cadence®

Virtuoso Simulation Driven Interactive Routing User Guide

Product Version IC23.1
September 2023

Printed in the United States of America.

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

Contents

1

Introduction to Simulation Driven Routing

The Virtuoso Simulation Driven Routing (SDR) solution is a step towards correct-by-construction routing driven by electrical requirements. It provides an environment to consider current density and maximum resistance design rules during interactive routing. It is a solution that lets you take into consideration the current information and automatically resizes wires and vias during interactive routing.

The Virtuoso Simulation Driven Interactive routing flow enhances and improves Virtuoso from an electrically aware design flow to simulation-driven routing, provides a powerful new way to meet the current density constraints, and improves the productivity and design reliability.

As part of this flow, you can capture the current data from simulations extracted values (data set) coupled with a parasitic extractor (ICT file), use it to perform electromigration (EM) checks with the physical design layout, and update the layout to correct any violations.

Virtuoso Simulation-Driven Interactive Routing is available in Layout EXL and higher tiers and uses the **Virtuoso_Layout_Suite_EXL (95800),** Product Name: Virtuoso Layout Suite EXL license.

The key features of Virtuoso® simulation-driven routing are as follows:

- visualize the current distribution per net. It provides an easy way to look at the net topology and current distribution before routing.
- control simulation-driven routing to calculate the current according to the net topology.
- auto size wires and vias according to the estimated current.

- auto connect devices according to the estimated current. It provides an easy and flexible way to connect devices as you route, especially for multi-finger devices.

Related Topics

Prerequisites for the SDR Flow

The SDR Flow in Virtuoso

Prerequisites for the SDR Flow

Simulation-driven interactive routing is performed using the current of a net terminal coupled with EM reliability rules (ICT file).

The minimum set of files and data that you need to use SDR are as follows:

- ICT file with EM rules (same as Voltus-Fi)

  If only an ICT file is provided, SDR can run and display the maximum current passed by a wire. However, if the current of a terminal is also provided, SDR resizes the wire according to the simulation data. The terminal current extracted after simulation is generated using ADE.

- Dataset

  The dataset can be generated from:

  - ADE-XL with Spectre
  - ADE-XL with an integrated third-party simulator

    In case of a third-party simulator or no simulation information, this approach does not support Dynamic EM because only average current is captured.

- `.CSV` file with terminal currents

  The terminal current can also be initialized manually in the EAD assistant tables or imported from a `.csv` file. Alternatively, you can specify the terminal current directly in the EAD browser.

Related Topics

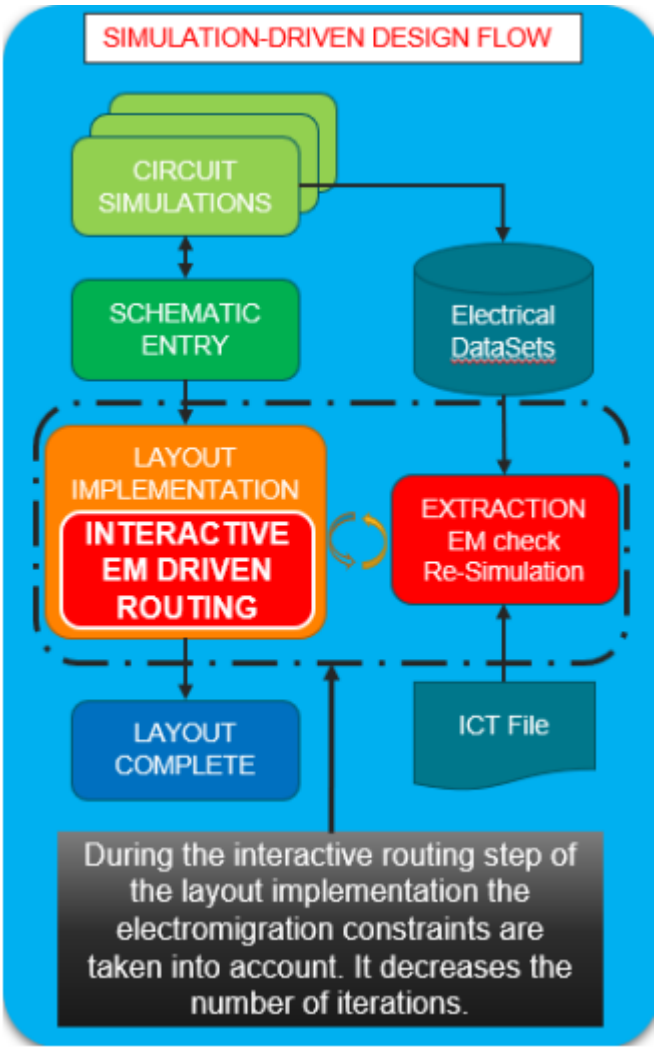
Introduction to Simulation Driven Routing

The SDR Flow in Virtuoso

The SDR Flow in Virtuoso

Before you start working with the SDR solution, ensure that EAD is initialized with an EAD setup file that defines the location of the ICT file and the EAD dataset and the analysis settings. When SDR is run using the setup information, it checks whether the specified EM data source file (EM data file and ICT file) exists and is readable.

The following diagram provides a snapshot of how SDR fits into the layout design flow.



In the SDR flow, the parasitic extraction and electromigration checks happen concurrently while the layout is implemented. In case of a classical flow (without SDR), the electrical impact is unknown until the layout is complete. In simulation-driven routing, during layout implementation, electromigration (EM) constraints are considered, which in turn, decreases the number of iterations.

Related Topics

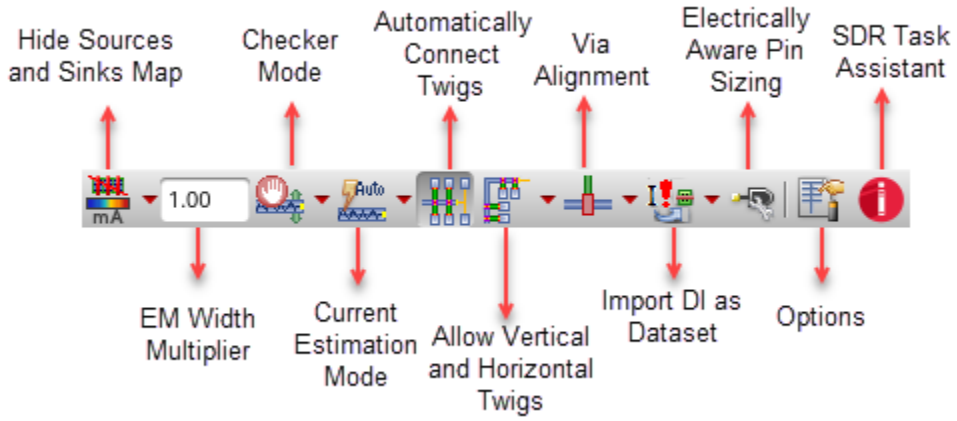
Introduction to Simulation Driven Routing

Prerequisites for the SDR Flow

SDR Toolbar

SDR is an effective way to find and display nets with accurate current and resistance estimation.

The SDR toolbar is designed to help you with the various simulation driven interactive routing features and settings. The SDR toolbar is installed automatically when you launch Layout EXL from the layout or schematic view. In case it is not available by default in Layout EXL, you can open it using *Windows – Toolbars – SDR Toolbar*. The buttons on the *SDR Toolbar* let you access SDR features with a single click.



The following table describes the SDR toolbar buttons.

Icon	Command	Description
	Sources and Sinks Map	Specifies whether or not the source and sink map should be displayed. **Environment Variable**: weSdrSourceSinkMap
	*Display Sources and Sinks Map Clusters*	Shows the current distribution across pin clusters as a thermal map.

	*Display Sources and Sinks Map Pins*	Shows the current distribution on a single pin as a thermal map.
	*Hide Sources and Sinks Map*	Hides the thermal map showing current distribution across pins and clusters. This is the default option.
	*EM Width Multiplier*	Helps in increasing or reducing the wires width compared to the estimated width required to satisfy the EM value **Environment Variable**: weSdrWidthMultiplier
	Checker Mode	Lets you choose the SDR checker mode to be used for current and resistance estimation. **Environment Variable**: weSdrCheckMode
	*Enforce*	Estimates the current or resistance in the edited wire and vias according to the EAD settings (dataset, temperature, and current scaling) and automatically calculates the wire width and via cuts to avoid EM or `maxResistance` violations.
	*Notify*	Estimates the current in the edited wire using a color code based on EAD EM violations color settings. The color is computed to represent the EM or `maxResistance` violation that is reported by the EAD checker after routing.
	*Off*	Retains the existing wire width and there is no feedback on estimated current or resistance.
	Current Estimation Mode	Lets you control how the current in the edited wire and via is estimated and calculated. **Environment Variable**: weSdrCurrentEstimationMode
	*Auto*	Estimates the current automatically in the last section of the wire according to the connected objects and flightline targets.
	*Sum Connected Pins Current*	Sums up the current of all the physically connected pins.
	*Maintain Constant Current*	Inherits the current from the previous wires.

	*Nearest Island Current*	Uses the current of all the pins connected by the nearest flightline.
	*Automatically Connect Twigs*	Toggles the twig modes between ON and OFF. **Environment Variable**: weAutoTwigMode
	Allow Twig Connection	Sets the allowed direction of twigs to either the vertical direction, the horizontal direction, or both. This toolbar button is enabled only when the *Automatically Connect Twigs* option is set to on. **Environment Variable**: weAutoTwigDirection
	*Allow vertical and horizontal twigs*	Allows twigs to be inserted in both vertical and horizontal direction.
	*Allow vertical twigs*	Set the twigs in vertical direction.
	*Allow horizontal twigs*	Set the twigs in horizontal direction.
	Via Alignment	Controls the alignment and orientation of vias or via stacks created on each twig over the trunk. You can choose the via alignment depending upon the via direction mode that is selected. The *Via Alignment* button on the *SDR* toolbar is enabled only when *Automatically Connect Twigs* is ON. **Environment Variable**: weAutoTwigTrunkViaAlignment, weAutoTwigTrunkViaDirection
	*Via Alignment: Left Or Bottom* (For Twig Via Direction)	Creates vias or via stacks in the direction of the twig that is vertical and bottom-aligned to the trunk.
	*Via Alignment: Center* (For Twig Via Direction)	Creates vias or via stacks in the direction of the twig that is vertical and aligned to the trunk center. This is the default alignment mode.
	*Via Alignment: Right Or Top* (For Twig Via Direction)	Creates vias or via stacks in the direction of the twig that is vertical and top-aligned to the trunk.
	*Via Alignment: Left Or Bottom* (For Trunk Via Direction)	Creates vias or via stacks in the direction of the trunk that is horizontal and left-aligned to the twig.

	*Via Alignment: Center* (For Trunk Via Direction)	Creates vias or via stacks in the direction of the trunk that is horizontal and aligned to the twig center. This is the default alignment mode.
	*Via Alignment: Right Or Top* (For Trunk Via Direction)	Creates vias or via stacks in the direction of the trunk that is horizontal and right-aligned to the trunk.
	DI as Dataset	Sets the EAD dataset called `DI` with the design intent information. **Environment Variable**: weSdrDIDatasetName, weSdrDatasetNamingMethod
	*Import DI as Dataset*	Imports or updates the EAD dataset called `DI` with the design intent information in the layout view.
	*Sync and import DI as Dataset*	Syncs the DI high current information from the schematic to the layout and then converts it into a dataset.
	*Electrically Aware Pin Sizing*	Displays the Electrically Aware Pin Sizing form and the *Pin Tool – Pin Browser* window. The Electrically Aware Pin Sizing form lets you identify pins that do not meet electrical requirements and automatically fix the reported violations.
	*Options*	Displays the SDR Options form.
	*SDR Task Assistant*	Displays the SDR documentation in task assistant.

Related Topics

Introduction to Simulation Driven Routing

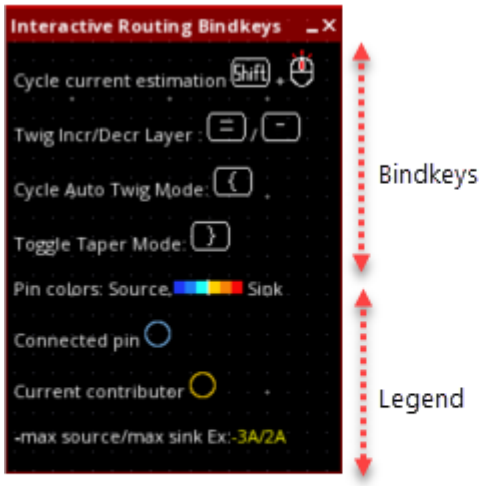
SDR Bindkeys and Visuals

Sources and Sinks Map

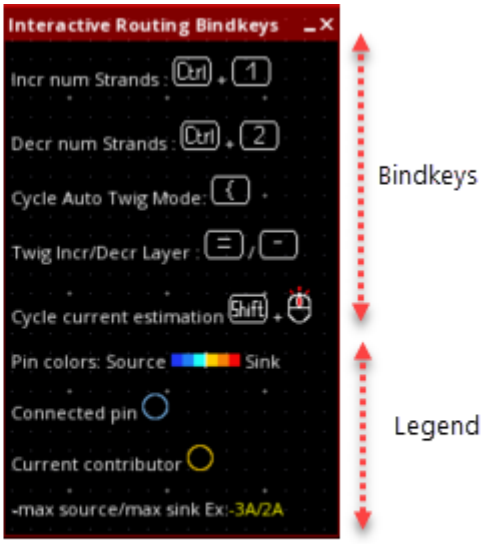
Electrically Aware Pin Sizing

SDR Bindkeys and Visuals

When you start the *Create Wire* command or the *Create Stranded Wire* command, the *Interactive Routing Bindkeys* info balloon is displayed at the top-left corner of the layout canvas. This info balloon shows specific bindkeys related to SDR and a legend to understand the visuals that get displayed in the layout canvas during simulation driven routing.



Bindkeys for the
Create Wire command

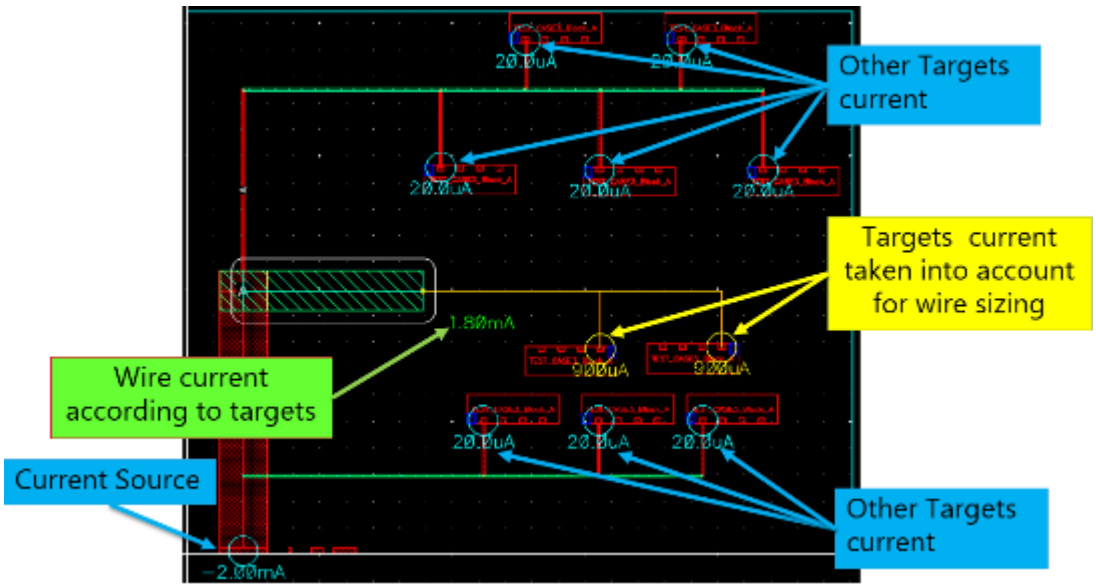


Bindkeys for the Create
Stranded Wire command

The following visuals are shown on the layout canvas during interactive routing:

- Thermal map representation
- Yellow and blue halos around connected pins and current contributors
- Estimated current value close to the cursor
- Connection style

Legend to show pin colors source and sink 



You can minimize the *Interactive Routing Bindkeys* info balloon and then maximize it when required. However, if you close the info balloon, it does not appear again. To view the info balloon again, go to another layout tier and then again open Layout EAD.

Related Topics

SDR Toolbar

Sources and Sinks Map

Sources and Sinks Map

The net connectivity provides information about the components that can be connected but not about the order in which they should be connected. The sources and sinks map representation and visualization helps in determining this order to create a net topology that is EM compliant.

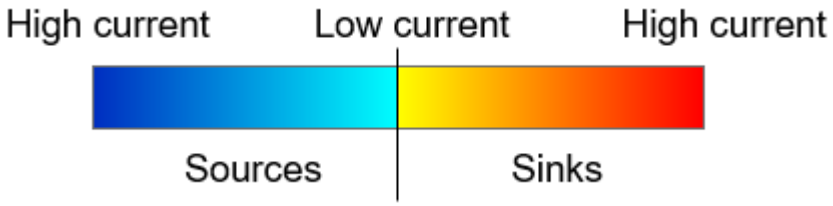
Use the *Display Sources and Sinks map* button on *SDR Toolbar* to:

- represent the current distribution as a thermal map by using color code.
- identify the individual pins or clusters that are the biggest current producers and consumers.

- view the estimated current value that is displayed on the screen over the devices or clusters.
- determine the connection style.
- identify the best net topology by connecting the major producers with fat wires to carry the current and then add small current producers.

The color code for the sources and sinks map is defined according to the current distribution. The color code uses the two gradients of three colors each, one for sources and the other for sinks.

- Red is for a sink with a high current (66% to 100%).
- Orange is for a sink with a medium current (33% to 66%).
- Yellow is for a sink with a low current (0% to 33%).
- Dark blue is for a source with a high current (66% to 100%).
- Blue is for a source with a medium current (33% to 66%).
- Cyan is for a source with a low current (0% to 33%)



Related Topics

Visualizing the Current Distribution Per Net

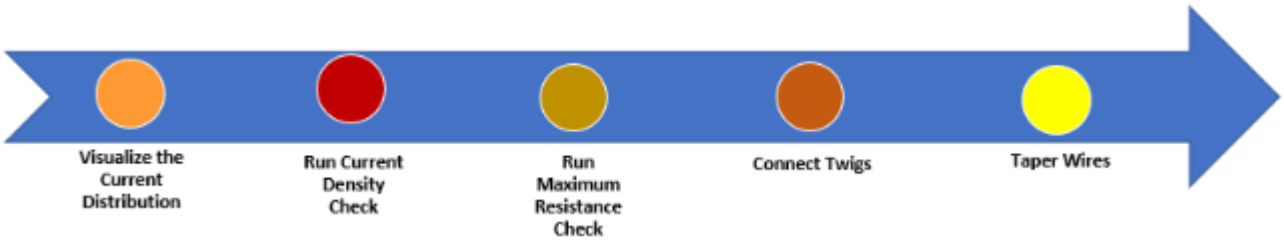
SDR Toolbar

SDR Bindkeys and Visuals

2

Performing Interactive SDR Checks

During interactive routing, you use SDR for two modes of routing checks, one based on the current density and the other on the maximum resistance. The following diagram depicts the steps for interactive SDR checks to avoid EM and `maxResistance` violations in the design while performing Simulation Driven Interactive Routing:



1. Visualize the Current Distribution.

   Identifies the net connections that either consume or generate the most current. See Visualizing the Current Distribution Per Net.

2. Run the Current Density Check.

   Estimates the current in the edited wire and lets you create a design with the appropriate width of the wire based on the estimated EM value. See Running Interactive SDR Current Density Checks.

