cadence®

Virtuoso Voltage Dependent Rules Flow Guide

Product Version IC23.1
November 2023

Printed in the United States of America.

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

Contents

B
Voltage Dependent Rules Functions

1

Virtuoso Voltage Dependent Rules Flows

This topic describes three Virtuoso® voltage dependent rules (VDR) flows available in Virtuoso Layout Suite XL and higher tiers.

The complete simulation driven VDR flow lets you capture minimum and maximum voltage values from a simulation run in Virtuoso ADE and propagate the values to a layout view. They are stored in the OpenAccess database for the design and can be referenced by design rule driven (DRD) editing, the interactive wire editor, and the Virtuoso space-based router (VSR) to ensure that minimum voltage spacing constraints are honored during the physical implementation of your design. You can also use them to generate labels or markers in the layout view, which in turn can be used by tools like the Cadence® Pegasus physical verification system or Cadence PVS to verify the correctness of the implementation.

Alongside the simulation driven VDR flow, this document also covers schematic driven and layout-centric VDR flows, which let you add voltage values as properties directly on nets in a schematic or layout view. The property values are saved to the OpenAccess database for the design and can be used by DRD, the wire editor, and VSR.

This topic describes only the VDR flow and the various enhancements made to the Virtuoso Analog Design Environment and Virtuoso Layout Suite tools to support them. Links are provided to more detailed information on specific products available elsewhere in the Virtuoso documentation set.

Many foundry processes base the minimum spacing requirements between shapes on metal layers on the maximum voltage difference between those shapes. These minimum spacing requirements are defined in the technology file as `minVoltageSpacing` constraints, which can be used by design rule checkers and by interactive and automatic routers to ensure that the circuit designer's intent is maintained in the physical implementation of the design.

This document describes three voltage dependent rules (VDR) flows that help automate the process and reduce the number of costly design iterations.

- The Simulation Driven VDR Flow lets the designer capture minimum and maximum voltage values on nets derived from simulation runs in Virtuoso ADE and propagate the

data to the OpenAccess layout view. Because this information is stored in the OpenAccess database, it can be used by Virtuoso tools, such as design rule driven (DRD) editing or the interactive wire editor, to verify in real time that the design is DRC correct. The Virtuoso space-based router (VSR) also considers the voltage values and avoids violating the minimum spacing rules when routing the design.

You can also use this flow to automatically create voltage labels or markers in the layout view based on the values captured during a simulation. This eliminates the need to manually annotate the layout design with voltage information, lets you visualize the voltages that apply in different areas of your design, and facilitates the use of DRC signoff tools, such as the Cadence® Pegasus physical verification system or Cadence PVS, to check that design rules are being honored and that the design is correct-by-construction.

If the schematic design is changed, you can rerun the simulation to update the voltage values in the simulation datasets and update the referenced voltage data in the layout view to take account of the changes.

- The Schematic Driven VDR Flow lets the circuit designer specify the required minimum and maximum voltage values as properties directly on the nets in the schematic, eliminating the need for guesswork on the part of the layout designer. The properties are saved in the OpenAccess schematic view and are transferred to the layout view using the usual Layout XL generation and update commands. Once in Layout XL, the values can be used by DRD, the interactive wire editor, and VSR as described above.

  Note: You cannot use the schematic driven VDR flow to automatically create labels or markers from voltage values specified as schematic properties. They must first be transferred to the layout view.

- The Layout-centric VDR Flow lets the layout designer set minimum and maximum voltage values as properties directly on the nets in the layout view. This data is then saved in the OpenAccess layout view and can be used by DRD, the interactive wire editor, and VSR, as described above. You can generate voltage labels or markers for all the nets at the current level of layout hierarchy or for a set of nets selected in the Navigator assistant.

Licensing Requirements

The VDR flows require the following licenses:

- **Invoking Virtuoso**
  - `111`: Cadence Design Framework II
  - `95011`: Virtuoso Advanced Node Framework

- **Capturing Voltages and Saving Datasets in Virtuoso ADE**
  - `95250`: Virtuoso ADE Explorer
  - `95260`: Virtuoso ADE Assembler

  Plus **one** of the following:

  - `95510`: Virtuoso Implementation Aware Design Option
  - `95600`: Virtuoso Layout Suite EAD
  - `95800`: Virtuoso Layout Suite EXL

  You also need appropriate licenses for the simulation engines you use to generate simulation data (For example, Spectre).

- **Populating Voltages in Layout XL**
  - `95310`: Virtuoso Layout Suite XL or `95321`: Virtuoso Layout Suite - GXL
  - `95511`: Virtuoso Advanced Node Option for Layout

For detailed information on the license requirements for Virtuoso and other Cadence tools, see the *Virtuoso Software Licensing and Configuration Guide*.

Types of VDR Labels

The minimum and maximum voltage values specified for each net are displayed in the canvas as labels attached to the in question. There are two types of VDR labels.

- **Generic VDR labels** are generated by Virtuoso and are identified by a `CDNS_VDR_LABEL` property.

  They are based on voltage data captured in simulation datasets, specified manually for nets in schematic or layout designs, or specified in a voltage information CSV file. Virtuoso reads the values from the specified source and creates labels for Vmin and Vmax values. The layer on which the label is drawn is derived from the parent net shape at create time, while the purpose is specified in the Voltage Dependent Rules form or the voltage purpose file.

  During ECO, Virtuoso first deletes all existing labels and then creates a new set of labels from the updated source data.

- **Constrained VDR labels** can be generated by Virtuoso, or they can be created, copied, or manually edited by the user directly in the layout view.

  They are created on the layer-purpose pairs defined in a `voltageLabelMapping` constraint in a technology file constraint group specified by the vdrConstraintGroupName environment variable. If `vdrConstraintGroupName` is defined and the specified constraint group contains a `voltageLabelMapping` constraint for at least one layer, then the constrained label flow is automatically enabled.

  During ECO, Virtuoso does not delete constrained labels, but rather updates any existing labels with new values and creates any labels that are missing.

  The consistency of constrained labels can be checked using the VDR Debugger. See Sanity Checking Voltage Values in Constrained Labels for information.

The two types of labels cannot be mixed in a the same design. If constrained labels are defined for layers `Metal1` through `Metal5` and a generic label needs to generated by VDR for `Metal6`, then Virtuoso issues an error.

Prerequisites for the VDR Flows

To allow DRD editing, the interactive wire editor, and Virtuoso space-based router to recognize violations of voltage dependent spacing rules, you must first specify the required minimum voltage spacing rules in the process technology file used by your design. For information, see Specifying a Minimum Voltage Spacing Constraint.

To create labels and markers in the layout view which can be used by Pegasus or PVS to verify the design, you must additionally specify the layer purposes on which voltage labels or markers are to be created. For more information, see Specifying Layers and Purposes for Generic Voltage Labels and Specifying Layers and Purposes for Voltage Markers.

If your process features nets the voltage of which must transition in phase with each other, you can also define *Voltage Synced Nets* constraints for the nets in question. You can then use the constraints to draw *vsync* shapes in the layout view and check them using the VDR Sanity Checker. For more information, see Specifying Layers and Purposes for Synced Nets.

Specifying a Minimum Voltage Spacing Constraint

The minimum spacing allowed between two wires with different voltages on the same metal layer is defined using a `minVoltageSpacing` constraint. The example below defines minimum voltage spacings for wire shapes on `Metal1` and `Metal2`:

```
spacingTables(
 ;( constraint                layer1                [layer2]
```

```
 ;   (( index1Definitions    [index2Definitions]) [defaultValue] )
 ;   ( table) )
;( ----------------------------------------)
  ( minVoltageSpacing          "Metal1"
     (( "voltage"     nil     nil ))
     (
       0.0   0.38
       1.8   0.39
       3.3   0.4
       )
  )
  ( minVoltageSpacing          "Metal2"
     (( "voltage"     nil     nil ))
     (
       0.0   0.48
       1.5   0.49
       3.3   0.5
       )
  ) ;spacingTables
```

In this example, wires on `Metal1`

- With voltages between 0.0V and 1.8V must be spaced at least 0.38 microns apart
- With voltages between 1.8V and 3.3V must be spaced at least 0.39 microns apart
- With voltages higher than 3.3V must be spaced at least 0.4 microns apart

For detailed information, see minVoltageSpacing in the *Virtuoso Technology Data ASCII Files Reference*.

Specifying Layers and Purposes for Generic Voltage Labels

By default, voltage labels are always generated on the same layer as the shape over which they are placed, but they use a different layer purpose. All specified layers and purposes must be defined in the technology file and the LPP must be set as valid in the techDisplays section. If there is no valid LPP for a net on a particular layer, the label is generated on the valid layer with the lowest mask number.

You control the LPPs on which labels are drawn using the following options.

- You can limit the *layers* on which labels are drawn by using the vdrSetValidLayers SKILL API to specify a list of valid layers for VDR label generation. The software generates labels only for nets on one of the listed layers.

  Note: Use vdrGetValidLayers to see the current list of valid layers.

- You control the *purposes* on which a label is drawn by using the *High Voltage Purpose* and *Low Voltage Purpose* options on the Voltage Dependent Rules form. The defaults

are `"vlo"` for minimum and `"vhi"` for maximum voltage labels and are set using the following environment variables:

- vdrHighVoltagePurpose
- vdrLowVoltagePurpose

- If your process requires greater control over the specific purpose on which a label is drawn, you can define a *Special Voltage LPP File*, which lets you override the default layer and purpose settings.

  The file is a simple text file stored at an accessible location in your file system with the format indicated below. You can specify it using the vdrLayerPurposeFile environment variable.

```
#This is vdrLayerPurpose.map

# layerName   low_volt_purpose/lpp   high_volt_purpose/lpp
OD              (ODV_low sp_low)      (ODV_high sp_high)
PO              (POV_low sp_low)      (POV_high sp_high)
M1              sp_low                sp_high
M2              sp_low                sp_high
M3              sp_low                sp_high
```

  Each entry can specify a layer name and the purposes or LPPs on which to draw high and low voltage labels for that layer. You can override the layer by specifying a layer-purpose pair in the purpose or LPP column as shown in the first two rows of the example above. If the same layer is listed twice, the first entry is taken and a warning issued to indicate that subsequent mappings were ignored. If the LPP is not defined in the technology file, no label is created and an appropriate warning is issued. Wildcard (*), tab (`\t`), and newline (`\n`) characters are also supported.

See Generating Voltage Labels from Simulation Data for All Nets for more information on how to use the options.

Specifying Layers and Purposes for Voltage Markers

In the marker-based VDR flows, the voltage purpose file lists the layer-purpose pairs on which markers for different voltage values are to be created.

The file is a simple text file stored in an accessible location in your file system with the format indicated below. Tab (`\t`) and newline (`\n`) characters are also supported.

```
#This is volt_LPP.map

#LayerPur      Metal   Voltage
#----------------------------
HVD test0      Metal1  -5
Metal1 vlo     Metal1  0.0
Metal1 dummy2  Metal1  0.1
```

```
Metal1 dummy3  Metal1  0.2
Metal1 dummy4  Metal1  0.3
Metal1 dummy5  Metal1  0.4
Metal1 dummy6  Metal1  0.5
Metal1 dummy7  Metal1  0.6
Metal1 dummy8  Metal1  0.7
Metal1 dummy9  Metal1  0.8
Metal1 dummyb  Metal1  1.0
Metal1 dummya  Metal1  0.9
Metal1 drawing Metal1  1.5
Metal1 vhi     Metal1  2.5
```

Each entry specifies the layer and purpose on which to create a marker based on the metal layer and voltage specified for the net in question. For example:

If a net voltage is...	The marker is drawn on...
> –5.0 && <= 0.0	`Metal1 vlo`
> 0.0 && <= 0.1	`Metal1 dummy2`
> 0.1 && <= 0.2	`Metal1 dummy3`
> 0.9 && <= 1.5	`Metal1 drawing`
> 1.5 && <= 2.5	`Metal1 vhi`

See Generating Voltage Markers from Simulation Data for All Nets and Generating Voltage Markers for Manually Entered Voltages for information about how the file is specified and used.

Specifying Layers and Purposes for Synced Nets

Some advanced node processes feature the concept of *synchronized* (or *synced*) *nets*, for which voltage values must always transition in phase. The voltage difference that triggers a DRC violation for the synced nets is lower than in mature node processes.

The ICADVM20.1 release supports the enhanced voltage check by allowing you to tag pairs of nets as *synced* using a *Voltage Synced Nets* constraint in the schematic, and to specify in the process technology file the layer-purpose pairs on which to draw shapes for the nets in the layout view. For example:

- To specify that the synced net delta voltage calculation should be performed for nets on a single layer (for example, `M1`), use the voltageLayerMarkerMapping constraint to specify that shapes are drawn for synced nets on layer `M1` and purpose `vsync`:

  ```
  (voltageLayerMarkerMapping "M1" 'layer "M1" 'purpose "vsync" )
  ```

Shapes are drawn on the specified layer and purpose provided at least one of the following constraints is defined in the technology file for the layer in question:

- minCornerSpacing
- minCornerVoltageSpacing
- minSideSpacing
- minSpacing
- minVoltageSpacing

- If the synced nets in the constraint are on two different layers, you need to specify on which layer the marker should be drawn. For example, for synced nets on layers `MD` and `PO`, you can use the voltageLayerPairMarkerMapping constraint to specify that markers are drawn on layer `MD` and purpose `vsync` as follows:

  ```
  (voltageLayerPairMarkerMapping "MD" "PO" 'layer "MD" 'purpose "vsync" )
  ```

  Shapes are drawn on the specified layer and purpose provided at least one of the following constraints is defined in the technology file for the layer in question:

  - minCornerSpacing (Two layers)
  - minCornerVoltageSpacing (Two layers)
  - minSideSpacing (Two layers)
  - minSpacing (Two layers)
  - minVoltageSpacing (Two layers)

See the *Virtuoso Technology Data Constraints Reference* for more information on these constraints.

You can then use the VDR synced nets flow to generate the required marker shapes in the layout view. When DRD detects shapes on the specified LPPs it automatically applies the synced delta voltage calculation for the nets in question. See Defining and Checking Voltage Synced Nets for more information.

Simulation Driven VDR Flow

The simulation driven VDR flow automates the creation of voltage labels or markers in the layout by leveraging the voltage values derived from simulations run in Virtuoso ADE. The flow comprises the following steps:

1. Setting Up Testbenches and Corners in Virtuoso ADE
2. Enabling Voltage Capture in Virtuoso ADE
3. Running Simulations and Capturing Voltage Data
4. Populating Voltages and Generating Labels or Markers

For more information about how you can view voltage information in canvas info balloons, use DRD editing to report when minimum spacing rules are violated during interactive editing, and how you can write your own SKILL procedures to exclude transient spikes during simulation runs, see the following topics:

- Showing Voltage Information in Info Balloons
- Checking for Voltage Dependent Spacing Violations using DRD
- Customizing the Voltage Calculation

Setting Up Testbenches and Corners in Virtuoso ADE

Open your design in Virtuoso ADE and set up tests and corners for simulation. These define a combination of variables or process models to describe a scenario in which you want to measure the performance of your design. See the following topics in the *Virtuoso ADE Explorer User Guide* and *Virtuoso ADE Assembler User Guide* for more information:

- Specifying Tests and Analyses
- Working with Design Variables and Instance Parameters
- Adding Corners

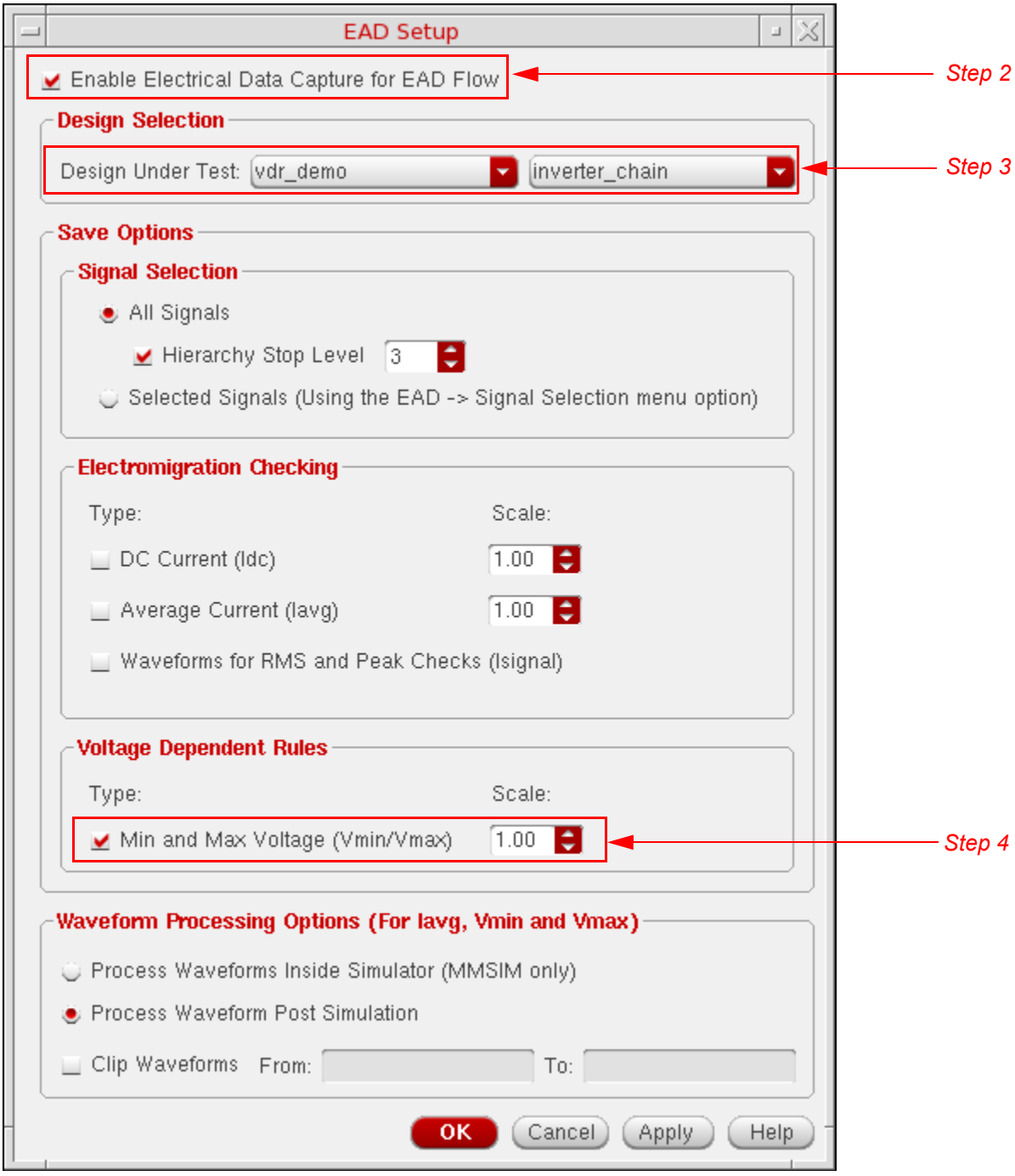
When you have completed the tasks, your design is ready to run simulations in Virtuoso.

