.device.constraint" "allowM0OnUIControlsWhenApplicable" 'boolean
nil)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

aprAssistantMode

```
APR aprAssistantMode cyclic { "Device" | "stdcell" }
```

Description

Specifies the mode in which the Auto P&R assistant is to be run. The valid values are:

- `Device`: Loads the device placement options in the Auto P&R assistant.
- `stdcell`: Loads the standard cell placement options in the Auto P&R assistant.

The default value is `Device`.

GUI Equivalent

Command	Auto P&R assistant
Field	 icon

Examples

```
envGetVal("APR" "aprAssistantMode")
envSetVal("APR" "aprAssistantMode" 'cyclic "stdcell")
```

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

Auto Device Placement and Routing Workspace

apAwareMoveThreshold

`APR.device.place apAwareMoveThreshold int` *`int_MoveThreshold`*

Description

Specifies the maximum number of instances that can be moved during assisted placement.

The default value is `10`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "apAwareMoveThreshold")
envSetVal("APR.device.place" "apAwareMoveThreshold" 'int 30)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

aprCreateModgens

```
APR.device.constraint aprCreateModgens boolean { t | nil }
```

Description

Creates Modgens in the *View and Edit Constraints* section of the *Constraints* tab of the Auto P&R assistant. When set to `nil`, symmetry constraints are created.

The default value is `t`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.constraint" "aprCreateModgens")
envSetVal("APR.device.constraint" "aprCreateModgens" 'boolean nil)
```

Related Topics

Generating Constraints and Constraint Groups

Auto P&R Assistant User Interface for Device-Level Placement

aprCreateModgenFromTemplate

```
APR.device.constraint aprCreateModgenFromTemplate boolean { t | nil }
```

Description

Specifies whether the default Modgen reuse template file is applied to all the Modgens that are generated in the *Constraints* step of the automated device placement and routing flow.

The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.constraint" "aprCreateModgenFromTemplate")
envSetVal("APR.device.constraint" "aprCreateModgenFromTemplate" 'boolean t)
```

Related Topics

Applying Default Modgen Reuse Template

Generating Constraints and Constraint Groups

Auto P&R Assistant User Interface for Device-Level Placement

aprEnableSetPPConn

```
APR.device.initialize aprEnableSetPPConn boolean { t | nil }
```

Description

Specifies whether pseudoparallel nets are to be generated in the Virtuoso automated device placement and routing flow. The default value is `t`.

For more information about pseudoparallel nets, see Defining Pseudoparallel Connections.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.initialize" "aprEnableSetPPConn")
envSetVal("APR.device.initialize" "aprEnableSetPPConn" 'boolean nil)
```

Related Topics

Generating Constraints and Constraint Groups

Auto P&R Assistant User Interface for Device-Level Placement

apStartR0EvenRows

```
APR.device.initialize apStartR0EvenRows cyclic { "Follow" | "Auto" | "Off" }
```

Description

Specifies the virtual grouping behavior while running the placer in the presence of guard rings. The available options are:

- `Follow`: Honors placement settings in even rows and R0 cell categories, even if guard rings are present in the design.
- `Auto`: Ignores placement settings in even rows and R0 cell categories if there are guard rings in the design.
- `Off`: Switches off all special placement settings before running the placer.

The default value is `Auto`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.initialize" "apStartR0EvenRows")
envSetVal("APR.device.initialize" "apStartR0EvenRows" 'cyclic "Off")
envSetVal("APR.device.initialize" "apStartR0EvenRows" 'cyclic "Follow")
```

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

apUseCustomSettings

```
APR.device.initialize apUseCustomSettings boolean { t | nil }
```

Description

Specifies whether custom PDK settings are to be loaded from a file. This option is useful for loading additional parameters and values, for example fill overrides, environment variables, and device registration that are applicable only at lower nodes.

The default value is `nil`. When set to `t`, use pdkMapping to specify the file that contains custom PDK definitions.

GUI Equivalent

Command	Auto P&R assistant – *Initialize* tab
Field	*Use custom settings*

Examples

```
envGetVal("APR.device.initialize" "apUseCustomSettings")
envSetVal("APR.device.initialize" "apUseCustomSettings" 'boolean t)
```

Related Topics

Initializing a Layout in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

arrangeByVGs

```
APR.device.constraint arrangeByVGs boolean { t | nil }
```

Description

Specifies whether all devices are to be arranged in columns according to their virtual groups after running the constraints step of the Virtuoso automated device placement flow.

The default value is `nil`.

GUI Equivalent

Command	Auto P&R assistant – *Constraints* tab
Field	*Arrange By Virtual Groups*

Examples

```
envGetVal("APR.device.place" "arrangeByVGs")
envSetVal("APR.device.place" "arrangeByVGs" 'boolean t)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

arrayTemplateFileDir

```
APR.device.migrate arrayTemplateFileDir string "templateFileDirectory"
```

Description

Specifies the reference directory for Virtuoso layout reuse flows to either load or extract array template files. Subdirectories and array template files can be read and written in the specified reference directory.

The default directory is `"./.cadence/dfII/Migrate/templateLib".`

GUI Equivalent

None

Examples

```
envGetVal("APR.device.migrate" "arrayTemplateFileDir")
envSetVal("APR.device.migrate" "arrayTemplateFileDir" 'string "./.cadence/dfII/
MigrateTemplates/multiplePatterns")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceAbutInstances

```
APR.device.place autoPlaceAbutInstances boolean { t | nil }
```

Description

Abuts single instances after running the placer. Two single instances can be abutted only when:

- They are in the same virtual group.
- Their positions along the y-axis are the same.
- They are adjacent in their positions along the x-axis, which means there is no figGroup between them.

The default value is `nil`, in which case single instances are not automatically abutted.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "autoPlaceAbutInstances")
envSetVal("APR.device.place" "autoPlaceAbutInstances" 'boolean t)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceAbutmentSymAxis

```
APR.device.place autoPlaceAbutmentSymAxis string "abutSymAxis"
```

Description

Specifies the symmetry axis to be used to abut single instances when autoPlaceAbutInstances is set to `t`. You can either set the value to `none` or specify a value in the float format, for example, `"3.5"`. The x-coordinate of the symmetry axis is set to the specified value.

The default value is `none`, in which case the tool determines the symmetry axis in the following order:

1. The tool checks for a symmetry constraint in the design and applies its symmetry axis position.
2. If a symmetry constraint is not found, the tool assigns the center of the PR boundary as the symmetry axis position.
3. If no PR boundary is found, the instances are abutted to the left.

You can use useDeviceOrder to control whether device abutment must follow device order.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "autoPlaceAbutmentSymAxis")
envSetVal("APR.device.place" "autoPlaceAbutmentSymAxis" 'string "3.6")
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceAdjustBdryToUnplaceable

```
APR.device.place autoPlaceAdjustBdryToUnplaceable boolean { t | nil }
```

Description

Control whether to consider excluded instances when adjusting boundary at the end of placement.

The default value is `t`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "autoPlaceAdjustBdryToUnplaceable")
envSetVal("APR.device.place" "autoPlaceAdjustBdryToUnplaceable" 'boolean nil)
```

Related Topics

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceAdjustBoundary

```
APR.device.place 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("APR.device.place" "autoPlaceAdjustBoundary")
envSetVal("APR.device.place" "autoPlaceAdjustBoundary" 'boolean t)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceCustomArea

`APR.device.place autoPlaceCustomArea string` *`areaCoordinates`*

Description

Specifies coordinates that define a custom area boundary in the design canvas. All objects are placed inside the area boundary.

The default value is `none`, which indicates that no area is selected.

GUI Equivalent

Command	Auto P&R assistant – *Place tab*
Field	*Boundary – Custom Area*

Examples

```
envGetVal("APR.device.place" "autoPlaceCustomArea")
envSetVal("APR.device.place" "autoPlaceCustomArea" 'string "((120 185) (1020 875))")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceSimilarityCost

```
APR.device.place autoPlaceSimilarityCost float float_number
```

Description

Specifies the similarity value, depending on which the placer attempts to achieve a placement similar to the placed devices.

The default is `0.0`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "autoPlaceSimilarityCost")
envSetVal("APR.device.place" "autoPlaceSimilarityCost" 'float 2.0)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceAdjustRowRegion

```
APR.device.place 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("APR.device.place" "autoPlaceAdjustRowRegion")
envSetVal("APR.device.place" "autoPlaceAdjustRowRegion" 'boolean t)
envSetVal("APR.device.place" "autoPlaceAdjustRowRegion" 'boolean nil)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceAreaCost

`APR.device.place 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("APR.device.place" "autoPlaceAreaCost")
envSetVal("APR.device.place" "autoPlaceAreaCost" 'float 2.0)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceExcludeList

```
APR.device.place autoPlaceExcludeList string "devices"
```

Description

Specifies a list of devices to be excluded during placement. You can specify either individual device names or the name of the master that contains the devices to be excluded.

The default is `""`, which implies that no devices are excluded.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "autoPlaceExcludeList")
```

The following example specifies the names of the devices to be excluded:

```
envSetVal("APR.device.place" "autoPlaceExcludeList" 'string, "device deviceName1
deviceName2")
```

The following example specifies the name of the device masters:

```
envSetVal("APR.device.place" "autoPlaceExcludeList" 'string "master library1 cell1
library2 cell2")
```

Related Topics

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceFixedAR

`APR.device.place autoPlaceFixedAR float` *`float_number`*

Description

Specifies the aspect ratio for generating the PR boundary.

The default is `-1.0`.

GUI Equivalent

Command	Auto P&R assistant – *Place tab*
Field	*Aspect Ratio*

Examples

```
envGetVal("APR.device.place" "autoPlaceFixedAR")
envSetVal("APR.device.place" "autoPlaceFixedAR" 'float 0.5)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceFixedW

`APR.device.place autoPlaceFixedW float` *`float_number`*

Description

Specifies the width of the PR boundary.

The default is `-1.0`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "autoPlaceFixedW")
envSetVal("APR.device.place" "autoPlaceFixedW" 'float 0.5)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceIgnorePassive

```
APR.device.place autoPlaceIgnorePassive boolean { t | nil }
```

Description

Ignores passive devices while running the placer. Set this option to `t` when the design contains large passive devices that might impact the placement quality of active components.

The default is `nil`.

GUI Equivalent

Command	Auto P&R assistant – *Place tab*
Field	*Ignore Passive Device*

Examples

```
envGetVal("APR.device.place" "autoPlaceIgnorePassive")
envSetVal("APR.device.place" "autoPlaceIgnorePassive" 'boolean t)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceInsertTrims

```
APR.device.place autoPlaceInsertTrims boolean { t | nil }
```

Description

Specifies whether trims are to be inserted after running the automatic placer. This environment variable is valid only in certain advanced node flows.

The default value is `t`. When set to `nil`, trims are not inserted.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "autoPlaceInsertTrims")
envSetVal("APR.device.place" "autoPlaceInsertTrims" 'boolean nil)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceLockFGAR

```
APR.device.place 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 Array Assistant are honored.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "autoPlaceLockFGAR")
envSetVal("APR.device.place" "autoPlaceLockFGAR" 'boolean t)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

Automatic Generation of Modgens using the Array Assistant

placePOverN

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

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceRegion

`APR.device.place autoPlaceRegion string` *`boundary`*

Description

Specifies the placement region in which all objects are to be placed. Valid values are all placement regions defined in the current design.

The default value is `Boundary`, which refers to the PR boundary.

GUI Equivalent

Command	Auto P&R assistant – *Place tab*
Field	*Boundary*

Examples

```
envGetVal("APR.device.place" "autoPlaceRegion")
envSetVal("APR.device.place" "autoPlaceRegion" 'string "MyBoundary")
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceSelectedOnly

```
APR.device.place autoPlaceSelectedOnly boolean { t | nil }
```

Description

Runs the device-level interactive placer only on the objects that are selected in the layout canvas.

The default is `nil`.

GUI Equivalent

Command	Auto P&R assistant – *Place tab*
Field	*Place Selected Only*

Examples

```
envGetVal("APR.device.place" "autoPlaceSelectedOnly")
envSetVal("APR.device.place" "autoPlaceSelectedOnly" 'boolean t)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

autoPlaceSymAxis

`APR.device.place autoPlaceSymAxis string` *`axis`*

Description

Specifies the symmetry axis to be applied when the Virtuoso device-level automatic placer places devices. Ensure that the specified value is within the range of the PR boundary.

The default value is `none`, in which case, the center of the PR boundary is used as the symmetry axis.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "autoPlaceSymAxis")
envSetVal("APR.device.place" "autoPlaceSymAxis" 'string "prBLeft")
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

autoPlaceWireLenCost

```
APR.device.place 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("APR.device.place" "autoPlaceWireLenCost")
envSetVal("APR.device.place" "autoPlaceWireLenCost" 'float 2.0)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

backAnnotateAll

```
layoutXL.APAssist backAnnotateAll boolean { t | nil }
```

Description

Backannotates active and inactive dummies from the layout to the schematic view.

The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL.APAssist" "autoPlaceSelectedOnly")
envSetVal("layoutXL.APAssist" "autoPlaceSelectedOnly" 'boolean nil)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

bottomWSPLayer

```
layoutXL.autoWSPGen bottomWSPLayer string "layerName"
```

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 P&R assistant – *Setup* tab
Field	*Bottom Layer*

Examples

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

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

calcBBoxIgnoreMaterialTypes

```
APR.device.place calcBBoxIgnoreMaterialTypes string "List_of_Materials"
```

Description

Specifies the materials to be ignored while running the placer. The default value lists the following materials: `"trim cut nWell pWell nImplant pImplant recognition other"`.

When you click *Adjust Boundary* on the *Place tab* of the Auto P&R assistant, the layers that contain the specified materials are excluded during row height calculation.

In lower node designs that have a PR boundary, set this environment variable to avoid shorts and overlaps.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "calcBBoxIgnoreMaterialTypes")
envSetVal("APR.device.place" "calcBBoxIgnoreMaterialTypes" 'string "cut nWell pWell nImplant")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Place tab of the Auto P&R assistant

calcFGBoxFromMember

```
APR.device.place calcFGBoxFromMember boolean { t | nil }
```

Description

Specifies whether the Modgen bounding box must be computed based on instance membership.

The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "calcFGBoxFromMember")
envSetVal("APR.device.place" "calcFGBoxFromMember" 'boolean t)
```

Related Topics

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

check_displayLog

```
APR.device.route check_displayLog boolean { t | nil }
```

Description

Controls the display of the checker log window once the checks are run. When set to `nil`, the log window is not displayed.