3. Connect Twigs.

   Connects multiple pins automatically with the appropriate wires and vias. See Connecting Twigs Automatically.

4. Taper Wires.

   Adjusts the width of each segment independently. See Tapering in SDR.

Related Topics

Checker Modes

Current Estimation Modes

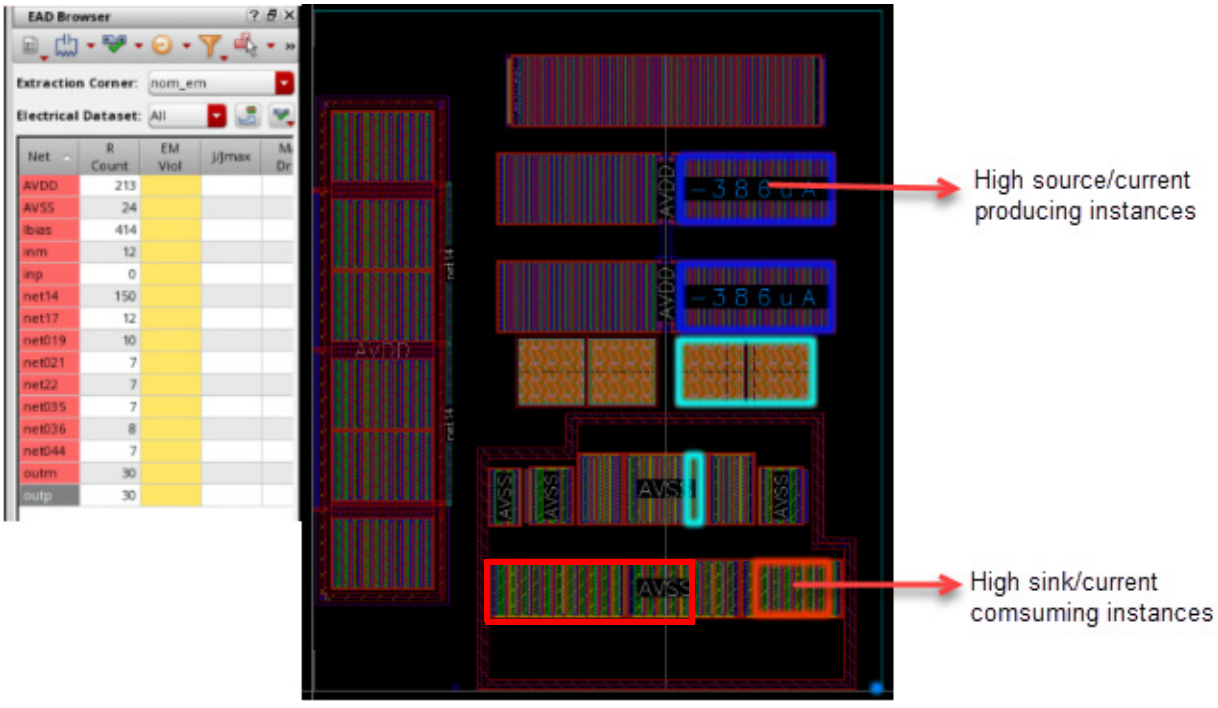
Visualizing the Current Distribution Per Net

Before starting interactive SDR routing, it is important to identify the net connections that either consume or generate the most current. Identifying the main current consumers and producers helps you to estimate the wire widths and vias cuts to create an EM-compliant topology.

To visualize the pins of a net according to their current distribution:

1. Select a net from the Navigator assistant or EAD Browser.
2. Click the *Hide Sources and Sinks Map* button on *SDR Toolbar*.

   The layout canvas displays the group of abutted instances of the `outp` net. The colored halo around the grouped instances indicates whether it is a current consumer or a producer. The pins or instance pins producing the current are shown with a blue halo, and the sinks, pins, or instance pins consuming the current are shown with a red halo. The dark blue and dark red indicate higher current, whereas, lighter colors indicate a lower current.

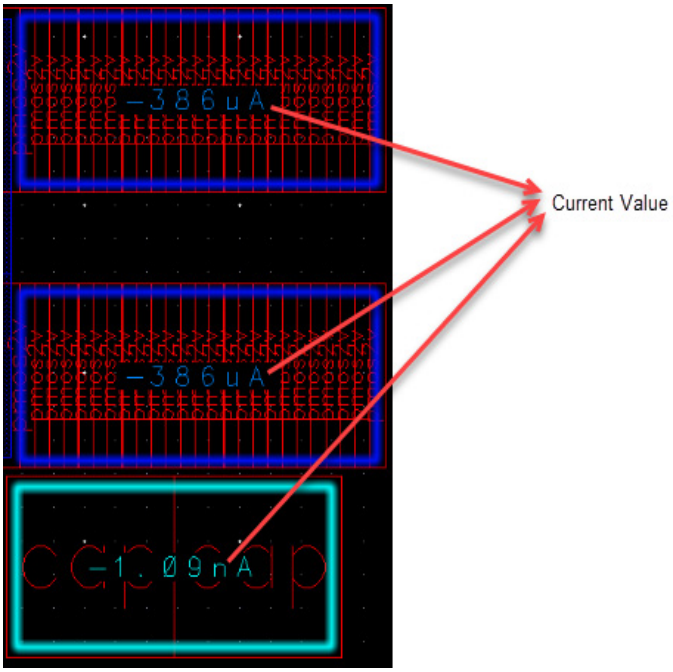


   The labels depicting the current values in source sink map are displayed according to the zoom level.

- At high altitude, the labels are not visible.
- At intermediate altitude, only single current value is displayed.

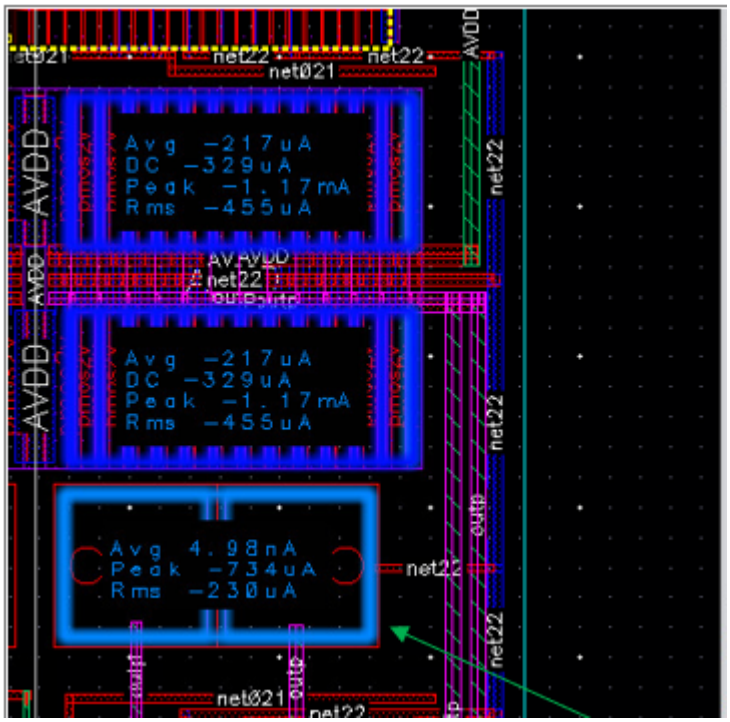
If the pin is big enough and the `weSdrSourceSinkMapAllCurrents` environment variable is set to `true`, all the current types (Avg/Peak/Rms) are displayed.

In case, you are unable to view the current values of the cluster, you can zoom in to the cluster and view the current value.



The source and sink map displays the maximum current value, which is highest value of each Avg/Peak/RMS, of the selected dataset. However, if multiple datasets are selected and they have different minimum and maximum values, the displayed currents are

irrespective of dataset and the maximum values of each current type AVG/PEAK/Rms from all available datasets is displayed.



Related Topics

Sources and Sinks Map

weSdrDisplayClusters

weSdrSourceSinkMap

weSdrSourceSinkMapAllCurrents

Checker Modes

You can specify a checker mode to be used for current and resistance estimation. The following checker modes are available: *Checker Mode: Enforce*, *Checker Mode: Notify, and Checker Mode: Off*. These are described as follows.

Environment variable: weSdrCheckMode

Using Enforce Checker Mode

The enforce mode estimates the current in the edited wire and vias and automatically calculates the wire width to avoid EM or `maxResistance` violations.

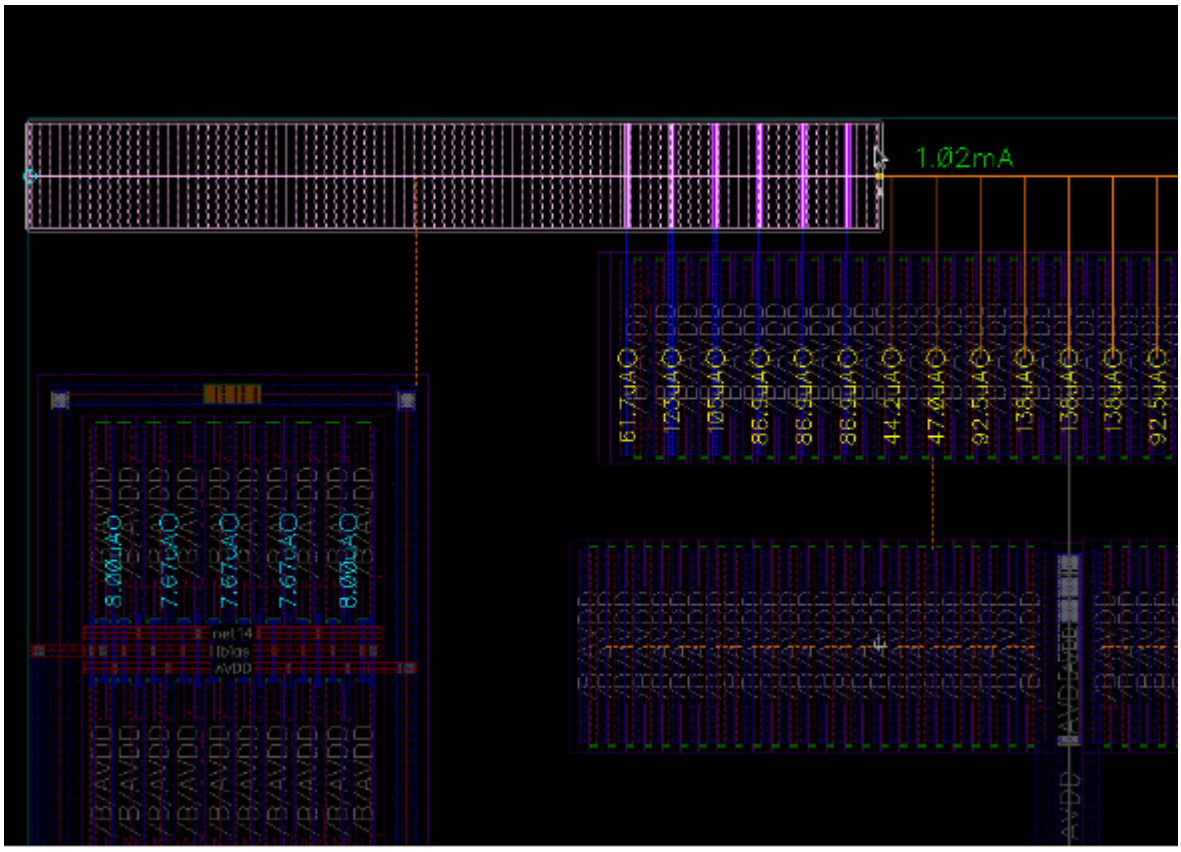
Lets see the difference in the *Enforce* checker mode when specified for current estimation using the Create Wire and Create Stranded Wire commands and for resistance estimation.

- Result of Enforce Mode Using the Create Wire Command

  The width of the wire and vias is automatically adjusted to avoid EM violations. The adjusted width respects the `minWidth` constraint and the width defined in the Wire Assistant. When starting from a pin, if the wire width required to avoid EM violations is smaller than the pin width and if the *Use Width – Tap Shapes and Pins Width* option in the *Create Wire* context-sensitive menu is enabled, then the wire matches the pin width. However, if the *Tap Shapes and Pins Width* option is disabled or the width required to avoid EM violations is larger than the pin width, then the estimated width is used.

  The estimated current of the wire is displayed using the color coding based on the color settings for EAD EM violations. The color is computed to represent the EM violation that

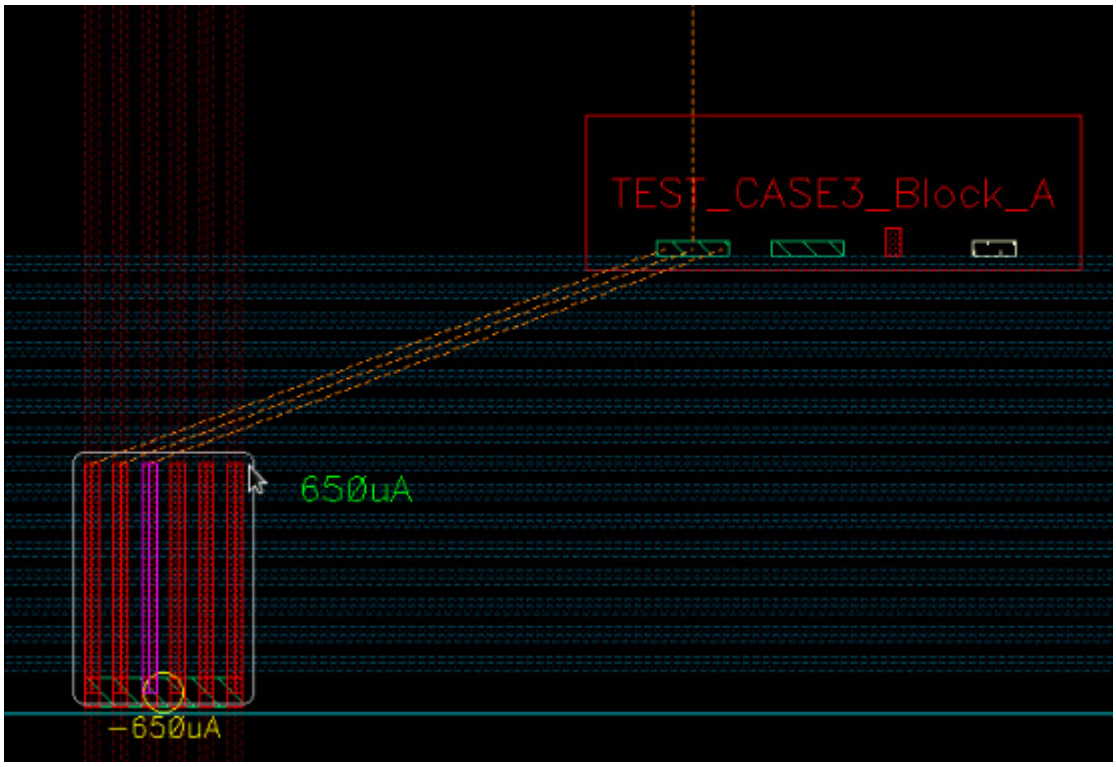
will be reported by the EAD checker after routing. The text in green in the following figure indicates that there is no EM violation and the current density check is satisfied.



- Result of Enforce Mode Using the Create Stranded Wire Command

  The number of stranded wires is automatically adjusted to avoid EM violations. The estimated current of the wire is displayed using the color coding based on the color settings for EAD EM violations. The color is computed to represent the EM violation that

is reported by the EAD checker after routing. The text in green, in the following figure, indicates that there is no EM violation and the current density check is satisfied.



If you change the number of stranded wires, then the checker mode is automatically updated to Notify. Also, the label color of the estimated current of the stranded wires is changed. The label color of the estimated current indicates that the number of stranded wires is either more or less than required.

Using Notify Checker Mode

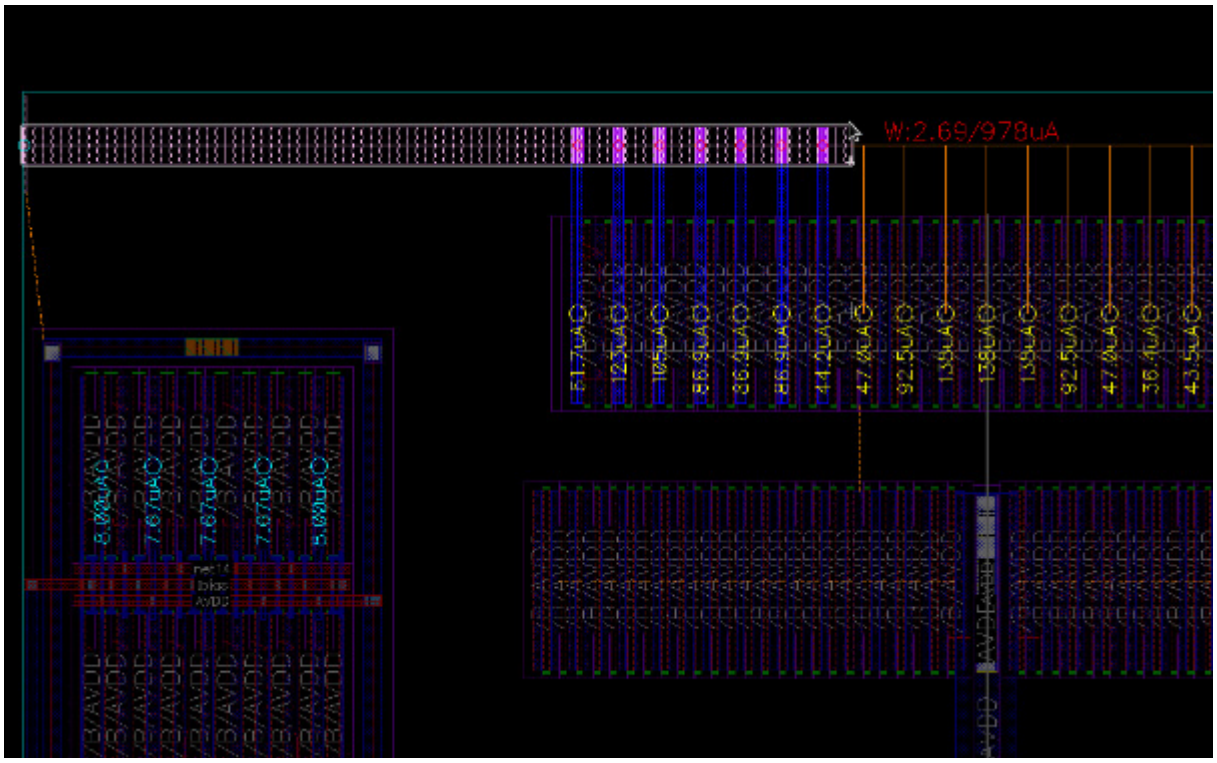
The notify mode estimates the current in the edited wire using a color coding based on EAD EM violations color settings.

Lets see the difference in the *Notify* checker mode when specified for current estimation and when specified for resistance estimation.