Enabling Voltage Capture in Virtuoso ADE

When your testbenches and corners are in place, you need to enable voltage capture for the design under test. To do this:

1. In the Virtuoso ADE window, choose *EAD — Setup* from the menu bar.

   The **EAD Setup** form appears.



2. Ensure that *Enable Electrical Data Capture for EAD Flow* is selected at the top of the form.

3. Choose the required *Design Under Test* in the Design Selection group box.
4. Check the *Min and Max Voltage (Vmin/Vmax)* option in the Voltage Dependent Rules group box to capture minimum and maximum voltage values during the simulation run.

   If needed, use the *Scale* option to specify a multiplier by which the voltage data is scaled before it is transferred to the layout view. The default is 1.00, which means that voltage data is not scaled.
5. Click *OK* to accept the settings and enable voltage capture.

For more information on the other options available in the EAD Setup form, see The EAD Setup Form and Preparing the EAD Setup for Simulation in the *Virtuoso Electrically Aware Design Flow Guide*.

Running Simulations and Capturing Voltage Data

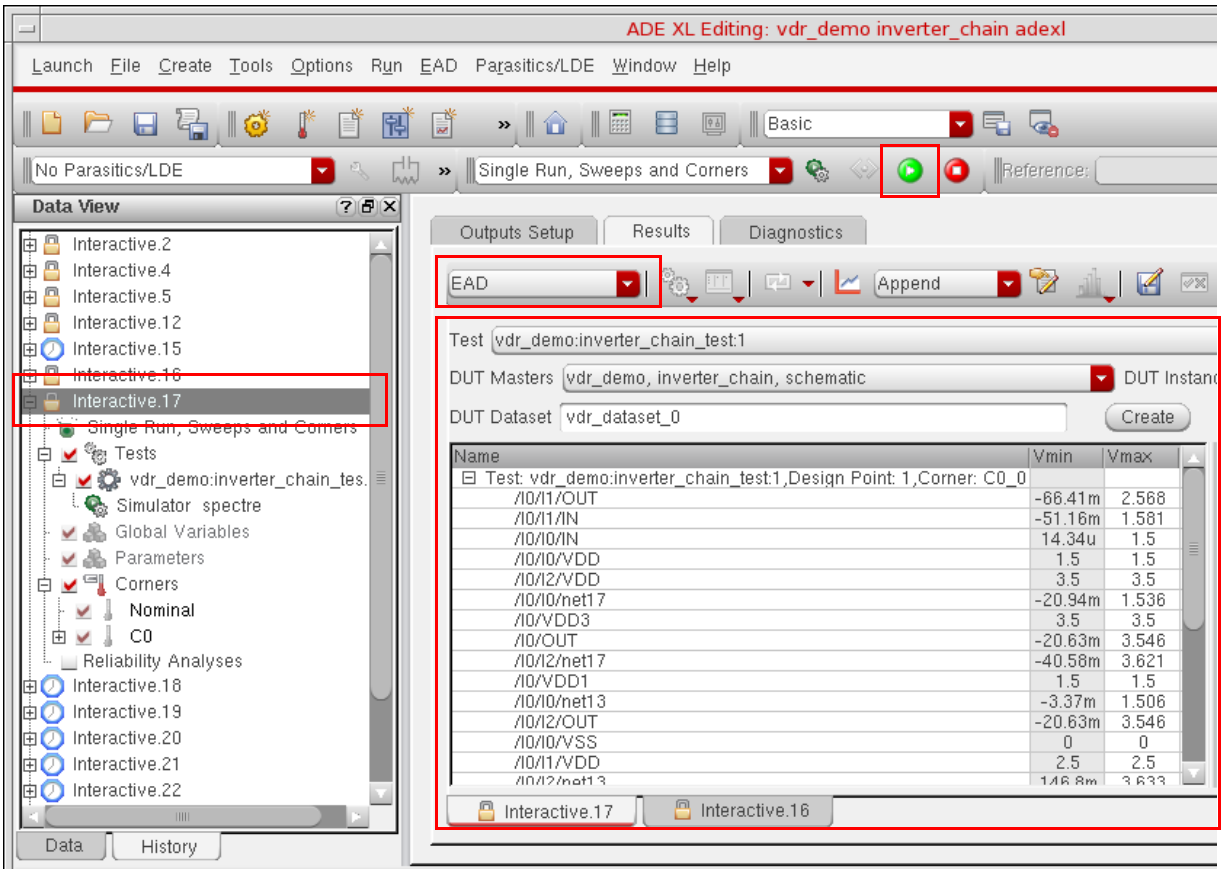
After specifying the test and corner details, enabling voltage capture by EAD, and specifying any custom calculation you need, click *Run Simulation* on the Run toolbar in Virtuoso ADE to run the simulation and display the results on the Results tab. You can now view the voltage data in the results and create datasets that can be transferred and used in Layout XL. The following sections describe how to view the voltage data from the simulation results and how to create datasets that can be transferred to the layout view.

- Viewing Voltage Data
- Creating Datasets for the Voltage Data

Viewing Voltage Data

The voltage data for the selected nets is saved in the testbench results. To view the data:

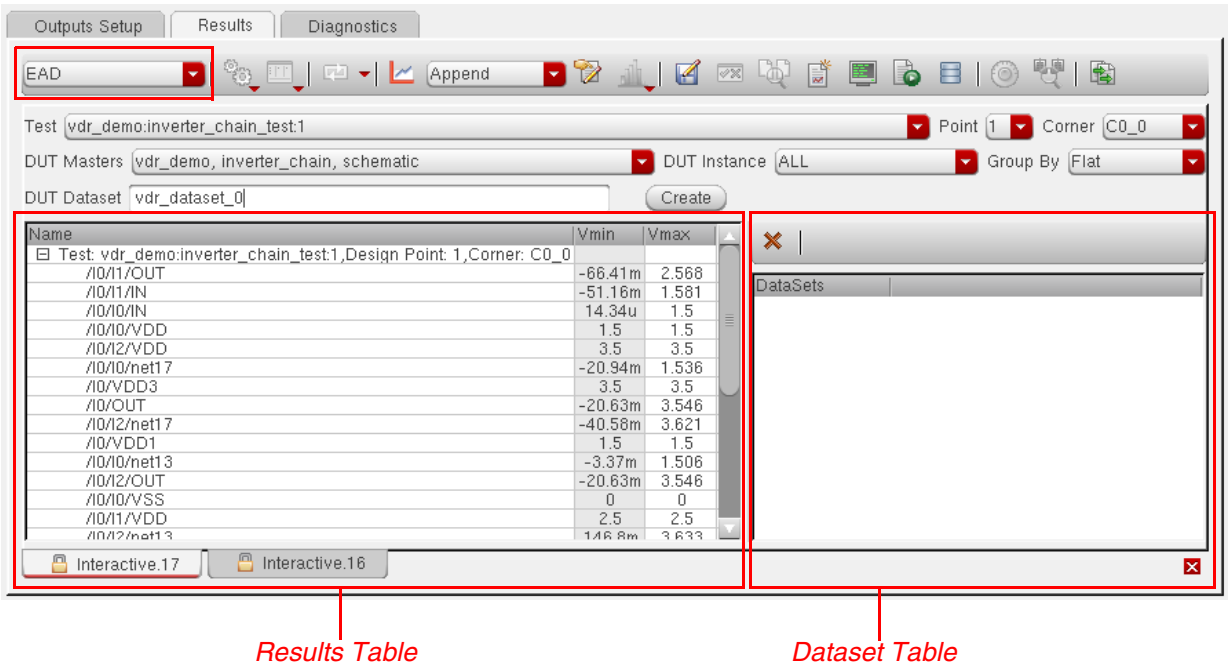
1. Ensure that the correct results history is loaded.



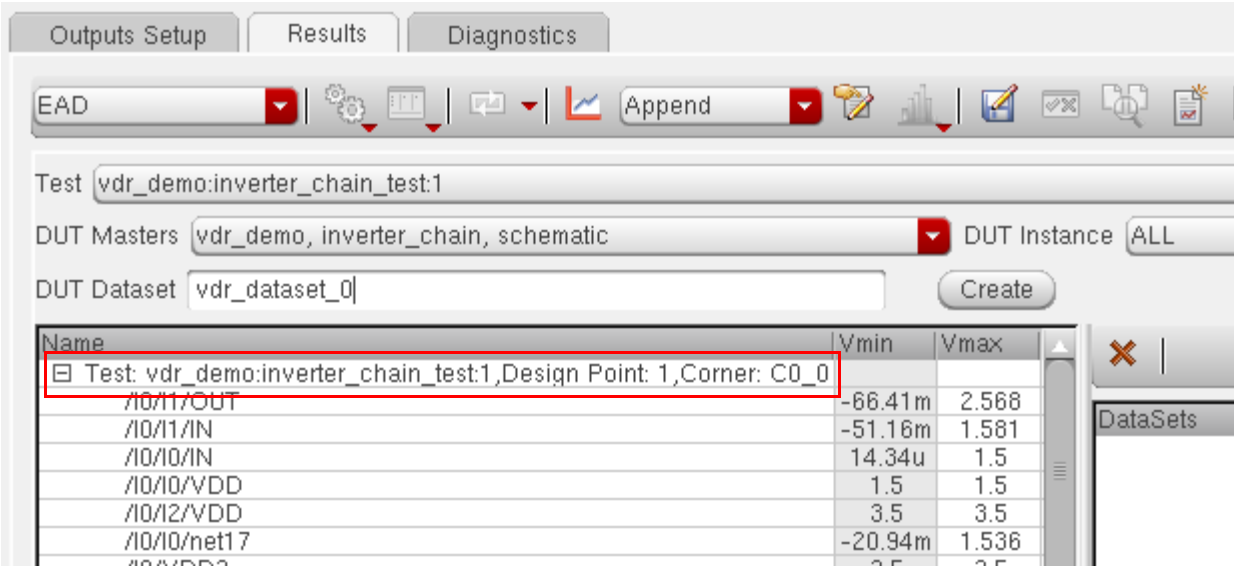
2. Select the *EAD* view on the *Results* tab of the Design Environment XL window.

   The EAD view appears.

By default, the data for the first test, first design point, and the first corner in the Virtuoso ADE results database is shown in the Results Table:



The test name, design point number, and the corner name are displayed in the title of the result tree node:



You can filter the results by changing the values in the various drop-down lists shown above the Results Table.

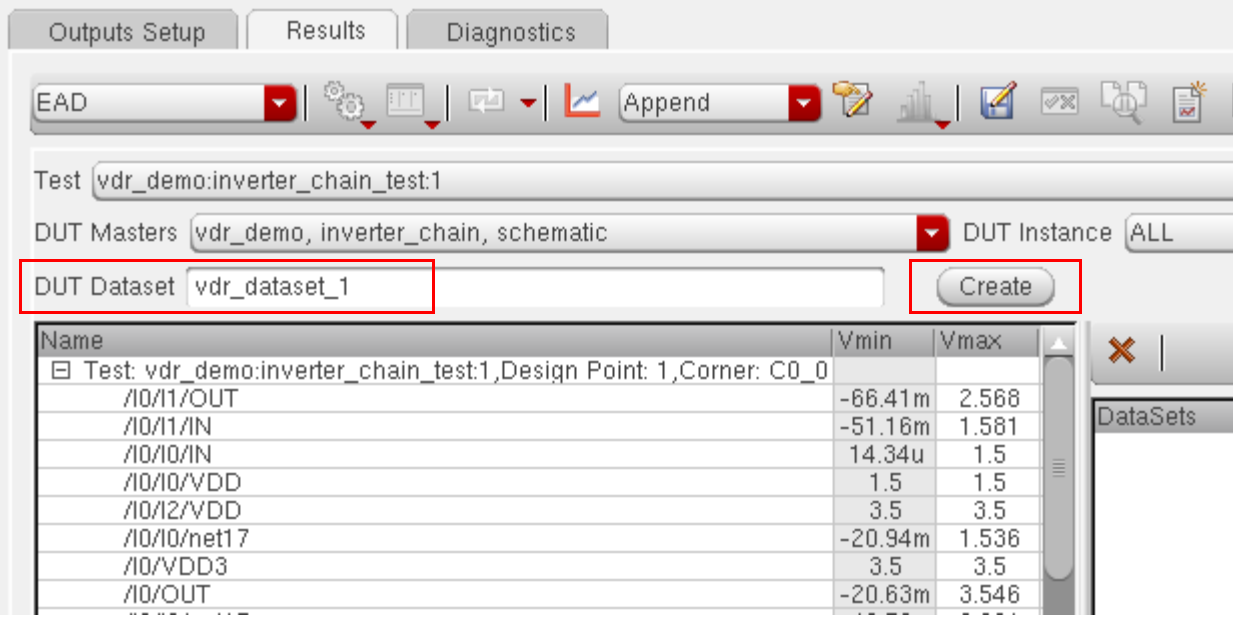
For more information, see Viewing the Current Data in the *Virtuoso Electrically Aware Design Flow Guide*.

Creating Datasets for the Voltage Data

To reference the voltage data in the layout view, you must first save the information into voltage datasets that can be transferred to and referenced by Layout XL.

To save the voltage data in a dataset:

1. Specify a name for the dataset in the *DUT Dataset* field and click *Create*.

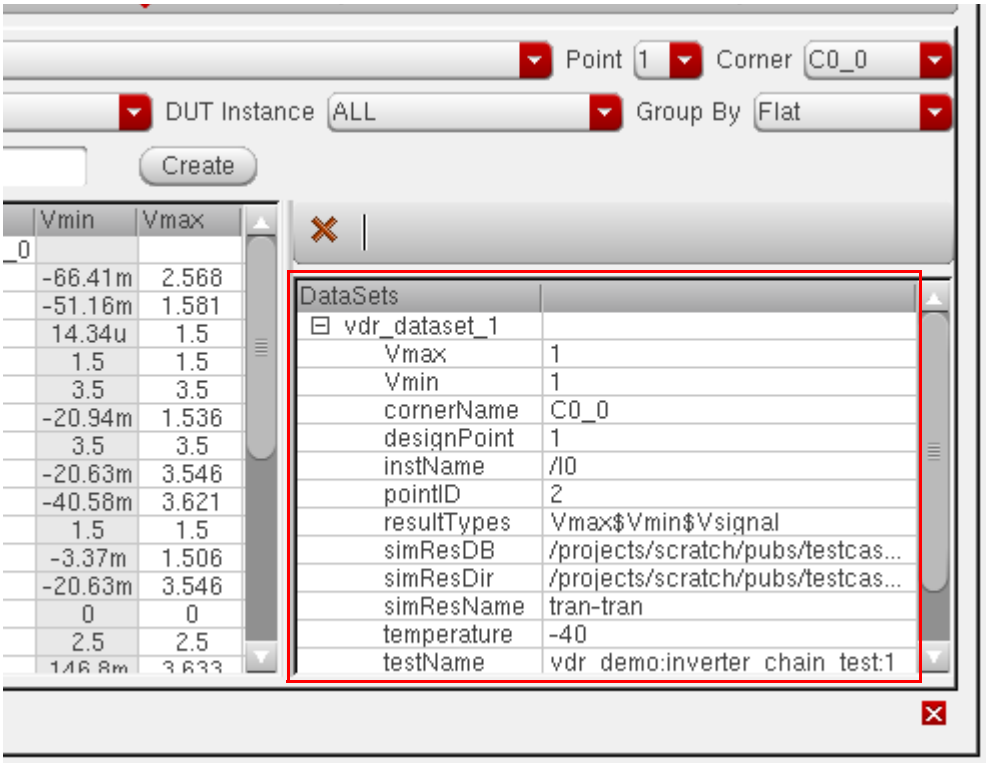


Virtuoso ADE creates a new dataset with the specified name and saves it in the constraint view for the design.

Note: If both current and voltage capture are enabled in the EAD Setup form, Virtuoso ADE creates two different datasets, one each for current measurements and voltage measurements. It appends "_I" to the dataset that contains current data and "_V" to the dataset that contains voltage data.

For more information on creating datasets, see Creating Datasets for the Current Data in the *Virtuoso Electrically Aware Design Flow Guide*.