The default is `t`.

GUI Equivalent

Command	Routing assistant – *Check* tab
Field	*Display log*

Example

```
envGetVal("APR.device.route" "check_displayLog")
envSetVal("APR.device.route" "check_displayLog" 'boolean nil)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

check_existingDRCs

```
APR.device.route check_existingDRCs boolean { t | nil }
```

Description

Specifies whether to run design rule checks for DRD.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Check* tab
Field	*Existing DRCs*

Examples

```
envGetVal("APR.device.route" "check_existingDRCs")
envSetVal("APR.device.route" "check_existingDRCs" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

check_generateMarkers

```
APR.device.route check_generateMarkers boolean { t | nil }
```

Description

Controls the generation of markers for errors. These error markers can be viewed in the *Misc* tab of the Annotation Browser. When set to `t`, the markers are generated.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Check* tab
Field	*Markers*

Examples

```
envGetVal("APR.device.route" "check_generateMarkers")
envSetVal("APR.device.route" "check_generateMarkers" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

check_logDir

```
APR.device.route check_logDir string "directory"
```

Description

Specifies the path to the directory that saves the checker log file from the *Check* tab of the device-level router.

The default directory is an empty string `""`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "check_logDir")
envSetVal("APR.device.route" "check_logDir" 'string "logdirectory1")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

check_logFile

```
routing.device.route check_logFile string "logprefixname"
```

Description

Specifies the prefix for the checker log file names. The checker log files follow the pattern of *logprefixname*.*libName*.*cellName*.*viewName*.log.

The default is `deviceCheckResults.log`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "check_logFile")
envSetVal("APR.device.route" "check_logFile" 'string "devicechecker")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

check_overwriteLog

```
APR.device.route check_overwriteLog boolean { t | nil }
```

Description

Controls the overwriting of the last log file. When set to `nil`, the existing log file is retained.

The default is `t`.

GUI Equivalent

Command	Routing assistant – *Check* tab
Field	*Overwrite last log*

Examples

```
envGetVal("APR.device.route" "check_overwriteLog")
envSetVal("APR.device.route" "check_overwriteLog" 'boolean nil)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

colorToStackedMetals

```
routing.device.route colorToStackedMetals string "colorStyle"
```

Description

Defines color style for stacked wires.

The default is an empty string.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "colorToStackedMetals")
envSetVal("APR.device.route" "colorToStackedMetals" 'string "RedBlue")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

connectAllMetalizedPins

```
APR.device.route connectAllMetalizedPins boolean { t | nil }
```

Description

Forces to route more than one metalized pins from a single device pin. Metalized pins must be on the first routing layer. The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "connectAllMetalizedPins")
envSetVal("APR.device.route" "connectAllMetalizedPins" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

controlDeviceLayers

```
APR.device.route controlDeviceLayers string "{ "Y" | "N" }"
```

Description

Forces device layers to be included or excluded for signal and spine routing. This overrides any argument provided to the router or PDK-specific default behavior.

To always include device layers:

```
envSetVal(APR.device.router" "controlDeviceLayers" 'string "Y")
```

To always remove device layers:

```
envSetVal("APR.device.router" "controlDeviceLayers" 'string "N")
```

Any other settings (including the default `""`) has no effect.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "controlDeviceLayers")
envSetVal("APR.device.route" "controlDeviceLayers" 'string "Y")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

createAlternateColoredWSP

```
APR.device.grid createAlternateColoredWSP boolean { t | nil }
```

Description

Specifies whether alternate-colored width spacing patterns (WSPs) are to be generated when flipped rows are created.

The default value is `t`, in which case alternate-colored WSPs are generated. When set to `nil`, the value specified in the WSP Manager is used.

GUI Equivalent

Command	*Create – P&R Objects – Width Spacing Patterns*
Field	*Track Color – Alternate*

Examples

```
envGetVal("APR.device.grid" "createAlternateColoredWSP")
envSetVal("APR.device.grid" "createAlternateColoredWSP" boolean nil)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Using the WSP Manager

createBidirectionalHalos

```
APR.device.infra createBidirectionalHalos boolean { t | nil }
```

Description

Specifies whether or not to use the full via halo in both directions as opposed to restricting it to be in the preferred layer direction. This behavior is limited to metal layers and does not affect halos on cut layers.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "createBidirectionalHalos")
envSetVal("APR.device.infra" "createBidirectionalHalos" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

createDiffusionGrid

```
APR.device.grid createDiffusionGrid boolean { t | nil }
```

Description

Specifies whether a diffusion grid is to be generated as part of grid generation in the automated device placement and routing flow.

The default value is `nil`, which implies that a diffusion grid is not generated by default.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "createDiffusionGrid")
envSetVal("APR.device.grid" "createDiffusionGrid" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Using the WSP Manager

createMinimumN3Halos

```
APR.device.infra createMinimumN3Halos boolean { t | nil }
```

Description

Specifies whether or not only minimum spacing halos are created for vias with N3 processes.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "createMinimumN3Halos")
envSetVal("APR.device.infra" "createMinimumN3Halos" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

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 nil)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

createPolyPattern

```
APR.device.grid createPolyPattern boolean { t | nil }
```

Description

Specifies whether poly snap patterns must be generated.

The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "createPolyPattern")
envSetVal("APR.device.grid" "createPolyPattern" 'boolean nil)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

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)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

createRowRegion

```
APR.device.grid createRowRegion boolean { t | nil }
```

Description

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

The default is `t`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Create Row Region*

Examples

```
envGetVal("APR.device.grid" "createRowRegion")
envSetVal("APR.device.grid" "createRowRegion" 'boolean nil)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

createViaKeepoutMetalHalos

```
APR.device.infra createViaKeepoutMetalHalos boolean { t | nil }
```

Description

Specifies whether or not halos are created for viaKeepoutMetal rules.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "createViaKeepoutMetalHalos")
envSetVal("APR.device.infra" "createViaKeepoutMetalHalos" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

createViaKeepoutZoneHalos

```
APR.device.infra createViaKeepoutZoneHalos boolean { t | nil }
```

Description

Specifies whether or not halos are created for viaKeepoutZone rules.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "createViaKeepoutZoneHalos")
envSetVal("APR.device.infra" "createViaKeepoutZoneHalos" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

definePNPattern

```
APR.device.grid definePNPattern boolean { t | nil }
```

Description

Specifies whether the distribution of P and N devices in rows is based on the rowHeight setting.

The default value is `nil`, in which case the row region has a single row attribute.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Define P/N Pattern*

Examples

```
envGetVal("APR.device.grid" "definePNPattern")
envSetVal("APR.device.grid" "definePNPattern" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

diffLayerNames

`APR.device.grid diffLayerNames string` *`cellName`*

Description

Specifies the cell of the instance to be considered as the master for creating dummy fill. This environment variable is applicable only when dummyFillNeighbor is set to `nil`.

The default value is `""` because dummyFillNeighbor is set to `t` by default.

GUI Equivalent

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

Examples

```
envGetVal("APR.device.grid" "diffLayerNames")
envSetVal("APR.device.grid" "diffLayerNames" 'string "myCell")
```

Related Topics

Generating and Deleting Base Layer Fill

Auto P&R Assistant User Interface for Device-Level Placement

dummyFillCell

```
layoutXL.APAssist dummyFillCell string cellName
```

Description

Specifies the cell of the instance to be considered as the master for creating dummy fill. This environment variable is applicable only when dummyFillNeighbor is set to `nil`.

The default value is `""` because dummyFillNeighbor is set to `t` by default.

GUI Equivalent

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

Examples

```
envGetVal("layoutXL.APAssist" "dummyFillCell")
envSetVal("layoutXL.APAssist" "dummyFillCell" 'string "myCell")
```

Related Topics

Generating and Deleting Base Layer Fill

Auto P&R Assistant User Interface for Device-Level Placement

dummyFillEnableFillEmptyRow

```
layoutXL.APAssist dummyFillEnableFillEmptyRow boolean { t | nil }
```

Description

Specifies whether dummy fill are to be inserted in the gaps between instances.

The default value is `t`.

GUI Equivalent

Command	Auto P&R assistant – *Fill* tab
Field	*Dummy Fill – Enable Dummy Fill*

Examples

```
envGetVal("layoutXL.APAssist" "dummyFillEnableFillEmptyRow")
envSetVal("layoutXL.APAssist" "dummyFillEnableFillEmptyRow" 'boolean nil)
```

Related Topics

Generating and Deleting Base Layer Fill

Auto P&R Assistant User Interface for Device-Level Placement

dummyFillLib

```
layoutXL.APAssist dummyFillLib string libName
```

Description

Specifies the library of the instance to be considered as the master for creating dummy fill. This environment variable is applicable only when dummyFillNeighbor is set to `nil`.

The default value is `""` because dummyFillNeighbor is set to `t` by default.

GUI Equivalent

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

Examples

```
envGetVal("layoutXL.APAssist" "dummyFillLib")
envSetVal("layoutXL.APAssist" "dummyFillLib" 'string "myLib")
```

Related Topics

Generating and Deleting Base Layer Fill

Auto P&R Assistant User Interface for Device-Level Placement

dummyFillNeighbor

```
layoutXL.APAssist dummyFillNeighbor boolean { t | nil }
```

Description

Creates dummies that match the neighboring active devices. This option is selected by default.

The default value is `t`.

GUI Equivalent

Command	Auto P&R assistant – *Fill* tab
Field	*Dummy Fill – Neighbor*

Examples

```
envGetVal("layoutXL.APAssist" "dummyFillNeighbor")
envSetVal("layoutXL.APAssist" "dummyFillNeighbor" 'boolean nil)
```

Related Topics

Generating and Deleting Base Layer Fill

Auto P&R Assistant User Interface for Device-Level Placement

dummyFillNeighborMultipleMode

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

Description

Specifies the number of fingers in the dummy fill.

- `singleFinger`: Creates multiple single-fingered dummy instances.
- `exactCopy`: Dummy fill has the same number of fingers as the neighboring device.

The default is `singleFinger`.

GUI Equivalent

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

Examples

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

Related Topics

Generating and Deleting Base Layer Fill

Auto P&R Assistant User Interface for Device-Level Placement

dummyFillNetMatchNeighbor

```
layoutXL.APAssist dummyFillNetMatchNeighbor boolean [ t | nil ]
```

Description

Specifies whether the connectivity of dummy devices must match the connectivity of the bulk net of their neighboring devices.

The default value is `t`. When set to `nil`, you can use dummyFillLib, Auto P&R Assistant User Interface for Device-Level Placement, and dummyFillView to specify the cellview of the instance to be considered as the master for creating dummy fill.

GUI Equivalent

Command	Auto P&R assistant – *Fill* tab
Field	*Dummy Fill – Match Neighbor*

Examples

```
envGetVal("layoutXL.APAssist" "dummyFillNetMatchNeighbor")
envSetVal("layoutXL.APAssist" "dummyFillNetMatchNeighbor" 'boolean nil)
```

Related Topics

Generating and Deleting Base Layer Fill

Auto P&R Assistant User Interface for Device-Level Placement

dummyFillNetName

```
layoutXL.APAssist dummyFillNetName string netName
```

Description

Specifies the net to which the dummy fill must be connected.

The default value is “”, which indicates that the dummies are unconnected.

GUI Equivalent

Command	Auto P&R assistant – *Fill* tab
Field	*Dummy Fill – Net Name*

Examples

```
envGetVal("layoutXL.APAssist" "dummyFillNetName")
envSetVal("layoutXL.APAssist" "dummyFillNetName" 'string "Net_1")
```

Related Topics

Generating and Deleting Base Layer Fill

Auto P&R Assistant User Interface for Device-Level Placement

dummyFillView

```
layoutXL.APAssist dummyFillView string viewName
```

Description

Specifies the view of the instance to be considered as the master for creating dummy fill. This environment variable is applicable only when dummyFillNeighbor is set to `nil`.

The default value is `""` because dummyFillNeighbor is set to `t` by default.

GUI Equivalent

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

Examples

```
envGetVal("layoutXL.APAssist" "dummyFillView")
envSetVal("layoutXL.APAssist" "dummyFillView" 'string "myView")
```

Related Topics

Generating and Deleting Base Layer Fill

Auto P&R Assistant User Interface for Device-Level Placement

echoViaHalos

```
APR.device.infra echoViaHalos boolean [ t | nil ]
```

Description

Prints each via halo generated for each unique via variant. If numerous vias are created using three unique via variants, the halos for the three unique via variants is printed only once.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "echoViaHalos")
envSetVal("APR.device.infra" "echoViaHalos" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

echoViaVariants

```
APR.device.infra echoViaVariants boolean [ t | nil ]
```

Description

Prints each unique via variant used during routing.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "echoViaVariants")
envSetVal("APR.device.infra" "echoViaVariants" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

enableAlternateVias

```
APR.device.infra enableAlternateVias boolean [ t | nil ]
```

Description

Lets you generate additional via solutions in certain cases where the desired via was not generated.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "enableAlternateVias")
envSetVal("APR.device.infra" "enableAlternateVias" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

enableDRCCompliantGridGen

```
APR.device.grid enableDRCCompliantGridGen boolean [ t | nil ]
```

Description

Generates DRC-clean tracks when no wire type pattern is specified for grid generation.

The default value is `t`. When set to `nil`, the feature is turned off.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "enableDRCCompliantGridGen")
envSetVal("APR.device.grid" "enableDRCCompliantGridGen" 'boolean nil)
```

Related Topics

Generating and Deleting Base Layer Fill

Auto P&R Assistant User Interface for Device-Level Placement

enableDummyFill

```
layoutXL.APAssist enableDummyFill boolean [ t | nil ]
```

Description

Specifies whether dummy fill to be inserted in the gaps between instances.