- Result of Notify Mode Using the Create Wire Command

  The default width of the wire is considered. You can also specify the width of the wire in the Create Wire form. The wire and vias provide the feedback using the label color, which displays the estimated current in the edited wire. Also, note that the width of the wire and

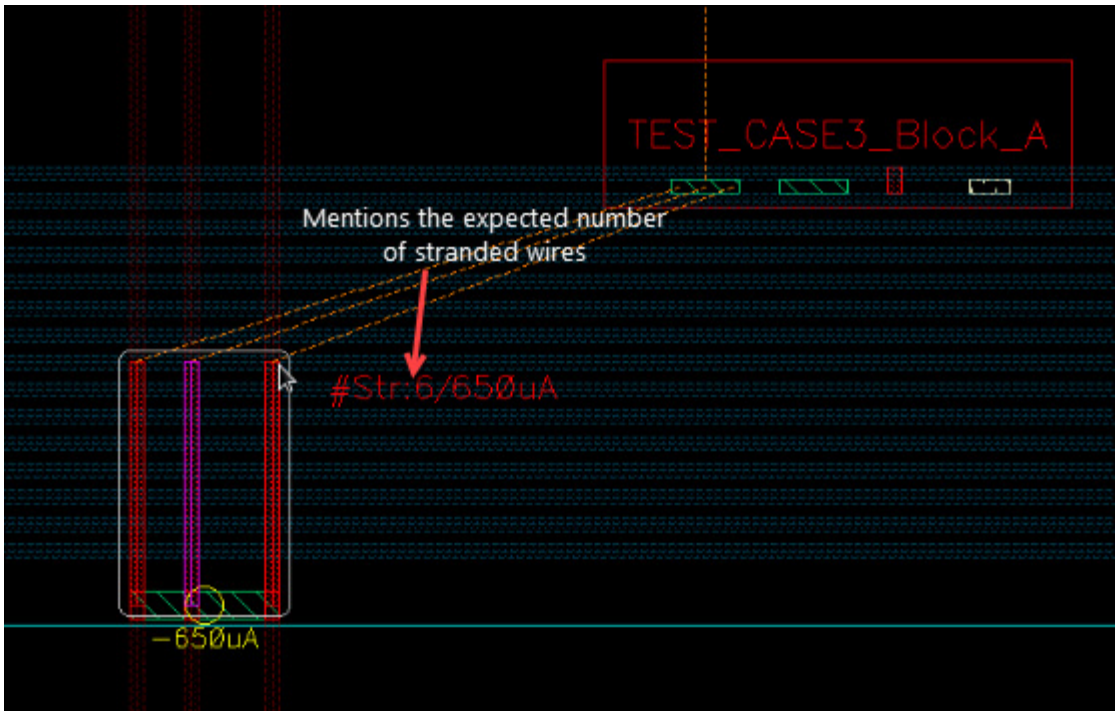
via is not updated automatically and the EM violation can continue to exist. The text in red reports an EM violation.



- Result of Notify Mode Using the Create Stranded Wire Command

  The required number of stranded wires are displayed without changing the number of stranded wires. Because the number of stranded wires is not changed automatically, the EM violation continues to exist. You can specify the number of stranded wires in the Create Stranded Wire form. The stranded wires provide the feedback using the label

color, which displays the required number of stranded wires for the estimated current. The text in red reports an EM violation.



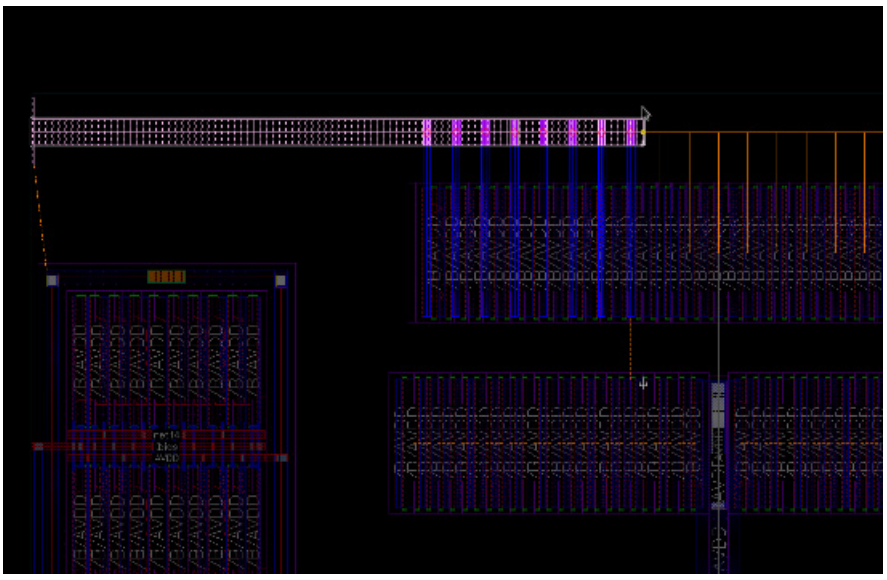
Using Off Checker Mode

The *Off* mode does not update the wire width and there is no feedback on the estimated current or resistance.

Lets see the difference in the *Off* checker mode when specified for current estimation using the Create Wire and Create Stranded Wire commands, and for resistance estimation.

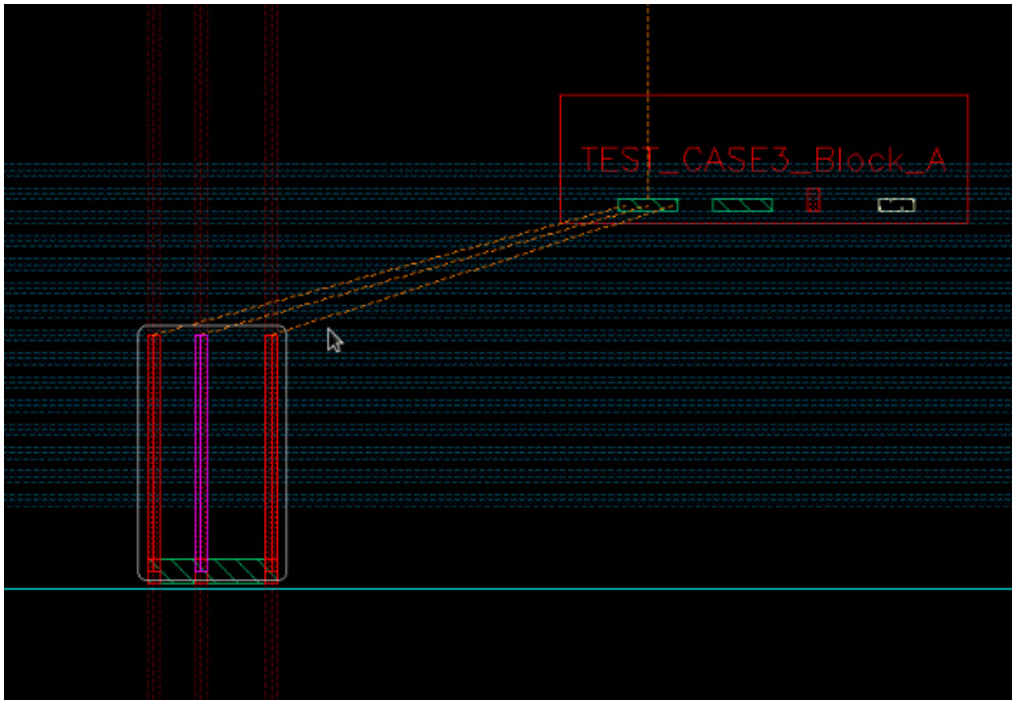
- Result of Off Mode Using the Create Wire Command

The Create Wire does not update the width nor does it provide any current feedback.



- Result of Off Mode Using the Create Stranded Wire Command

  The number of stranded wires is not updated, nor does it provide any current feedback.



Related Topics

Running Interactive SDR Current Density Checks

Current Estimation Modes

Current Estimation Modes

You can control how create wire estimates the current in the edited pathSeg and via. The following current estimation modes are available: *Auto*, *Sum Connected Pins Currents*, *Maintain Constant Current*, *Nearest Island Current*. These are described as follows:

Environment variable: weSdrCurrentEstimationMode

Using Auto Current Estimation Mode

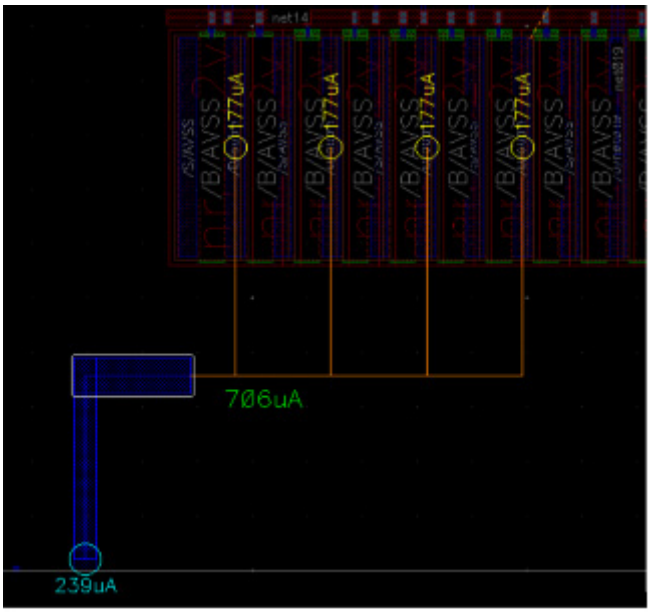
Estimates the current automatically in the last section of the wire according to the connected objects and flightline targets. Lets see how the difference in the result of current estimation using the *Create Wire* and *Create Stranded Wire* commands.

- Current Estimation Using the Create Wire Command

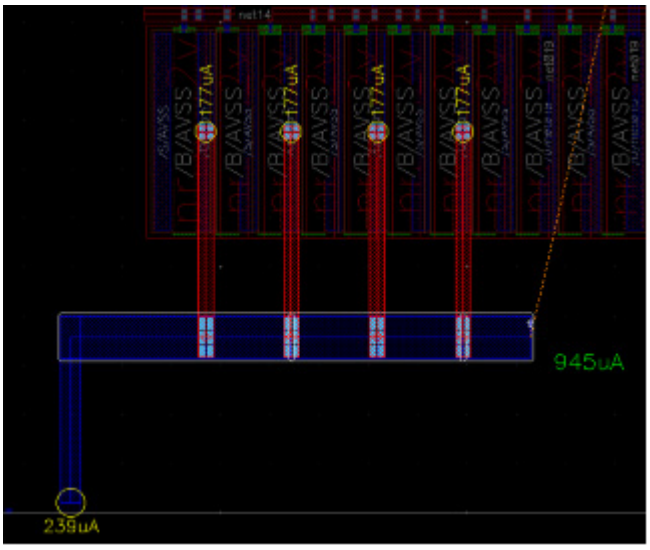
  Calculates the current based on all the connected target pins identified in the channel separating two rows of instances or `nmos` and `pmos` devices.

  When entering a channel, the flightlines are displayed to show all the identified targets in the channel. The current estimation is based on the sum of all targets, for example, `706uA = 4 * 177uA`, assuming the wire is routed only until the last target and then restart the routing from the bend. The following figure shows how the wire width is adjusted when the current estimation mode is *Automatically estimate current* and the

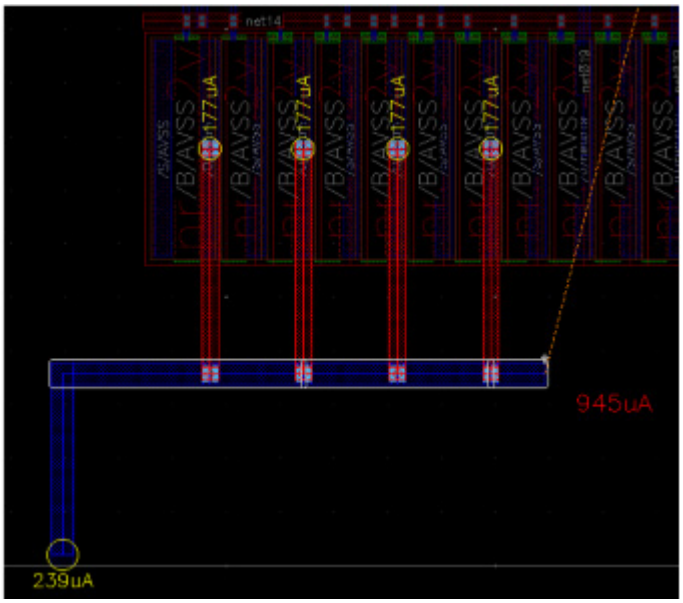
checker mode is *Enforce*. The pins used to estimate the current and wire size are highlighted in yellow (current contributors).



When passing through the last target of the channel, the current estimation is based on the sum of all connected pins, for example, `945uA = 4 * 177uA + 239uA`. The wire transfers all the current attached to the wire to the right.



Similarly, the following figure shows the adjustment of the wire width when the checker mode is selected as *Notify*. The *945uA* label is red to indicate that the wire is undersized compared to the estimated current.

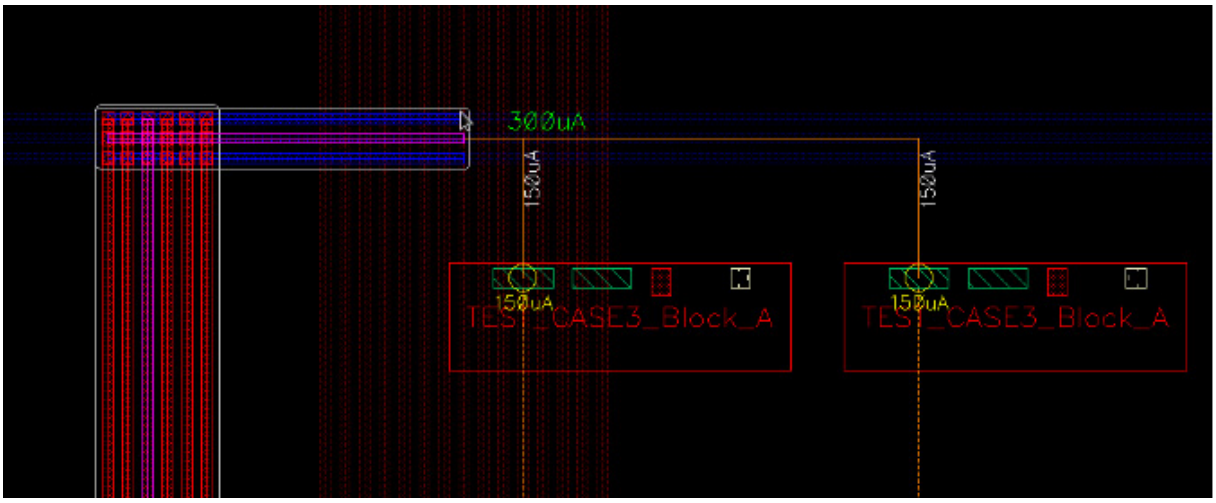


- Current Estimation Using the Create Stranded Wire Command

  Calculates the current based on all the connected target pins identified in the channel separating two rows of instances. When entering a channel, the flightlines are displayed to show all the identified targets in the channel. The current estimation is based on the sum of all targets, for example, `300uA = 2 * 150uA`, assuming the wire is routed only until the last target and then restart the routing from the bend.

  The following figure shows how the number of stranded wires is adjusted when the current estimation mode is specified as *Auto* and the checker mode is *Enforce*. The pins

that are used for current estimation and for adjusting the number of strands are highlighted in yellow (current contributors).



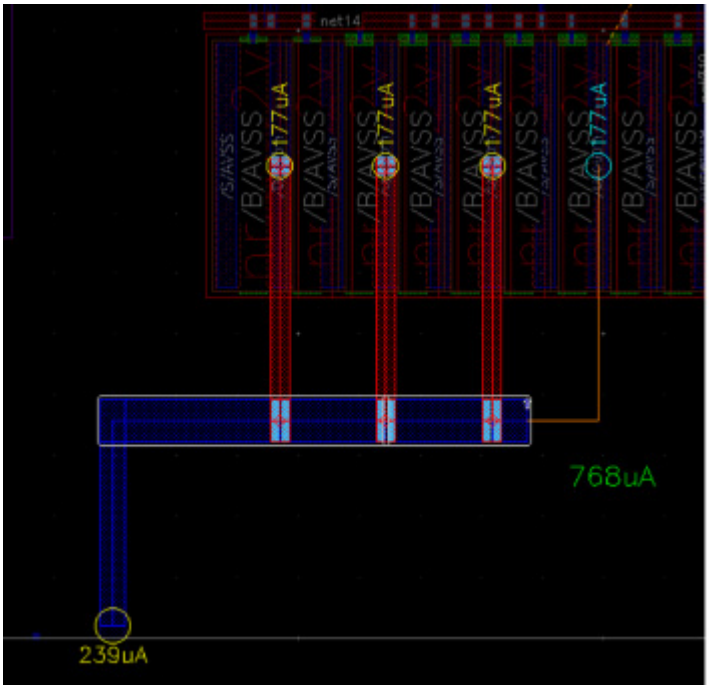
Using Sum Connected Pins Currents Estimation Mode

Estimates the current after adding the current of all the connected pins. Lets see how the difference in the result of current estimation using the *Create Wire* and *Create Stranded Wire* commands.

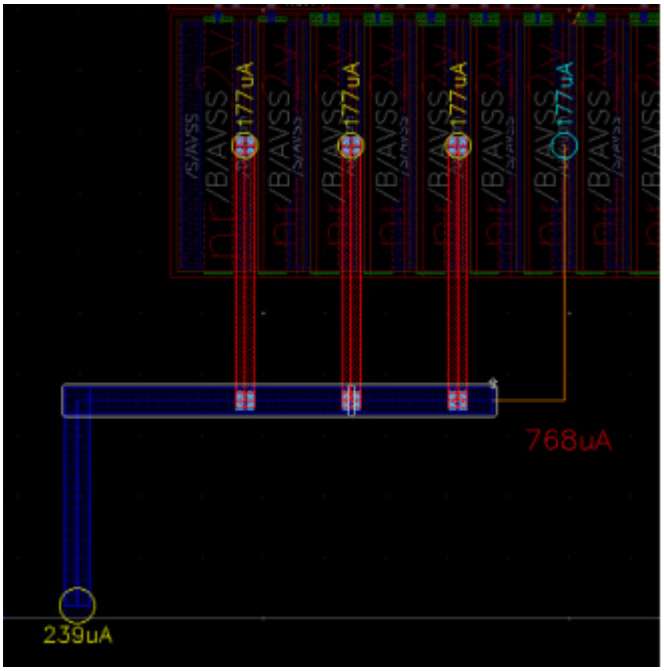
- Current Estimation Using the Create Wire Command

  Keeps the wire width based on the estimated current of all the pins already connected to the wire. The current at the end of the last segment is the sum of the current for all connected pins. For example, `768uA = 239uA + 3 * 177uA`. The following figure shows how the wire width is adjusted when the current estimation mode is *Sum*

connected pins current and the checker mode is *Enforce*. The pins used to estimate the current and wire size are highlighted in yellow (current contributors).