2. You can view the datasets for a design in the Dataset Table in the EAD results view, as shown below.

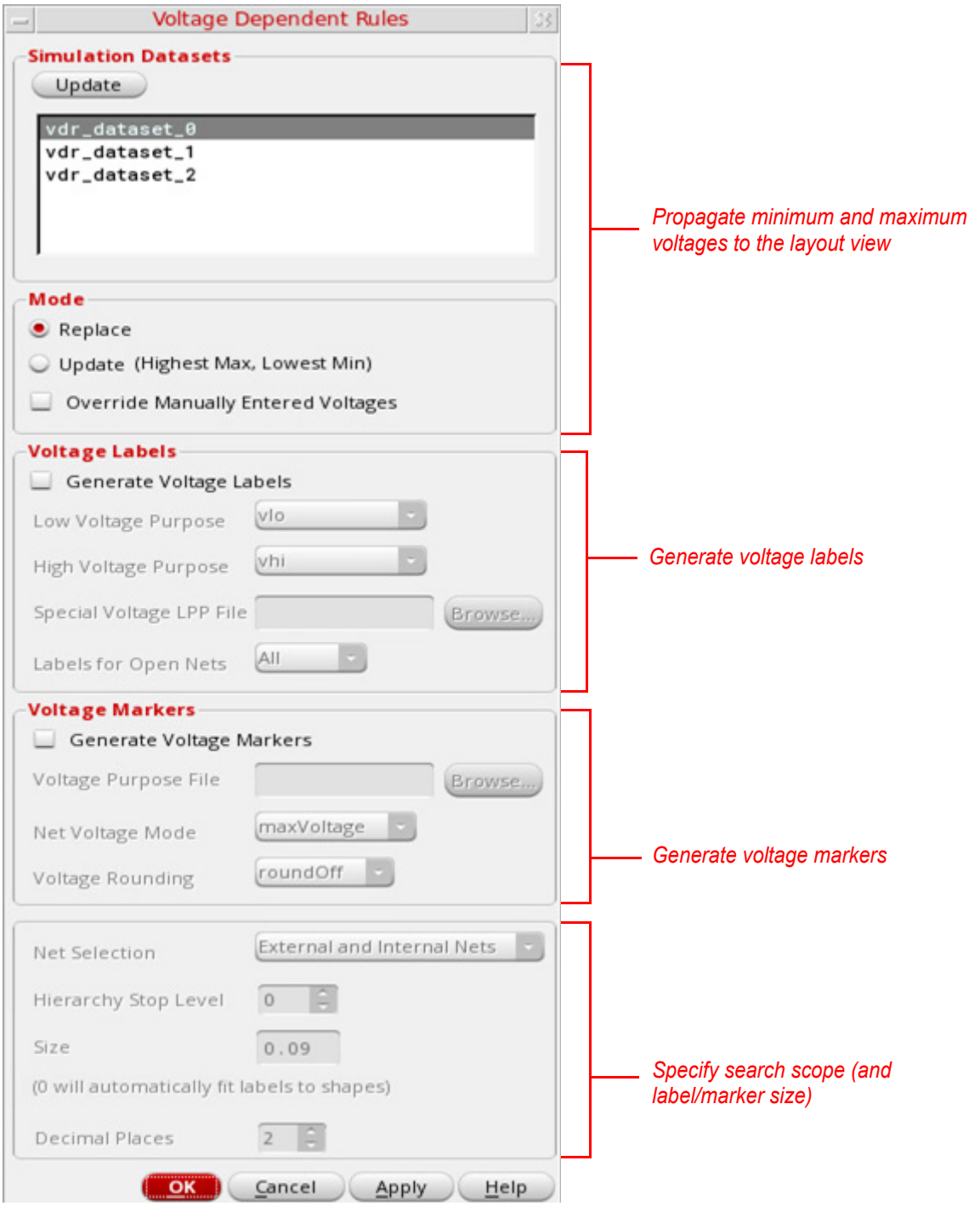


3. Expand the dataset to view details of the test name, corner name, voltages, temperature values, and more.

   For more information, including alternative methods of viewing the contents of datasets, see Viewing Datasets in the *Virtuoso Electrically Aware Design Flow Guide*.

Populating Voltages and Generating Labels or Markers

To reference the datasets created during the simulation runs and generate corresponding voltage labels or markers for all the nets in the layout design, use the *Tools – Voltage Dependent Rules – Create Labels/Markers From Simulation Voltages* command.



- Use the options in the *Simulation Datasets* and *Mode* group boxes to propagate the minimum and maximum voltages to the OpenAccess database for the layout design. This makes the voltage data available to layout editor tools such as design rule driven (DRD) editing, the interactive wire editor, and the Virtuoso space-based router, which help ensure that the minimum voltage spacing constraints defined in the technology file are honored. See Storing Voltages in the OpenAccess Database for more information.
- Use the options in the *Voltage Labels* group box to generate voltage labels for all nets in the layout design. the labels can then be used by DRC sign-off tools such as Pegasus or PVS to check that the minimum voltage spacing rules are met. See Generating Voltage Labels from Simulation Data for All Nets for more information.

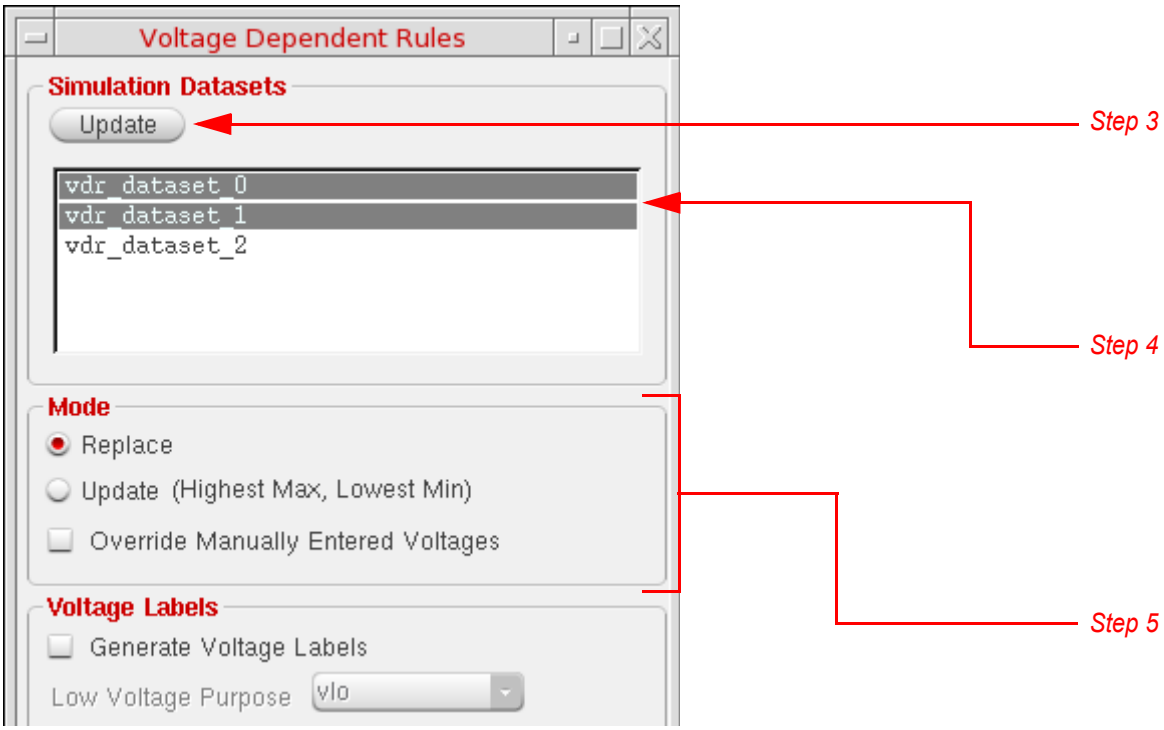
  Alternatively, if required by your process, you can use the options in the *Voltage Markers* group box to generate voltage markers for all nets in the layout design. See Generating Voltage Markers from Simulation Data for All Nets for more information.
- Use the common options at the bottom of the form to specify for which nets labels or markers are to be generated and how far down the hierarchy to search for those nets.

Storing Voltages in the OpenAccess Database

To propagate the minimum and maximum voltage values from the datasets to the OpenAccess database for the layout design:

1. Open the layout design in Layout XL.
2. Choose *Tools – Voltage Dependent Rules – Create Labels/Markers From Simulation Voltages* from the layout window menu bar.

   The Voltage Dependent Rules form appears.



3. (Optional) Click *Update* to transfer all the latest voltage dataset information from Virtuoso ADE. This is important if simulation data has changed since it was last referenced by Layout XL.
4. Choose the datasets containing the data you want to use from the list.
5. Choose the *Mode* you require:

   - *Replace* the minimum and maximum voltage values for the nets in the selected datasets. This is the default.

     Note: Voltages are replaced only for the nets that are present in the selected datasets. All other nets are left unchanged.

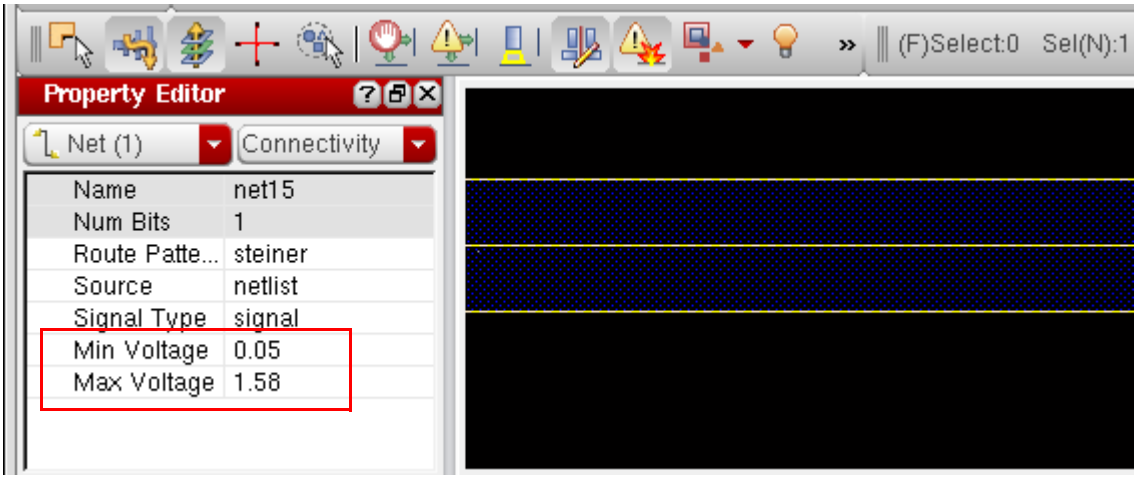
- *Update* the minimum and maximum voltages for the nets in the selected datasets, but only if the minimum voltage value specified for the net is lower than the value currently in the design and the maximum value specified for the net is higher than the voltage currently in the design.

Use *Override Manually Entered Voltages* to override any voltage values that were entered manually on nets using the Property Editor assistant in VLS or VSE (and then propagated to the layout by using *Generate All From Source*).

Note: By default, the option is switched off and user-specified voltages are not overridden. The only exception is if you set both minimum and maximum voltages to 0 manually in the Property Editor assistant and then generate labels directly from the Navigator using these values. Those labels *will* be overwritten if you subsequently run the simulation-driven flow (provided the dataset you use contains values for the nets in question and even if *Override Manually Entered Voltages* is switched off).

6. Click *OK* to populate the minimum and maximum voltages on nets in the OpenAccess database.

   You can view the voltages in the Property Editor assistant in Layout XL.



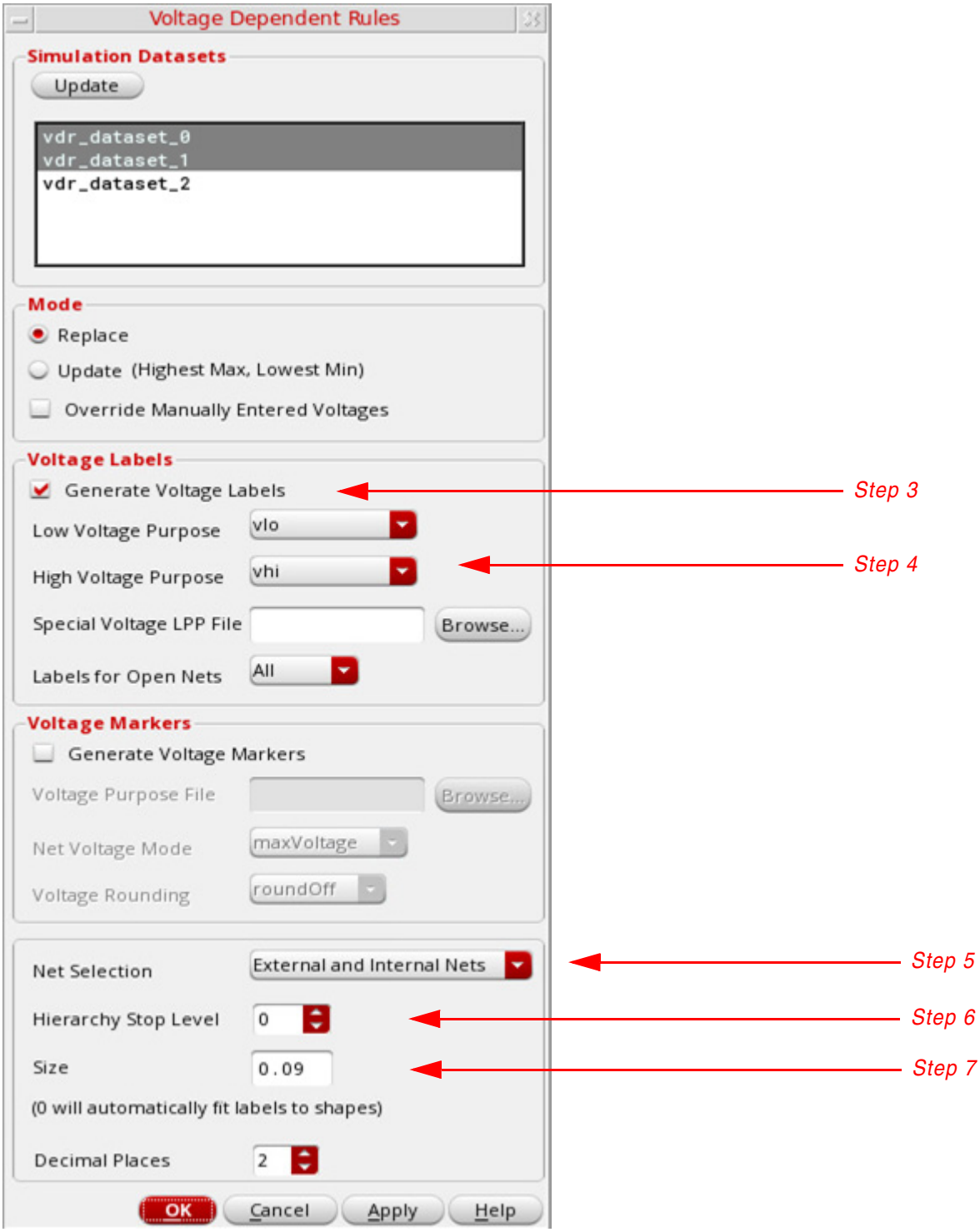
   Observe that no labels or markers have been generated on the net in the layout canvas yet.

Generating Voltage Labels from Simulation Data for All Nets

To generate voltage labels from simulation data for all the nets in your design:

1. Choose *Tools – Voltage Dependent Rules – Create Labels/Markers From Simulation Voltages* to display the Voltage Dependent Rules form appears.

2. Select the datasets and mode as described in Storing Voltages in the OpenAccess Database.



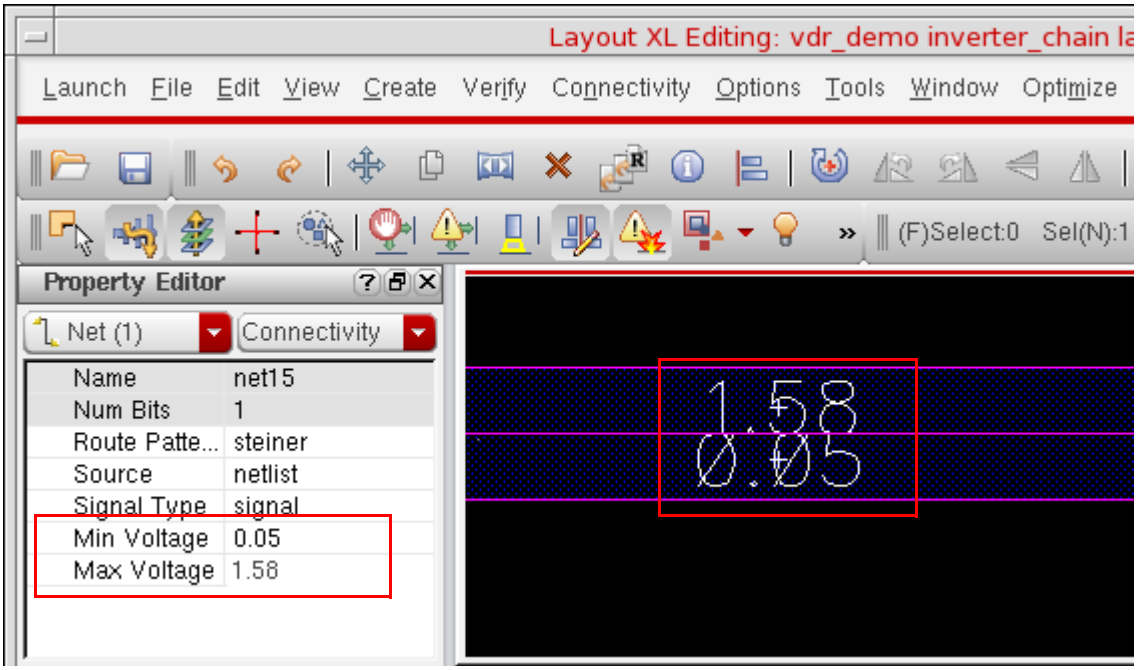
3. Check the *Generate Voltage Labels* box to enable the controls in the *Voltage Labels* group box.
4. Set the layer purposes on which the voltage labels are to be drawn.

   Specify a *Special Voltage LPP File* if you require greater control over the purposes on which labels are drawn.
5. Specify the nets for which labels are to be generated. Choose between:
   - *External and Internal Nets* to generate labels for all nets (the default).
   - *External Nets Only* to generate labels only for nets that are connected to the terminals of the cellview to which they belong.
   - *Internal Nets Only* to generate labels only for nets that are internal to the cellview in which they belong.
6. Specify how many hierarchy levels to search for nets on which to generate labels.

   The default is 0, which means that labels are generated only for top-level nets only; 1 means top-level nets and nets located one level below in the hierarchy; and so on.
7. Specify the size of the labels to be generated.

   Type 0.0 to automatically size labels to match the height of the shape to which they are attached.
8. Click *OK* to generate the specified labels.

The voltage labels for the specified nets are added in the layout canvas, with the maximum value above and the minimum value underneath it; for example:



Label generation depends on the *Mode* setting.

- In *Replace* mode, all voltage labels are replaced for the nets in the datasets.
- In *Update* mode:
  - Minimum voltages are updated only if the existing label shows a higher voltage.
  - Maximum voltages are updated only if the existing label shows a lower voltage.

By default, labels are generated for a net only if geometry exists on that net. Set the vdrZeroShapeNets environment variable to also generate labels for nets on which no shapes exist. The software does not consider geometry inside parameterized cells.

Labels are also generated for individual bits of bus nets.

Mosaics are not supported.