The default value is `nil`.

GUI Equivalent

Command	Auto P&R assistant – *Fill* tab
Field	*Dummy Fill – Enable Dummy Fill*

Examples

```
envGetVal("layoutXL.APAssist" "enableDummyFill")
envSetVal("layoutXL.APAssist" "enableDummyFill" 'boolean t)
```

Related Topics

Generating and Deleting Base Layer Fill

Auto P&R Assistant User Interface for Device-Level Placement

enableGFGRowRegionSnapping

```
APR.device.grid enableGFGRowRegionSnapping boolean { t | nil }
```

Description

Snaps row regions to the Global Fin Grid.

The default value is `nil`, in which case row regions are aligned to the PR boundary.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "enableGFGRowRegionSnapping")
envSetVal("APR.device.grid" "enableGFGRowRegionSnapping" boolean nil)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

enablePlaceWithWsp

```
APR.device.place enablePlaceWithWsp boolean { t | nil }
```

Description

Displays the *Create Diffusion Grid* option on the *Setup* tab of the Auto P&R assistant. Set this environment variable if there are multi-height cells in your design. This option lets you create a diffusion grid based on the heights of the diffusion layers of the multi-height cells.

The default value is `nil`, in which case the *Create Diffusion Grid* option does not appear on the *Setup* tab of the Auto P&R assistant.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Create Diffusion Grid*

Examples

```
envGetVal("APR.device.place" "enablePlaceWithWsp")
envSetVal("APR.device.place" "enablePlaceWithWsp" 'boolean t)
```

Related Topics

Placing Multi-Height Devices Using Automatic Device Placer

Auto P&R Assistant User Interface for Device-Level Placement

setup placementRegion

enablePolyViaHalos

```
APR.device.infra enablePolyViaHalos boolean [ t | nil ]
```

Description

Adds via halos for poly and local interconnect layers.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "enablePolyViaHalos")
envSetVal("APR.device.infra" "enablePolyViaHalos" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

extendPaths

```
APR.device.route extendPaths boolean { t | nil }
```

Description

Enables paths on higher layers to extend further than paths on lower layers. This optimization is used to connect to multiple wires in a mesh and to collect pins that are already logically connected.

The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "extendPaths")
envSetVal("APR.device.route" "extendPaths" 'boolean nil)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

extractDepth

`APR.device.route extractDepth int` *`extractionDepth`*

Description

Controls the pre- and post-route extraction depth.

The default value is `32`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "extractDepth")
envSetVal("APR.device.route" "extractDepth" 'int 5)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

extractStopLevel

`APR.device.route extractStopLevel int` *`stopLevel`*

Description

Controls the pre- and post-route extraction stop-level.

The default value is `32`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "extractStopLevel")
envSetVal("APR.device.route" "extractStopLevel" 'int 7)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

fillRegionHonorTransitionSpacing

```
APR.device.fill fillRegionHonorTransitionSpacing cyclic { "XY" | "X" | "Y" | "None"
     | "Unset" }
```

Description

Specifies the transition spacing mode to be used when inserting device fill. You can use this environment variable to specify the direction along which the transition spacing value, specified on the *Place* tab of the Auto P&R assistant, is to be applied when inserting device fill.

The available transition spacing modes are:

- `XY`: Transition spacing is honored in both X and Y directions.
- `X`: Transition spacing is honored in only in the X-direction.
- `Y`: Transition spacing is honored in only in the Y-direction.
- `None`: Transition spacing is set to 0.
- `Unset`: The tool default value (`XY`) is honored.

The default is `Unset`.

GUI Equivalent

Examples

```
envGetVal("APR.device.fill" "fillRegionHonorTransitionSpacing")
envSetVal("APR.device.fill" "fillRegionHonorTransitionSpacing" 'cyclic "X")
```

Related Topics

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Generating and Deleting Base Layer Fill

Auto P&R Assistant User Interface for Device-Level Placement

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 Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

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 Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

flipFirstRow

```
APR.device.grid flipFirstRow boolean { t | nil }
```

Description

Specifies that the bottom-most row must start with the MX orientation, instead of R0 orientation, when there are flipped rows in a design.

The default value is `nil`, where the bottom-most row starts with R0 orientation.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Generate Flipped Rows*

Examples

```
envGetVal("APR.device.grid" "flipFirstRow")
envSetVal("APR.device.grid" "flipFirstRow" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Using the WSP Manager

flippedCellTypes

```
APR.device.grid flippedCellTypes string "cdfParam, value"
```

Description

Specifies a list of cells to be flipped during design initialization. Cell type can specified using a CDF parameter and its value.

The default is `""`, which implies that there are no user-specific cells to be flipped.

This environment variable works only in certain advanced node flows.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "flippedCellTypes")
envSetVal("APR.device.grid" "flippedCellTypes" 'string "Param1, HalfCell")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Verifying Placement

forceAbutM0Regen

```
APR.device.fill forceAbutM0Regen boolean { t | nil }
```

Description

Forces the Modgen to be regenerated on every abutment during dummy fill.

The default value is `nil`, where the Modgen is not regenerated. It is not recommended to set this environment variable to `t` for normal tool runs.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.fill" "forceAbutM0Regen")
envSetVal("APR.device.fill" "forceAbutM0Regen" 'boolean t)
```

Related Topics

Generating and Deleting Base Layer Fill

ftiToActiveMinFingers

```
APR.device.fill ftiToActiveMinFingers int minFingers
```

Description

Specifies the minimum gap between active instances and the FTI in terms of the minimum number of fingers of the dummy fills.

The default value is `3`, which indicates that the gap must be at least 3 fingers wide.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.fill" "ftiToActiveMinFingers")
envSetVal("APR.device.fill" "ftiToActiveMinFingers" 'int 7)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Auto P&R Assistant User Interface for Device-Level Placement

activeToPRBoundaryMinFingers

activeToActiveMinFingers

ftiToPRBoundaryMinFingers

ftiToFtiMinFingers

`APR.device.fill ftiToFtiMinFingers int` *`minFingers`*

Description

Specifies the minimum gap between FTIs in terms of the minimum number of fingers of the dummy fills.

The default value is `1`, which indicates that the gap must be at least 1 finger wide.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.fill" "ftiToFtiMinFingers")
envSetVal("APR.device.fill" "ftiToFtiMinFingers" 'int 3)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Auto P&R Assistant User Interface for Device-Level Placement

activeToPRBoundaryMinFingers

activeToActiveMinFingers

ftiToActiveMinFingers

ftiToPRBoundaryMinFingers

ftiToPRBoundaryMinFingers

`APR.device.fill ftiToPRBoundaryMinFingers int` *`stopLevel`*

Description

Specifies the minimum gap between PR boundary and the FTI in terms of the minimum number of fingers of the dummy fills.

The default value is `4`, which indicates that the gap must be at least 4 fingers wide.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.fill" "ftiToPRBoundaryMinFingers")
envSetVal("APR.device.fill" "ftiToPRBoundaryMinFingers" 'int 7)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Auto P&R Assistant User Interface for Device-Level Placement

activeToPRBoundaryMinFingers

activeToActiveMinFingers

ftiToActiveMinFingers

ftiToFtiMinFingers

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 P&R assistant – *Setup* tab
Field	*Gate Track Spacing*

Examples

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

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

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 P&R assistant – *Setup* tab
Field	*Gate Track Width*

Examples

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

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

genFlippedRows

```
APR.device.grid genFlippedRows boolean { t | nil }
```

Description

Specifies whether flipped patterns and rows are generated.

The default value is `nil`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Generate Flipped Rows*

Examples

```
envGetVal("APR.device.grid" "genFlippedRows")
envSetVal("APR.device.grid" "genFlippedRows" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

globalRowHeightSnappingMode

```
APR.device.grid globalRowHeightSnappingMode cyclic { "none" |
     "localFinOfLargestDevice" | "globalFin" }
```

Description

Specifies the mode in which the global row height is to be adjusted.

The default is `"globalFin"`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "globalRowHeightSnappingMode")
envSetVal("APR.device.grid" "globalRowHeightSnappingMode" 'cyclic "globalFin")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

gridOffset

```
APR.device.grid specifyGridOffset float gridOffset
```

Description

Specifies the the offset of poly grids. This environment variable is valid only when specifyGridOffset is set to `t`.

The default value is `0.00`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Specify Grid Offset*

Examples

```
envGetVal("APR.device.grid" "gridOffset")
envSetVal("APR.device.grid" "gridOffset" float 1.00)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Auto P&R Assistant User Interface for Device-Level Placement

specifyGridOffset

groupPattern

```
APR.device.constraint groupPattern cyclic { "Uniform" | "Interdigitate"| "Compact"
     }
```

Description

Specifies the patterns to be set for Modgens after creating the constraints. The available options are `Uniform`, `Interdigitate`, and `Compact`.

The default value is `Compact`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.constraint" "groupPattern")
envSetVal("APR.device.constraint" "groupPattern" 'cyclic "Interdigitate")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

ignoreColorCheck

```
APR.device.fill ignoreColorCheck boolean { t | nil }
```

Description

Specifies whether color checks are to be performed while inserting device fill.

The default value is `nil`, in which case color checks are not run.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.fill" "ignoreColorCheck")
envSetVal("APR.device.fill" "ignoreColorCheck" 'boolean t)
```

Related Topics

Base Layer Fill in the Automated Device Placement and Routing Flow

ignoreInstSelfSymmetry

```
APR.device.constraint ignoreInstSelfSymmetry boolean { t | nil }
```

Description

Restricts propagation of self-symmetry of multiple grouped instances into a single symmetry constraint.

The default value is `nil`.

GUI Equivalent

Command	Auto P&R assistant – *Constraints* tab
Field	*Ignore Instance Self-symmetry*

Examples

```
envGetVal("APR.device.constraint" "ignoreInstSelfSymmetry")
envSetVal("APR.device.constraint" "ignoreInstSelfSymmetry" 'boolean t)
```

Related Topics

Constraints in the Automated Device Placement and Routing Flow

Generating Constraints and Constraint Groups

APR Circuit Finders

Auto P&R Assistant User Interface for Device-Level Placement

ignoreOverlapCheckProp

```
APR.device.place ignoreOverlapCheckProp string "property"
```

Description

Sets a property that can be used to ignore overlaps between top-level instances or figGroups when verifying the placement.

The default is `""`, which implies that no property is defined. Irrespective of this setting, if the instances or figGroups have the Boolean property `verifyPlaceIgnoreOverlap` set, overlap checks are ignored.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "ignoreOverlapCheckProp")
envSetVal("APR.device.place" "ignoreOverlapCheckProp" 'string
"IgnoreAllOverlaps")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Verifying Placement

includeCellNameInRRName

```
APR.device.grid includeCellNameInRRName boolean { t | nil }
```

Description

Specifies whether the name of the row region, generated using the *Setup* tab of the Auto P&R assistant, must include the cell name.

The default is `nil`, in which case the row region name does not include the cell name.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "includeCellNameInRRName")
envSetVal("APR.device.grid" "includeCellNameInRRName" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

includePassiveComponentPoly

```
APR.device.grid includePassiveComponentPoly boolean { t | nil }
```

Description

Recognizes the poly layer patterns of passive devices while generating WSPs.

The default value is `nil`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Include Passive Component Poly*

Examples

```
envGetVal("APR.device.grid" "includePassiveComponentPoly")
envSetVal("APR.device.grid" "includePassiveComponentPoly" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

importFromCell

```
APR.device.grid importFromCell string "cell_name"
```

Description

Specifies the name of the cell that contains the row template based on which a row region is to be created.

The default is `""`, where no row template is imported.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Use row template*

Examples

```
envGetVal("APR.device.grid" "importFromCell")
envSetVal("APR.device.grid" "importFromCell" 'string "MyCell")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

importFromLib

```
APR.device.grid importFromLib string "lib_name"
```

Description

Specifies the name of the library that contains the row template based on which a row region is to be created.

The default is `""`, where no row template is imported.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Use row template*

Examples

```
envGetVal("APR.device.grid" "importFromLib")
envSetVal("APR.device.grid" "importFromLib" 'string "MyLib")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

importFromView

```
APR.device.grid importFromView string "lib_name"
```

Description

Specifies the name of the view that contains the row template based on which a row region is to be created.