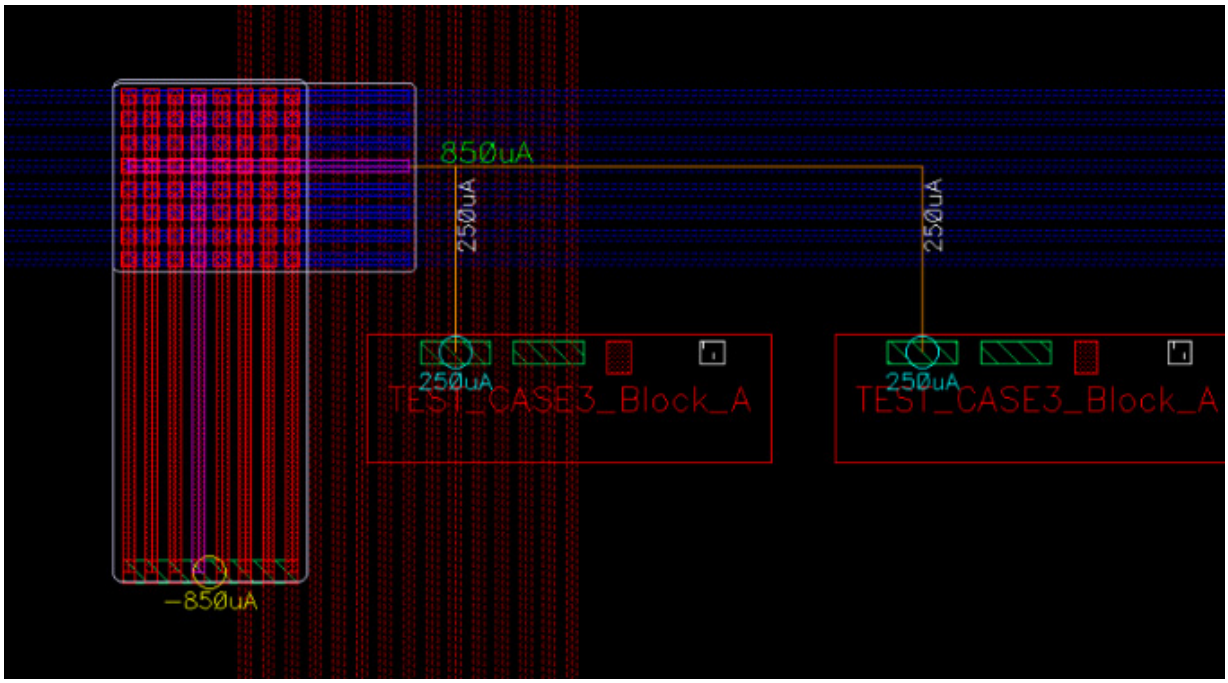
Similarly, the following figure shows the adjustment of the wire width when the checker mode is selected as *Notify*. The *768uA* label is red to indicate that the wire is undersized compared to the estimated current.



- Current Estimation Using the Create Stranded Wire Command

  Keeps the number of stranded wires based on the estimated current of all the pins already connected to the wire. The current at the end of the last segment is the sum of current for all the connected pins. For example, `650uA`, if there is only one pin producing `650uA` of current connected to the wire.

  The following figure shows how the number of stranded wires is adjusted when the current estimation mode is *Sum Connected Pins Current* and the checker mode is *Enforce*. The pins used to estimate the current and the number of stranded wires are highlighted in yellow (current contributors).



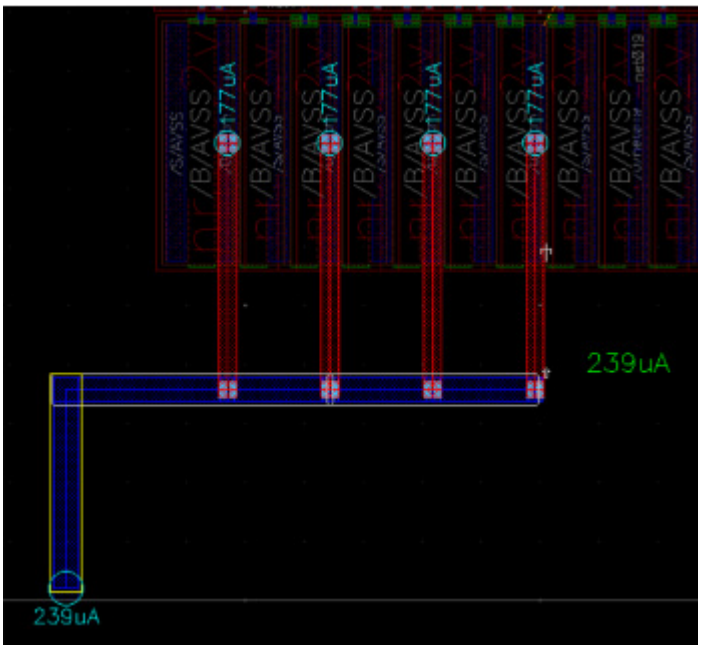
Using Maintain Constant Current Estimation Mode

Estimates the current after inheriting the current from the previous wire. Lets see the difference in the result of current estimation using the *Create Wire* and *Create Stranded Wire* commands.

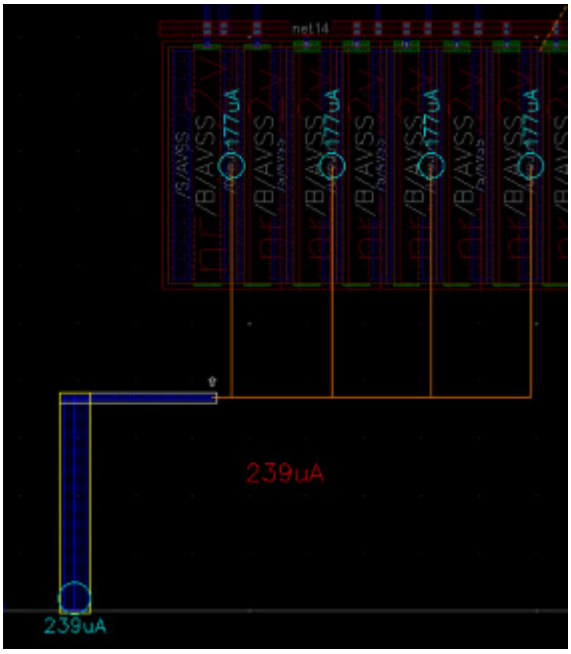
- Current Estimation Using the Create Wire Command

  Adjusts the wire width based on the current of the previous segment. The vertical pathSeg is created with the current of the connected pin. Therefore, the width of the pathSeg is adjusted for `239uA`. The horizontal pathSeg is also adjusted to maintain the same current as the vertical pathSeg, which is `239uA`. The following figure shows how

the wire width is adjusted when the current estimation mode is *Maintain Constant Current* and the checker mode is *Enforce*.

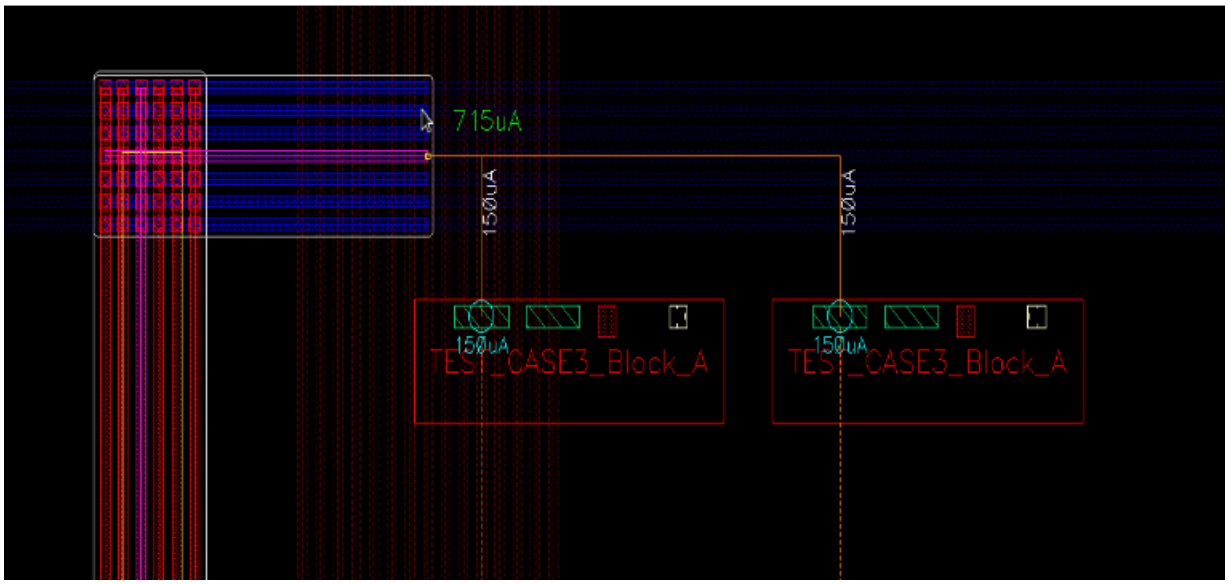


Similarly, the following figure shows the adjustment of the wire width when the checker mode is selected as *Notify*.



- Current Estimation Using the Create Stranded Wire Command

Adjusts the number of stranded wires based on the current of the previous segments. The following figure shows how the number of stranded wires is adjusted when the current estimation mode is *Maintain Constant Current* and the checker mode is *Enforce*. The vertical stranded wires in red gets created for a current of `715uA`. Thus, the stranded wires in blue are adjusted for the same amount of current, which is `715uA`.

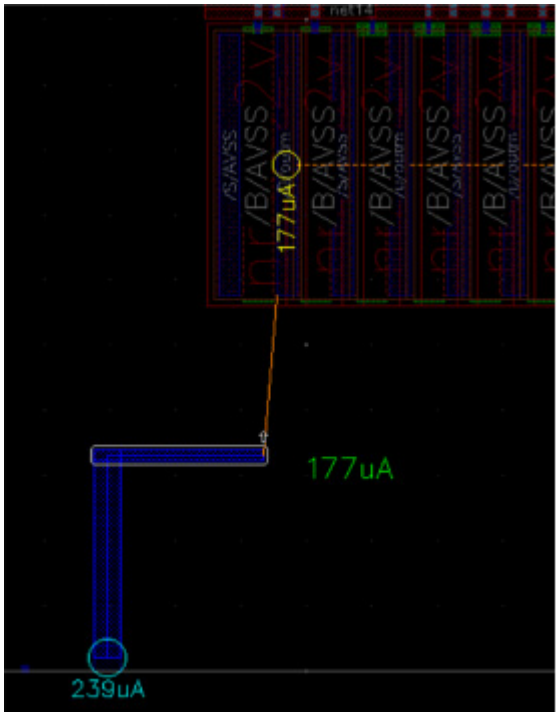


Using Nearest Island Current Estimation Mode

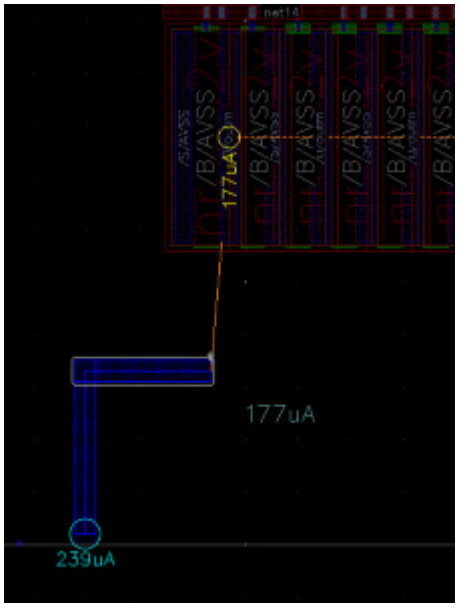
Estimates the current based on all the pins connected by the flightline. Lets see the difference in the result of current estimation using the *Create Wire* and *Create Stranded Wire* commands.

- Current Estimation Using the Create Wire Command

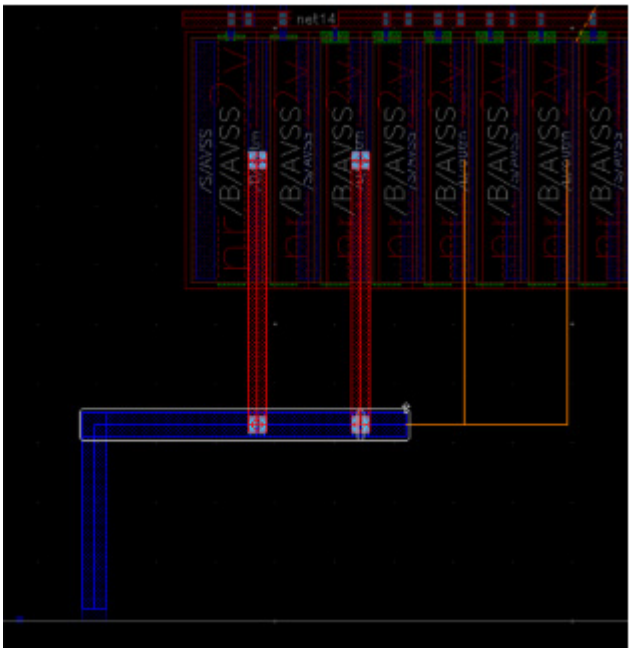
Current of the closest pin is considered to estimate the width of the wire. The following figure shows how the wire width is adjusted when the current estimation mode is *Nearest Island Current* and the checker mode is *Enforce*.



Similarly, the following figure shows the adjustment of the wire width when the checker mode is selected as *Notify*. The blue label indicates that the last wire segment is oversized compared to the target current (177uA).



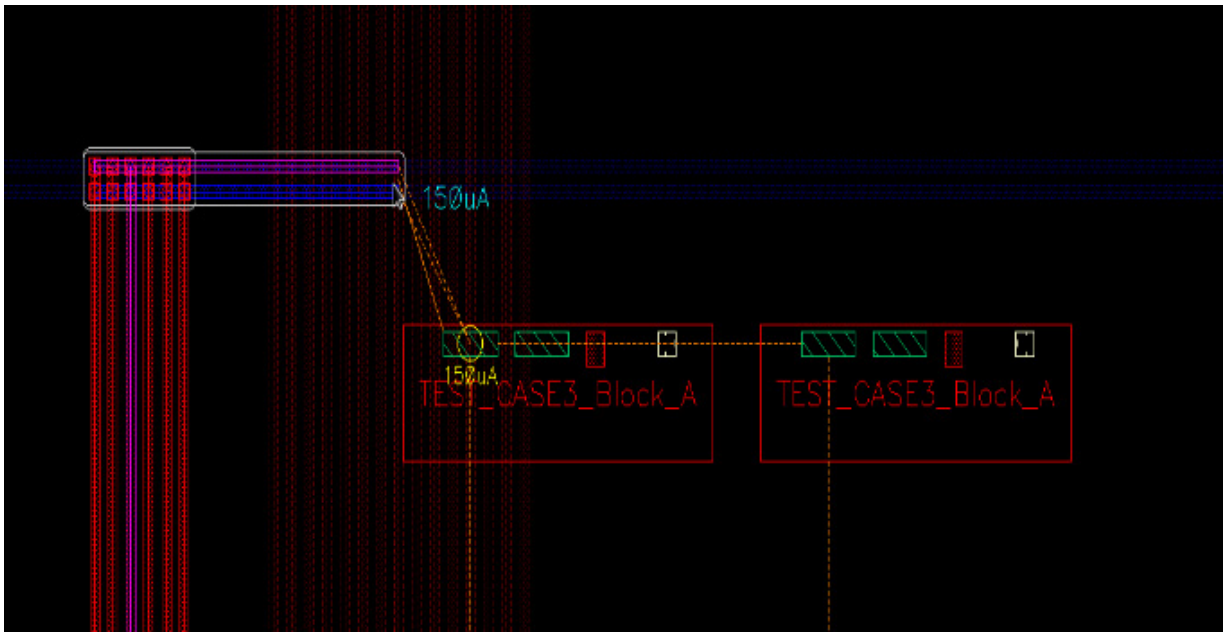
The following figure shows the wire width and the routing when the checker mode is selected as *Off*.



- Current Estimation Using the Create Stranded Wire Command

  Calculates the number of stranded wires based on the current of the closest pin. The following figure shows how the number of stranded wires is adjusted when the current

estimation mode is *Nearest Island Current* and the checker mode is *Enforce*, and the wire is targeting one pin at `150uA`.



Related Topics

Running Interactive SDR Current Density Checks

Connecting Twigs Automatically

Running Interactive SDR Current Density Checks

Based on the simulation results, Create Wire estimates the current in the edited wire and vias according to the EAD settings, such as dataset, temperature, and current scaling. The current density check lets you create a design with the appropriate width of the wire based on the estimated EM value.

To run the current density check:

1. Select a net in the Navigator assistant or the EAD browser.
2. Select `Metal2` as the active layer in the Palette.
3. To control the wire size and the number of strands compared to the estimated width required to avoid EM violations, specify a value in the *Width Multiplier* text box on the SDR toolbar. For example, if you specify the width multiplier as 2.0, it means that the

stranded width is multiplied by 2 compared to the estimated width to reach 100% of EM. Either the width of each strand is increased or the number of strands are increased.

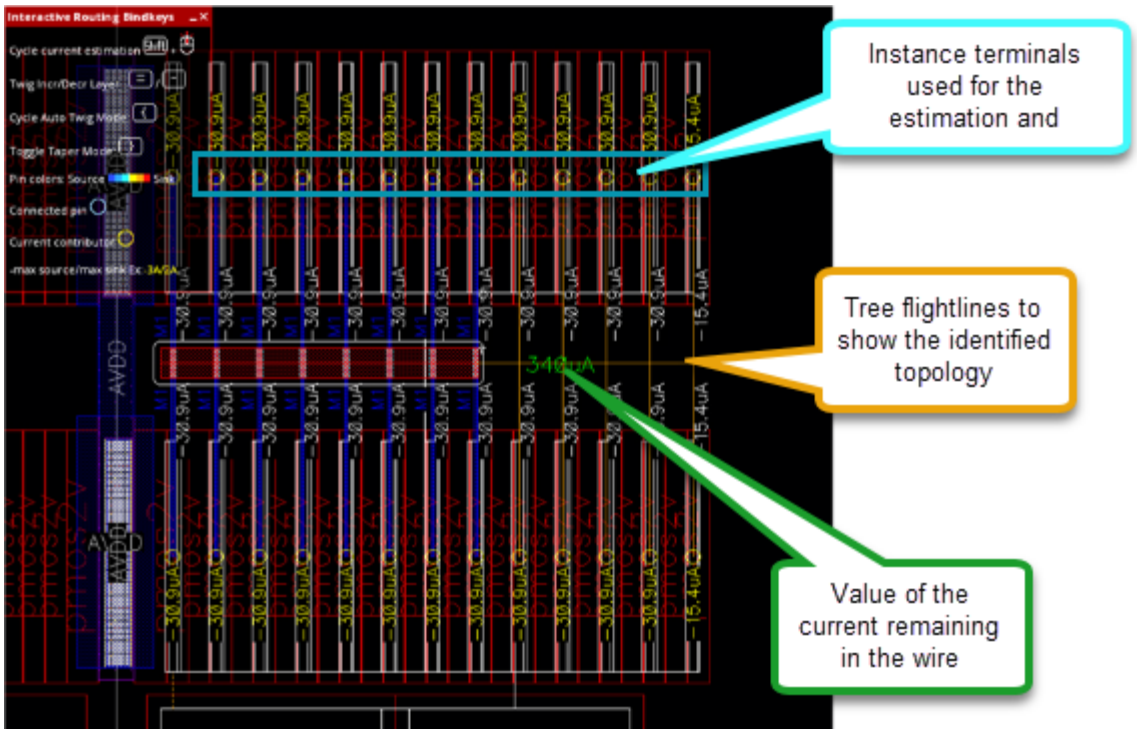
Environment variable: weSdrWidthMultiplier

4. Start to create a wire or a stranded wire from the selected pin or instance.

   a. *Choose Create – Wiring – Wire* or press `P`.

   b. *Choose Create – Wiring – Create Stranded Wire* or press `Ctrl + Shift + S`.

   When you start the *Create Wire* or the *Create Stranded Wire* command, the wires and vias interactively connect the pins and automatically estimates the current between the two pins.