Generating Voltage Labels from a Voltage Information File

As an alternative to using a simulation dataset, you can specify a text file with comma-separated values as an input for label generation. If the same net is listed multiple times in the file, the generated labels reflect the highest maximum and lowest minimum values specified for that net.

Note: This feature is available only through the vdrCreateVoltageLabel and vdrCreateVoltageLabelEx SKILL functions. There is no GUI equivalent.

The following command runs label generation in *Replace* mode for only the top-level nets in the current cellview. Voltage values are taken from a CSV file called `voltages.csv`, and labels are drawn on purpose `drawing`. The code then automatically executes a user-defined callback named `_myPostVdrCB` to perform some post-processing tasks.

```
vdrCreateVoltageLabel(geGetEditCellView() nil "drawing" "drawing" 0.0 t nil
nil t nil 0 nil "_myPostVdrCB" "./voltages.csv")
```

The format of the voltage information file is illustrated below.

```
#Net, minV, maxV
#All comments start with '#'

#Top-level Nets
AVDD,           1.0,    1.5
net15,          -10,    10
net57,          1.4,    1.8

#Wildcard * is supported,
V*,             0.4,    0.5
|I1/*,          -1.0,   1.0

#Hierarchical Nets
#Full hierarchical path must be specified with '/' as delimiter.
A0/A1/net31,    0.7,    0.5
|I2/VDD,        -2.0,   2.0
|I0/OUT,        -3.0,   3.0

#Bus Nets
|I0/VSS<1:5>,   -4.0,   4.0

#Bit Nets
|I0/VSS1<10>,   -4.0,   4.0
```

If the CSV file contains a DUT instance name before net names, the DUT instance name must be defined in the CSV header so that VDR strips off the DUT instance name before the net names to read voltage information.

For example:

```
# DUT_INST:/I0
#RESULT_TYPES(SCALE_FACTOR): Vmin(1),Vmax(1)
#Name,Vmin,Vmax
/I0/net15,-50.05m,1.58
/I0/net14,-65.88m,2.562
/I0/VSS,0,0
/I0/VDD3,3.5,3.5
/I0/VDD2,2.5,2.5
```

```
/I0/VDD1,1.5,1.5
/I0/OUT,-20.09m,3.543
/I0/IN,141.5n,1.5
/I0/|I2/net17,-39.46m,3.61
/I0/|I2/net13,160.2m,3.627
/I0/|I2/VSS,0,0
/I0/|I2/VDD,3.5,3.5
```

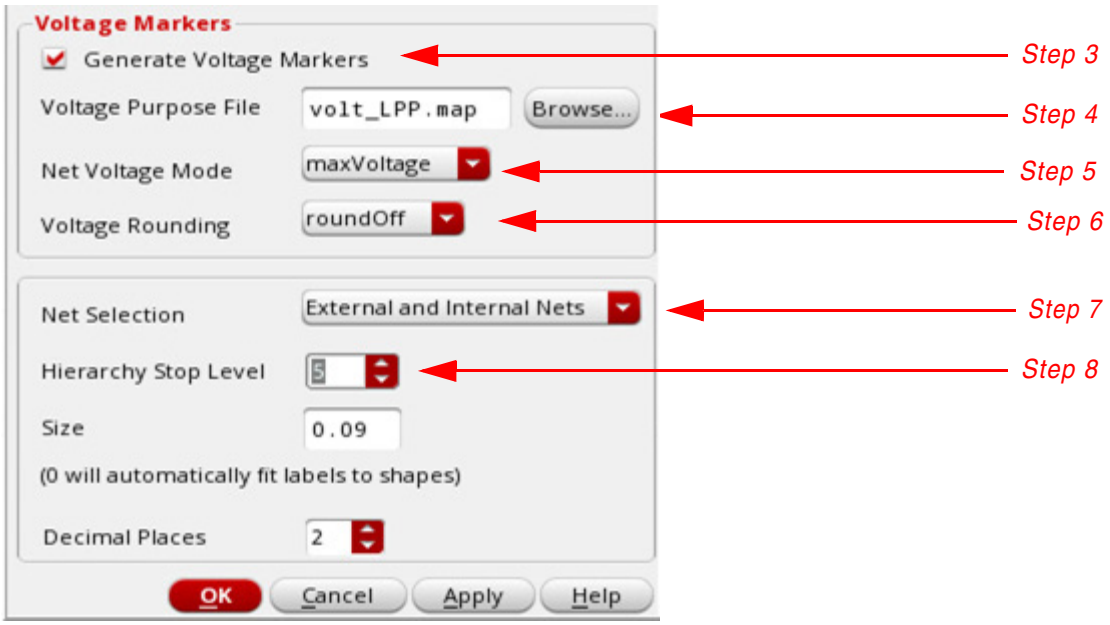
Generating Voltage Markers from Simulation Data for All Nets

To generate voltage markers from simulation data for all the nets in your design:

1. Choose *Tools – Voltage Dependent Rules – Create Labels/Markers From Simulation Voltages* from the layout window menu bar (or type `vdrGenerateLabelsGUI()` in the CIW).

   The Voltage Dependent Rules form appears.

2. Select the datasets and mode as described in Storing Voltages in the OpenAccess Database.

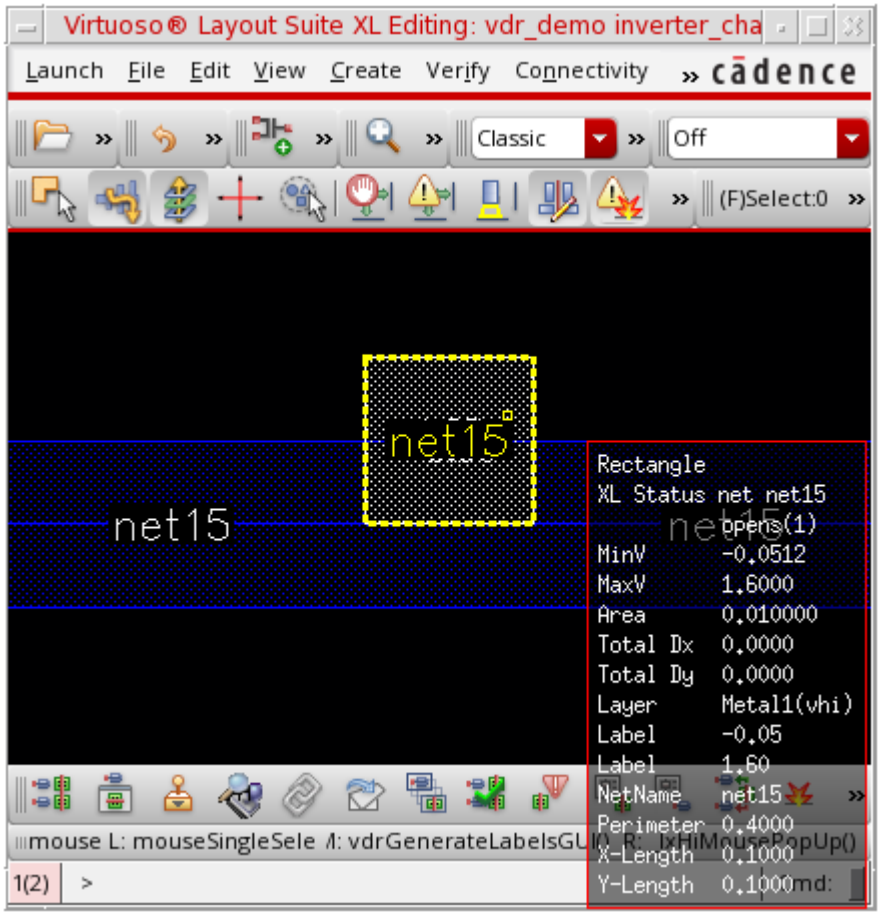


3. Check the *Generate Voltage Markers* box to enable the controls in the *Voltage Markers* group box.

4. Specify the location of the *Voltage Purpose File*, which lists the layer-purpose pairs on which markers for different voltage values are to be created. See Specifying Layers and Purposes for Voltage Markers for more information.

5. Choose whether markers are to be created for maximum, minimum, or all voltage values.
6. Choose the rounding rule to follow for voltage values.
   - *roundOff* rounds the voltage value to the nearest 0.01 (the default)
   - *ceiling* rounds up the voltage value to the nearest 0.01
   - *floor* rounds down the voltage value to the nearest 0.01
7. Specify the nets for which markers are to be generated. Choose between:
   - *External and Internal Nets* to generate labels for all nets (the default)
   - *External Nets Only* to generate labels only for nets that are connected to the terminals of the cellview to which they belong
   - *Internal Nets Only* to generate labels only for nets that are completely internal to the cellview in which they belong
8. Specify how many hierarchy levels to search for nets on which to generate markers.
9. Click *OK* to generate the specified markers.

The voltage markers for the specified nets are added in the layout canvas. See Showing Voltage Information in Info Balloons to learn how to view information about each marker.



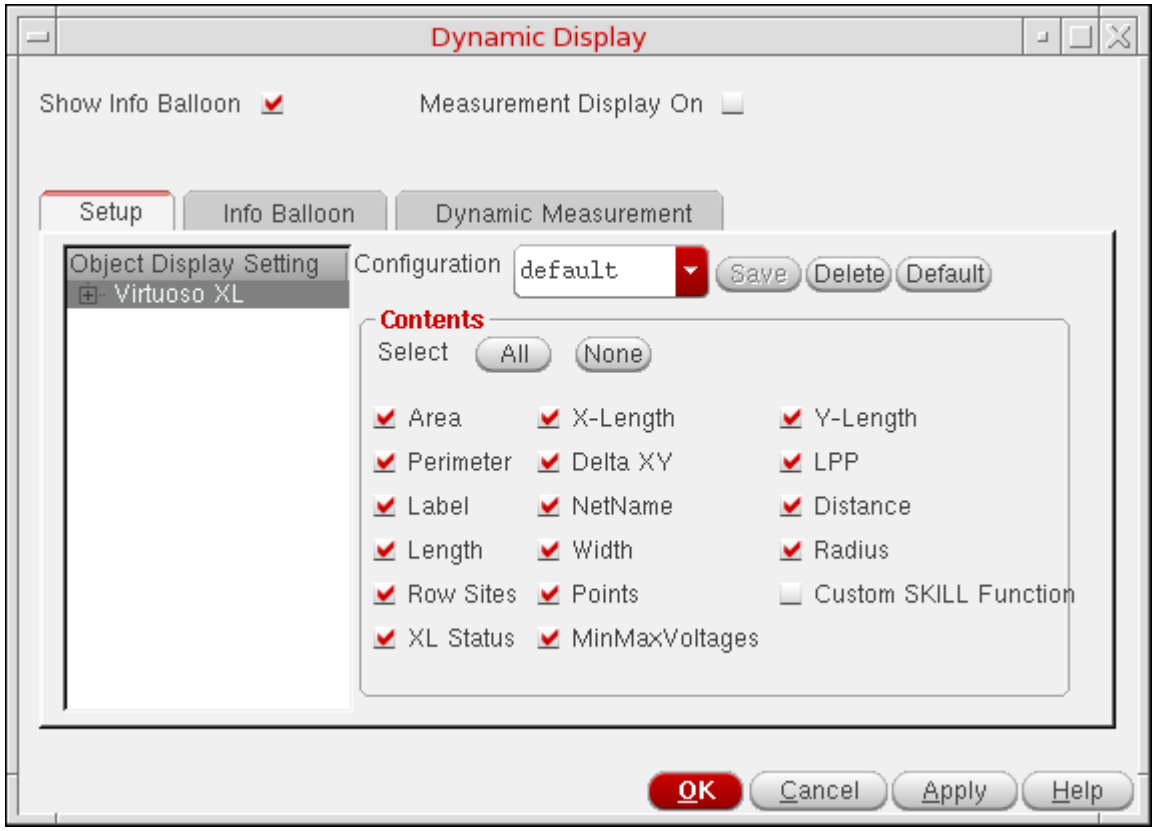
Marker generation depends on the *Mode* setting.

- In *Replace* mode, all voltage markers are replaced for the nets in the datasets.
- In *Update* mode, minimum voltages are updated only if the existing marker shows a higher voltage; maximum voltages are updated only if the existing marker shows a lower voltage.

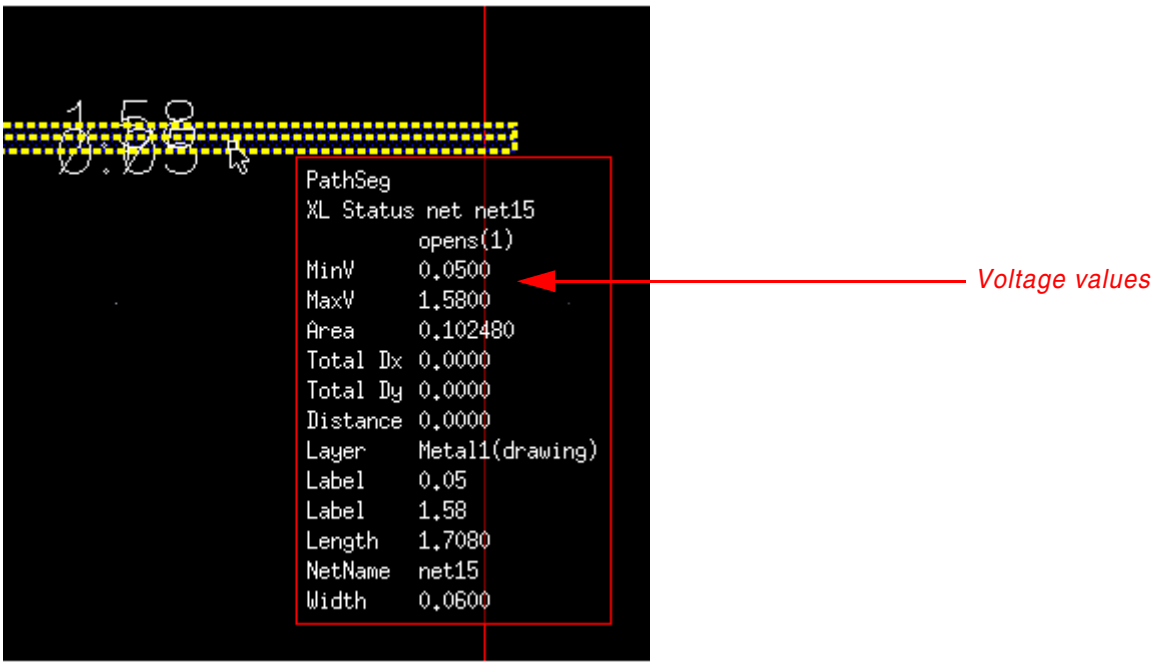
Showing Voltage Information in Info Balloons

You can use the layout editor's *Show Info Balloon* feature to see voltage information when you move the mouse pointer over nets in the layout canvas. To do this:

1. From the layout window menu bar, choose *Options – Dynamic Display* and enable *Show Info Balloon* at the top of the form.
2. Make sure the *MinMaxVoltages* option is selected (this is the default) in the *Contents* pane and click *OK*.



3. Move the mouse pointer over a net in your design to see an info balloon containing the voltage information for the highlighted net.



4. Move the pointer over a different net.

   The original info balloon disappears and a new one opens containing information on the new net.

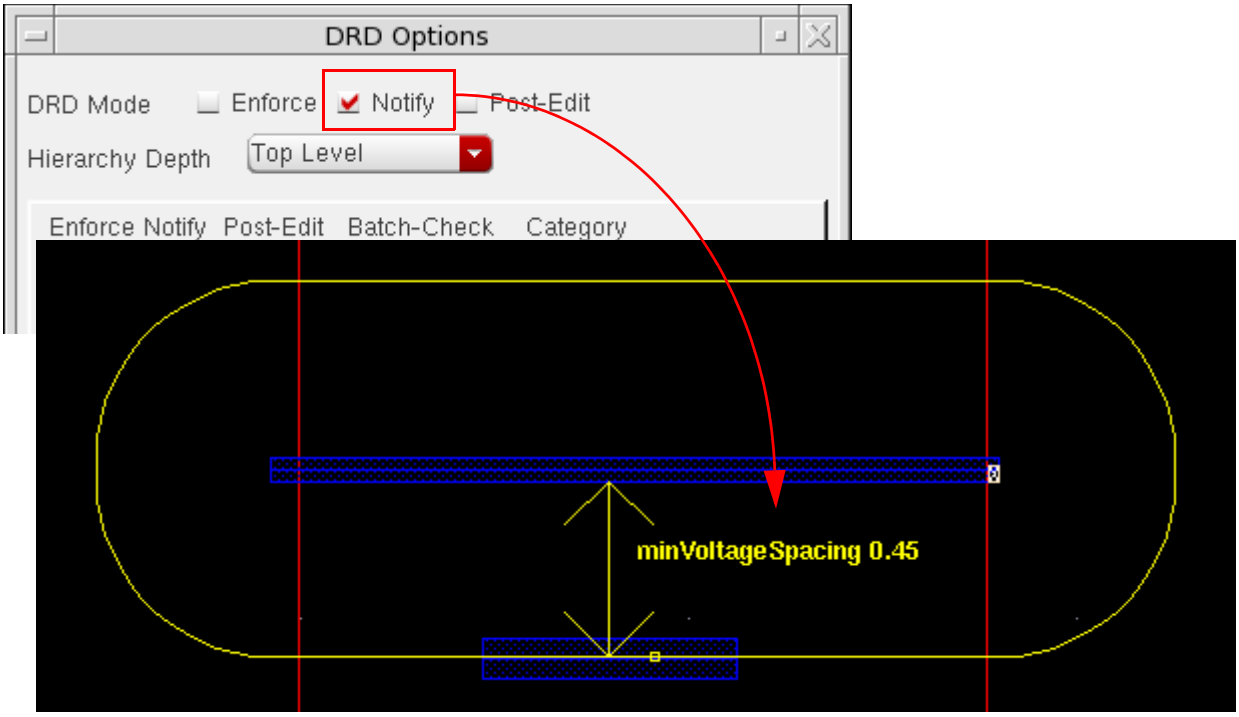
Checking for Voltage Dependent Spacing Violations using DRD

You can use Layout Editor features such as DRD editing to check for and report any minimum voltage spacing violations as you edit your design. To do this:

1. Choose *Options – DRD Edit* from the layout window menu bar.

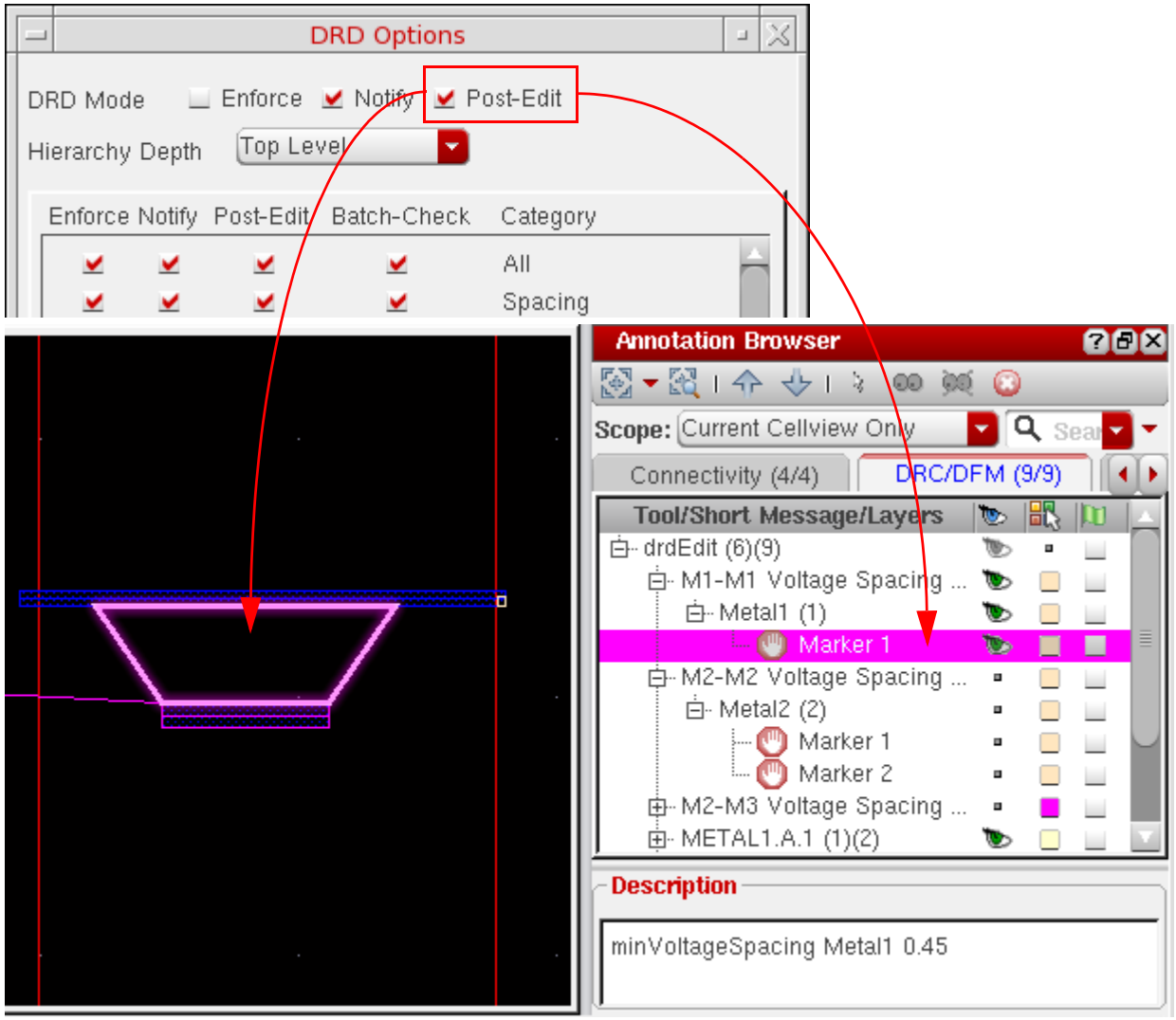
   The DRD Options form appears.

2. Set *Notify* to display interactive notifications of violations as they occur.



If you violate the minimum voltage spacing rules during interactive editing, the system provides immediate visual feedback on the layout canvas.

3. Set *Post-Edit* to see violation markers on the canvas and Annotation Browser assistant after you have finished editing.



For detailed information on design rule driven layout editing, see the *Virtuoso Design Rule Driven Editing User Guide*.

Customizing the Voltage Calculation

Voltage values captured during a transient simulation can sometimes contain short pulses (spikes) that distort the final result. You can exclude these spikes from the final voltage calculation by writing your own custom SKILL procedures to perform the required filtering operations on the voltage waveforms before the minimum and maximum voltage values are extracted. To do this:

1. Write two custom SKILL procedures, one for the calculation of Vmax and one for the calculation of Vmin, which define the filtering you need.

   The basic structure of the procedures is as indicated below:

```
procedure( My_Vmax(w)
    <perform filtering operations>
        <return Vmax>
); end procedure

procedure( My_Vmin(w)
    <perform filtering operations>
        <return Vmin>
); end procedure
```

2. Assign the custom SKILL procedures to the environment variables indicated below by typing the following in the CIW:

```
envSetVal("elec.gui" "customVmaxCalc" 'string "My_VMax")
envSetVal("elec.gui" "customVminCalc" 'string "My_VMin")
```

   For details of these variables, see customVmaxCalc and customVminCalc in the *Virtuoso Electrically Aware Design Flow Guide*.

3. Switch on the custom voltage calculation feature by typing the following in the CIW:

```
envSetVal("elec.gui" "enableCustomVminVmaxCalc" 'boolean t)
```

   For details of this variable, see enableCustomVminVmaxCalc in the *Virtuoso Electrically Aware Design Flow Guide*.

4. Run a transient simulation with voltage capture enabled in the EAD Setup form.

   Virtuoso replaces the standard voltage calculation with the procedures defined in the custom SKILL procedures.

Schematic Driven VDR Flow

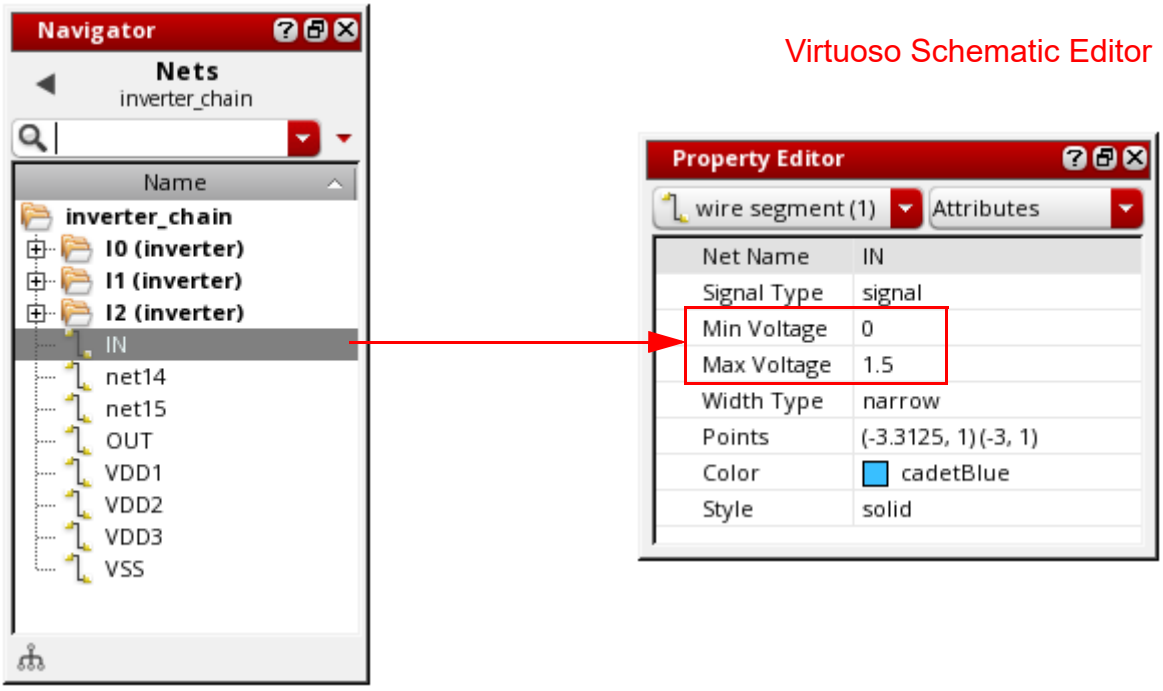
You can also populate the OpenAccess database for your layout design with voltage data by manually entering the minimum and maximum voltages as properties on the nets in the schematic.

When you generate or update your layout view from the schematic, the voltage values are propagated to the layout design and made available to DRD editing, the interactive wire editor, and the Virtuoso space-based router, which you can use to ensure that the minimum voltage spacing constraints defined in the technology file are honored.

Propagating Voltage Values from Schematic to Layout

To define voltage values in the schematic and propagate them to the layout view:

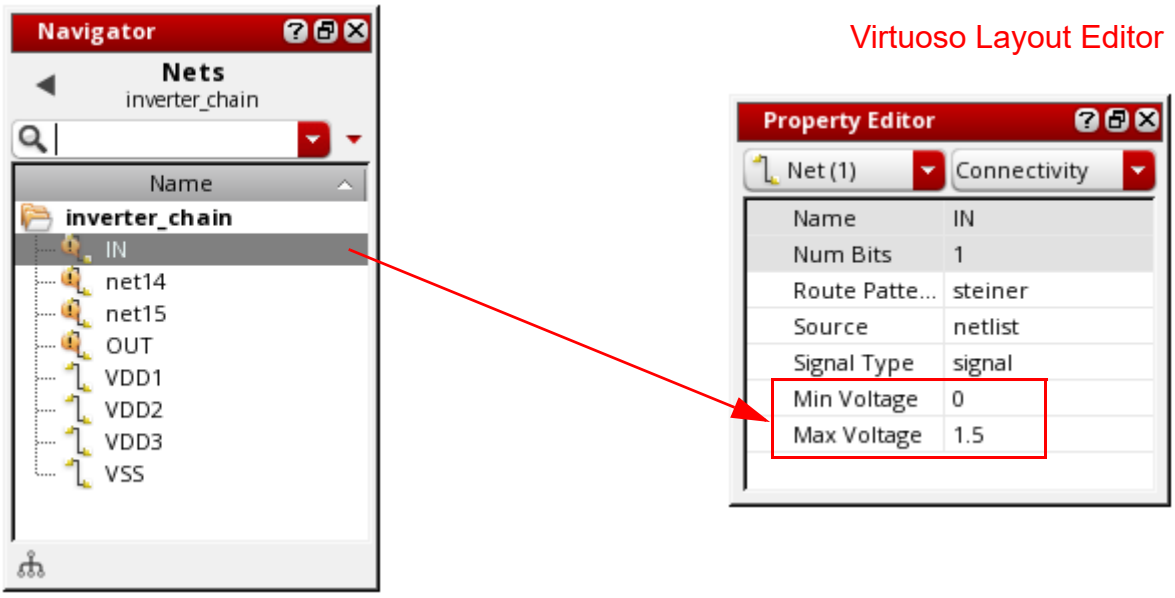
1. Launch Layout XL and open the schematic view in the Virtuoso Schematic Editor.



2. Select the net you require in the Navigator assistant.
3. Type the *Min Voltage* and *Max Voltage* values into the Property Editor assistant.
4. Choose *File – Save* to save the schematic the do one of the following,
   - ❑ To create a new layout, choose *Connectivity – Generate – All From Source* from the layout window menu bar.

- To update an existing layout, see Updating Voltage Values in the Layout to Match the Schematic.

The layout view is created or updated and the new minimum and maximum voltage values are transferred from the schematic nets to the layout nets.



Layout editor tools such as DRD, the interactive wire editor, and VSR consider the voltages when checking that minimum voltage spacing constraints defined in the technology file are not being violated. See Specifying a Minimum Voltage Spacing Constraint for more information.

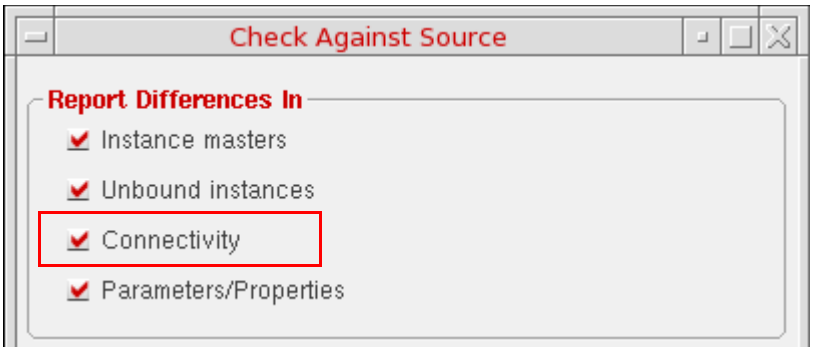
For information on how to generate labels or markers for these values, see

- Generating Voltage Labels for Manually Entered Voltages
- Generating Voltage Markers for Manually Entered Voltages

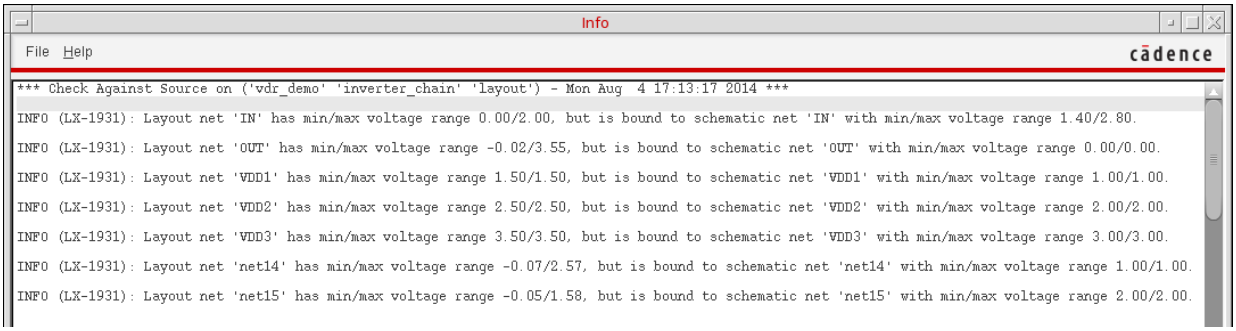
Checking Voltage Values between Schematic and Layout

You can use the Layout XL *Check Against Source* command to ensure that the voltage values specified in the schematic match those in the corresponding layout:

1. In the layout window menu bar, select *Connectivity – Check – Against Source*.
2. Make sure the *Connectivity* option is selected in the Check Against Source form and then click *OK*.



Layout XL checks the schematic against the layout and reports any mismatches found:

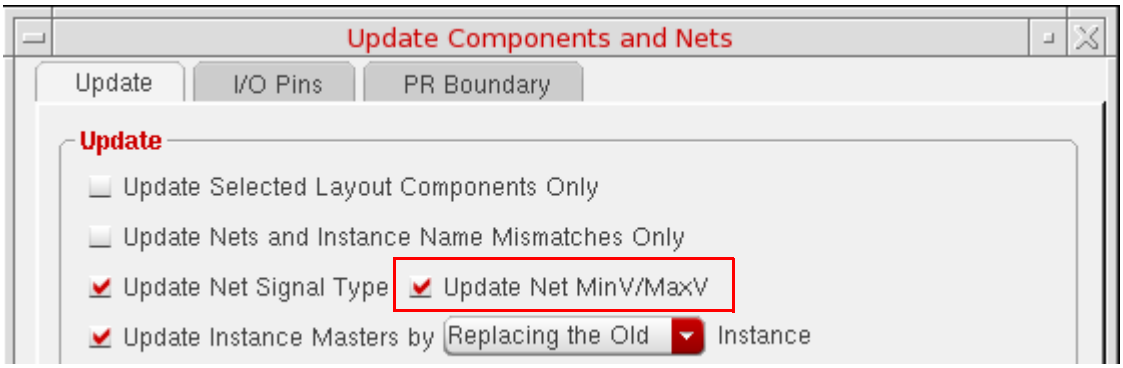


Updating Voltage Values in the Layout to Match the Schematic

To update the values in the layout to match those in the schematic:

1. Choose *Connectivity – Update – Components And Nets* from the layout window menu bar.

2. Ensure the *Update Net MinV/MaxV* option is selected, and click *OK*.



   The layout view is updated with the minimum and maximum voltage values from the schematic nets and any existing voltage labels for that net in the layout canvas are updated automatically.

Note: The *Update Net MinV/MaxV* option is switched on by default. To switch it off by default, set the `updateNetMinMaxVoltage` environment variable to `nil` in your `.cdsenv` file.

Backannotating Voltage Values from Layout to Schematic

When working on the physical implementation of your design in Layout XL, it might be necessary to update the voltage values for certain nets to meet specific design requirements. To update the voltage values in the schematic to match those in the layout:

➡ Choose *Connectivity – Back Annotate – Net MinV/MaxV* from the layout window menu bar.

   Layout XL backannotates the minimum and maximum voltage values for all the top-level nets in the layout to the corresponding nets in the schematic.

Note: When backannotating voltage values for buses, provided you have set the same minV/maxV value pair for all the bits of a given bus in the layout, backannotation will be performed to the corresponding schematic bus.