The default is `""`, where no row template is imported.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Use row template*

Examples

```
envGetVal("APR.device.grid" "importFromView")
envSetVal("APR.device.grid" "importFromView" 'string "MyLib")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

init_boundaryAspectRatioOrHeight

```
APR.device.place init_boundaryAspectRatioOrHeight cyclic { "Aspect Ratio" |
     "Height" }
```

Description

Specifies the method in which the PR boundary is to be defined. The valid values are:

- `Aspect Ratio`: Uses the init_boundaryAspectRatioVal setting to specify the width-to-height ratio of the PR boundary.
- `Height`: Uses the init_boundaryHeightVal setting to specify the exact height of the PR boundary.

The default value is `Aspect Ratio`.

GUI Equivalent

Command	Auto P&R assistant – *Initialize* tab
Field	*Boundary – Aspect Ratio, Height*

Examples

```
envGetVal("APR.device.place" "init_boundaryAspectRatioOrHeight")
envSetVal("APR.device.place" "init_boundaryAspectRatioOrHeight" 'cyclic "Height")
```

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

Initializing a Layout in the Automated Device Placement and Routing Flow

init_boundaryAspectRatioVal

```
APR.device.place init_boundaryAspectRatioVal float utilizationVal
```

Description

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.

This setting is valid only when init_boundaryAspectRatioOrHeight is set to `Aspect Ratio`.

The default value is `1.0`.

GUI Equivalent

Command	Auto P&R assistant – *Initialize* tab
Field	*Boundary – Aspect Ratio*

Examples

```
envGetVal("APR.device.place" "init_boundaryAspectRatioVal")
envSetVal("APR.device.place" "init_boundaryAspectRatioVal" 'float 2.0)
```

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

Initializing a Layout in the Automated Device Placement and Routing Flow

init_boundaryHeightVal

```
APR.device.place init_boundaryHeightVal float boundaryWidth
```

Description

Specifies the exact height of the PR boundary. This setting is valid only when init_boundaryAspectRatioOrHeight is set to `Height`.

The default value is `10.00`.

GUI Equivalent

Command	Auto P&R assistant – *Initialize* tab
Field	*Boundary – Height*

Examples

```
envGetVal("APR.device.place" "init_boundaryHeightVal")
envSetVal("APR.device.place" "init_boundaryHeightVal" 'float 15.00)
```

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

Initializing a Layout in the Automated Device Placement and Routing Flow

init_boundaryUtilizationOrWidth

```
APR.device.place init_boundaryUtilizationOrWidth cyclic { "Utilization" | "Width"
     }
```

Description

Specifies the method in which the fillable area within the cell boundary is to be defined. The valid values are:

- `Utilization`: Uses the init_boundaryUtilizationVal setting to specify the percentage of area within the cell boundary that can be filled with objects.
- `Width`: Uses the init_boundaryWidthVal setting to specify the exact width of the PR boundary.

The default value is `Utilization`.

GUI Equivalent

Command	Auto P&R assistant – *Initialize* tab
Field	*Boundary – Utilization, Width*

Examples

```
envGetVal("APR.device.place" "init_boundaryUtilizationOrWidth")
envSetVal("APR.device.place" "init_boundaryUtilizationOrWidth" 'cyclic "Width")
```

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

Initializing a Layout in the Automated Device Placement and Routing Flow

init_boundaryUtilizationVal

`APR.device.place init_boundaryUtilizationVal float` *`utilizationVal`*

Description

Specifies the percentage of area within the cell boundary that can be filled with objects. This setting is valid only when init_boundaryUtilizationOrWidth is set to `Utilization`.

The default value is `25`.

GUI Equivalent

Command	Auto P&R assistant – *Initialize* tab
Field	*Boundary – Utilization*

Examples

```
envGetVal("APR.device.place" "init_boundaryUtilizationVal")
envSetVal("APR.device.place" "init_boundaryUtilizationVal" 'float 35)
```

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

Initializing a Layout in the Automated Device Placement and Routing Flow

init_boundaryWidthVal

```
APR.device.place init_boundaryWidthVal float boundaryWidth
```

Description

Specifies the exact width of the PR boundary. This setting is valid only when init_boundaryUtilizationOrWidth is set to `Width`.

The default value is `10.00`.

GUI Equivalent

Command	Auto P&R assistant – *Initialize* tab
Field	*Boundary – Width*

Examples

```
envGetVal("APR.device.place" "init_boundaryWidthVal")
envSetVal("APR.device.place" "init_boundaryWidthVal" 'float 15)
```

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

Initializing a Layout in the Automated Device Placement and Routing Flow

init_createPowerPins

```
APR.device.place init_createPowerPins boolean { t | nil }
```

Description

Specifies whether power pins are to be created.

The default value is `t`.

GUI Equivalent

Command	Auto P&R assistant – *Initialize* tab
Field	*Pins*

Examples

```
envGetVal("APR.device.place" "init_createPowerPins")
envSetVal("APR.device.place" "init_createPowerPins" 'boolean nil)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

init_generateBoundary

```
APR.device.place init_generateBoundary boolean { t | nil }
```

Description

Specifies whether a PR boundary is to be generated.

The default value is `t`.

GUI Equivalent

Command	Auto P&R assistant – *Initialize* tab
Field	*Generate – Boundary*

Examples

```
envGetVal("APR.device.place" "init_generateBoundary")
envSetVal("APR.device.place" "init_generateBoundary" 'boolean nil)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

init_generateInstances

```
APR.device.place init_generateInstances boolean { t | nil }
```

Description

Specifies whether instances are to be generated.

The default value is `t`.

GUI Equivalent

Command	Auto P&R assistant – *Initialize* tab
Field	*Generate – Instances*

Examples

```
envGetVal("APR.device.place" "init_generateInstances")
envSetVal("APR.device.place" "init_generateInstances" 'boolean nil)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

init_generatePins

```
APR.device.place init_generatePins boolean { t | nil }
```

Description

Specifies whether pins are to be generated.

The default value is `t`.

GUI Equivalent

Command	Auto P&R assistant – *Initialize* tab
Field	*Generate – Pins*

Examples

```
envGetVal("APR.device.place" "init_generatePins")
envSetVal("APR.device.place" "init_generatePins" 'boolean nil)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

init_mode

```
APR.device.place init_mode cyclic { "Generate" | "Update" }
```

Description

Specifies a device generation mode. The available options are:

- `Generate`: Generates new devices in the layout design.
- `Update`: Updates existing objects in the layout design.

The default value is `Generate`.

GUI Equivalent

Command	Auto P&R assistant – *Initialize* tab
Field	*Generation Options – Generate, Update*

Examples

```
envGetVal("APR.device.place" "init_mode")
envSetVal("APR.device.place" "init_mode" 'cyclic "Update")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

init_scope

```
APR.device.place init_scope cyclic { "All" | "Selected" }
```

Description

Specifies the scope of layout generation.

- `All`: Generates all objects present in the source.
- `Selected`: Generates only the selected objects.

The default value is `All`.

GUI Equivalent

Command	Auto P&R assistant – *Initialize* tab
Field	*Generation Options – All, Selected*

Examples

```
envGetVal("APR.device.place" "init_scope")
envSetVal("APR.device.place" "init_scope" 'cyclic "Selected")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

init_useSourceLayout

```
APR.device.place init_useSourceLayout boolean { t | nil }
```

Description

Specifies whether the settings from the source layout are to be reused in the target.

The default value is `nil`. When set to `t`, use the following environment settings to specify the layout reuse settings:

- init_useSourceLayoutBoundary
- init_useSourceLayoutConstraints
- init_useSourceLayoutInstances
- init_useSourceLayoutPins

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "init_useSourceLayout")
envSetVal("APR.device.place" "init_useSourceLayout" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

init_useSourceLayoutBoundary

```
APR.device.place init_useSourceLayoutBoundary boolean { t | nil }
```

Description

Adjusts the PR boundary in the current cellview as per its setting in the reference cellview.

The default value is `t`. Ensure that init_useSourceLayout is also set to `t`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "init_useSourceLayoutBoundary")
envSetVal("APR.device.place" "init_useSourceLayoutBoundary" 'boolean nil)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

init_useSourceLayoutConstraints

```
APR.device.place init_useSourceLayoutConstraints boolean { t | nil }
```

Description

Uses constraints and process overrides from the source cellview.

The default value is `nil`. When set to `t`, ensure that init_useSourceLayout is also set to `t`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "init_useSourceLayoutConstraints")
envSetVal("APR.device.place" "init_useSourceLayoutConstraints" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

init_useSourceLayoutInstances

```
APR.device.place init_useSourceLayoutInstancePositions boolean { t | nil }
```

Description

Updates the positions of standard cells, custom cells, and macros in the current cellview as per the reference cellview.

The default value is `t`. Ensure that init_useSourceLayout is also set to `t`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "init_useSourceLayoutInstancePositions")
envSetVal("APR.device.place" "init_useSourceLayoutInstancePositions" 'boolean
nil)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

init_useSourceLayoutPins

```
APR.device.place init_useSourceLayoutPins boolean { t | nil }
```

Description

Places pins in the current cellview as per as per their positions in the reference cellview.

The default value is `t`. Ensure that init_useSourceLayout is also set to `t`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "init_useSourceLayoutPins")
envSetVal("APR.device.place" "init_useSourceLayoutPins" 'boolean nil)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

init_useCustomSettings

```
APR.device.place init_useCustomSettings boolean { t | nil }
```

Description

Places pins in the current cellview as per as per their positions in the reference cellview.

The default value is `nil`. When set to `t`, ensure that init_useSourceLayout is also set to t.

GUI Equivalent

Command	Auto P&R Assistant – *Initialize* tab
Field	*PDK Settings Use – Custom Settings*

Examples

```
envGetVal("APR.device.place" "init_useCustomSettings")
envSetVal("APR.device.place" "init_useCustomSettings" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

isOnlyAnalogCellFlow

```
APR.device.initialize isOnlyAnalogCellFlow cyclic { "Unset" | "True" | "False" }
```

Description

Specifies whether the current PDK supports the analog cell flow or the primitive flow.

- `Unset`: Honors the tool default value that is set in the initialize step.
- `True`: The current PDK supports only analog cell flow.
- `False`: The current PDK supports only primitive flow.

The default value is `Unset`.

This environment variable is applicable only to nodes that support both analog and primitive flows.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.initialize" "isOnlyAnalogCellFlow")
envSetVal("APR.device.initialize" "isOnlyAnalogCellFlow" 'cyclic "False")
```

Related Topics

Initializing a Layout in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

matureOrAdvancedNode

```
APR.device.place matureOrAdvancedNode cyclic { "auto" | "mature" | "advanced" }
```

Description

Specifies the design process node for running the placer. The placement options displayed on the *Constraints* tab of the Auto P&R assistant vary depending on the selected design process node.

The available design process nodes are `mature` and `advanced`. When set to `auto`, the tool automatically identifies an appropriate node based on the constraint definitions in the design.

The default value is `advanced`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "matureOrAdvancedNode")
envSetVal("APR.device.place" "matureOrAdvancedNode" 'cyclic "auto")
```

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

mergeWeakPins

```
APR.device.infra mergeWeakPins boolean { t | nil }
```

Description

Enables gate pins to be connected over diffusion layer.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "mergeWeakPins")
envSetVal("APR.device.infra" "mergeWeakPins" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

meshViaStackLimit

```
APR.device.route meshViaStackLimit int viaStackLimit
```

Description

Controls the depth for stack via creation.

The default value is `-1`. This means no value is specified.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "meshViaStackLimit")
envSetVal("APR.device.route" "meshViaStackLimit" 'int 0)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

metal_Spacing

`APR.device.grid metal_Spacing float` *`rowSpacing`*

Description

Specifies the spacing between tracks. The default value is `0.00`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "metal_Spacing")
envSetVal("APR.device.grid" "metal_Spacing" 'float 1.00)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

metal_Width

`APR.device.grid metal_Width float` *`rowWidth`*

Description

Specifies the track width. The default value is `0.00`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "metal_Width")
envSetVal("APR.device.grid" "metal_Width" 'float 2.00)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

metal_WSPMode

```
APR.device.grid metal_WSPMode cyclic { "deriveMetal_WSP" | "useExistingMetal_WSP"
     }
```

Description

Specifies the mode for creating WSPs on metal layers. The available options are:

- `deriveMetal_WSP`: Creates new WSPs on metal layers.
- `useExistingMetal_WSP`: Uses existing WSPs on metal layers.

The default value is `deriveMetal_WSP`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "metal_WSPMode")
envSetVal("APR.device.grid" "metal_WSPMode" 'cyclic "useExistingMetal_WSP")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

migrationDir

```
migrate.layout migrationDir string "directory_name"
```

Description

This environment variable works only in Layout MXL, as part of the assisted custom layout design migration flow. It specifies the initial path to the working directory in which the captured source data must be stored. Changes to the UI do not change environment variable. The target layout references this location for applying the captured source data.

If `migrationDir` is set to the current working directory, the value is automatically updated to `"/migrate_reuse_<libName>"` when initializing the UI. This is because using the current working directory is not recommended.

The default value is ".", which refers to `"/migrate_reuse_<libName>"`.

GUI Equivalent

Command	Auto P&R Assistant – *Initialize* and *Constraint* tabs
Field	*Migration Directory*

Examples

```
envGetVal("layoutXL" "migrationDir")
envSetVal("layoutXL" "migrationDir" 'string "(/home/myFiles/reuse_data)")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Auto P&R Assistant User Interface for Device-Level Placement

Automatic Generation of Modgens using the Array Assistant

modgenDummyFillType

```
APR.device.fill modgenDummyFillType cyclic { "Default" | "AnalogCell" |
     "StackCell"}
```

Description

Specifies the type of cells to be used as dummies. The available options are:

- *Default*: Uses the default dummy defined in the Modgen Dummy Options form.
- *Analog Dummy*: Inserts dummies of type analog cell.
- *Stacked Dummy*: Insert dummies of type stack cell.

This environment variable is available only in certain advanced node flows.

The default value is `Default`.

GUI Equivalent

Command	Array Assistant – *Placement* tab
Field	*Dummy Type*

Examples

```
envGetVal("APR.device.fill" "modgenDummyFillType")
envSetVal("APR.device.fill" "modgenDummyFillType" 'cyclic "AnalogCell")
```

Related Topics

Automatic Generation of Modgens using the Array Assistant

modgenDummyTypeAnalogLCV

```
layoutXL modgenDummyTypeAnalogLCV string "analog_LCV"
```

Description

Specifies the library, cell, and view for Modgen dummies of type analog cell to be inserted when the *Fill Modgen Dummies* button is clicked in the Array Assistant.

This environment variable is available only in certain advanced node flows.

The default value is "", with no library, cell, or view specified.

GUI Equivalent

Command	Array Assistant – *Placement* tab
Field	*Dummy Type – Analog Dummy*

Examples

```
envGetVal("layoutXL" "modgenDummyTypeAnalogLCV")
envSetVal("layoutXL" "modgenDummyTypeAnalogLCV" 'string "(lib1 cell1 layout)")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Auto P&R Assistant User Interface for Device-Level Placement

Automatic Generation of Modgens using the Array Assistant

modgenDummyTypeStackLCV

```
layoutXL modgenDummyTypeStackLCV string "stack_LCV"
```

Description

Specifies the library, cell, and view for Modgen dummies of type stack to be inserted when the *Fill Modgen Dummies* button is clicked in the Array Assistant.