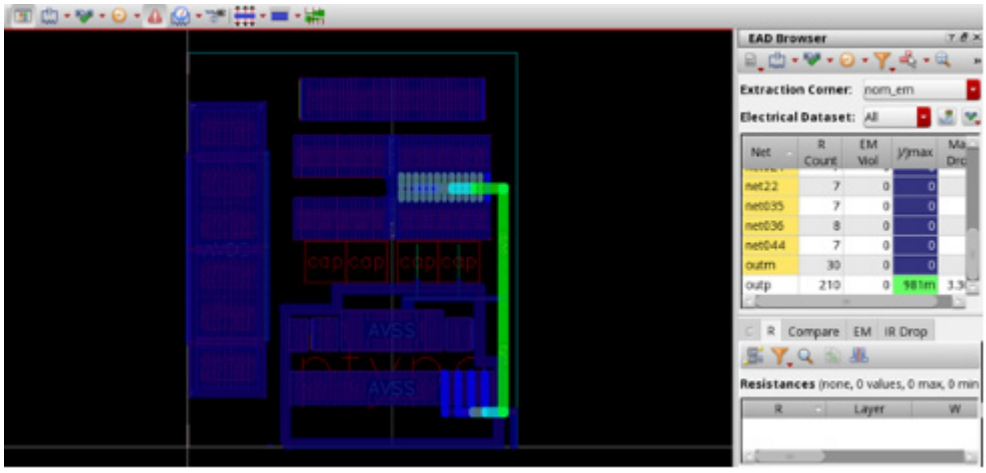
   Based on the estimated current, the wire width and number of via cuts are automatically resized so that the estimated current of the wire is below the value of the final current of the wire.

5. Select a checker mode from the *Checker Mode* drop-down list. You can select one of the following checker modes: *Checker Mode: Enforce*, *Checker Mode: Notify,* and *Checker Mode: Off.*

   Environment variable: weSdrCheckMode

6. To control the current estimation of wire, select a current estimation mode. You can select one of the following current estimation mode: *Auto*, *Sum Connected Pins Current*, *Maintain Constant Current*, and *Nearest Island Current*.
7. To complete interactive routing, press `Enter`.
8. Run EM checks as follows.
   a. Click the *Extract Parasitics* button on *EAD Toolbar*.
   b. Click the *Run EM Checks* button on *EAD Toolbar*.
9. To display the EM violations, click *Highlight EM violations* button on *EM* tab of the *Details* pane of *EAD Browser*.

   This completes simulation-driven routing and EM checks for the design.



Related Topics

Connecting Twigs Automatically

Current Estimation Modes

Connecting Twigs Automatically

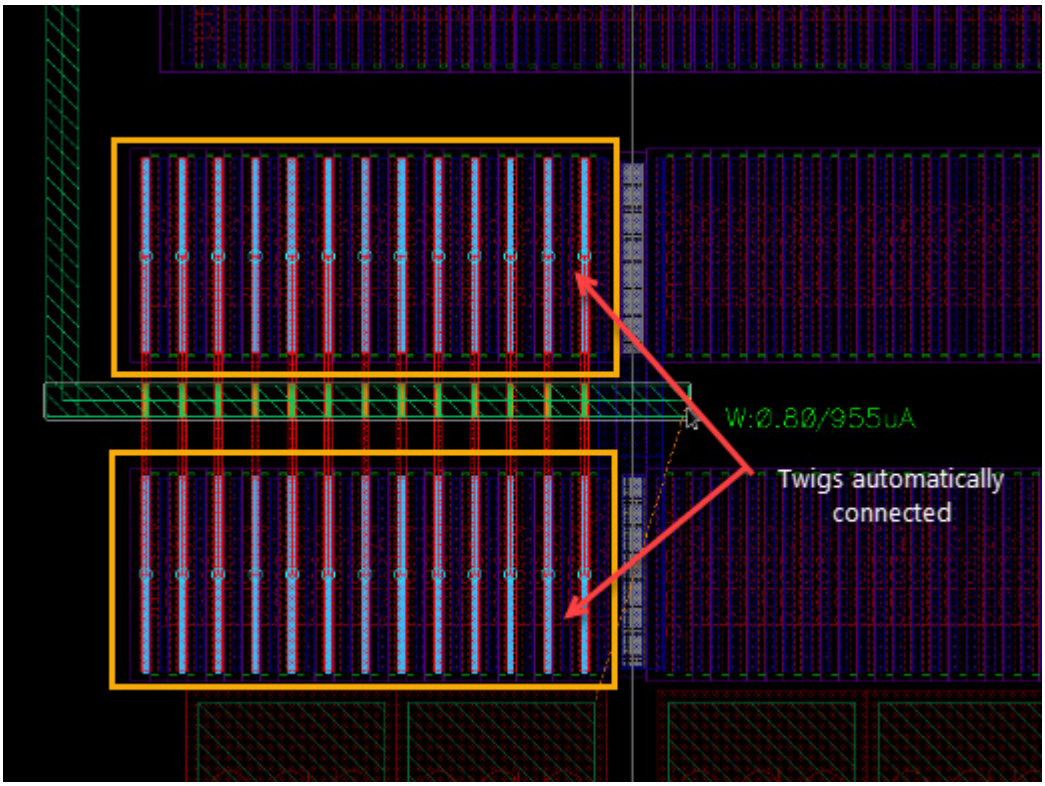
In interactive SDR, you can identify the device pins (target pins identified by the flightlines) and automatically connect multiple pins with the appropriate wires and vias. The twig (wires from the device to the main wire) layer is automatically defined but can be changed. The automatic twigs connection is independent of the current estimation mode and the checker

modes. Also, the number of cuts in the vias and the width of the twigs are automatically adjusted.

To automatically connect twigs:

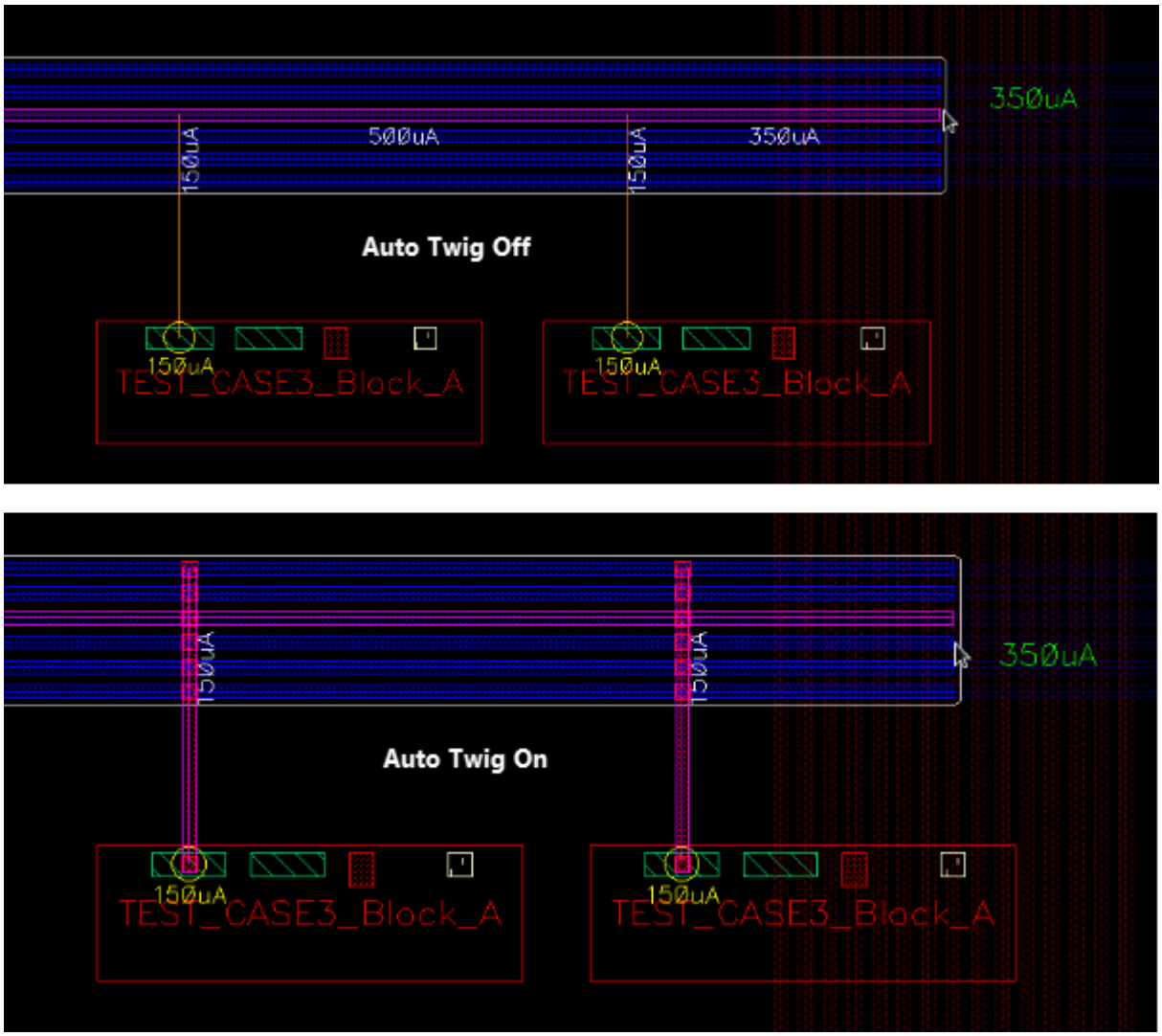
1. Select a net in the Navigator assistant or EAD Browser.
2. Select a source pin of the selected net in the layout design.
3. Choose *Create – Wiring – Wire*.
4. On *SDR Toolbar*, click the *Automatically Connect Twigs* button. The status of the button is modified to *Automatically Connect Twigs: ON*.

   When the mouse pointer moves close to an object on the same net, the twig connections are automatically created.



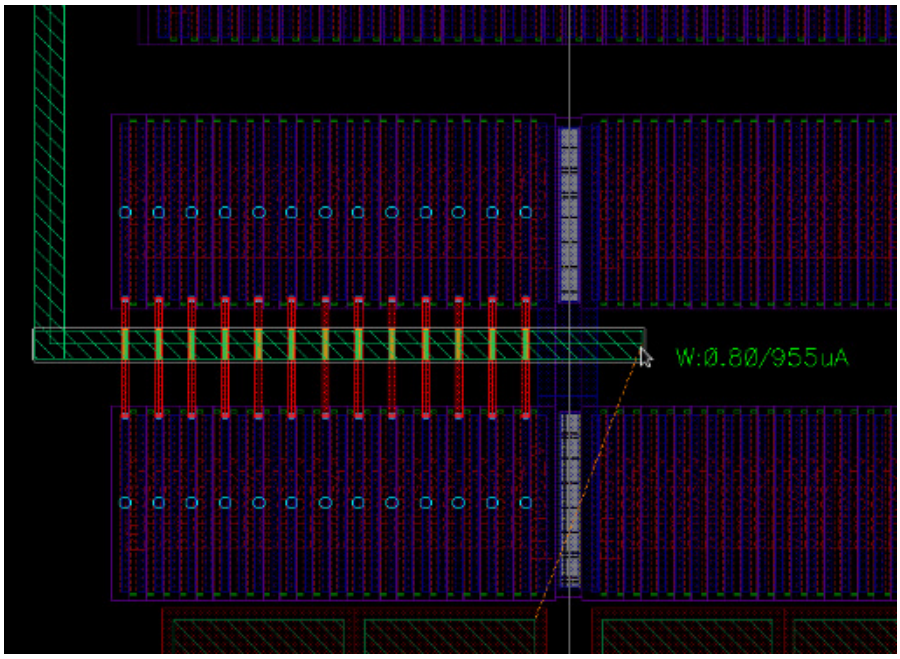
By default, twigs are created on the layer above the layer on which the wire is created. However, you can change the twig layer anytime while creating the wire (See step 9).

The following figure shows how twigs are automatically created when the *Create Stranded Wire* command is enabled.

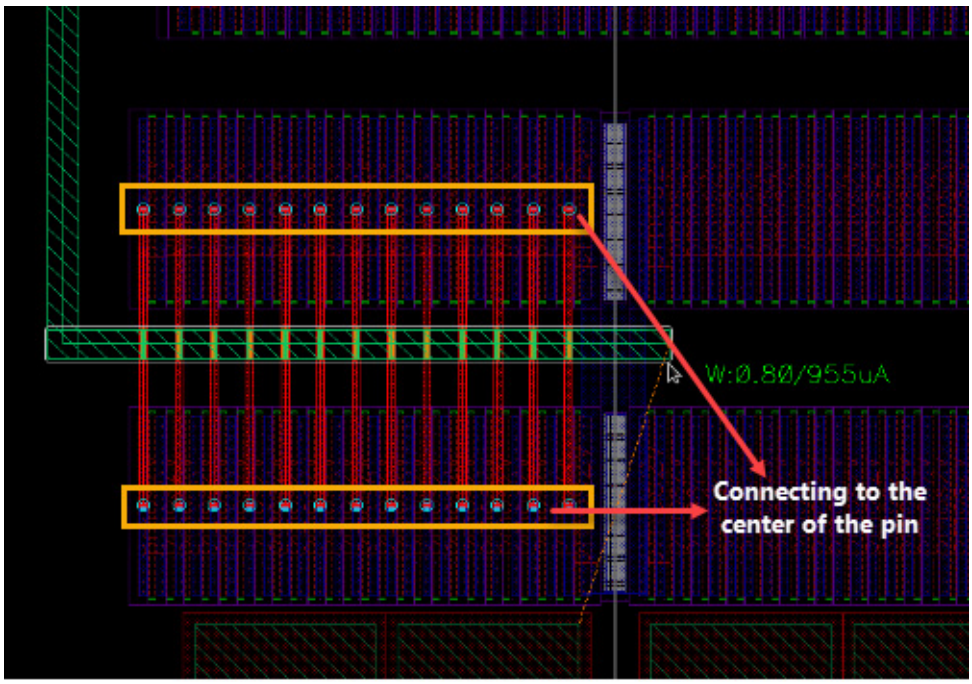


5. Press `F3` to open the Create Wire form.

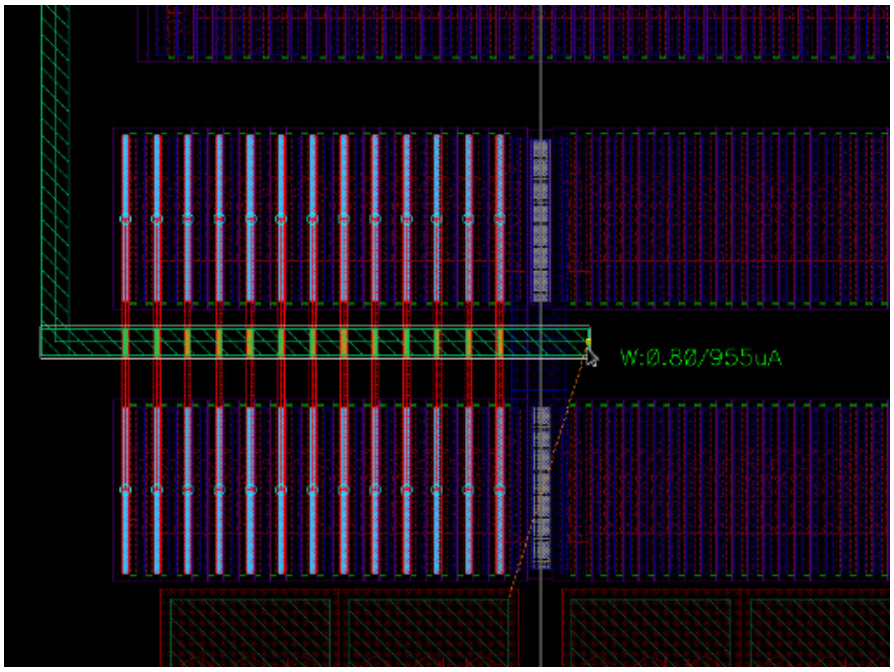
By default, the *Snap to Pin Center* and the *Cover Pin* options are deselected in the Create Wire form. In this case, the twigs are automatically created and snapped to the edge of the pin.



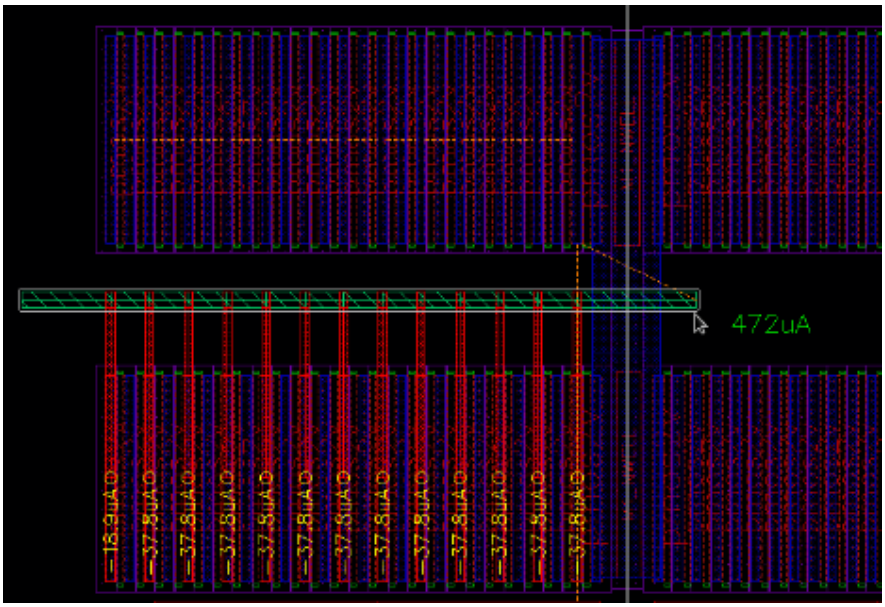
6. Select the *Snap to Pin Center* option in the Create Wire form. This automatically connects the twigs to the center of the pin of the created wire segment.



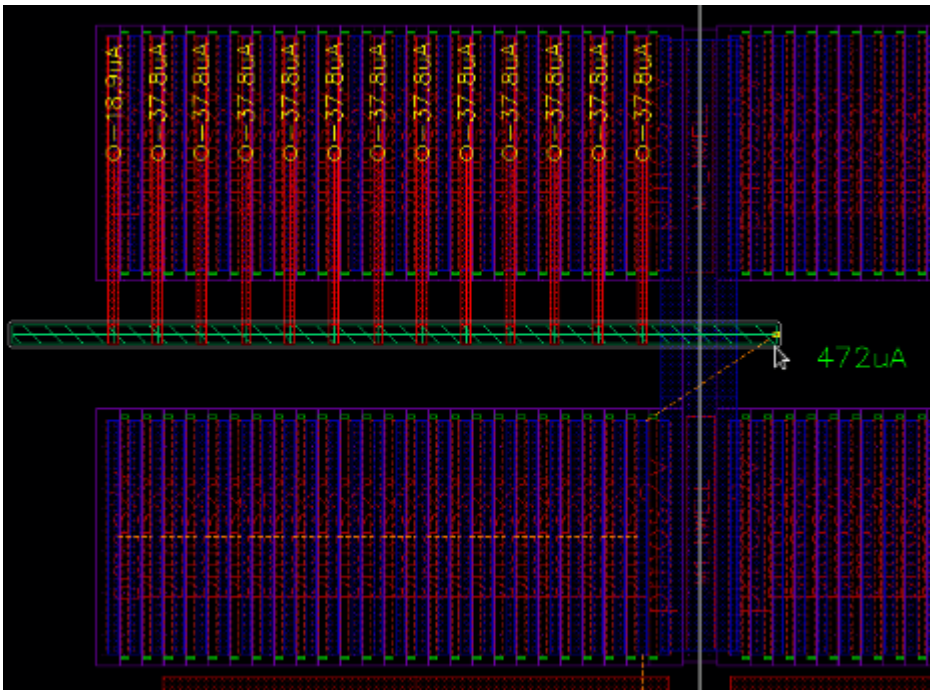
7. Select the *Cover Pin* option. This adjusts the twigs to automatically cover the source and target pins of the wire segment.