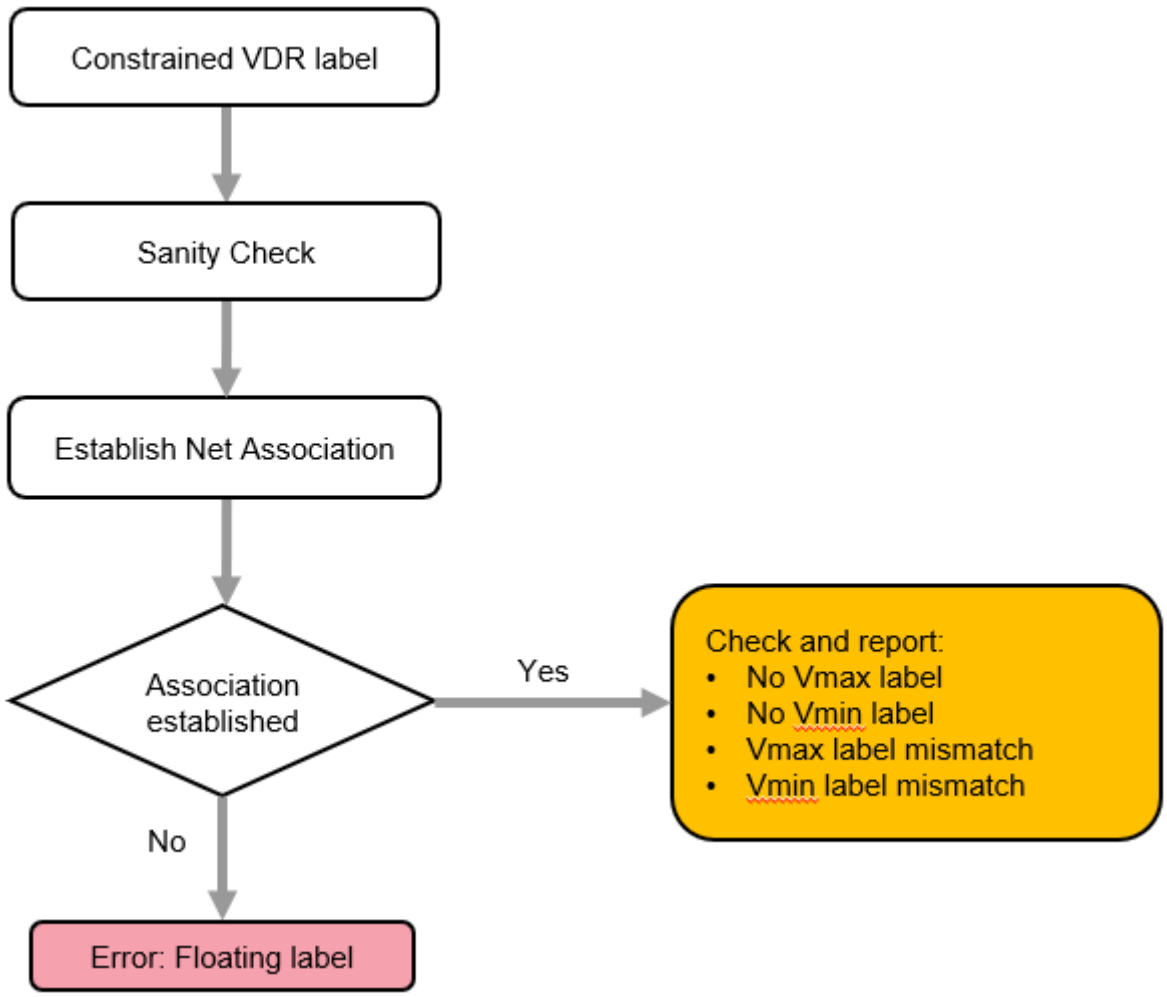
If you have set different minV/maxV value pairs on the different bits of a layout bus and the corresponding schematic bus is not expanded, then Layout XL takes the highest maximum voltage value and the lowest minimum voltage value across all the bits of the layout bus and backannotates that minV/maxV value pair to the corresponding unexpanded bus in the schematic.

Sanity Checking Voltage Values in Constrained Labels

You can use the VDR Sanity Checker to check constrained VDR voltage labels in the layout view against the values stored in the schematic or layout net properties and report any discrepancies between them. Missing labels, labels with value differences greater than a specified tolerance, and multiple labels with different values on a single net are reported either in a user-specified log file or printed in the CIW and listed in the Annotation Browser.

Note: In ICADVM20.1 Layout EXL, you can additionally check that vsync shapes in the layout canvas all have a corresponding *Voltage Synced Nets* constraint in the layout view. See Checking Synced Nets in the Layout View (ICADVM20.1 EXL Only) for more information.

The constrained VDR label sanity check flow is illustrated below.



Establishing the Net Association

The label-net association for constrained labels is established by searching the design for a net shape that overlaps the label shape in question.

- If a net shape is found at the top level, Virtuoso reads the net name from that shape.
- If no net shape is found at the top level, Virtuoso traverses the hierarchy to establish if there is an underlying terminal or pin shape inside the instance and, if so, makes the association to the top-level net connected to that terminal or pin shape.
- If a non-terminal/non-pin net shape is found one level down the hierarchy, or any net shape is found at two or more levels down the hierarchy, the label is ignored on the assumption that it is intended for some hierarchical net.
- If no overlapping net shape is found in the design, including cases where the label is created on a via layer and does not overlap any net, the label is tagged as floating.

Types of Violations

Type	Description	Notes
No Vmax label	There is a maximum voltage defined for a net in the design but no corresponding label drawn on the canvas.	Although the sanity checker expects only one label each for Vmax and Vmin for each net, multiple Vmax or Vmin labels on a net are treated as a single label provided the values are the same. Markers are drawn on one of the net shapes.
No Vmin label	There is a minimum voltage defined for a net in the design but no corresponding label drawn on the canvas.	
Vmax label mismatch	The maximum voltage value in the label is different from the value specified in the design.	For both mismatch violations, the values are first rounded to two decimal places and the specified tolerance applied. Markers are drawn on the label in question.
Vmin label mismatch	The minimum voltage value in the label is different from the value specified in the design.	
Floating label	If a label does not overlap any net or instance or the instance terminal is missing and no net association is possible, the label is a *floating label*.	Markers for these violations are drawn on the label in question.

Performing a Label Sanity Check

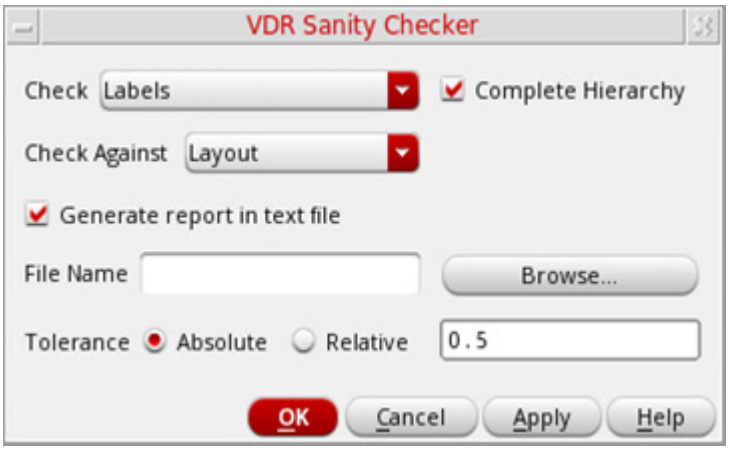
To perform a sanity check of constrained VDR labels:

1. Specify the vdrConstraintGroupName environment variable before you launch Virtuoso.

   The specified constraint group must contain a valid `voltageLabelMapping` constraint to specify the layer-purpose pair on which labels are drawn.

2. Choose *Tools – Voltage Dependent Rule – Sanity Checker* from the layout window menu bar.

   The VDR Sanity Checker form appears.



3. From the *Check* drop-down list, select *Labels*.
4. Select the *Complete Hierarchy* check box to perform checks on the complete hierarchy. If you do not select this check box, checks are performed only on the current hierarchy.

   Note: Sanity checker checks a cell only once even if it is instantiated more than once in the same or different hierarchy to help save run time.

5. From the *Check Against* drop-down list, specify whether to check against the voltage values stored in the schematic, layout, CSV file, or dataset.

   - If you want to check the voltage values against the values in a CSV file, specify the path to the CSV file in the *CSV File Name* field or use the *Browse* command to select the CSV file.
   - If you want to check the voltage values against the values in a dataset, select the dataset from the *Datasets* list.

6. Specify a tolerance within which any differences are ignored.

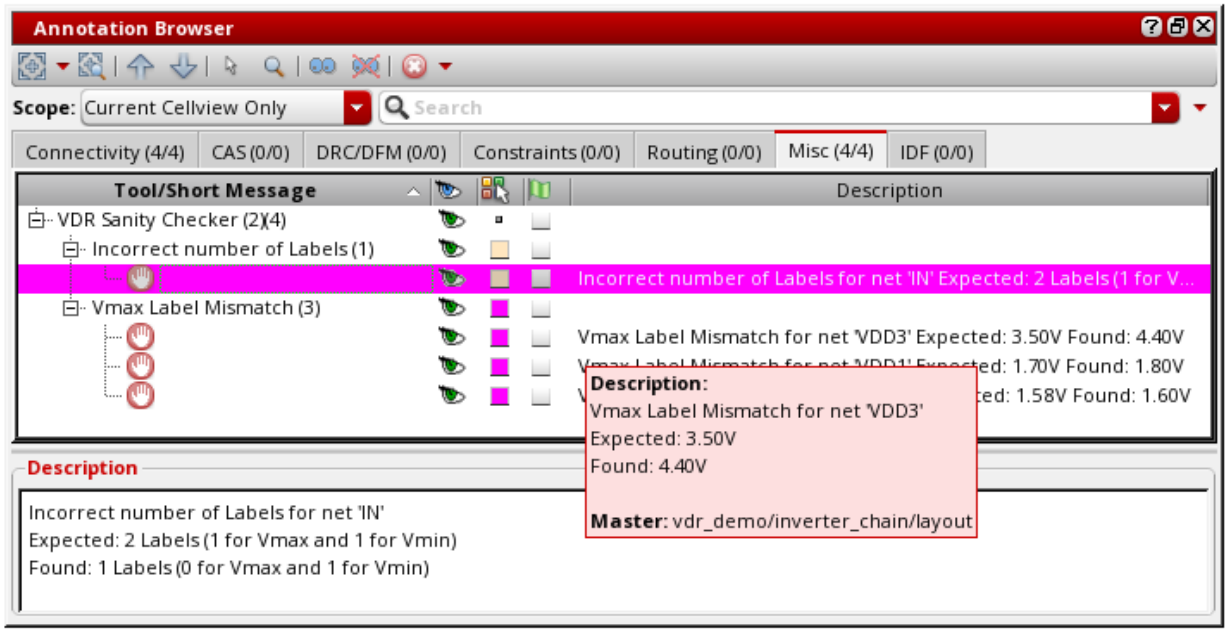
   You can also specify whether the tolerance value is absolute or relative to the voltage.

7. Specify whether the results are to be captured in a text file and click *OK* to perform the sanity check.

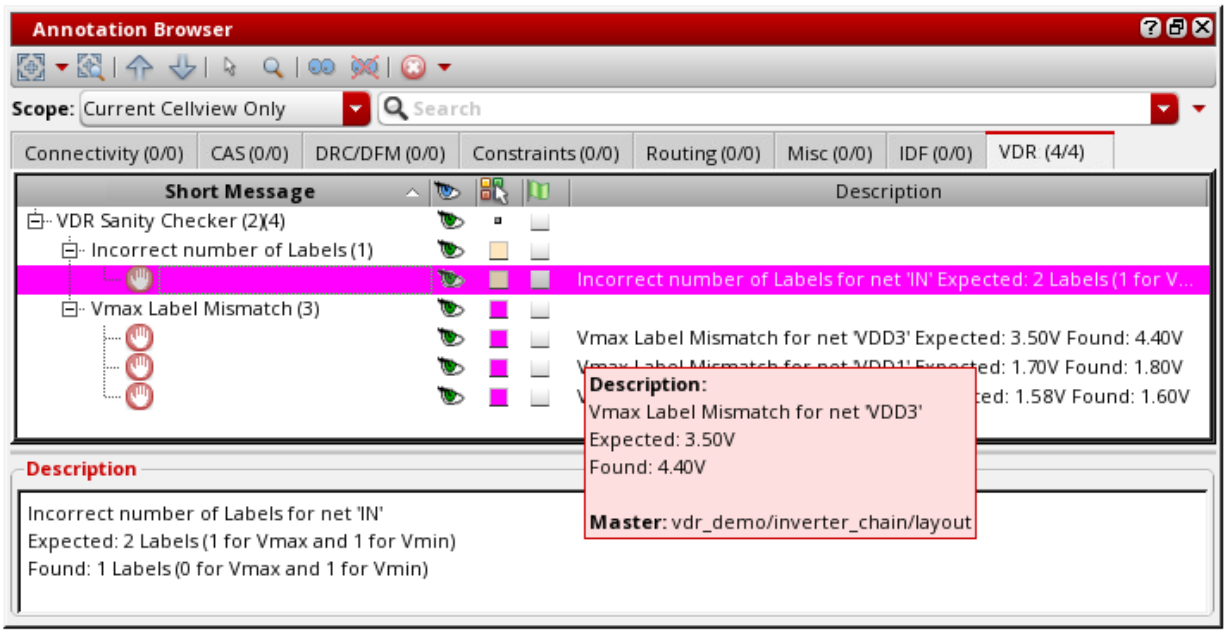
   When you capture the results in a log file, only a summary message is printed in the CIW. If you do not specify a log file, discrepancies are reported in a table printed in the CIW. In both cases, the format is similar to that shown below. Values are rounded to two decimal places before comparison.

```
=============================================================================
NetName        Net Voltage Values      Label Voltage Values
               (Vmin)   (Vmax)         (Vmin)  (Vmax)
=============================================================================
net15          (-0.05)  (1.57)         (0.05)  (1.75)
net14          (1.25)   (2.56)         (1.15)  (2.35)
=============================================================================
```

   Corresponding markers are drawn on the canvas and entries are displayed on the *Misc* tab in the Annotation Browser assistant. When you select a violation in the assistant, Virtuoso zooms to the marker in question on the canvas.



In ICADVM20.1 EXL, entries for these markers are displayed on the *VDR* tab instead of *Misc* tab in the Annotation Browser.



Tip

You can also run the sanity check procedurally using the vdrRunSanityChecker SKILL API.

When analyzing sanity checker output, keep the following in mind:

- *Zero Voltage Nets* specified by the user during label generation are ignored by the sanity checker provided there are either no labels at all on the specified net or both Vmin and Vmax values are set to 0. If one or both values is nonzero, or the number of labels is other than 0 or 2, the sanity checker reports an error.
- Where there is no geometry for a net on the canvas, and labels have been generated on all the Pcell and instance terminals connected to the net, there could be multiple discrepancies related to incorrect numbers of labels or label mismatches. The sanity checker reports such discrepancies in terms of both instance name and net name to allow them to be easily identified.
- Sanity Checker reports VDR objects (voltage labels, voltage markers, VSync shapes, and Userdv shapes) found inside the shared cells. This helps in reminding users to delete these VDR objects from shared cells.
- Sanity Checker does not perform any checks on existing VDR labels inside the shared cells.

Related Topics

vdrSharedCellList

Rules for Creating Voltage Labels in Shared Cells

Defining and Checking Voltage Synced Nets

Some processes feature the concept of *synced nets*, the voltages of which must always transition in phase with each other. The voltage difference that triggers a DRC violation for synced nets is much lower than in conventional processes. To support the enhanced voltage check, you can:

1. Define *Voltage Synced Nets* constraints in the schematic for the nets whose voltages must transition in phase.
2. Generate vsync shapes connecting these synced nets in the layout view.

   Note: Vsync shape generation is instance-based. Markers are created at the top level for synced nets inside the instance hierarchy.
3. Use the VDR Sanity Checker to check that the vsync shapes match the constraints in the layout.

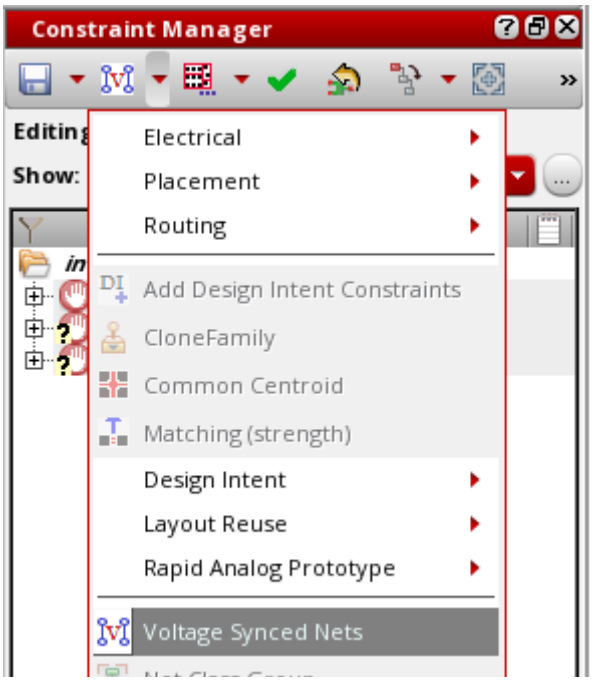
Note: This functionality is available only in Layout EXL when the vdrConstraintGroupName environment variable is set.

Creating a Voltage Synced Nets Constraint By Using the Constraint Manager (ICADVM20.1 EXL Only)

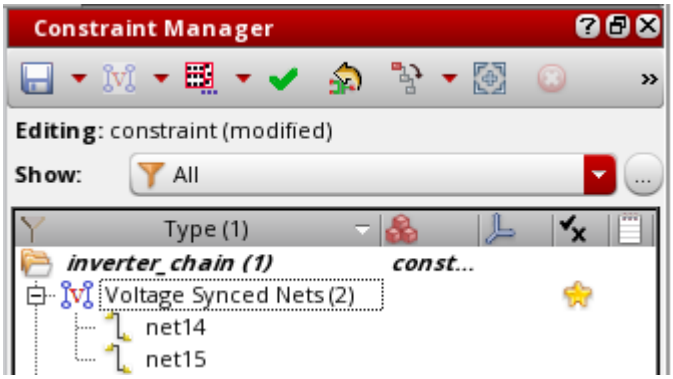
To create a *Voltage Synced Nets* constraint using the Constraint Manager:

1. Select the pair of nets to be tagged as synced in the schematic Navigator assistant.

2. In the Constraint Manager assistant, choose *Voltage Synced Nets* from the toolbar:



The *Voltage Synced Nets* constraint is created on the selected nets.