This environment variable is available only in certain advanced node flows.

The default value is "", with no library, cell, or view specified.

GUI Equivalent

Command	Array Assistant – *Placement* tab
Field	*Dummy Type – Stack Dummy*

Examples

```
envGetVal("layoutXL" "modgenDummyTypeStackLCV")
envSetVal("layoutXL" "modgenDummyTypeStackLCV" 'string "(lib1 cell1 layout)")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Auto P&R Assistant User Interface for Device-Level Placement

Automatic Generation of Modgens using the Array Assistant

modgenDummyTypeStackNum

```
layoutXL modgenDummyTypeStackLCV string "stackNum"
```

Description

Specifies the number of stacks to be included in dummies of type stack gate analog cell when the *Fill Modgen Dummies* button is clicked in the Array Assistant.

This environment variable is available only in certain advanced node flows.

The default value is `decompCnt`.

GUI Equivalent

Command	Array Assistant – *Placement* tab
Field	*Dummy Type – Stack Dummy*

Examples

```
envGetVal("layoutXL" "modgenDummyTypeStackLCV")
envSetVal("layoutXL" "modgenDummyTypeStackLCV" 'string "decompCnt1")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Auto P&R Assistant User Interface for Device-Level Placement

Automatic Generation of Modgens using the Array Assistant

modgenDummyStacksMinSize

`APR.device.fill modgenDummyStacksMinSize int` *`integer`*

Description

Specifies the minimum number of stacks to be included in dummies of type stack gate when the *Fill Modgen Dummies* button is clicked in the Array Assistant.

This environment variable is available only in certain advanced node flows.

The default value is `2`.

GUI Equivalent

Command	Array Assistant – *Placement* tab
Field	*Dummy Type – Stack Dummy*

Examples

```
envGetVal("APR.device.fill" "modgenDummyStacksMinSize")
envSetVal("APR.device.fill" "modgenDummyStacksMinSize" 'int 3)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

multiConn

```
APR.device.route multiConn string "connName"
```

Description

Lets you specify a name for multiple connections to a single pin.

The default value is an empty string "". This means the connection name is not specified.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "multiConn")
envSetVal("APR.device.route" "multiConn" 'string "connection1")
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

multiCutsOnPin

```
APR.device.route multiCutsOnPin boolean { t | nil }
```

Description

Replaces a via with a multi-cut via on pins with long overlapping regions.

The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "multiCutsOnPin")
envSetVal("APR.device.route" "multiCutsOnPin" 'boolean t)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

multiFingerTransFill

```
APR.device.fill multiFingerTransFill boolean { t | nil }
```

Description

Specifies whether multi-fingered stack gate analog cells are to be used as transition fill in the *Fill* step of the automated device placement flow.

The default is `nil`. In this state, single-fingered analog cells are used.

When set to t, use transitionFillFingerCount to specify the number of fingers.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "multiFingerTransFill")
envSetVal("APR.device.grid" "multiFingerTransFill" 'boolean t)
```

Related Topics

transitionFillFingerCount

Generating and Deleting Base Layer Fill

Auto P&R Assistant User Interface for Device-Level Placement

multiPolyWSP

```
APR.device.grid 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("APR.device.grid" "multiPolyWSP")
envSetVal("APR.device.grid" "multiPolyWSP" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

nullViaVariantAnError

```
APR.device.infra nullViaVariantAnError boolean { t | nil }
```

Description

Considers a NULL via variant in the via service as an error (non-fatal) when asking for its halo. Via variants should only be NULL when a via cannot be found for a given layer and width pairs,

The default value is `t`. When set to `nil`, it prevents artificial blocking on one or both of the layers.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "nullViaVariantAnError")
envSetVal("APR.device.infra" "nullViaVariantAnError" boolean nil)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

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 P&R assistant – *Setup* tab
Field	*#S/D Tracks*

Examples

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

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

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 P&R assistant – *Setup* tab
Field	*#Gate Tracks*

Examples

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

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

optimization

```
layoutXL.APAssist optimization cyclic { "Optimize Area and Wire Length" | "Optimize
     Area Only" | "Optimize Wire Length Only" }
```

Description

Specifies how the design is to be optimized during placement. The available options are:

- `Optimize Area and Wire Length`: Achieves a balance between compact placement and reduced wire length.
- `Optimize Area Only`: Place devices to optimize compactness.
- `Optimize Wire Length Only`: Minimizes the wire length for better routability.

The default is `Optimize Area and Wire Length`.

GUI Equivalent

Command	Auto P&R assistant – *Place tab*
Field	*Optimize*

Examples

```
envGetVal("layoutXL.APAssist" "optimization")
envSetVal("layoutXL.APAssist" "optimization" 'cyclic "Optimize Area Only")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

patternName

```
APR.device.grid patternName string pattern
```

Description

Specifies the pattern to define distribution of P and N devices in rows.

The default value is `""`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Define P/N Pattern*

Examples

```
envGetVal("APR.device.grid" "patternName")
envSetVal("APR.device.grid" "patternName" string "pattern1")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

pdkMapping

`APR.device.initialize pdkMapping string` *`pattern`*

Description

Specifies the mapping between PDKs to be honored in the Virtuoso automated device placement and routing flow. PDK settings, for example the WSP, fill, router, auto placer, and assisted placer settings, are loaded from the mapped PDK.

The default value is `""`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.initialize" "pdkMapping")
envSetVal("APR.device.initialize" "pdkMapping" 'string "pdkNameFrom pdkNameTo")
envSetVal("APR.device.initialize" "pdkMapping" 'string "pdkNameFrom1 pdkNameTo1
pdkNameFrom2 pdkNameTo2")
```

Related Topics

Initializing a Layout in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

pinPurposes

`APR.device.infra pinPurposes string "`*`pinPurposeName`*`"`

Description

Provides a list of recognized pin purposes for routing.

The default value is `pin drawing dummy net`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "pinPurposes")
envSetVal("APR.device.infra" "pinPurposes" 'string "purposeName")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

polyFillRegion

```
layoutXL.APAssist polyFillRegion cyclic { "CellView" | "RowRegion" | "CustomArea" }
```

Description

Specifies the region in which poly fill are to be inserted. The available options are:

- `CellView`: Selects the current cellview to insert poly fill.
- `RowRegion`: Selects the current region for inserting poly fill.
- `CustomArea`: Lets you specify a custom area for inserting poly fill.

The default is `"RowRegion"`.

GUI Equivalent

Command	Auto P&R assistant – *Fill* tab
Field	*Poly Fill – Select As*

Examples

```
envGetVal("layoutXL.APAssist" "polyFillRegion")
envSetVal("layoutXL.APAssist" "polyFillRegion" 'cyclic "CellView")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

polyPurposes

`APR.device.grid polyPurposes string` *`pattern`*

Description

Specifies a list of purposes to consider for the poly pitch calculation when creating poly layer patterns as WSPs.

The default value is `"drawing pin"`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "polyPurposes")
envSetVal("APR.device.grid" "polyPurposes" string "boundary pin")
```

Related Topics

Initializing a Layout in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

postRouteTrigger

`APR.device.route postRouteTrigger string "`*`procedurename`*`"`

Description

Specifies the SKILL procedure to be executed after routing has completed.

The default is `""`, which means no additional processing is done.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "postRouteTrigger")
envSetVal("APR.device.route" "postRouteTrigger" 'string "postRouteProcedure")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

preferredLayerStrength

```
APR.device.route preferredLayerStrength int layerStrength
```

Description

Controls the extent to which the router favors preferred layers compared to layers outside the preferred range. This environment variable can be adjusted to fine tune routing results.

- 0 means preferred layers are not treated specially at all.
- 1-5 adjusts the preference strength, with 5 being most preferred.
- 6 imposes a hard constraint on the maximum length for pathSegs on non-preferred layers and leaves opens if no solution that honors this constraint can be found. At this strength, each new pathSeg on a non-preferred layer can be at most the length of the layer's pitch.

The default value is `3`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "preferredLayerStrength")
envSetVal("APR.device.route" "preferredLayerStrength" 'int 2)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

preferSmallestCutClass

```
APR.device.infra preferSmallestCutClass boolean { t | nil }
```

Description

Considers the via with the smallest legal cut class for a given wire intersection.

The default value is `t`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "preferSmallestCutClass")
envSetVal("APR.device.infra" "preferSmallestCutClass" boolean nil)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

preferStandardVias

```
APR.device.infra preferStandardVias boolean { t | nil }
```

Description

Uses standard vias over custom vias, whenever possible.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "preferStandardVias")
envSetVal("APR.device.infra" "preferStandardVias" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

preRouteTrigger

```
APR.device.route preRouteTrigger string "procedurename"
```

Description

Specifies the SKILL procedure to be executed before routing.

The default is `""`, which means no additional processing is done.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "preRouteTrigger")
envSetVal("APR.device.route" "preRouteTrigger" 'string "preRouteProcedure")
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

prouteGroupByLayer

```
APR.device.route prouteGroupByLayer boolean { t | nil }
```

Description

Creates a child figGroup to group path and via by layer for the supply routing result. This is besides the original single parent figGroup.

The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "prouteGroupByLayer")
envSetVal("APR.device.route" "prouteGroupByLayer" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

prouteUsePinPurposeForCreatedPin

```
APR.device.route prouteUsePinPurposeForCreatedPin boolean { t | nil }
```

Description

Enables to use pin as the layer purpose for the created pin.

The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "prouteUsePinPurposeForCreatedPin")
envSetVal("APR.device.route" "prouteUsePinPurposeForCreatedPin" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

prouteWaivedMarkerLayer

```
routing.device.route prouteWaivedMarkerLayer string "makerLayer"
```

Description

Specifies a layer to create the waived marker overlapping path segment with layer-purpose pair as `PG_marker/layer` to ignore DRC checker for certain stacking by constraint rule checking. The default is an empty string.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "prouteWaivedMarkerLayer")
envSetVal("APR.device.route" "prouteWaivedMarkerLayer" 'string "marker1")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

regenModgenPostProcess

```
APR.device.place 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("APR.device.place" "regenModgenPostProcess")
envSetVal("APR.device.place" "regenModgenPostProcess" 'boolean t)
envSetVal("APR.device.place" "regenModgenPostProcess" 'boolean nil)
```

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

results_nets

```
APR.device.route results_nets cyclic { "All" | "Selected" | "Open" }
```

Description

Specifies whether you want all nets, selected nets, or only open nets to be included in the results table.

The default is `"All"`.

GUI Equivalent

Command	Routing assistant – *Results* tab
Field	*Nets – All*, *Selected*, *Open*

Examples

```
envGetVal("APR.device.route" "results_nets")
envSetVal("APR.device.route" "results_nets" 'cyclic "Selected")
envSetVal("APR.device.route" "results_nets" 'cyclic "Open")
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

results_netsWithin

```
APR.device.route results_netsWithin cyclic { "PR boundary" | "Device array/FigGroup
     }
```

Description

Specifies whether to route everything inside the PR boundary or within a device array or figGroup.

The default is `"PR boundary"`.

GUI Equivalent

Command	Routing assistant – *Results* tab
Field	*Within – PR boundary*, *Device array/FigGroup*

Examples

```
envGetVal("APR.device.route" " results_netsWithin")
envSetVal("APR.device.route" " results_netsWithin" 'cyclic "Device array/
FigGroup")
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

results_supplyNets

```
APR.device.route results_supplyNets boolean { t | nil }
```

Description

Includes power and ground nets in the results table. When set to `nil`, the power and ground nets are not included in the results table.

The default is `t`.