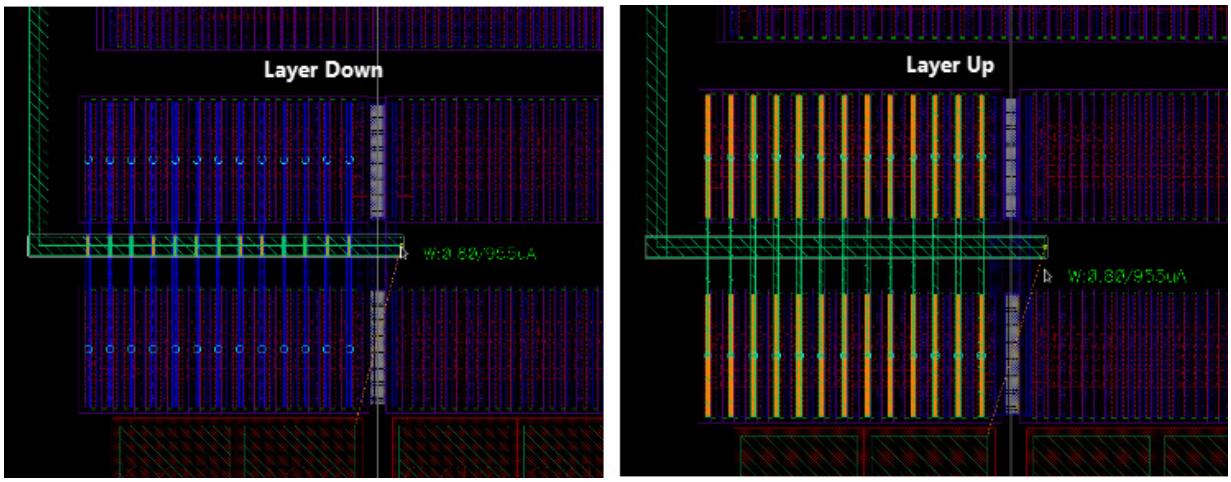
8. (Optional) Press the { bindkey to automatically connect the twigs to the top-left or bottom-right targets of the wire. By default, the twigs are automatically connected to all targets. The following figure shows the automatic twig connection to the bottom targets.



Similarly, the following figure shows the automatic twig connection to the top targets.

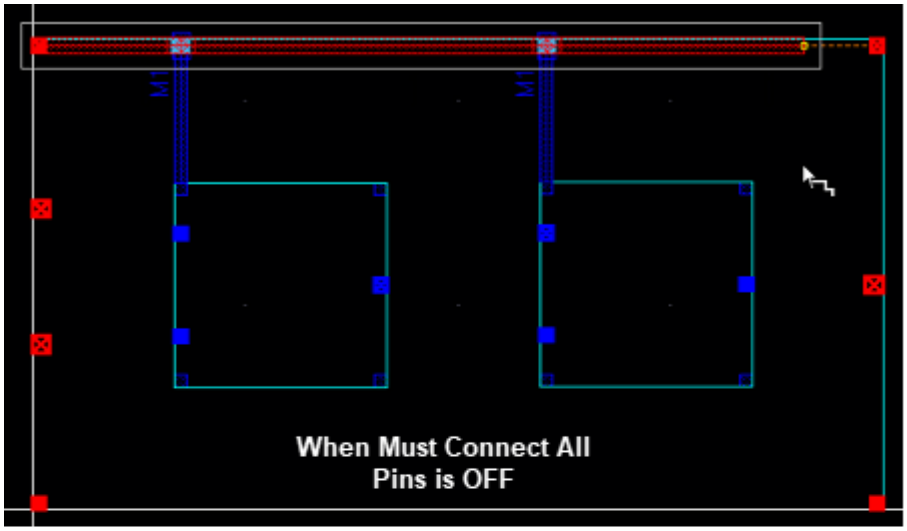


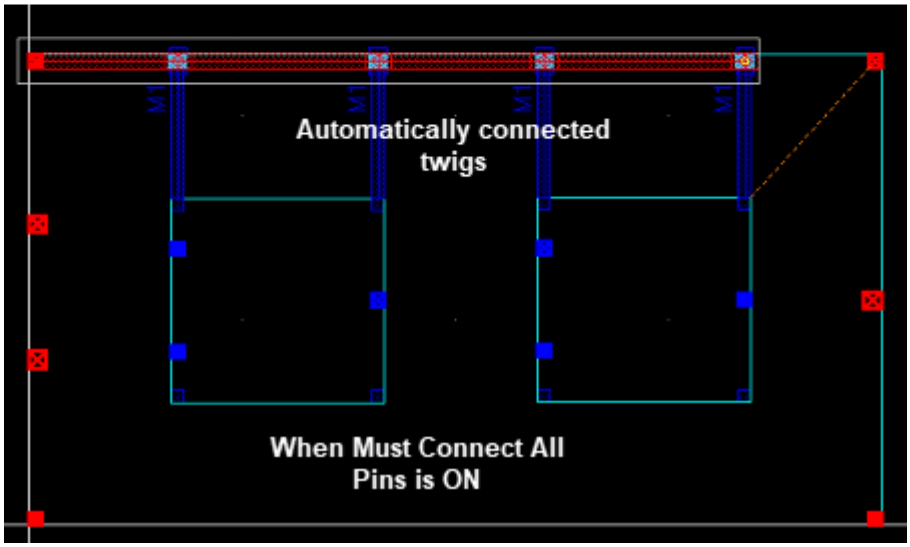
9. Modify the twig layer by pressing the = key to move the twig layer up or the - key to move the twig layer down. When the twig layer is changed, the vias are automatically created on the wire being created.



10. To finish the connecting the twigs automatically, press `Enter` or click the last twig connection.

In addition, the *Create Wire* command identifies the *Must Connect All Pins* terminal instances and automatically makes twig connections for all the pins that must be connected.





Related Topics

weAutoTwigMode

weAutoTwigTrunkViaMode

weAutoTwigMeshRoutingEnabled

Automatic Twig Mesh Routing

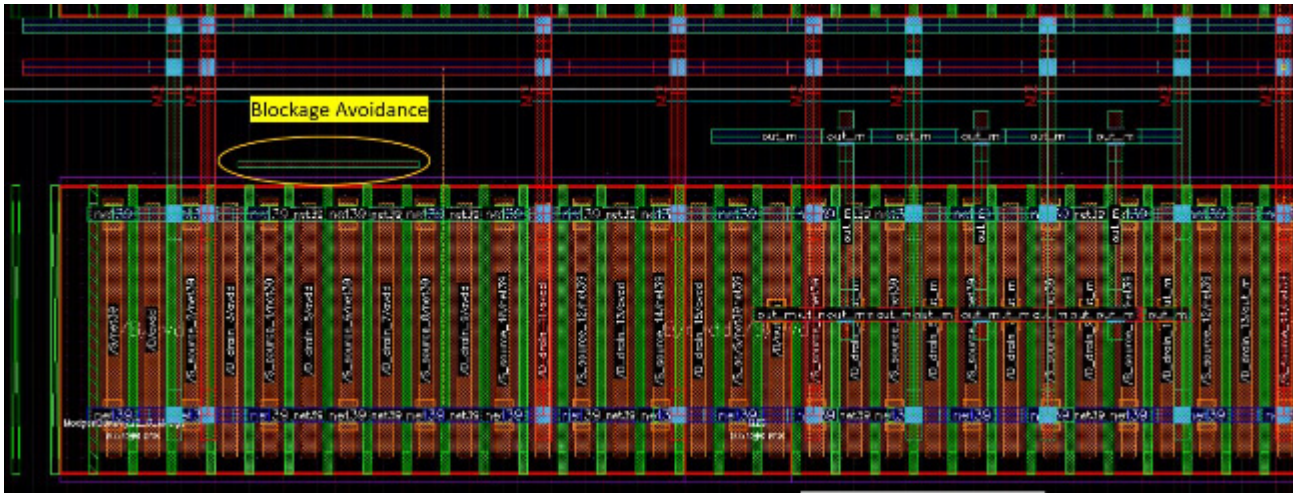
SDR Toolbar

Automatic Twig Mesh Routing

Automatic twig mesh routing lets you automatically connect a new wire to a parallel or orthogonal pre-routed wire segment. The pre-routed wire segment can either be an over-the-device route or it can be within the channel route that already connects the device pins. The number of twigs generated in mesh routing depends on the aspect ratio of the device and the electrical data of the targeted object.

In comparison to mesh routing, standard automatic twig routing only allows connections to the device pins or to an already existing pre-route on the same net with no flexibility to spread and distribute the current over existing wires. This leads to the stacking requirements.

The following example shows mesh routing with two rows of horizontal pre-routes on `net39` on a large device. When a new horizontal wire is created on the selected net, multiple twigs (the vertical red segments) are created to connect both rows of pre-routes. With standard automatic twig routing, only one connection would be created on the whole device.



The number of twig connections can be modified by using the bindkeys. Use:

- `Ctrl` + `=` bindkey to add the twig connections
- `Ctrl` + `-` bindkey to remove the twig connections.

Mesh routing also supports the standard automatic twig bindkeys to increase and decrease the twig layer.

For twig mesh routing, an option is also available in the bindkeys info balloon. This option is only available when the weAutoTwigMeshRoutingEnabled environment variable is set in create wire.

Related Topics

weAutoTwigMode

weAutoTwigTrunkViaMode

weAutoTwigMeshRoutingEnabled

Connecting Twigs Automatically

SDR Toolbar

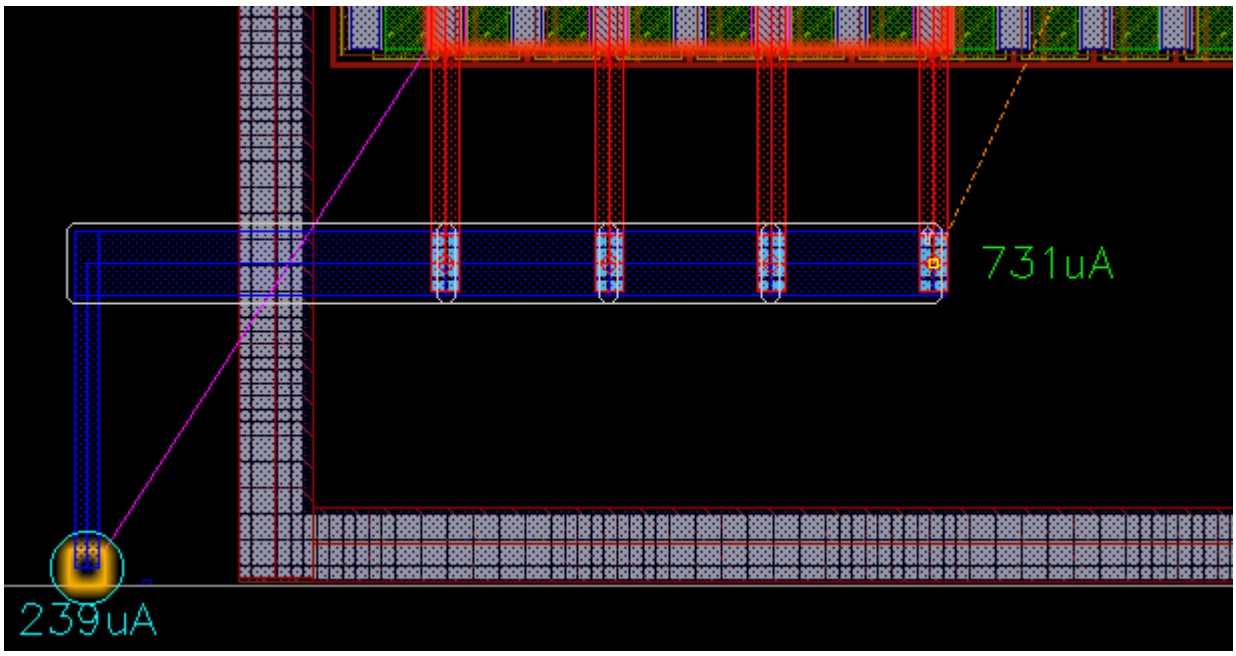
Tapering in SDR

Tapering is available only in *Enforce* checker mode for all current estimation modes. Use the tapering feature to show how each segment can be optimized by SDR. To adjust the width and taper each segment independently:

1. Select a net from the Navigator Assistant or the EAD Browser.
2. Choose *Create – Wiring – Wire*.

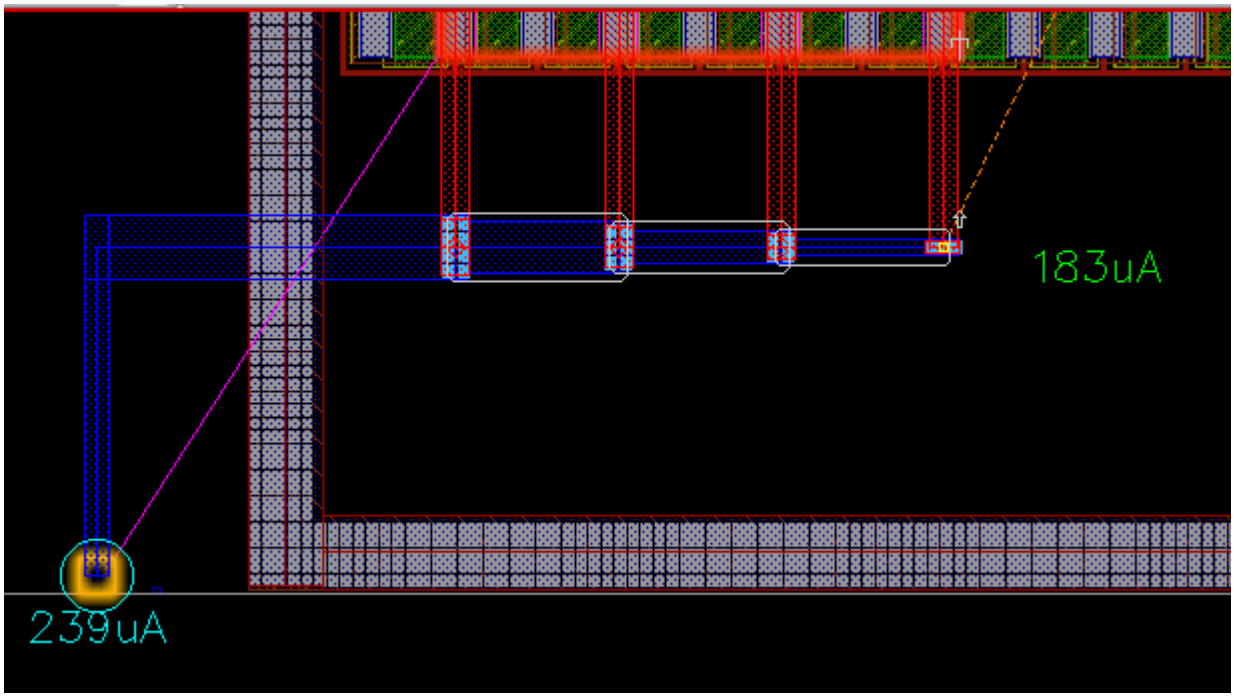
   Note: The tapering is available only for the *Create Wire* command.
3. Select an instance from where you want to start creating the wire.

By default, the taper mode is `noTaper`. This means that the wire width remains constant even after connecting the pins.

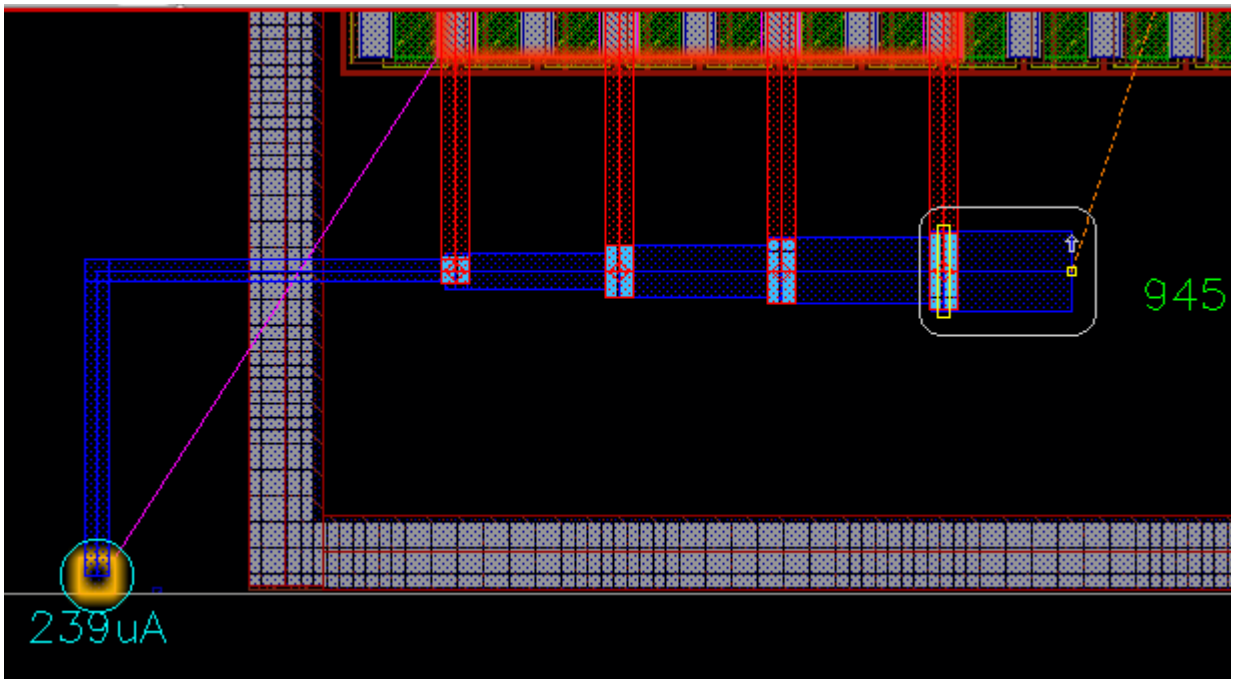


4. To enable tapering, press `}`.

When tapering is enabled, the width of each segment is independently adjusted to match the estimated current value. The following figure shows how a wire width is adjusted when tapering is enabled.



When there are no additional connected shapes for the edited wire and you continue to move the mouse pointer in the same direction, then after crossing the last target, the tapering of the wire moves in the reverse direction, as shown in the following figure.



Related Topics

weSdrTaperMode

SDR Toolbar

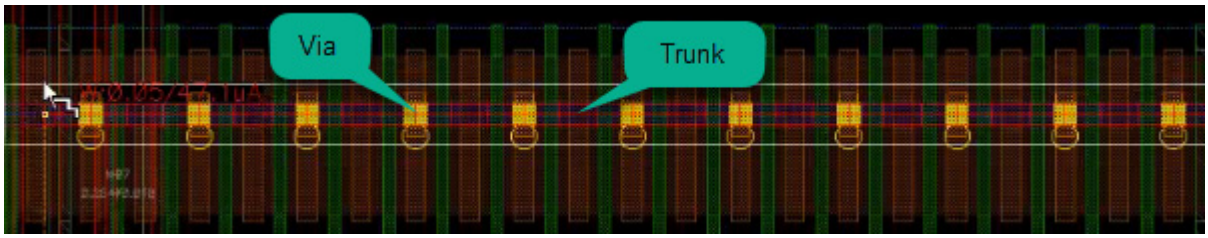
Routing Over Device

Simulation Driven Interactive Routing is designed to work in channel routing style. In the mature node, channel routing is recommended because over-device routing creates parasitics that can lead to changes in device behavior.

However, with FinFET technology, the routing density is more important than other criteria. Devices (especially MOS) have a different structure than before and are less sensitive to over-device routes. Therefore, Simulation Driven Interactive Routing now supports over-device routing, which is automatically enabled when a trunk is over device pins.