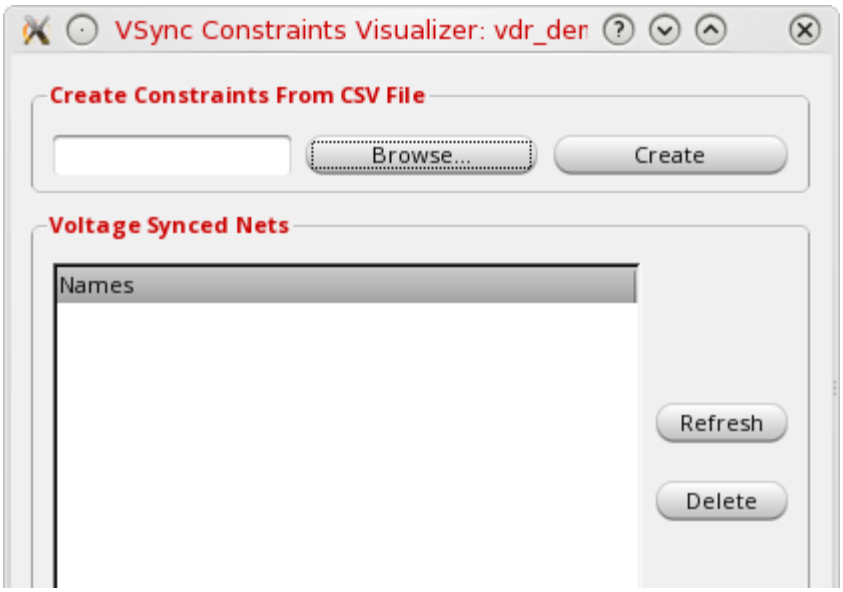
See Voltage Synced Nets in the *Virtuoso Unified Custom Constraints* for more information about the constraint.

Creating Voltage Synced Nets Constraints from a File (ICADVM20.1 EXL Only)

You can also create *Voltage Synced Nets* constraints from a comma-separated file capturing the nets to be constrained and the minimum and maximum voltages for those nets. Each line in the file creates a Voltage Synced Nets constraint with members as defined by the comma-separated nets captured in that line. To do this:

1. Choose *Tools – Voltage Dependent Rules – VSync Visualizer* from the layout window menu bar.

   The VSync Constraints Visualizer form appears.



2. Browse to the location of the file you require and click *Create* to create constraints from the information in that file.
3. (Optional) Click *Refresh* to update the list of voltage synced nets currently in the design.
4. (Optional) Click *Delete* to remove the selected voltage synced nets from the design.

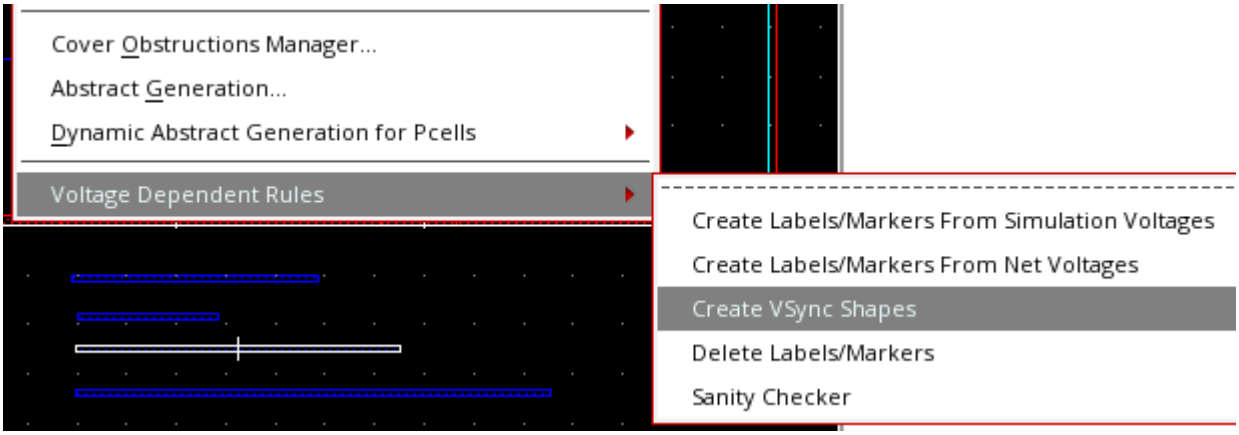
Tip

You can perform the same operation procedurally using the vdrCreateVSyncConstraintsFromFile SKILL API.

Generating Vsync Shapes in the Layout View (ICADVM20.1 EXL Only)

To generate vsync shapes in the layout view:

➡ Choose *Tools – Voltage Dependent Rules – Create VSync Shapes* from the layout window menu bar.



Note: If the vdrConstraintGroupName environment variable is not set, the menu item is grayed out.

Virtuoso removes all existing vsync shapes created by the tool and generates one vsync shape for each pair of nets tagged in a *Voltage Synced Nets* constraint in the layout view.

- For pairs of nets on the *same layer*, the vsync shapes are created on the layer and purpose defined in the voltageLayerMarkerMapping technology file constraint.
- For pairs of nets on *different layers*, the vsync shapes are created on the layers and purposes defined in the voltageLayerPairMarkerMapping technology file constraint.
- If there are multiple possible layers on which the shapes could be created, the vsync shapes are created on the lowest layer to preserve routing resources on higher metal layers.

See Specifying Layers and Purposes for Synced Nets for more information.

Checking Synced Nets in the Layout View (ICADVM20.1 EXL Only)

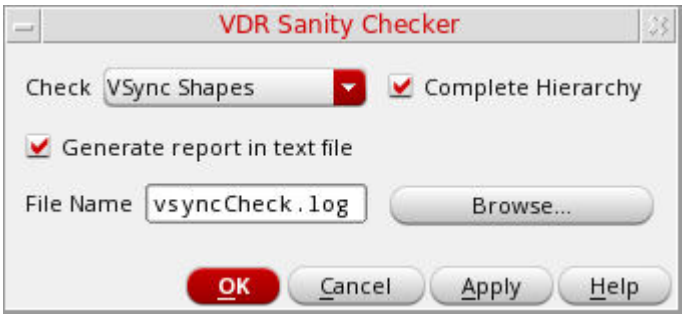
You can use the VDR Sanity Checker to check that each vsync shape on the canvas corresponds to a *Voltage Synced Nets* constraint in the layout.

Note: The sanity checker checks vsync shapes against constraints in the layout view, not in the schematic, so you must first ensure that *Voltage Synced Nets* constraints defined in the schematic are current in the layout view.

To do this:

1. In the layout window, choose *Connectivity – Update – Layout Constraints* to transfer the constraints from the schematic view to the layout view.
2. Choose *Tools – Voltage Dependent Rules – Sanity Checker* from the layout window menu bar.

   The VDR Sanity Checker form appears.



3. Select *VSync Shapes* from the *Check* drop-down list.

   Note: If the `vdrConstraintGroupName` environment variable is not set, the menu item is grayed out.
4. Select the *Complete Hierarchy* check box to perform checks on the complete hierarchy. If you do not select this check box, checks are performed only on the current hierarchy.
5. Select the *Generate report in text file* check box to captured results in a text file.

   When you capture the results in a log file, only a summary message is printed in the CIW. If you do not specify a log file, discrepancies are reported in a table printed in the CIW.
6. Click *OK* to check the vsync shapes in the design.

   Two types of errors are reported:

   - Vsync shapes with no corresponding *Voltage Synced Nets* constraint in the layout.

     Markers for this violation type are created on the vsync shape in the layout view. The text report prints the bounding box of the vsync shape.
   - A *Voltage Synced Nets* constraint exists in the layout, but there is no corresponding vsync shape.

     Markers for this violation type are created on the corresponding cell to reduce visual clutter on the canvas. The description in the Annotation Browser provides the names of the nets in the constraint. The text report captures the constraint name and its member nets.

   Note: Sanity Checker reports VDR objects (voltage labels, voltage markers, VSync

shapes, and Userdv shapes) found inside the shared cells. This helps in reminding users to delete these VDR objects from shared cells.

Creating Delta Voltage Constraints between Nets

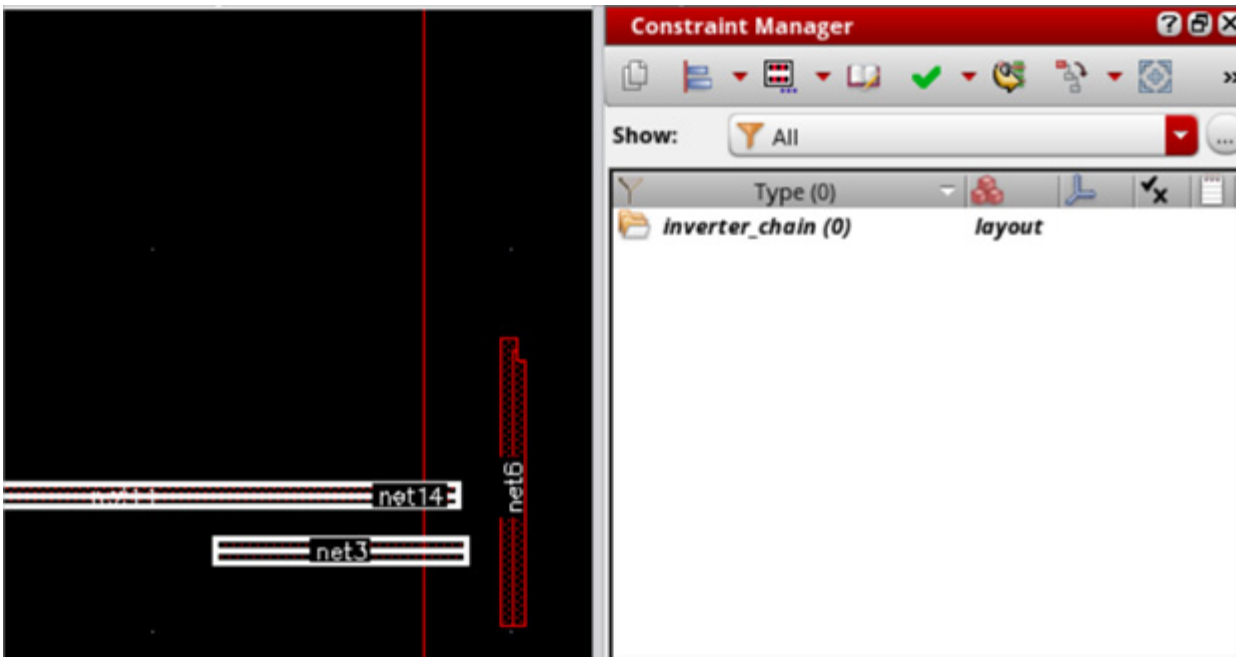
You can create a Delta Voltage constraints between nets by using either the Constraint Manager assistant or the CSV file that contains information about nets and delta voltage values.

The entries for the nets and delta voltage values between them in the CSV file are as follows:

```
# Comma-separated nets and delta voltage values
net14, net3, 0.4
net14, net6, 0.6
net3, net6, 0.8
```

To create a delta voltage constraints between nets using the Constraint Manager assistant:

1. Select the pair of nets between which you want to create a delta voltage constraints in the Layout EXL or MXL. For example, `net14` and `net3`.

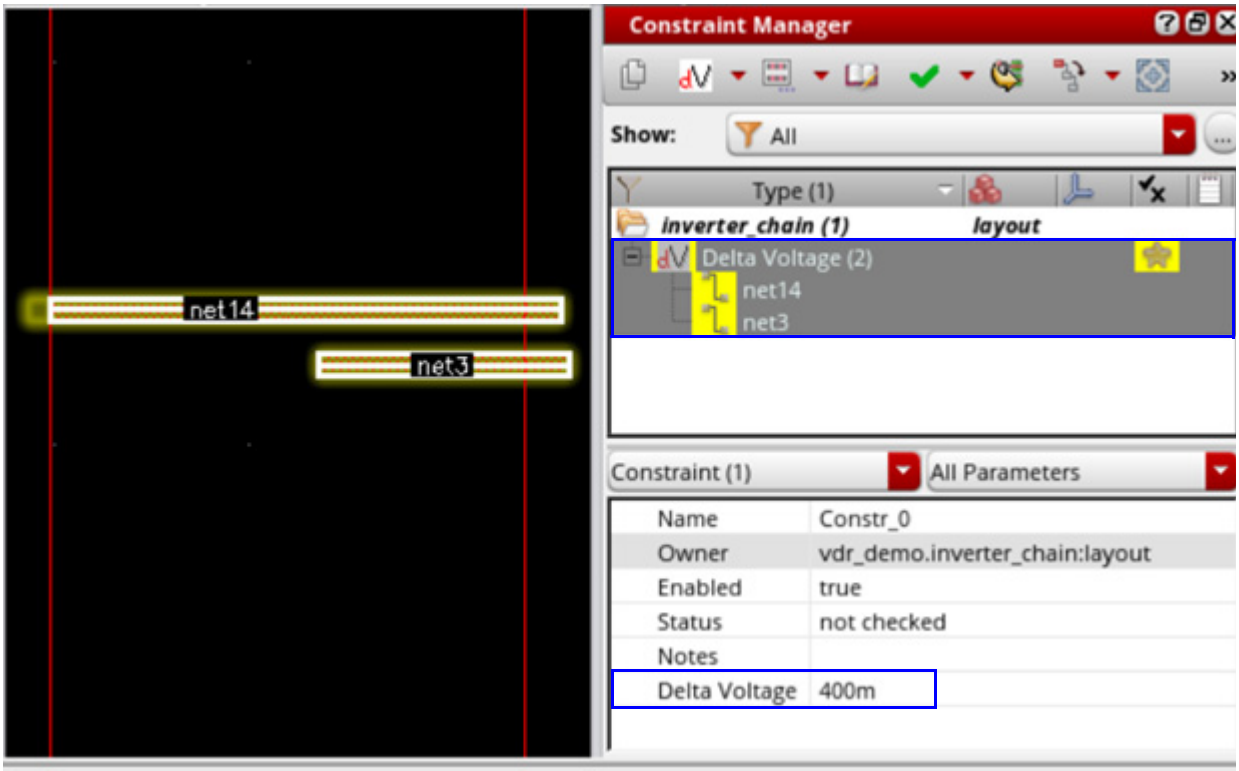


2. From the toolbar of the Constraint Manager, click the arrow next to the *Alignment* icon to display additional options.
3. Select *Electrical – Delta Voltage*.

The Delta Voltage dialog box appears.

4. In the *Delta Voltage* field, specify the delta voltage value. For example, `0.4`.
5. Click *OK*.

   The delta voltage is created between the selected nets, `net14` and `net3` with a delta voltage value of `0.4` (`400m`), as shown in the following figure.



To create delta voltage constraints between the nets with the delta voltage value specified using the CSV file:

1. Open the design in Layout EXL and MXL. In this example, the library cellview is `vdr_demo/inveret_chain/layout`.
2. Create a CSV file that contains the information about nets and delta voltage values between them.

```
# Comma-separated nets and delta voltage values
net14, net3, 0.4
net14, net6, 0.6
net3, net6, 0.8
```

3. Save this CSV file in a directory. In this example, the CSV files is saved with the name `userdvData.csv` in the current working directory.

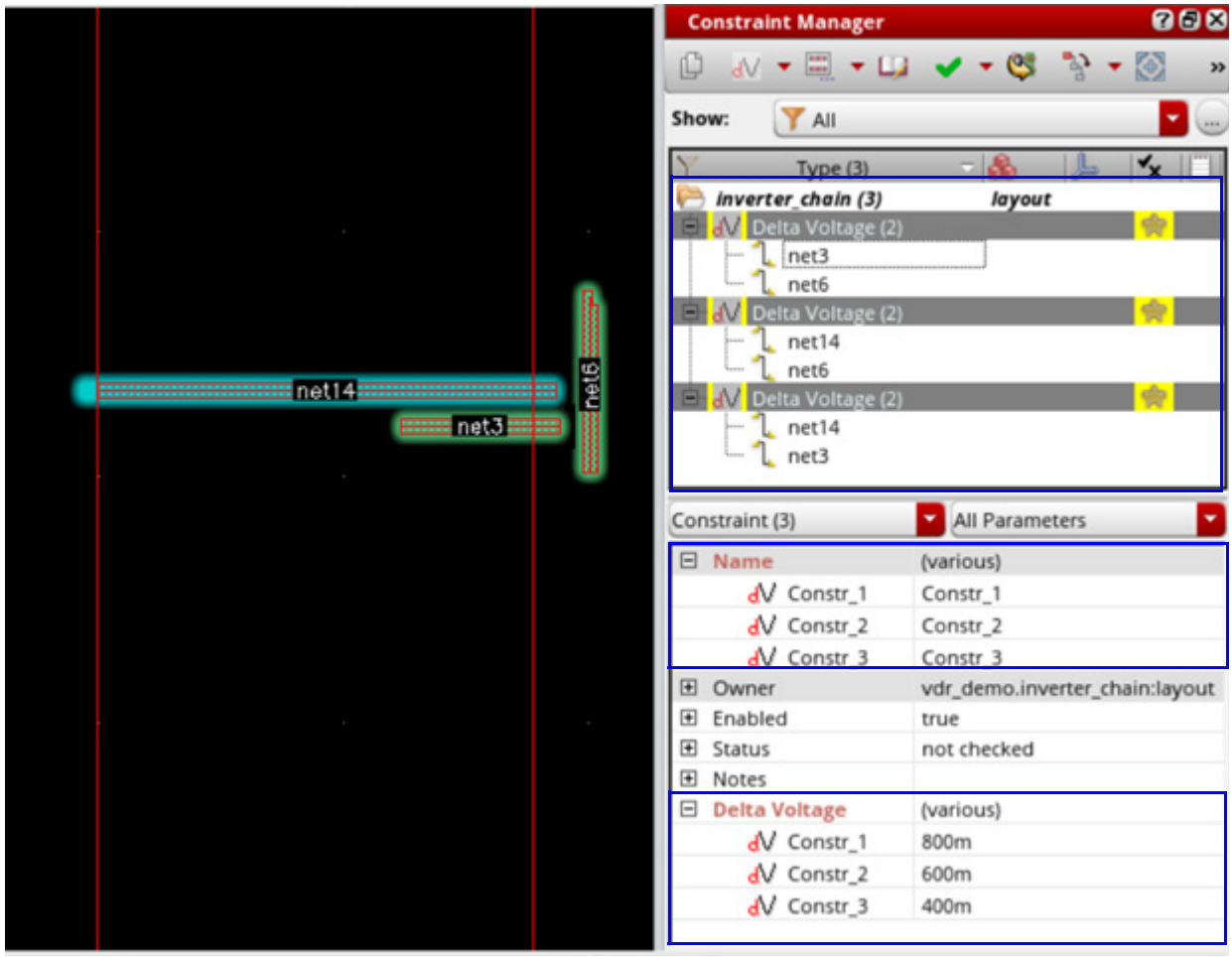
4. In the Virtuoso Studio CIW, run the following command to create delta voltage constraints using the function `vdrCreateUserdvConstraintsFromFile` from the the information contained in the CSV file.

```
scv = dbOpenCellViewByType(
                            "vdr_demo"
                            "inverter_chain"
                            "layout"
                            "" "r"
                            )
srcCache = ciCacheGet(scv)
;gives constraint cache of cellview
listOfCons = nil
foreach(elm srcCache~>constraints
          when(elm~>type == 'deltaVoltage listOfCons = cons(elm listOfCons)))
vdrCreateUserdvConstraintsFromFile("vdr_demo" "inverter_chain" "layout" "./
userdvData.csv")
;Creates User Delta Voltage constraints for the specified nets in the file
userdvData.csv file in the library cellview: vdr_demo inverter_chain layout
```