GUI Equivalent

Command	Routing assistant – *Results* tab
Field	*Supply Nets*

Examples

```
envGetVal("APR.device.route" "results_supplyNets")
envSetVal("APR.device.route" "results_supplyNets" 'boolean nil)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_connectBothSidesOfGate

```
APR.device.route route_connectBothSidesOfGate boolean { t | nil }
```

Description

Routes both ends of poly gate pins and makes the poly gate pins must connect even if they are weak connect.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Connect on both sides of gate*

Examples

```
envGetVal("APR.device.route" "route_connectBothSidesOfGate")
envSetVal("APR.device.route" "route_connectBothSidesOfGate" 'boolean t)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_connectInstsToTrunks

```
APR.device.route route_connectInstsToTrunks boolean { t | nil }
```

Description

Connects instance pins to trunks. If the trunk is directly over a source pin, it drops a via to connect a pin to the trunk.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Connect instances to trunks*

Examples

```
envGetVal("APR.device.route" "route_connectInstsToTrunks")
envSetVal("APR.device.route" "route_connectInstsToTrunks" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

route_createRoutingAsAGroup

```
APR.device.route route_createRoutingAsAGroup boolean { t | nil }
```

Description

Specifies whether or not to create routing as a figGroup. When set to `t`, the routing is created as a figGroup.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Create routing as a group*

Examples

```
envGetVal("APR.device.route" "route_createRoutingAsAGroup")
envSetVal("APR.device.route" "route_createRoutingAsAGroup" 'boolean t)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_createTrunks

```
APR.device.route route_createTrunks boolean { t | nil }
```

Description

Creates trunks for selected or all nets in the design.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Create trunks*

Examples

```
envGetVal("APR.device.route" "route_createTrunks ")
envSetVal("APR.device.route" "route_createTrunks " 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

route_defaultRoutedView

```
APR.device.route route_defaultRoutedView string "defaultViewName"
```

Description

Specifies the default name for the routed view if signal routing is written to another cellview.

The default is `layout.routed`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Other cellview*

Examples

```
envGetVal("APR.device.route" "route_defaultRoutedView")
envSetVal("APR.device.route" "route_defaultRoutedView" 'string "route1")
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_deleteManualRouting

```
APR.device.route route_deleteManualRouting boolean { t | nil }
```

Description

Deletes the manually routed wires and vias.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Delete – Manual routing*

Examples

```
envGetVal("APR.device.route" "route_deleteManualRouting")
envSetVal("APR.device.route" "route_deleteManualRouting" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

route_deleteTrunks

```
APR.device.route route_deleteTrunks boolean { t | nil }
```

Description

Deletes the routed trunks.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Delete – Trunks*

Examples

```
envGetVal("APR.device.route" "route_deleteTrunks")
envSetVal("APR.device.route" "route_deleteTrunks" 'boolean t)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_deleteWiresAndVias

```
APR.device.route route_deleteWiresAndVias boolean { t | nil }
```

Description

Deleted the wires and vias created by the router.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Delete – Wires and vias*

Examples

```
envGetVal("APR.device.route" " route_deleteWiresAndVias")
envSetVal("APR.device.route" " route_deleteWiresAndVias" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

route_displayLog

```
APR.device.route route_displayLog boolean { t | nil }
```

Description

Controls the display of the log window when signal routing is run. When set to `nil`, the log window is not displayed.

The default is `t`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Display Log*

Examples

```
envGetVal("APR.device.route" "check_displayLog")
envSetVal("APR.device.route" "check_displayLog" 'boolean nil)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_enableHierarchicalTrims

```
APR.device.route route_enableHierarchicalTrims boolean { t | nil }
```

Description

Enables trim insertion in hierarchical designs. This environment variable is available for certain advance node designs only.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Enable hierarchical trims*

Examples

```
envGetVal("APR.device.route" "route_enableHierarchicalTrims")
envSetVal("APR.device.route" "route_enableHierarchicalTrims" 'boolean nil)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_finishOverConstraints

```
APR.device.route route_finishOverConstraints boolean { t | nil }
```

Description

Enables the router to prefer finishing the routing of nets over considering the constraints. There maybe situations seen in some designs where some constraints may not be honored completely when the environment variable is set to `t`, which is the default.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Prefer finishing over constraints*

Examples

```
envGetVal("APR.device.route" "route_finishOverConstraints")
envSetVal("APR.device.route" "route_finishOverConstraints" 'boolean nil)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_finishOverDRCClean

```
APR.device.route route_finishOverDRCClean boolean { t | nil }
```

Description

Enables the router to prioritize finishing routing over DRC-correctness, when set to `t`.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Prefer finishing over constraints*

Examples

```
envGetVal("APR.device.route" "route_finishOverDRCClean")
envSetVal("APR.device.route" "route_finishOverDRCClean" 'boolean t)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_meshExtendPreroutes

```
APR.device.route route_meshExtendPreroutes boolean { t | nil }
```

Description

Lets you select whether or not to extend pre-routes to the active meshing layer.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Extend preroutes for mesh*

Examples

```
envGetVal("APR.device.route" "route_meshExtendPreroutes")
envSetVal("APR.device.route" "route_meshExtendPreroutes" 'boolean t)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_meshMinSkipTracks

```
APR.device.route route_meshMinSkipTracks boolean { t | nil }
```

Description

Determines whether or not to consider the minimum skip track value for mesh count mode.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab – *Mesh Layer Setup*
Field	*Min tracks to skip*

Examples

```
envGetVal("APR.device.route" "route_meshMinSkipTracks")
envSetVal("APR.device.route" "route_meshMinSkipTracks" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

route_meshMinSkipTracksCount

```
APR.device.route route_meshMinSkipTracksCount int skipLevelCount
```

Description

Specifies the minimum skip track value for mesh count mode.

The default is `1`.

GUI Equivalent

Command	Routing assistant – *Route* tab – *Mesh Layer Setup*
Field	*Min tracks to skip – Value*

Examples

```
envGetVal("APR.device.route" "route_meshMinSkipTracksCount")
envSetVal("APR.device.route" "route_meshMinSkipTracksCount" 'int 2)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_meshNets

```
APR.device.route route_meshNets cyclic { "Selected" }
```

Description

Specifies whether to route the selected nets for mesh routing in the *Assisted* mode.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Nets – Selected*

Examples

```
envGetVal("APR.device.route" " route_meshNets")
envSetVal("APR.device.route" " route_meshNets" 'cyclic "Selected")
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_meshNetsWithin

```
APR.device.route route_meshNetsWithin cyclic { "PR boundary" | "Device array/
     FigGroup | Area" }
```

Description

Specifies whether to route everything inside the PR boundary or area or within a device array or figGroup.

The default is `"PR boundary"`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Within – PR boundary*, *Device array/FigGroup*, *Area*

Examples

```
envGetVal("APR.device.route" " route_meshNetsWithin")
envSetVal("APR.device.route" " route_meshNetsWithin" 'cyclic "Area")
envSetVal("APR.device.route" " route_meshNetsWithin" 'cyclic "Device array/
FigGroup")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

route_minimizeM0OnNets

```
APR.device.route route_minimizeM0OnNets boolean { t | nil }
```

Description

Lets you minimize the capacitance on `M0` nets in addition to the regular trim insertion. When set to `t`, the length of the `M0` wires is trimmed down using multiple trims to minimize the capacitance.

This environment variable is available for certain advance node designs only.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Minimize M0 on nets*

Examples

```
envGetVal("APR.device.route" "route_minimizeM0OnNets")
envSetVal("APR.device.route" "route_minimizeM0OnNets" 'boolean t)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_nets

```
APR.device.route route_nets cyclic { "All" | "Selected" | "Open" | "Shorted" }
```

Description

Specifies whether to route all nets, selected nets, or only nets with opens or shorted nets.

The default is `"All"`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Nets – All*, *Selected*, *Open, Shorted*

Examples

```
envGetVal("APR.device.route" "route_nets")
envSetVal("APR.device.route" "route_nets" 'cyclic "Selected")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

route_netsWithin

```
APR.device.route route_netsWithin cyclic { "PR boundary" | "Device array/FigGroup
     | Area" }
```

Description

Specifies whether to route everything inside the PR boundary or only within a specified area, device array, or figGroup.

The default is `"PR boundary"`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Within – PR boundary*, *Device array/FigGroup*, *Area*

Examples

```
envGetVal("APR.device.route" " route_meshNetsWithin")
envSetVal("APR.device.route" " route_meshNetsWithin" 'cyclic "Device array/
FigGroup")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

route_objective

```
APR.device.route route_objective cyclic { "Wire length" | "Vias" }
```

Description

Specifies how the routing algorithm evaluates the quality of a solution. This is adjusted to assist the router to find solutions with different topologies.

- `Wire Length`: When specified, solutions that use more vias but less wire are favorable.
- `Vias`: When set, solutions that use fewer vias but more wires are favorable. This is the default option.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Optimize – Wire length, Vias*

Examples

```
envGetVal("APR.device.route" " route_objective")
envSetVal("APR.device.route" " route_objective" 'cyclic "Wire length")
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_overwriteLog

```
APR.device.route route_overwriteLog boolean { t | nil }
```

Description

Specifies whether to overwrite or keep the existing log. When set to `t`, the existing log is overwritten.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Overwrite last log*

Examples

```
envGetVal("APR.device.route" "route_overwriteLog")
envSetVal("APR.device.route" "route_overwriteLog" 'boolean t)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_routedLoc

```
APR.device.route route_routedLoc cyclic { "Current cellview" | "Other cellview" }
```

Description

Specifies whether to write the supply routing to the current cellview or to a different cellview.

The default is `"Current cellview"`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Current cellview, Other cellview*

Examples

```
envGetVal("APR.device.route" "route_routedLoc")
envSetVal("APR.device.route" "route_routedLoc" 'cyclic "Other cellview")
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_routingMode

```
APR.device.route route_routingMode cyclic { "Auto" | "Assisted" }
```

Description

Lets you select the routing mode. The default is `"Auto"`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Auto, Assisted*

Examples

```
envGetVal("APR.device.route" " route_routingMode")
envSetVal("APR.device.route" " route_routingMode" 'cyclic "Assisted")
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_saveRoutingOnly

```
APR.device.route route_saveRoutingOnly boolean { t | nil }
```

Description

Specifies whether to copy only the supply grid or all initial data and the supply grid to the new cellview.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Save routing only*

Examples

```
envGetVal("APR.device.route" "route_saveRoutingOnly")
envSetVal("APR.device.route" "route_saveRoutingOnly" 'boolean t)
```

Related Topics

Routing Assistant User Interface for Device-Level Routing

route_supplyNets

```
APR.device.route route_supplyNets boolean { t | nil }
```

Description

Specifies whether to route power and ground nets. When set to `nil`, the power and ground nets are not routed.

The default is `t`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Supply Nets*

Examples

```
envGetVal("APR.device.route" "route_supplyNets")
envSetVal("APR.device.route" "route_supplyNets" 'boolean nil)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

route_updatePins

```
APR.device.route route_updatePins boolean { t | nil }
```

Description

Controls pin placement in Innovus. When set to `nil`, the pin placement does not happen.

The default is `t`.

GUI Equivalent

Command	Routing assistant – *Route* tab
Field	*Update Pins (This option is currently inactive)*

Examples

```
envGetVal("APR.device.route" "route_updatePins")
envSetVal("APR.device.route" "route_updatePins" 'boolean nil)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

route_weightedSumDistance

```
APR.device.route route_weightedSumDistance boolean { t | nil }
```

Description

Controls the distance of spine generation. The location of spine from pins depends on whether or not the computed distance is valid and is not occupied or blocked. The closest valid distance is selected for generating a spine.

The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" " route_weightedSumDistance")
envSetVal("APR.device.route" " route_weightedSumDistance" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

rowHeight

`APR.device.grid rowHeight float` *`rowHeight`*

Description

Specifies the height of the row region. This environment variable is applicable only when rowRegionMode is set to `specifyRowHeight`. The default value is `0.00`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Specify row height*

Examples

```
envGetVal("APR.device.grid" "rowHeight")
envSetVal("APR.device.grid" "rowHeight" 'float 2.00)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

rowRegionMode

```
APR.device.grid rowRegionMode cyclic { "deriveRowRegion" | "specifyRowHeight" |
     "specifyRowTemplate" }
```

Description

Specifies the mode to derive the row height when creating row regions. The available options are:

- `deriveRowRegion`: Automatically calculates the row height based on the devices in the design.
- `specifyRowHeight`: Lets you specify a row height value using the rowHeight environment variable.
- `specifyRowTemplate`: Lets you use the useRowTemplate environment variable to select an existing row template from which the row height value is to be used.

The default value is `deriveRowRegion`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Auto compute row height, Specify row height, Use row template*

Examples

```
envGetVal("APR.device.grid" "rowRegionMode")
envSetVal("APR.device.grid" "rowRegionMode" 'cyclic "specifyRowHeight")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

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 P&R assistant – *Setup* tab
Field	*#S/D Tracks, S/D Track Width*

Examples

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

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

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 P&R assistant – *Setup* tab
Field	*S/D Track Width*

Examples

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

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

setup_checkDRCsAfterRouting

```
APR.device.route setup_checkDRCsAfterRouting boolean { t | nil }
```

Description

Specifies whether to automatically run design rule checks after routing. When set to `t`, the design rule checks are run automatically.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Setup* tab
Field	*Check DRCs after routing*

Examples

```
envGetVal("APR.device.route" "setup_checkDRCsAfterRouting")
envSetVal("APR.device.route" "setup_checkDRCsAfterRouting" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

setup_createPolyPattern

```
APR.device.place setup_createPolyPattern boolean { t | nil }
```

Description

Specifies whether poly snap patterns must be generated.

The default is `t`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Create Poly Grid (WSP)*

Examples

```
envGetVal("APR.device.place" "setup_createPolyPattern")
envSetVal("APR.device.place" "setup_createPolyPattern" 'boolean nil)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

setup_definePNPattern

```
APR.device.place setup_definePNPattern boolean { t | nil }
```

Description

Specifies whether the distribution of P and N devices in rows is based on the rowHeight setting.

The default value is `nil`, in which case the row region has a single row attribute.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Define P/N Pattern*

Examples

```
envGetVal("APR.device.grid" "setup_definePNPattern")
envSetVal("APR.device.grid" "setup_definePNPattern" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

setup_flippedRowsStartWith

```
APR.device.place setup_flippedRowsStartWith cyclic { "Start with R0" | "Start with
     MX" }
```

Description

Specifies the orientation of the first (bottom-most) row based on which flipped rows are generated. The available options are `Start with R0` and `Start with MX`.

The default is `"Start with R0"`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Generate Flipped Rows*

Examples

```
envGetVal("APR.device.place" "setup_flippedRowsStartWith")
envSetVal("APR.device.place" "setup_flippedRowsStartWith" 'cyclic "Start with MX")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

setup_genFlippedRows

```
APR.device.place setup_genFlippedRows boolean { t | nil }
```

Description

Specifies whether flipped patterns and rows are to be generated.

The default value is `nil`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Generate Flipped Rows*

Examples

```
envGetVal("APR.device.place" "setup_genFlippedRows")
envSetVal("APR.device.place" "setup_genFlippedRows" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

setup_gridOffset

`APR.device.place setup_gridOffset float` *`gridOffset`*

Description

Specifies the the offset of poly grids. This environment variable is valid only when setup_specifyGridOffset is set to `t`.