To automatically route over devices:

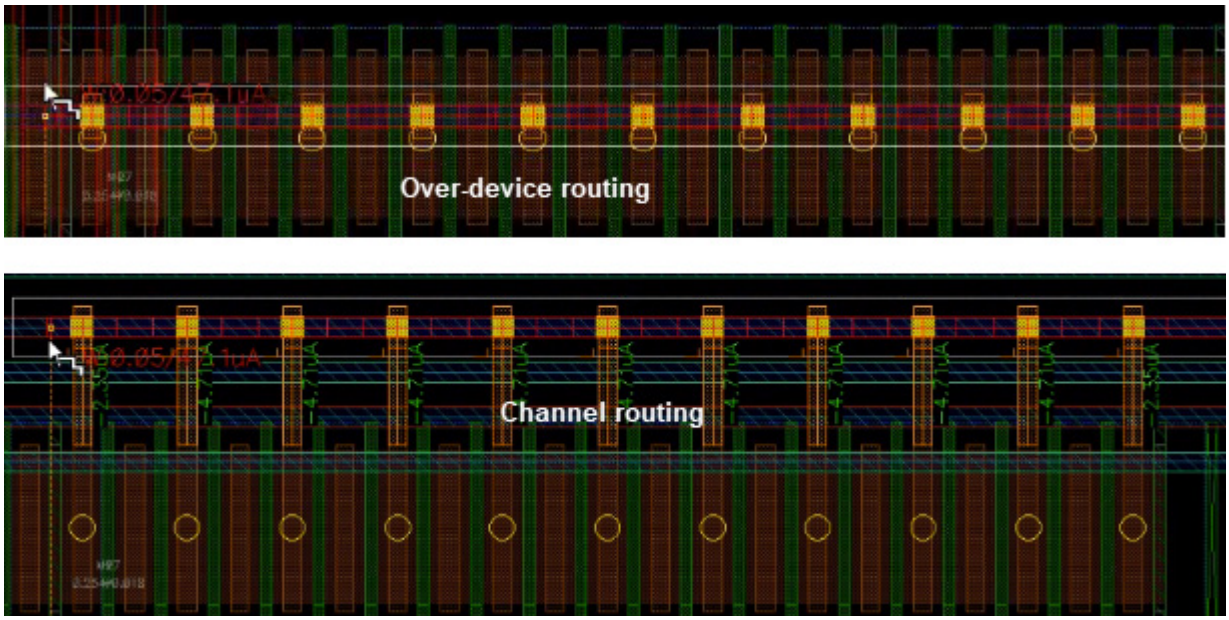
1. Choose *Create – Wiring – Wire*.
2. Select a location in line with the instance to start creating a wire.
3. While creating a wire, route over the device. The trunk is automatically connected to the device pins by adding vias, as shown in the following figure.



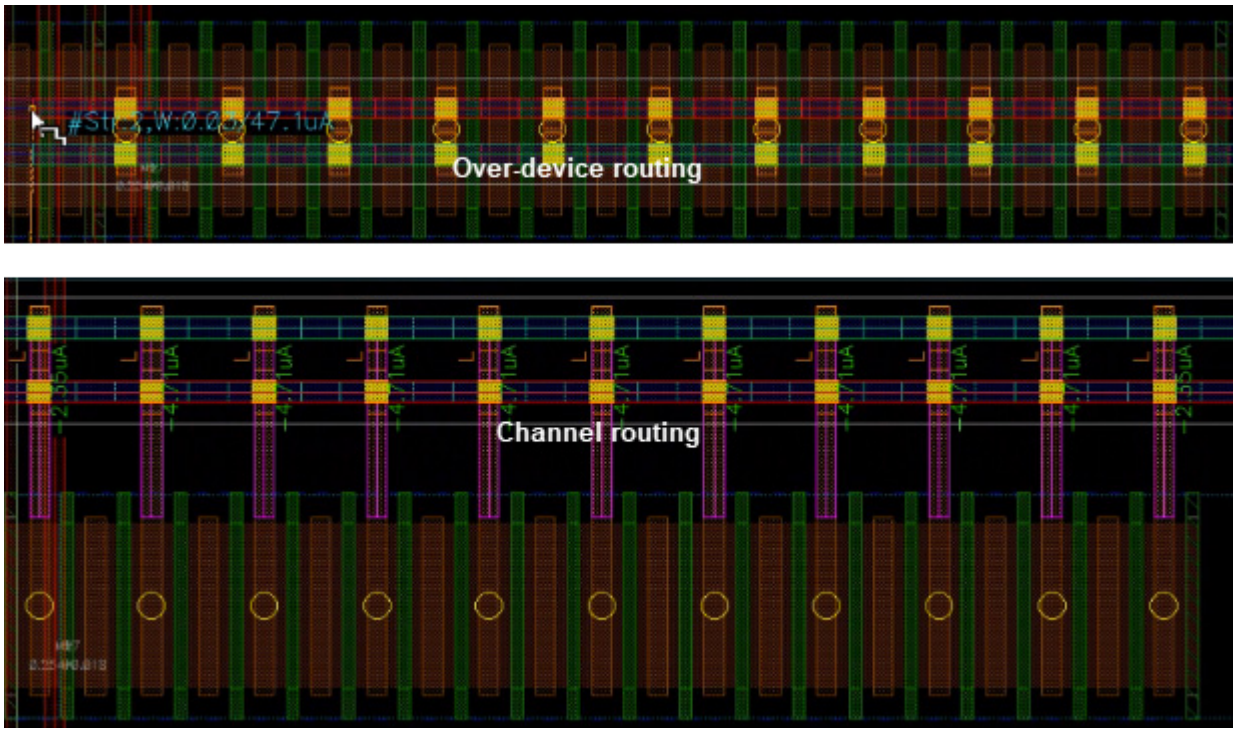
If both sides of the channel are connected, the connections to the devices that are not overlapped are removed. In addition, the current from them is ignored.

Similarly, when you run the *Create Stranded Wire* command, the trunk is automatically connected to the device pins by adding vias.

4. Hold the `Ctrl+Shift` keys and move the mouse pointer to a different current position. The wire is automatically switched between the channel routing and the over-device routing.



The following figure displays the switch between channel routing and over-device routing when the *Create Stranded Wire* command is used.



The *Cover Pin* and the *Snap to Pin Center* options are ignored in WSP. Otherwise, the path completely obstructs the pin and create shapes in the wrong direction. In addition, ignoring *Cover Pin* helps in the current distribution.

Related Topics

Performing Interactive SDR Checks

A

Simulation-driven Interactive Routing Environment Variables

This topic provides a list of graphical user interface equivalents for the environment variables used in Simulation-driven Interactive Routing (SDR).

Only the environment variables listed in this topic are supported for public use. All other environment variables, and undocumented aspects of the environment variables described below, are private and subject to change at any time.

- weAutoTwigCheckerMode
- weAutoTwigDefaultLayer
- weAutoTwigDirection
- weAutoTwigMatchPinWidth
- weAutoTwigMeshRoutingEnabled
- weAutoTwigMode
- weAutoTwigTargetsDetectionAccuracy
- weAutoTwigTargetsFinderObjectsLimit
- weAutoTwigTargetsType
- weAutoTwigTrunkViaAlignment
- weAutoTwigTrunkViaDirection
- weAutoTwigTrunkViaMode
- weSdrCheckMode
- weSdrCurrentEstimationMode
- weSdrDatasetNamingMethod
- weSdrDIDatasetName

- weSdrDisplayClusters
- weSdrElectricalMode
- weSdrWidthMultiplier
- weSdrMaxDisplayPins
- weSdrParasiticExtractionEffort
- weSdrResistanceScale
- weSdrShapeChasingLimit
- weSdrSourceSinkMap
- weSdrSourceSinkMapAllCurrents
- weSdrTaperMode

weAutoTwigCheckerMode

```
we weAutoTwigCheckerMode cyclic { "enforce" | "notifyMinSpacing" | "notify" }
```

Description

Specifies the checker mode to be used for automatic twig generation.

- `enforce` does not create twigs if doing so would result in DRC violations.
- `notifyMinSpacing` creates a twig even if doing so results in a `minSpacing` violation and displays a marker to highlight the violation. However, it does not generate twigs that would result in other types of DRCs violations.
- `notify` creates twigs even if doing so results in any type of DRC violation and displays markers to highlight the violations. This mode is recommended to understand why the twigs are not created automatically.

Default is `enforce`.

GUI Equivalent

Command	*SDR Toolbar – Options*
Field	*Allow violations on target*

Examples

```
envGetVal("we" "weAutoTwigCheckerMode")
envSetVal("we" "weAutoTwigCheckerMode" 'cyclic "notify")
envSetVal("we" "weAutoTwigCheckerMode" 'cyclic "notifyMinSpacing")
```

Related Topics

Simulation-driven Interactive Routing Environment Variables

weAutoTwigDefaultLayer

```
we weAutoTwigDefaultLayer cyclic { "auto" | "target" | "trunk" | "aboveTrunk" |
     "belowTrunk" }
```

Description

Controls the default layer for the automatic connection of twig creation, based on the specified criterion.

- `auto` uses the layer that is most suitable for the twig creation. This can be either the target pin layer, the trunk layer, the layer below the trunk, or the layer above the trunk.
- `target` uses the pin or target segment layer as the default twig layer.
- `trunk` uses the trunk layer as the default twig layer.
- `aboveTrunk` uses the layer above the trunk as the default twig layer.
- `belowTrunk` uses the layer below the trunk as the default twig layer.

The default is `auto`.

GUI Equivalent

None

Examples

```
envGetVal("we" "weAutoTwigDefaultLayer")
envSetVal("we" "weAutoTwigDefaultLayer" 'cyclic "trunk")
envSetVal("we" "weAutoTwigDefaultLayer" 'cyclic "target")
```

Related Topics

SDR Toolbar

weAutoTwigDirection

```
we weAutoTwigDirection cyclic { "both" | "vertical" | "horizontal" }
```

Description

Specifies the allowed direction of twigs.

- `both` allows the twigs in both vertical and horizontal direction.
- `vertical` allows the twigs in vertical direction.
- `horizontal` allows the twigs in horizontal direction.

Default is `both`.

GUI Equivalent

Command	*SDR Toolbar – Allow vertical and horizontal twigs*
Field	*NA*

Examples

```
envGetVal("we" "weAutoTwigDirection")
envSetVal("we" "weAutoTwigDirection" 'cyclic "vertical")
envSetVal("we" "weAutoTwigDirection" 'cyclic "horizontal")
```

Related Topics

Connecting Twigs Automatically

SDR Toolbar

weAutoTwigMatchPinWidth

```
we weAutoTwigMatchPinWidth boolean { t | nil }
```

Description

Controls the width of the automatically created twig segments. When set to `t`, the width of the twig segment always matches the pin width. If the pin width is smaller than the `minWidth` constraint or larger than the `maxWidth` constraint, the constraints are enforced.

However, when set to `nil`, the width of the twig segments is automatically computed to avoid EM violations. If the estimated width is smaller than the pin width but larger than pin width divided by 3, the twigs use the pin width to avoid DRC violations.

Default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("we" "weAutoTwigMatchPinWidth")
envSetVal("we" "weAutoTwigMatchPinWidth" 'boolean t)
```

Related Topics

Visualizing the Current Distribution Per Net

Connecting Twigs Automatically

SDR Toolbar

weAutoTwigMeshRoutingEnabled

```
we weAutoTwigMeshRoutingEnabled boolean { t | nil }
```

Description

Enables creation of mesh routing connecting the routes that exist over the device or in the channel. Mesh routing is realized using the automatically generated twigs. The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("we" "weAutoTwigMeshRoutingEnabled")
envSetVal("we" "weAutoTwigMeshRoutingEnabled" 'boolean t)
```

Related Topics

Connecting Twigs Automatically

SDR Toolbar

weAutoTwigMode

```
we weAutoTwigMode cyclic { "allTargets" | "allTargetsClosest" |
     "leftOrBottomTargets" | "leftOrBottomTargetsClosest" | "rightOrTopTargets"
     |"rightOrTopTargetsClosest" | "off" }
```

Description

Enables to toggle between twig modes. You can also use bindkey `t` to create and toggle between twig modes.

- `allTargets` creates twigs on both sides of the trunk.
- `allTargetsClosest` creates twigs on both sides of the trunk. However, connects only the closest row of targets above and below the trunk.
- `leftOrBottomTargets` connects twigs on the left of the trunk if it is vertical or on the bottom of the trunk if it is horizontal.
- `leftOrBottomTargetsClosest` connects the first column of twigs on the left of the trunk if it is vertical or the first row of twigs on the bottom of the trunk if it is horizontal.
- `rightOrTopTargets` connects twigs on the right of the trunk if it is vertical or on the top of the trunk if it is horizontal.
- `rightOrTopTargetsClosest` connects the first column of twigs on the right of the trunk if it is vertical or the first row of twigs on the top of the trunk if it is horizontal.
- `off` does not create twigs.

Default is `allTargets`.

GUI Equivalent

Command	*SDR Toolbar – Automatically Connect Twigs*
Field	*NA*

Examples

```
envGetVal("we" "weAutoTwigMode")
envSetVal("we" "weAutoTwigMode" 'cyclic "leftoOrBottomTargets")
envSetVal("we" "weAutoTwigMode" 'cyclic "rightOrTopTargets")
envSetVal("we" "weAutoTwigMode" 'cyclic "off")
```

Related Topics

Visualizing the Current Distribution Per Net

Connecting Twigs Automatically

SDR Toolbar

weAutoTwigTargetsDetectionAccuracy

```
we weAutoTwigTargetsDetectionAccuracy cyclic { "high" | "medium" | "low" }
```

Description

Relaxes the criteria, such as wider distance, components master, and so on, to help find a suitable target. The default is `high`.

GUI Equivalent

Command	*SDR Toolbar – Options*
Field	*Targets Detection Accuracy*

Examples

```
envGetVal("we" "weAutoTwigTargetsDetectionAccuracy")
envSetVal("we" "weAutoTwigTargetsDetectionAccuracy" 'cyclic "medium")
```

Related Topics

Simulation-driven Interactive Routing Environment Variables

SDR Options Form

weAutoTwigTargetsFinderObjectsLimit

```
we weAutoTwigTargetsFinderObjectsLimit int numObjects
```

Description

Limits the number of objects on a net that are processed while searching for pins that can be automatically routed by the automatic twig routing feature. The default is `500`. If this limit is reached then the automatic twig feature does not detect all pins that can be automatically routed but it will still connect to the first pins detected.

GUI Equivalent

None

Examples

```
envGetVal("we" "weAutoTwigTargetsFinderObjectsLimit")
envSetVal("we" "weAutoTwigTargetsFinderObjectsLimit" 'int 200)
```

Related Topics

Connecting Twigs Automatically

SDR Toolbar

weAutoTwigTargetsType

```
we weAutoTwigTargetsType cyclic { "instPinsOnly" | "pathSegsAndPins" }
```

Description

Enables the creation of automatic connections to unrouted pins and existing pathSegs.

- `instPinsOnly` creates automatic connections to pins that are not connected to any routing object.
- `pathSegsAndPins` creates automatic connections to any of these:
  - pins that are not connected to any routing object.
  - segments connected and aligned with pins.
  - trunk segments indirectly connecting (through segments or vias) multiple pins.

Default is `pathSegsAndPins`.

GUI Equivalent

Command	*SDR Toolbar – Options*
Field	*Targets Type*

Examples

```
envGetVal("we" "weAutoTwigTargetsType")
envSetVal("we" "weAutoTwigTargetsType" 'cyclic "instPinsOnly")
```

Related Topics

Connecting Twigs Automatically

SDR Options Form

SDR Toolbar

weAutoTwigTrunkViaAlignment

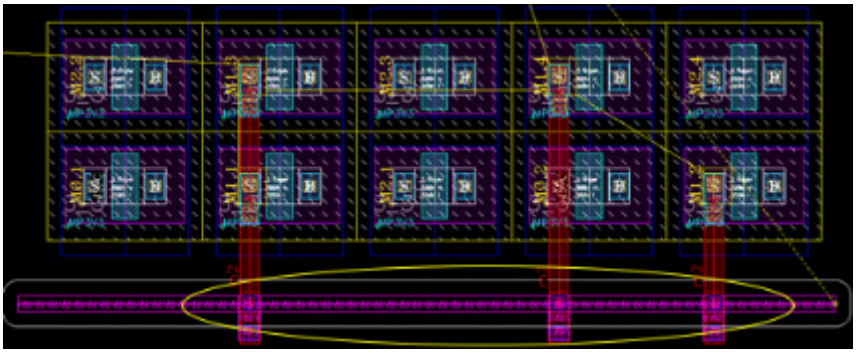
```
we weAutoTwigTrunkViaAlignment cyclic { "leftOrBottom" | "center" | "rightOrTop" }
```

Description

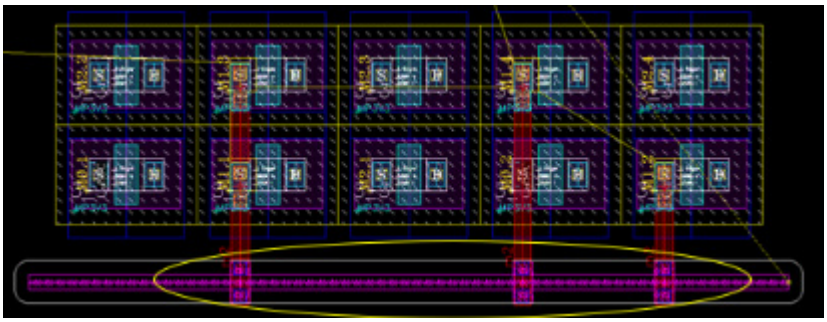
Controls the alignment of the via or via stacks created on each twig over the trunk. You can select one of the following methods for the via alignment to be used for twigs and trunks.

For twig via direction:

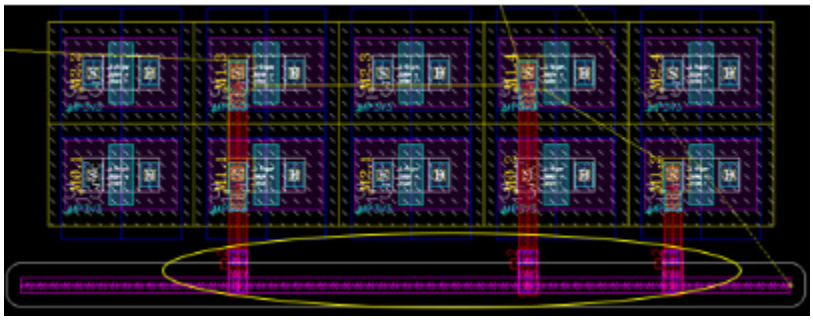
- `leftOrBottom`: Creates via or via stack in the direction of the twig that is vertical and bottom aligned to the trunk.



- `center`: Creates via or via stack in the direction of the twig that is vertical and aligned to the trunk center. This is the default alignment mode.

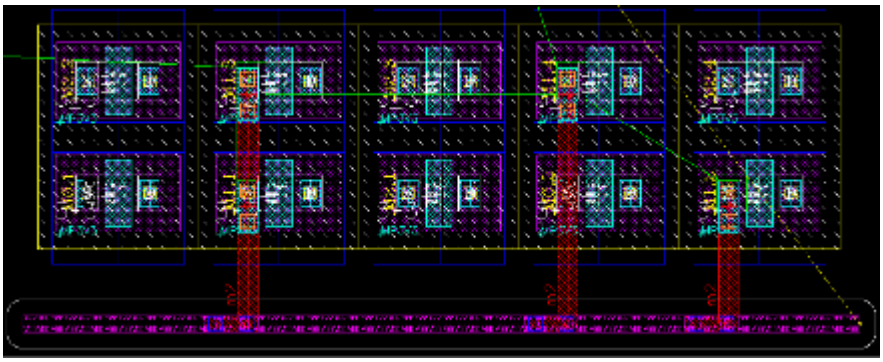


- `rightOrTop`: Creates via or via stack in the direction of the twig that is vertical and bottom aligned to the trunk.

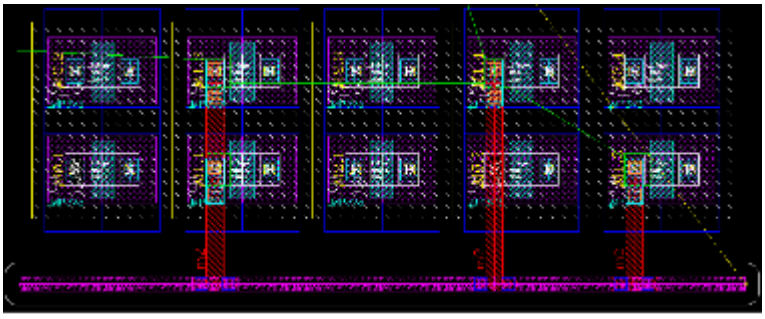


For trunk v*ia direction*:

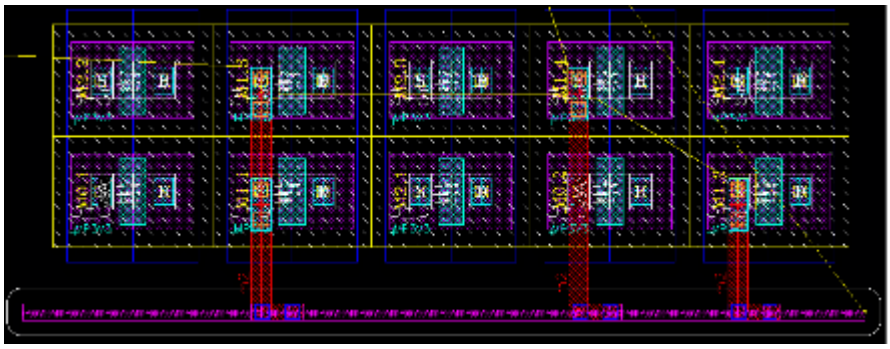
- `leftOrBottom`: Creates via or via stack in the direction of the trunk that is horizontal and left aligned to the twig.



- `center`: Creates via or via stack in the direction of the trunk that is horizontal and aligned to the twig center.



- `rightOrTop`: Creates via or via stack in the direction of the trunk that is horizontal and right aligned to the twig.



The default is `center`.

GUI Equivalent

Command	*SDR Toolbar – Via Alignment*
Field	NA

Examples

```
envGetVal("we" "weAutoTwigTrunkViaAlignment")
envSetVal("we" "weAutoTwigTrunkViaAlignment" 'cyclic "leftOrBottom")
```

Related Topics

SDR Options Form

SDR Toolbar

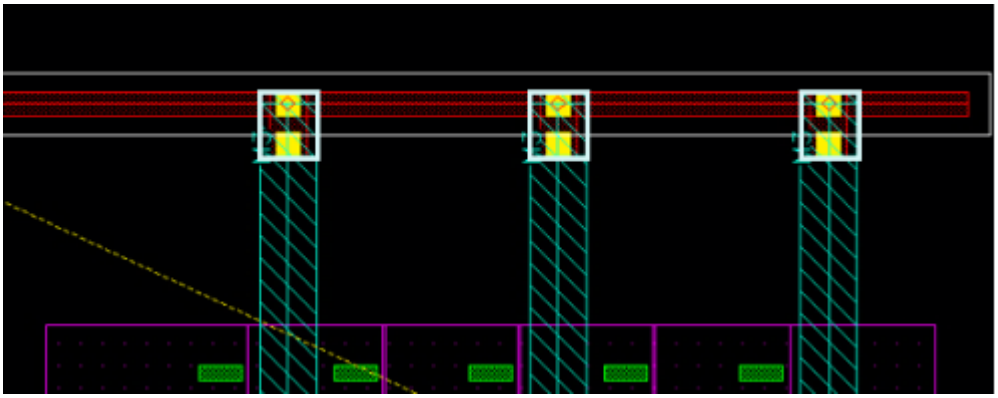
weAutoTwigTrunkViaDirection

```
we weAutoTwigTrunkViaDirection cyclic { "twig" | "trunk" }
```

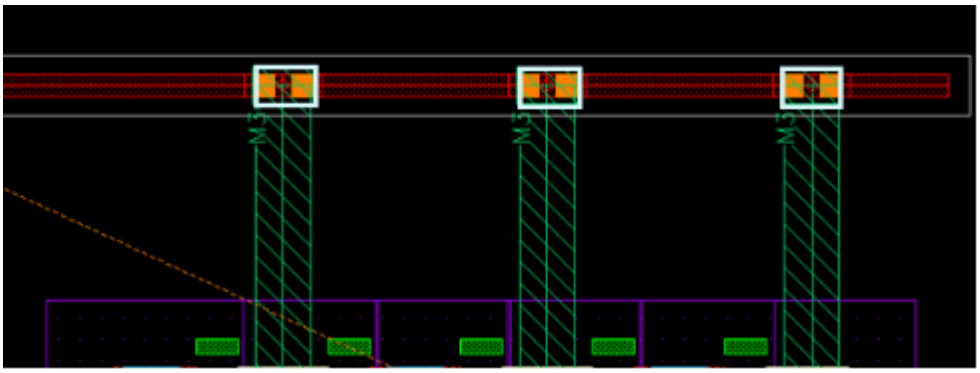
Description

Selects one of the objects to determine the via direction between the trunk and the twig:

- `twig`: The via enclosure is in the twig segment direction.



- `trunk`: The via enclosure is in the trunk segment direction.



The default is `twig`.

GUI Equivalent

Command	*SDR Toolbar – Options*
Field	*Via Direction*

Examples

```
envGetVal("we" "weAutoTwigTrunkViaDirection")
envSetVal("we" "weAutoTwigTrunkViaDirection" 'cyclic "trunk")
```

Related Topics

SDR Options Form

SDR Toolbar

weAutoTwigTrunkViaMode

```
we weAutoTwigTrunkViaMode cyclic { "oneViaPerTwig" | "oneViaForTrunk" |
     "oneViaIfViolations" }
```

Description

Selects one of the following methods to control how many vias are dropped on the trunk when twigs are generated.

- `oneViaPerTwig`: Creates one via for each twig segment.
- `oneViaForTrunk`: Creates a single via connecting multiple twig segments at the same time. This is useful to avoid spacing violations when the twig segments are close and creating one via per twig generates spacing issues between vias.
- `oneViaIfViolations`: Creates either one via per twig or one via for the trunk depending on the spacing violations between the vias.

The default is `oneViaPerTwig`.

GUI Equivalent

Command	*SDR Toolbar – Options*
Field	*Trunk Via Mode*

Examples

```
envGetVal("we" "weAutoTwigTrunkViaMode")
envSetVal("we" "weAutoTwigTrunkViaMode" 'cyclic "oneViaForTrunk")
```

Related Topics

Connecting Twigs Automatically

SDR Options Form

SDR Toolbar

weSdrCheckMode

```
we weSdrCheckMode cyclic { "enforce" | "notify" | "off" }
```

Description

Specifies the checker mode to be used.

Note: The violation depends on the electrical mode that is selected.

- `enforce` estimates the current in the edited wire and vias according to the EAD settings (dataset, temperature, and current scaling) and automatically calculates the wire width to avoid EM or `maxResistance` violations.
- `notify` estimates the current in the edited wire using a color coding based on EAD EM violations color settings. The color is computed to represent an EM or `maxResistance` violation that is reported by the EAD checker after routing.
- `off` does not update the wire width and there is no feedback on the estimated current or resistance.

Default is `enforce`.

GUI Equivalent

Command	*SDR Toolbar – Checker Mode*
Field	NA

Examples

```
envGetVal("we" "weSdrCheckMode")
envSetVal("we" "weSdrCheckMode" 'cyclic "notify")
envSetVal("we" "weSdrCheckMode" 'cyclic "off")
```

Related Topics

Visualizing the Current Distribution Per Net

Running Interactive SDR Current Density Checks

Connecting Twigs Automatically

SDR Toolbar

weSdrCurrentEstimationMode

```
we weSdrCurrentEstimationMode cyclic { "auto" | "connectedPins" | "override" |
     "flightline" }
```

Description

Controls how the wire being created estimates and calculates the current in the edited wire and via. To change the current estimation mode, you can also use the `Shift+middle mouse button` bindkey.

- `auto` estimates the current automatically in the last section of the wire according to the connected objects and flightline targets.
- `connectedPins` sums up the current of all the connected pins.

  Note: The violation depends on the electrical mode that is selected.
- `override` inherits the current from the previous wires.
- `flightline` uses the current of all the pins connected by the flightline.

Default is `auto`.

GUI Equivalent

Command	*SDR Toolbar – Current Estimation Mode*
Field	*NA*

Examples

```
envGetVal("we" "weSdrCurrentEstimationMode")
envSetVal("we" "weSdrCurrentEstimationMode" 'cyclic "connectedPins")
envSetVal("we" "weSdrCurrentEstimationMode" 'cyclic "override")
envSetVal("we" "weSdrCurrentEstimationMode" 'cyclic "flightline")
```

Related Topics

Visualizing the Current Distribution Per Net

Running Interactive SDR Current Density Checks

Connecting Twigs Automatically

Current Estimation Modes

SDR Toolbar

weSdrDatasetNamingMethod

```
we weSdrDatasetNamingMethod cyclic { "envvar" | "date" | "gui" }
```

Description

Configures how the design intent dataset is named.

- `envvar` uses the name specified by the weSdrDIDatasetName environment variable.
- `date` appends the current date and time to the name of the dataset. The default name that is generated is in the format of `DI_<date>_<hour>`.
- `gui` lets you specify a new dataset name.

Default is `envvar`.

GUI Equivalent

Command	*SDR Toolbar – Options*
Field	*Dataset Naming Method*

Examples

```
envGetVal("we" "weSdrDatasetNamingMethod")
envSetVal("we" "weSdrDatasetNamingMethod" 'cyclic "date")
envSetVal("we" "weSdrDatasetNamingMethod" 'cyclic "gui")
```

Related Topics

SDR Options Form

SDR Toolbar

weSdrDIDatasetName

```
we weSdrDIDatasetName string "datasetName"
```

Description

Lets you specify a default name for the design intent dataset.

GUI Equivalent

Command	*SDR Toolbar – Options*
Field	*Dataset Name*

Examples

```
envGetVal("we" "weSdrDIDatasetName")
envSetVal("we" "weSdrDIDatasetName" 'string "myDataset")
```

Related Topics

SDR Options Form

SDR Toolbar

weSdrDisplayClusters

```
we weSdrDisplayClusters boolean { t | nil }
```

Description

Controls the display mode and highlights the pins individually or by cluster. This function groups pins according to their location and type. The *Sources and Sink Map* highlights the boundary of the pin clusters with the sum total of the current for all the pins inside the same cluster. The current is calculated as `(max(sum(sinks),sum(sources))`.

Default is `t`. This means that the pins are highlighted by cluster.

GUI Equivalent

None

Examples

```
envGetVal("we" "weSdrDisplayClusters")
envSetVal("we" "weSdrDisplayClusters" 'boolean nil)
```

Related Topics

Sources and Sinks Map

Visualizing the Current Distribution Per Net

SDR Toolbar

weSdrElectricalMode

```
we weSdrElectricalMode cyclic { "EM" | "maxResistance" }
```

Description

Specifies if the edited wire width should be calculated according to the current or the resistance to avoid EM or `maxResistance` violations.

- `EM` estimates the current in the edited wire and calculates the wire width to avoid EM violations.
- `maxResistance` estimates the resistance in the edited wire and calculates the wire width to avoid `maxResistance` violations.

Default is `EM`.

GUI Equivalent

Command	*SDR Toolbar – Interactive SDR Current Density Check*
Field	*NA*

Examples

```
envGetVal("we" "weSdrElectricalMode")
envSetVal("we" "weSdrElectricalMode" 'cyclic "maxResistance")
```

Related Topics

Visualizing the Current Distribution Per Net

Performing Interactive SDR Checks

SDR Toolbar

weSdrWidthMultiplier

```
we weSdrWidthMultiplier float widthMultiplierValue
```

Description

Specifies a floating point value for the scaling wire width when SDR automatically adjust wires according to the current. The estimated width base on the current is a multiplier of this value. Default is `1`.

GUI Equivalent

Command	*SDR Toolbar – EM Width Multiplier*
Field	*NA*

Examples

```
envGetVal("we" "weSdrWidthMultiplier")
envSetVal("we" "weSdrWidthMultiplier" 'float 1.05)
```

Related Topics

SDR Toolbar

weSdrMaxDisplayPins

```
we weSdrMaxDisplayPins int numberofPins
```

Description

Specifies an integer value to control the maximum number of pins to be highlighted simultaneously by source and sink map. Default is `500`.

GUI Equivalent

None

Examples

```
envGetVal("we" "weSdrMaxDisplayPins")
envSetVal("we" "weSdrMaxDisplayPins" 'int 200)
```

Related Topics

SDR Toolbar

weSdrParasiticExtractionEffort

```
we weSdrParasiticExtractionEffort cyclic { "low" | "medium" | "high" }
```

Description

Extracts the resistance of the pins connected by the flightline accurately.

- `low` uses a fast method to extract the resistance of the pins connected by the flightline.
- `medium` and `high` extracts the resistance of the pins connected by the flightline accurately when one of the wires is already created.

Default is `high`.

GUI Equivalent

None

Examples

```
envGetVal("we" "weSdrParasiticExtractionEffort")
envSetVal("we" "weSdrParasiticExtractionEffort" 'cyclic "low")
envSetVal("we" "weSdrParasiticExtractionEffort" 'cyclic "medium")
```

Related Topics

SDR Toolbar

weSdrResistanceScale

```
we weSdrResistanceScale float resistaneScale
```

Description

Sets the resistance target percentage compared to the `maxResistance` constraint. Default is `0.95`.

GUI Equivalent

None

Examples

```
envGetVal("we" "weSdrResistanceScale")
envSetVal("we" "weSdrResistanceScale" 'float 0.9)
```

Related Topics

SDR Toolbar

weSdrShapeChasingLimit

```
we weSdrShapeChasingLimit int shapeChasingValue
```

Description

Specifies an integer value to control the number of connected shapes to compute wire resistance. Default is `10000`.

GUI Equivalent

None

Examples

```
envGetVal("we" "weSdrShapeChasingLimit")
envSetVal("we" "weSdrShapeChasingLimit" 'int 5000)
```

Related Topics

SDR Toolbar

weSdrSourceSinkMap

```
we weSdrSourceSinkMap boolean { t | nil }
```

Description

Creates a colored halo around the source and sink pins of the net according to the current distribution. Default is `nil`.

GUI Equivalent

Command	*SDR Toolbar – Display sources and sinks map*
Field	*NA*

Examples

```
envGetVal("we" "weSdrSourceSinkMap")
envSetVal("we" "weSdrSourceSinkMap" 'boolean t)
```

Related Topics

SDR Toolbar

Visualizing the Current Distribution Per Net

Sources and Sinks Map

weSdrSourceSinkMapAllCurrents

```
we weSdrSourceSinkMapAllCurrents boolean { t | nil }
```

Description

Displays details about the maximum current value of the pins or clusters according to each current type. When `t`, the source sink map displays for each pin or cluster, the maximum current value for each current types (Static, Avg, Rms, and Peak), if available. When `nil`, the source sink map only displays the maximum current value irrespective of the current type.

The default value is `nil`.

GUI Equivalent

Command	*SDR Toolbar – Display sources and sinks map*
Field	*NA*

Examples

```
envGetVal("we" "weSdrSourceSinkMapAllCurrents")
envSetVal("we" "weSdrSourceSinkMapAllCurrents" 'boolean t)
```

Related Topics

SDR Toolbar

Visualizing the Current Distribution Per Net

Sources and Sinks Map

weSdrTaperMode

```
we weSdrTaperMode cyclic { "taper" | "noTaper" }
```

Description

Controls resizing of each wire independently according to its current or if all collinear wires have the same width.

- `taper` calculates the wire width according to its current to avoid EM violations.
- `noTaper` prevents tapering. All collinear wires have the same width except when starting from an existing wire, which is too small compared to the current estimation for the edited segment.

Default is `noTaper`.

Note: Tapering is available only in the *Enforce* checker mode for all current estimation modes. Also, it is supported only for the *Create Wire* command.

GUI Equivalent

None

Examples

```
envGetVal("we" "weSdrTaperMode")
envSetVal("we" "weSdrTaperMode" 'cyclic "taper")
```

Related Topics

SDR Toolbar

Tapering in SDR

B

SDR Options Form

Use the SDR Options form to specify options that usually do not change for simulation driven interactive routing.

Field	Description
Automatic pin connections	This section lets you specify the options for automatically connecting pins.
Targets Type	Selects one of the following types for the creation of automatic connections to unrouted pins and existing pathSegs. ■ *Inst pins only*: Creates automatic connections to pins that are not connected to any routing object. ■ *PathSegs and pins*: Creates automatic connections to any of the following: ❑ pins that are not connected to any routing object ❑ segments connected and aligned with pins ❑ trunk segments indirectly connecting (through segments or vias) multiple pins Environment variable: weAutoTwigTargetsType

Field	Description
Via Mode	Selects one of the following methods to control how many vias are dropped on the trunk: ■ *One via per twig*: Creates one via for each twig segment. ■ *One via for trunk*: Creates a single via connecting multiple twig segments at the same time. This is useful to avoid spacing violations when the twig segments are close and creating one via per twig generates spacing issues between vias. ■ *One via if violation*: Creates either one via per twig or one via for the trunk depending on the spacing violations between the vias.
Via Direction	Selects one of the following methods for the via alignment between the trunk and the twig: ■ *Twig*: The via enclosure is in the twig segment direction. ■ *Trunk*: The via enclosure is in the trunk segment direction.
Allow violations on target	Lets you generate twigs even if there are spacing violations with existing routing. By default, this option is not selected. This means that twigs are not generated, which introduces DRC errors. When the option is selected, the twigs are generated even if they create `minSpacing` violations and a marker is generated to explain the reason of the violation. You can then fix the violation.
Match pin width	Controls the width of the automatically created twig segments. When set to `t`, the width of the twig segment always matches the pin width. Environment variable: weAutoTwigTrunkViaMode
Force all weak pins as must	Lets you establish connections with all weak pins in the design.
Targets Detection Accuracy	Relaxes the criteria, such as wider distance, components master, and so on, to help find a suitable target.
DI	This section lets you specify the naming method for the design intent dataset.
Dataset Name	Specifies the name of the dataset. Environment variable: weSdrDIDatasetName

Field	Description
Dataset Naming Method	Selects one of the following methods for the naming of design intent dataset: ■ *Use Dataset name*: Uses the specified dataset name when the *Import DI as Dataset* option is selected. ■ *Append date & time to Dataset Name*: Appends the current date and time to the specified name of the dataset. The default name that is generated is in the format of `DI_<date>_<hour>`. For example, DI_ ■ *User defined Dataset Name*: Lets you specify a new dataset name. Environment variable: weSdrDatasetNamingMethod
EM	
EAD Options	Displays the EAD Options form.
Consider scalar dynamic currents	Controls whether the peak and RMS current scalar values are visible in the EAD browser. These peak and RMS current scalar values are used by SDR for the current estimation and for the conversion of Design Intent into a dataset. Environment variable: supportSDRDynamicDataset

Related Topics

SDR Toolbar

Import DI As Dataset

Use the Import DI As Dataset form to specify a new dataset name.

Field	Description
Dataset Name	Specifies the name of the design intent dataset. Environment variable: weSdrDIDatasetName, weSdrDatasetNamingMethod

Related Topics

SDR Toolbar