The default value is `0.00`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Specify Grid Offset*

Examples

```
envGetVal("APR.device.place" "setup_gridOffset")
envSetVal("APR.device.place" "setup_gridOffset" float 1.00)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Auto P&R Assistant User Interface for Device-Level Placement

specifyGridOffset

setup_includePassiveComponentPoly

```
APR.device.place setup_includePassiveComponentPoly boolean { t | nil }
```

Description

Recognizes the poly layer patterns of passive devices while generating WSPs.

The default value is `nil`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Include Passive Component Poly*

Examples

```
envGetVal("APR.device.place" "setup_includePassiveComponentPoly")
envSetVal("APR.device.place" "setup_includePassiveComponentPoly" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

setup_insertTrim

```
APR.device.route setup_insertTrim boolean { t | nil }
```

Description

Controls trim insertion to fix DRC errors.

The default is `t`.

GUI Equivalent

Command	Routing assistant – *Setup* tab
Field	*Insert trim to fix DRCs*

Examples

```
envGetVal("APR.device.route" " setup_insertTrim")
envSetVal("APR.device.route" " setup_insertTrim" 'boolean nil)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

setup_lockColorsAfterRouting

```
APR.device.route setup_lockColorAfterRouting boolean { t | nil }
```

Description

Specifies whether to automatically lock colored shapes after routing. If the process has trim shapes that cut locked color shapes, then it is selected and disabled. If the process does not have trim shapes, it is unselected and disabled. If, the process has trim shapes and it cuts unlocked color shapes, it is off by default but is enabled.

The default value varies depending on the technology. If the technology does not support coloring, the default is `nil`. However, if the technology supports coloring, default is `t`.

GUI Equivalent

Command	Routing assistant – *Setup* tab
Field	*Lock colors after routing*

Examples

```
envGetVal("APR.device.route" "setup_lockColorAfterRouting")
envSetVal("APR.device.route" "setup_lockColorAfterRouting" 'boolean nil)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

setup_patternName

```
APR.device.place setup_patternName cyclic { "PNNP" | "NPPN" }
```

Description

Specifies the pattern to define distribution of P and N devices in rows.

The default value is `PNNP`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Define P/N Pattern*

Examples

```
envGetVal("APR.device.place" "setup_patternName")
envSetVal("APR.device.place" "setup_patternName" 'cyclic "NPPN")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

setup_placementRegion

```
APR.device.place setup_placementRegion cyclic { "Create Diffusion Grid" | "Create
     Row Region" }
```

Description

Specifies the region for placement. The available options are:

- `Create Diffusion Grid`: Creates a diffusion grid based on the heights of the diffusion layers of different devices in the design. This option is valid only if enablePlaceWithWsp is set to `t` before launching the session.
- `Create Row Region`: Creates a row region.

The default value is `Create Diffusion Grid` when enablePlaceWithWsp is set to t. When set to nil, `Create Row Region` is the default value.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Placement Region*

Examples

```
envGetVal("APR.device.place" "setup_placementRegion")
envSetVal("APR.device.place" "setup_placementRegion" 'cyclic "Create Row Region")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

setup_rowRegionMode

```
APR.device.place setup_rowRegionMode cyclic { "Auto Compute Row Height" | "Specify
     Row Height" | "Use Row Template" }
```

Description

Specifies the mode to derive the row height when creating row regions. The available options are:

- `Auto Compute Row Height`: Automatically calculates the row height based on the devices in the design.
- `Specify Row Height`: Lets you specify a row height value using the setup_specifyRowHeight environment variable.
- `Use Row Template`: Lets you use the setup_useRowTemplate environment variable to select an existing row template from which the row height value is to be used.

The default value is `Auto Compute Row Height`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Auto compute row height, Specify row height, Use row template*

Examples

```
envGetVal("APR.device.place" "setup_rowRegionMode")
envSetVal("APR.device.place" "setup_rowRegionMode" 'cyclic "Specify Row Height")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

setup_specifyGridOffset

```
APR.device.place setup_specifyGridOffset boolean { t | nil }
```

Description

Lets you use the setup_gridOffset environment variable to specify the offset of poly grids.

The default value is `nil`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Specify Grid Offset*

Examples

```
envGetVal("APR.device.place" "setup_specifyGridOffset")
envSetVal("APR.device.place" "setup_specifyGridOffset" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Auto P&R Assistant User Interface for Device-Level Placement

setup_specifyRowHeight

```
APR.device.place setup_specifyRowHeight float rowHeight
```

Description

Specifies the height of the row region. This environment variable is applicable only when setup_rowRegionMode is set to `Specify Row Height`.

The default value is `0.00`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Specify row height*

Examples

```
envGetVal("APR.device.place" "setup_specifyRowHeight")
envSetVal("APR.device.place" "setup_specifyRowHeight" 'float 2.00)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

setup_usePassiveComponentHeights

```
APR.device.place setup_usePassiveComponentHeights boolean { t | nil }
```

Description

Considers passive components, such as inductors, transformers, and transmission lines, for row height calculations during row region and poly grid generation.

The default value is `nil`, in which case passive components are ignored.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Use Passive Component Heights*

Examples

```
envGetVal("APR.device.place" "setup_usePassiveComponentHeights")
envSetVal("APR.device.place" "setup_usePassiveComponentHeights" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

setup_useRowTemplate

`APR.device.place setup_useRowTemplate string` *`templateName`*

Description

Specifies the row template from which the row height value is to be used for the WSPs. This environment variable is applicable only when setup_rowRegionMode is set to `Use Row Template`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Use row template*

Examples

```
envGetVal("APR.device.place" "setup_useRowTemplate")
envSetVal("APR.device.place" "setup_useRowTemplate" 'string "myTemplate1")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

shieldInsertBlockageInsideShieldGap

```
APR.device.route shieldInsertBlockageInsideShieldGap boolean { t | nil }
```

Description

Enables shielding to add blockages in gap between shield and signal. The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "shieldInsertBlockageInsideShieldGap")
envSetVal("APR.device.route" "shieldInsertBlockageInsideShieldGap" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

showOpensTreeBoxes

```
APR.device.route showOpensTreeBoxes boolean { t | nil }
```

Description

Adds an additional entry *Open Boxes* as part of the report that is displayed at the end of routing. Also, the bounding box of nets with opens is outputted to CIW. The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "showOpensTreeBoxes")
envSetVal("APR.device.route" "showOpensTreeBoxes" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

specifyGridOffset

```
APR.device.grid specifyGridOffset boolean { t | nil }
```

Description

Lets you use the gridOffset environment variable to specify the offset of poly grids.

The default value is `nil`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Specify Grid Offset*

Examples

```
envGetVal("APR.device.grid" "specifyGridOffset")
envSetVal("APR.device.grid" "specifyGridOffset" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Auto P&R Assistant User Interface for Device-Level Placement

gridOffset

stackDummyFingerCount

```
APR.device.fill stackDummyFingerCount int int_spacing
```

Description

Specifies the number of fingers to be included as stack dummy transition fill when they are inserted in the *Fill* step of the automated device placement flow.

The default value is `2`.

This environment variable is valid only in certain advanced node PDKs.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "stackDummyFingerCount")
envSetVal("APR.device.place" "stackDummyFingerCount" 'int 3)
```

Related Topics

Auto P&R Assistant User Interface for Device-Level Placement

strictShieldHalos

```
APR.device.infra strictShieldHalos boolean { t | nil }
```

Description

Controls whether the shield spacing checks are strictly applied. This means that it tries to achieve 100% shielding. However, it is not considered a failure to have less than 100% shielding.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "strictShieldHalos")
envSetVal("APR.device.infra" "strictShieldHalos" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

supply_connectToTermAndGR

```
APR.device.route supply_connectToTermAndGR boolean { t | nil }
```

Description

Specifies whether or not the supply router should connect to overlapping instance pins and guard rings.

The default is `t`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Connect to overlapped terminals and guardrings*

Examples

```
envGetVal("APR.device.route" "supply_connectToTermAndGR")
envSetVal("APR.device.route" "supply_connectToTermAndGR" 'boolean nil)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_createGridAsGroup

```
APR.device.route supply_createGridAsGroup boolean { t | nil }
```

Description

Creates the supply grid as a figGroup.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Create supply grid as a group*

Examples

```
envGetVal("APR.device.route" "supply_createGridAsGroup")
envSetVal("APR.device.route" "supply_createGridAsGroup" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_createOnPNRegions

```
APR.device.route supply_createOnPNRegions boolean { t | nil }
```

Description

Determines whether or not to only route inside P region or N region or not in shared track mode.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Create only on P/N regions*

Examples

```
envGetVal("APR.device.route" "supply_createOnPNRegions")
envSetVal("APR.device.route" "supply_createOnPNRegions" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_createPinLabel

```
APR.device.route supply_createPinLabel boolean { t | nil }
```

Description

Creates labels on pins when the *Pins Create* option is selected in the *Supply* tab of the Routing assistant.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Pins – Create label*

Examples

```
envGetVal("APR.device.route" "supply_createPinLabel")
envSetVal("APR.device.route" "supply_createPinLabel" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_createPins

```
APR.device.route supply_createPins boolean { t | nil }
```

Description

Creates a pin instead of a pathSeg on supply stripes.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Create pins on supply stripes*

Examples

```
envGetVal("APR.device.route" "supply_createPins")
envSetVal("APR.device.route" "supply_createPins" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_createPinsOnEnds

```
APR.device.route supply_createPinsOnEnds boolean { t | nil }
```

Description

Creates square pins on both ends of a power stripe instead of creating a long rectangular pin overlapping the stripe. When set to `t`, no power pins are created.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Create on ends*

Examples

```
envGetVal("APR.device.route" "supply_createPinsOnEnds")
envSetVal("APR.device.route" "supply_createPinsOnEnds" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_createPinsOnPinPurpose

```
APR.device.route supply_createPinsOnPinPurpose boolean { t | nil }
```

Description

Creates supply routing pins on pin purpose instead of drawing purpose.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Create on pin purpose*

Examples

```
envGetVal("APR.device.route" "supply_createPinsOnPinPurpose")
envSetVal("APR.device.route" "supply_createPinsOnPinPurpose" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_defaultRoutedCellExpression

```
APR.stdcell.route supply_defaultRoutedCellExpression string "viewName"
```

Description

Specifies the output to the other cell view. The cell is created as an instance in the design library, You can search for the cell name in the design library with <*cellname*>_proute.

The default is %_proute, where the symbol % is <*cellname*>.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Specify Routed Cellview*

Examples

```
envGetVal("APR.stdcell.route" "supply_defaultRoutedCellExpression")
envSetVal("APR.stdcell.route" "supply_defaultRoutedCellExpression" 'string
"dut_proute")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

supply_defaultRoutedView

```
APR.device.route supply_defaultRoutedView string "viewName"
```

Description

Specifies the routed view name if supply routing is written to another cellview.

The default is `layout.power`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Other cellview*

Examples

```
envGetVal("APR.device.route" "supply_defaultRoutedView")
envSetVal("APR.device.route" "supply_defaultRoutedView" 'string "routed1")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_deleteStripes

```
APR.device.route supply_deleteStripes boolean { t | nil }
```

Description

Deletes the stripes created by supply router.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Delete – Supply stripes*

Examples

```
envGetVal("APR.device.route" "supply_deleteStripes")
envSetVal("APR.device.route" "supply_deleteStripes" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_deleteVias

```
APR.device.route supply_deleteVias boolean { t | nil }
```

Description

Deletes the vias created by supply router.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Delete – Supply vias*

Examples

```
envGetVal("APR.device.route" "supply_deleteVias")
envSetVal("APR.device.route" "supply_deleteVias" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_displayLog

```
APR.device.route supply_displayLog boolean { t | nil }
```

Description

Controls the display of the log window when supply routing is run. When set to `nil`, the log window is not displayed.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Display Log*

Examples

```
envGetVal("APR.device.route" "supply_displayLog")
envSetVal("APR.device.route" "supply_displayLog" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_genSupplyStripes

```
APR.device.route supply_genSupplyStripes boolean { t | nil }
```

Description

Specifies that supply stripes should be generated. When set to `nil`, the supply stripes are not generated.

The default is `t`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Generate supply stripes*

Examples

```
envGetVal("APR.device.route" "supply_genSupplyStripes")
envSetVal("APR.device.route" "supply_genSupplyStripes" 'boolean nil)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_generateStaples

```
APR.device.route supply_generateStaples boolean { t | nil }
```

Description

Enables supply stapling.

The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "supply_generateStaples")
envSetVal("APR.device.route" "supply_generateStaples" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_IgnoreBoundaryTracks

```
APR.device.route supply_IgnoreBoundaryTracks boolean { t | nil }
```

Description

Creates power stripes on tracks that are coincident with the PR boundary. When set to `t`, it ignores generating power stripes on tracks that are on the prBoundary.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Ignore boundary tracks*

Examples

```
envGetVal("APR.device.route" "supply_IgnoreBoundaryTracks")
envSetVal("APR.device.route" "supply_IgnoreBoundaryTracks" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_IgnoreBoundaryVias

```
APR.device.route supply_IgnoreBoundaryVias boolean { t | nil }
```

Description

Creates power vias on tracks that are coincident with the PR boundary. When set to `t`, it ignores generating power vias on tracks that are on the prBoundary.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Ignore boundary vias*

Examples

```
envGetVal("APR.device.route" "supply_IgnoreBoundaryVias")
envSetVal("APR.device.route" "supply_IgnoreBoundaryVias" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_insertVias

```
APR.device.route supply_insertVias boolean { t | nil }
```

Description

Specifies that vias are inserted between the intersection of the layer above and below the via in the supply grid.

The default is `t`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Insert vias for supply stripes*

Examples

```
envGetVal("APR.device.route" "supply_insertVias")
envSetVal("APR.device.route" "supply_insertVias" 'boolean nil)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_nets

```
APR.device.route supply_nets cyclic { "All" | "Selected" }
```

Description

Specifies whether to route all or selected supply nets.

The default is `"All"`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Supply Nets – All*, *Selected*

Examples

```
envGetVal("APR.device.route" "supply_nets")
envSetVal("APR.device.route" "supply_nets" 'cyclic "Selected")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_netsWithin

```
APR.device.route supply_netsWithin cyclic { "PR boundary" | "Guardring/FigGroup" |
     "Area" }
```

Description

Specifies whether to route everything inside the PR boundary or a specified area or only within a guard ring or figGroup.

The default is `"PR boundary"`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Supply Nets Within – PR boundary*, *Guardring/FigGroup, Area*

Examples

```
envGetVal("APR.device.route" "supply_netsWithin")
envSetVal("APR.device.route" "supply_netsWithin" 'cyclic "Guardring/FigGroup")
envSetVal("APR.device.route" "supply_netsWithin" 'cyclic "Area")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_overwriteLog

```
APR.device.route supply_overwriteLog boolean { t | nil }
```

Description

Specifies whether you want to overwrite the last log file or keep the existing one. When set to `t`, the existing log is overwritten.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Overwrite last log*

Examples

```
envGetVal("APR.device.route" "supply_overwriteLog")
envSetVal("APR.device.route" "supply_overwriteLog" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_pinLayers

```
APR.stdcell.route supply_pinLayers string "pinLayer"
```

Description

Lets you specify list of layers to create pins on for supply routing.

The default is `""`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Does it have a corresponding GUI option*

Examples

```
envGetVal("APR.stdcell.route" "supply_pinLayers")
envSetVal("APR.stdcell.route" "supply_pinLayers" 'string "metal1")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

supply_routedLoc

```
APR.device.route supply_routedLoc cyclic { "Current cellview" | "Other cellview" }
```

Description

Specifies whether to write the supply routing to the current cellview or to a different cellview.

The default is `"Current cellview"`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Current cellview, Other cellview*

Examples

```
envGetVal("APR.device.route" "supply_routedLoc")
envSetVal("APR.device.route" "supply_routedLoc" 'cyclic "Other cellview")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_pinLayerSet

```
APR.device.route supply_pinLayerSet cyclic { "Use all supply stripe layers" | "Use
     selected layers" }
```

Description

Specifies the bottom and top layers of the supply stripes layer range that pins should be created on.

The default is `"Use all supply stripe layers"`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Pins Create*

Examples

```
envGetVal("APR.device.route" "supply_pinLayerSet")
envSetVal("APR.device.route" "supply_pinLayerSet" 'cyclic "Use selected layers")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_saveRoutingOnly

```
APR.device.route supply_saveRoutingOnly boolean { t | nil }
```

Description

Specifies whether to copy only the supply grid or all initial data and the supply grid to the new cellview.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Save routing only*

Examples

```
envGetVal("APR.device.route" "supply_saveRoutingOnly")
envSetVal("APR.device.route" "supply_saveRoutingOnly" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_shareTracks

```
APR.device.route supply_shareTracks boolean { t | nil }
```

Description

Allows PG tracks to be shared by each other when applicable in corresponding P and N device locations.

The default is `nil`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Share tracks for supply nets*

Examples

```
envGetVal("APR.device.route" "supply_shareTracks")
envSetVal("APR.device.route" "supply_shareTracks" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

supply_useExisitingPGTracks

```
APR.device.route supply_useExisitingPGTracks boolean { t | nil }
```

Description

Restricts supply routing to layers with existing power or ground tracks and allows all routing layers to be used.

The default is `t`.

GUI Equivalent

Command	Routing assistant – *Supply* tab
Field	*Only use layers with WSP P/G tracks*

Examples

```
envGetVal("APR.device.route" "supply_useExisitingPGTracks")
envSetVal("APR.device.route" "supply_useExisitingPGTracks" 'boolean nil)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

tieShieldTieCellList

```
APR.device.route tieShieldTieCellList string "libCellViewName"
```

Description

Specifies the library, cell, and view names of the tie cells to connect to during tie shielding.

The default is an empty string.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.route" "tieShieldTieCellList")
envSetVal("APR.device.route" "tieShieldTieCellList" 'string "lib1 cell1 view1 lib2 cell2 view2")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

Routing Assistant User Interface for Device-Level Routing

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 P&R assistant – *Setup* tab
Field	*Top Layer*

Examples

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

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

trackWidth

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

Description

Specifies the default width of each track.

The default is `0.0`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Compute Pattern Width*

Examples

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

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

transitionFillFingerCount

```
APR.device.fill transitionFillFingerCount int int_numberOfFingers
```

Description

Specifies the number of fingers to be included in the stack gate transition fill cells when multiFingerTransFill is set to `t`. These fill cells are inserted in the gaps between active devices in the *Fill* step of the automated device placement flow.

The default value is `1`, in which case analog cells are used as transition fill. When set to any value higher than `1`, analog stack gate cells with the specified number of fingers are used as transition fill.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.fill" "transitionFillFingerCount")
envSetVal("APR.device.fill" "transitionFillFingerCount" 'int 10)
```

Related Topics

multiFingerTransFill

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

transitionUseDummyCell

```
APR.device.fill transitionUseDummyCell boolean { t | nil }
```

Description

Specifies whether to insert transition dummy cells next to active cells while inserting device fill.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.fill" "transitionUseDummyCell")
envSetVal("APR.device.fill" "transitionUseDummyCell" 'boolean t)
```

Related Topics

Base Layer Fill in the Automated Device Placement and Routing Flow

useHaloFile

```
APR.device.infra useHaloFile boolean { t | nil }
```

Description

Writes JSON files called via_halos.json containing all the via halos, if the file does not exist. If the file does exist, the via halos are instead read from it and used. This provides a means for the halos to be overridden,

When the file exists, halos are used as is and generated only for new vias not in the halo file.

It also reads and writes a corresponding file called seg_halos.json for the segment halos.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "useHaloFile")
envSetVal("APR.device.infra" "useHaloFile" 'boolean t)
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

useLayoutAsSeed

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

Description

Runs the placer from its current layout. The placement results depend on the starting layout. Consider the same layout generated using two different methods. The placement results for these layouts might differ.

However, placement results will remain the same for a specific starting layout. For example, the placement result will remain the same for a layout that is always generated using Generating All Components from Source.

The default is `nil`.

GUI Equivalent

None

Examples

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

Related Topics

Virtuoso Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

usePassiveComponentHeights

```
APR.device.grid usePassiveComponentHeights boolean { t | nil }
```

Description

Considers passive components, such as inductors, transformers, and transmission lines, for row height calculations during row region and poly grid generation.

The default value is `nil`, in which case passive components are ignored.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Use Passive Component Heights*

Examples

```
envGetVal("APR.device.grid" "usePassiveComponentHeights")
envSetVal("APR.device.grid" "usePassiveComponentHeights" 'boolean t)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

useRowHeight

`APR.device.grid useRowHeight float` *`float_number`*

Description

Specifies the horizontal height.

The default is `0.0`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "useRowHeight")
envSetVal("APR.device.grid" "useRowHeight" 'float 1.0)
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

useRowTemplate

```
APR.device.grid useRowTemplate string templateName
```

Description

Specifies the row template from which the row height value is to be used for the WSPs. This environment variable is applicable only when rowRegionMode is set to `specifyRowTemplate`.

GUI Equivalent

Command	Auto P&R assistant – *Setup* tab
Field	*Use row template*

Examples

```
envGetVal("APR.device.grid" "useRowTemplate")
envSetVal("APR.device.grid" "useRowTemplate" 'string "myTemplate1")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

vgFillSpacePitchesOrUserUnit

```
APR.device.fill vgFillSpacePitchesOrUserUnit float float_number
```

Description

Specifies the number of poly pitches or user units in which fill devices are to be extended.

Before setting this environment variable, use vgFillSpaceSpecifyUnit to specify the unit in which fill extensions are the be specified.

`vgFillSpaceSpecifyUnit` and `vgFillSpacePitchesOrUserUnit` work only when dummyFillEnableFillEmptyRow is set to `true` or *Extend Fill* is selected on the *Fill* tab of the Auto P&R assistant.

The default value is `0.0`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "vgFillSpacePitchesOrUserUnit")
envSetVal("APR.device.place" "vgFillSpacePitchesOrUserUnit" 'float 2.0)
```

Related Topics

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

vgFillSpaceSpecifyUnit

```
APR.device.fill vgFillSpaceSpecifyUnit cyclic { "Poly pitches" | "User Unit" }
```

Description

Sets the unit in which fill extensions are to be specified. The valid values are:

- `Poly pitches`: Specifies the extension value in terms of the number of poly pitches.
- `User Unit`: Specifies the extension value in terms of user units.

Use vgFillSpacePitchesOrUserUnit to specify the number of poly pitches or user units. `vgFillSpaceSpecifyUnit` and `vgFillSpacePitchesOrUserUnit` work only when dummyFillEnableFillEmptyRow is set to `true` or *Extend Fill* is selected on the *Fill* tab of the Auto P&R assistant.

The default value is `Poly pitches`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "vgFillSpaceSpecifyUnit")
envSetVal("APR.device.place" "vgFillSpaceSpecifyUnit" 'cyclic "User Unit")
```

Related Topics

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

vgHorizontalIntraGroupSpaceInPitches

```
APR.device.place vgHorizontalIntraGroupSpaceInPitches int int_spacing
```

Description

Specifies the minimum spacing between Modgens, figGroups, or top-level devices within a virtual group. Spacing is measured between the diffusion edges.

Appropriate setting of this environment variable improves the dummy fill insertion between Modgens.

Valid values are `0` through `1000`.

The default value is `1`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "vgHorizontalIntraGroupSpaceInPitches")
envSetVal("APR.device.place" "vgHorizontalIntraGroupSpaceInPitches" 'int 10)
```

Related Topics

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

vgHorizontalSpaceSpecify

`APR.device.place vgHorizontalSpaceSpecify float` *`float_number`*

Description

Specifies the absolute horizontal spacing value to be applied above, below, or between the row regions. This setting is applied only when vgSpacingMode is set to `Specify`.

The default value is `1.0`.

GUI Equivalent

Command	Auto P&R assistant – *Place tab*
Field	*Transition Spacing – Specify – Horizontal*

Examples

```
envGetVal("APR.device.place" "vgHorizontalSpaceSpecify")
envSetVal("APR.device.place" "vgHorizontalSpaceSpecify" 'float 2.0)
```

Related Topics

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

vgPriority

```
APR.device.place viaParamOverrides string "priority_list"
```

Description

Specifies the priority to be followed while forming virtual groups in a design.

The default is `"type PN pitch height bulkNetName"`, which means that virtual groups are first separated based on their types—passive and active devices, then by their PN, followed by their poly pitches, and finally by their heights.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.place" "vgPriority")
```

```
envSetVal("APR.device.place" "vgPriority" 'string "type height PN pitch height bulkNetName")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

vgSpacingMode

```
APR.device.place vgSpacingMode cyclic { "Compact" | "Specify" }
```

Description

Adds transition rows or spacing above, below, or between the row regions. The available options are:

- `Compact`: The row vertical gap is set to `0` and the horizontal gap is set to the minimum value, following DRC rules.
- `Specify`: Lets you specify absolute vertical and horizontal spacing values using vgHorizontalSpaceSpecify and vgVerticalSpaceSpecify.

The default value is `Compact`.

GUI Equivalent

Command	Auto P&R assistant – *Place tab*
Field	*Transition Spacing*

Examples

```
envGetVal("APR.device.place" "vgSpacingMode")
envSetVal("APR.device.place" "vgSpacingMode" 'cyclic "Specify")
```

Related Topics

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

vgVerticalSpaceSpecify

`APR.device.place vgVerticalSpaceSpecify float` *`float_number`*

Description

Specifies the absolute vertical spacing value to be applied above, below, or between the row regions. This setting is applied only when vgSpacingMode is set to `Specify`.

The default value is `1.0`.

GUI Equivalent

Command	Auto P&R assistant – *Place tab*
Field	*Transition Spacing – Specify – Vertical*

Examples

```
envGetVal("APR.device.place" "vgVerticalSpaceSpecify")
envSetVal("APR.device.place" "vgVerticalSpaceSpecify" 'float 2.0)
```

Related Topics

Placing Devices Automatically in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement

viaParamOverrides

`APR.device.infra viaParamOverrides string "`*`viaParameterName`*`"`

Description

Overrides a given parameter on a specified via variant.

The default is `""`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.infra" "viaParamOverrides")
envSetVal("APR.device.infra" "viaParamOverrides" 'string "via1")
```

Related Topics

Automated Device Placement and Routing Flow Environment Variables

wireTypeAbbrev

```
APR.device.grid wireTypeAbbrev string "wire_type symbol"
```

Description

Assigns wire types to abbreviations, for example, abbreviation P is assigned to wire type power.

The default value is `""`.

GUI Equivalent

None

Examples

```
envGetVal("APR.device.grid" "wireTypeAbbrev")
envSetVal("APR.device.grid" "wireTypeAbbrev" string "power P, ground G, signal S")
```

Related Topics

Deriving Row Regions and Grids in the Automated Device Placement and Routing Flow

Auto P&R Assistant User Interface for Device-Level Placement