The following output is printed in the Virtuoso Studio CIW and delta voltage constraints created between the nets are displayed in the Constraint Manager assistant.

```
=> db:0x2b6dd01a
=> ci:0x3850fa80
=> nil
=> nil
=> t
=>
INFO (CMGR-5020): Created constraint of type 'deltaVoltage' in cache 'vdr_demo
inverter_chain layout' with name 'Constr_1'.
INFO (CMGR-5020): Created constraint of type 'deltaVoltage' in cache 'vdr_demo
inverter_chain layout' with name 'Constr_2'.
INFO (CMGR-5020): Created constraint of type 'deltaVoltage' in cache 'vdr_demo
inverter_chain layout' with name 'Constr_3'.
```

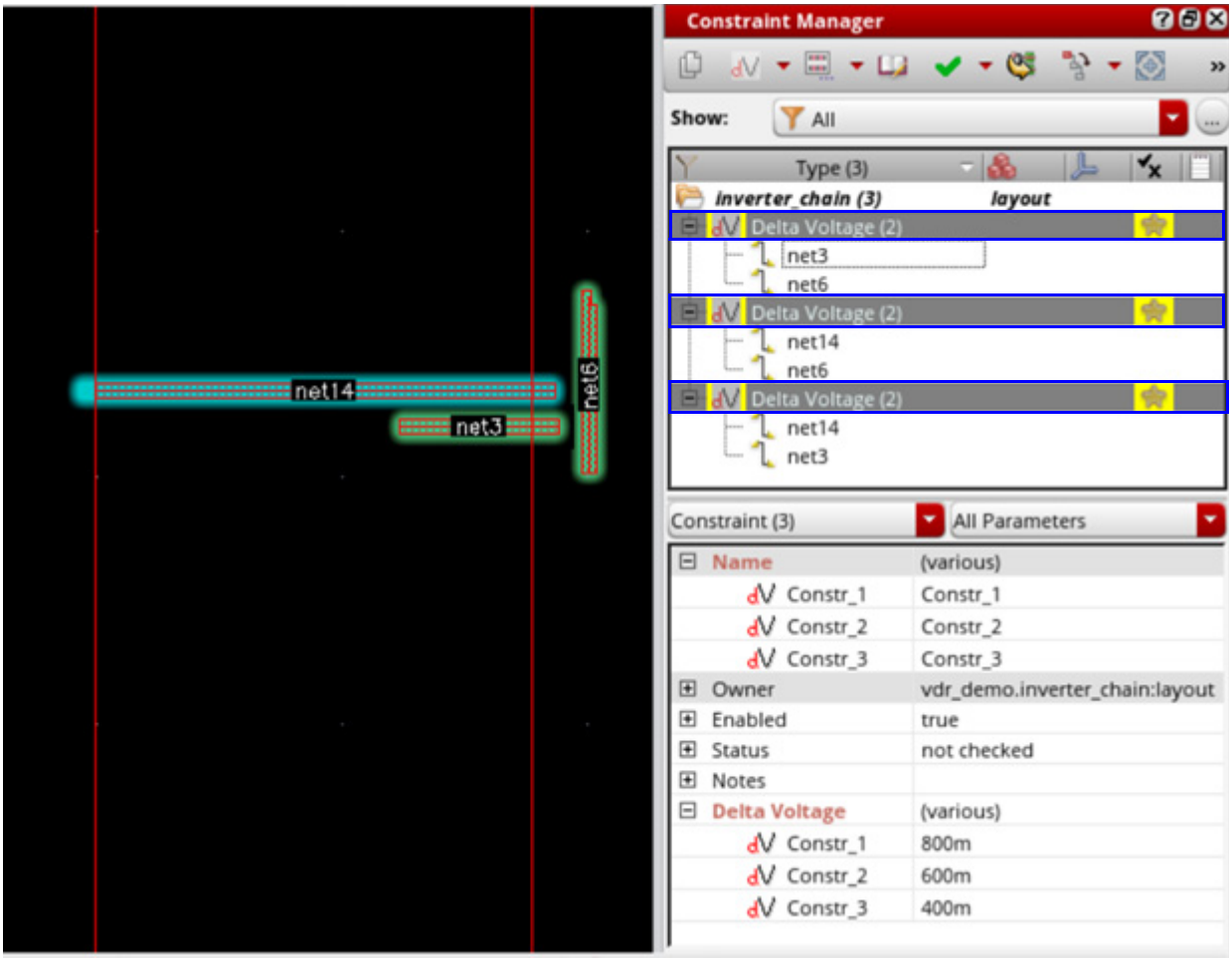
```
=> t
```



Creating Userdv Shapes in the Layout View

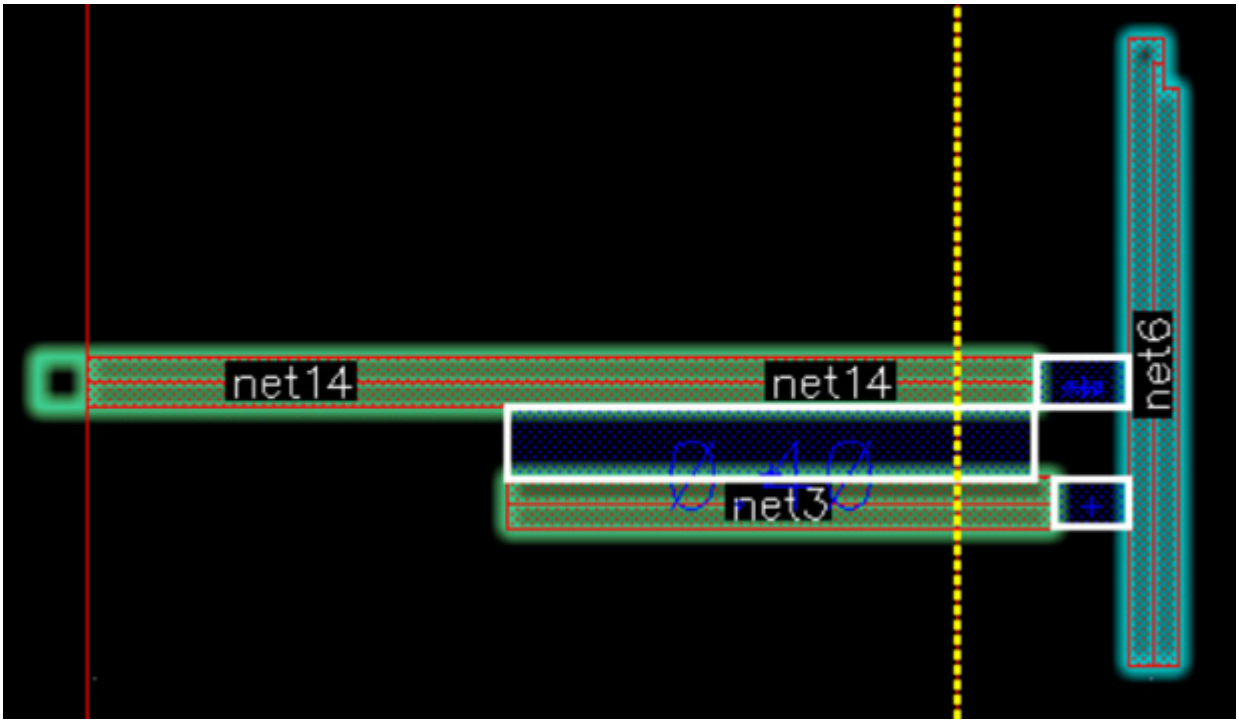
To create Userdv shapes between the nets constrained by the delta voltage constraints in the Layout view:

1. Create the delta voltage constraints between the nets by using the Constraint Manager assistant or by using a CSV file that contains the information about the nets and their delta voltage values.



2. Select the delta voltage constraints in the Constraint Manager assistant.
3. From the menu bar of Layout EXL or MXL, select *Tools – Voltage Dependent Rules – Create Userdv Shapes*.

The Userdv shapes between the nets constrained by the delta voltage constraints are created.



The following information is also printed in the Virtuoso Studio CIW.

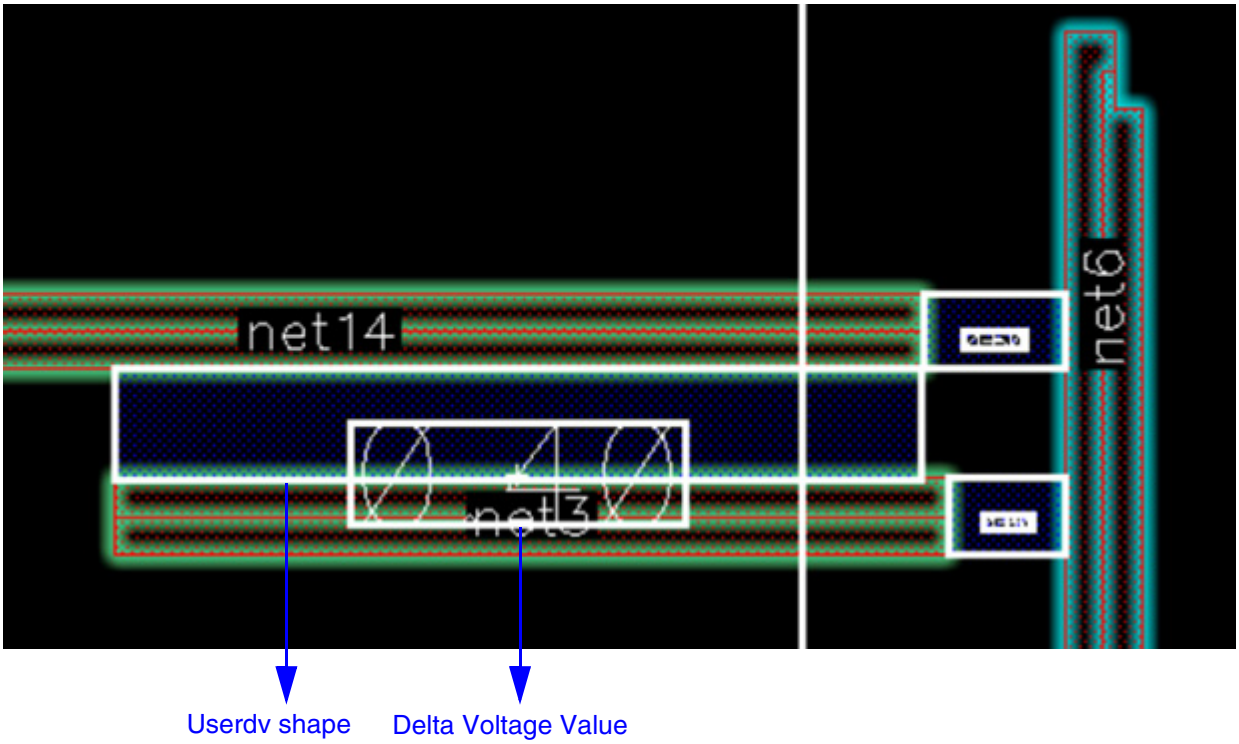
```
INFO (VDR-2011): VDR annotation deleted 0 Userdv Shapes.
INFO (VDR-2000): Successfully created Userdv Shape between 'net14' and 'net3'
on ('Metal1' 'vsync') for voltage 0.4 and bBox ((15.2305 3.239) (15.863
3.326)).


INFO (VDR-2000): Successfully created Userdv Shape between 'net14' and 'net6'
on ('Metal1' 'vsync') for voltage 0.6 and bBox ((15.864 3.3255) (15.976
3.3845)).


INFO (VDR-2000): Successfully created Userdv Shape between 'net3' and 'net6'
on ('Metal1' 'vsync') for voltage 0.8 and bBox ((15.884 3.1805) (15.976
3.2395)).


INFO (VDR-2012): VDR annotation created 3 Userdv Shapes.
=> t
```

Alternatively, you can also create the Userdv shapes by running the function `vdrGenerateUserdvShapes` in the Virtuoso Studio CIW.

```
vdrGenerateUserdvShapes(
                        scv
                        ?verbose t
```

```
                        ?logFile "./userdvLogFile.log"
                        )
;Created Userdv shapes between the nets constrained by the delta voltage
constraints. The debug messages are printed in the log file userdvLogFile,
which is saved in the current working directory.
```

The Userdv shapes are created between the nets.



The following messages, indicating the successful creation of Userdv shapes, are also printed in the Virtuoso Studio CIW.

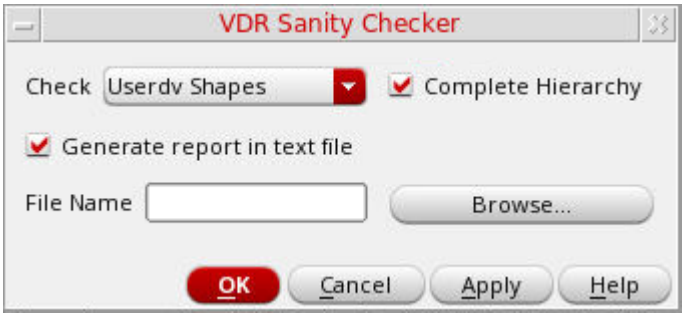
```
INFO (VDR-2011): VDR annotation deleted 0 Userdv Shapes.
INFO (VDR-2000): Successfully created Userdv Shape between 'net14' and 'net3'
on ('Metal1' 'vsync') for voltage 0.4 and bBox ((15.2305 3.239) (15.863
3.326)).


INFO (VDR-2000): Successfully created Userdv Shape between 'net14' and 'net6'
on ('Metal1' 'vsync') for voltage 0.6 and bBox ((15.864 3.3255) (15.976
3.3845)).


INFO (VDR-2000): Successfully created Userdv Shape between 'net3' and 'net6'
on ('Metal1' 'vsync') for voltage 0.8 and bBox ((15.884 3.1805) (15.976
3.2395)).


INFO (VDR-2012): VDR annotation created 3 Userdv Shapes.
```

```
=> t
```

Checking Userdv Shapes in the Layout View (ICADVM20.1 EXL Only)

1. Choose *Tools – Voltage Dependent Rules – Sanity Checker* from the layout window menu bar.

   The VDR Sanity Checker form appears.



2. Select *Userdv Shapes* from the *Check* drop-down list.

   Note: If the `vdrConstraintGroupName` environment variable is not set, the menu item is grayed out.

3. Select the *Complete Hierarchy* check box to perform checks on the complete hierarchy. If you do not select this check box, checks are performed only on the current hierarchy.

4. Select the *Generate report in text file* check box to captured results in a text file.

   When you capture the results in a log file, only a summary message is printed in the CIW. If you do not specify a log file, discrepancies are reported in a table printed in the CIW.

5. Click *OK* to check the Userdv shapes in the design.

   Note: Sanity Checker reports VDR objects (voltage labels, voltage markers, VSync shapes, and Userdv shapes) found inside the shared cells. This helps in reminding users to delete these VDR objects from shared cells.

   Userdv shapes can also be created for out-of-context hierarchical nets.

Violations Reported by Sanity Checker for VSync and Userdv Shapes

The following violations are reported by Sanity Checker for VSync and Userdv shapes:

- Shapes are present but Sanity Checker reports them invalid:
  - Floating VSync and Userdv shapes

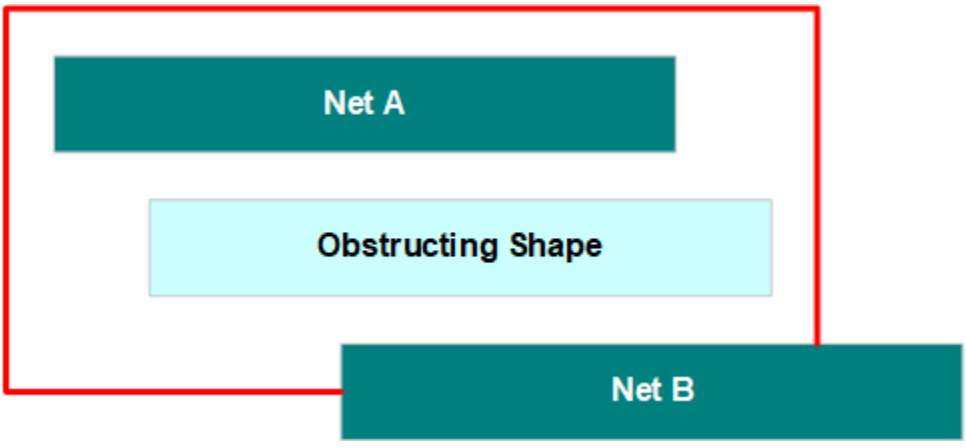
- VSync and Userdv shapes touching one net
- VSync and Userdv shapes touching shape without net
- VSync and Userdv shapes touching shape different net
- Userdv shapes touching more than two shapes
- Userdv label mismatch

- Shapes are missing:
  - No parallel run length found:

    Two net shapes (member nets of VSync and Userdv constraint) do not have any parallel run length where the VSync or Userdv shapes can be created.

  - No empty area found:

    When region between two net shapes (member nets of VSync and Userdv constraint) is completely obstructed by the other shapes.



  - No shape found for corresponding nets:

    No net shape is found in the design corresponding to member nets in the constraint.

- VSync net shapes far away:

  When there are no other shapes within the threshold spacing from the net shape (member nets of VSync and Userdv constraint).

  Threshold spacing value is calculated as the maximum value of following constraints:

  - `minVoltageSpacing`
  - `minSideSpacing`
  - `minCornerSpacing`
  - `minCornerVoltageSpacing`
  - `minSpacing`
  - `minVoltageClearance`
  - `minSideClearance`
  - `minCornerClearance`
  - `minClearance`

- Missing shapes:

  When Sanity Checker does not find a reason of why the VSync and Userdv shape are missing, the violation is reported as missing shapes.

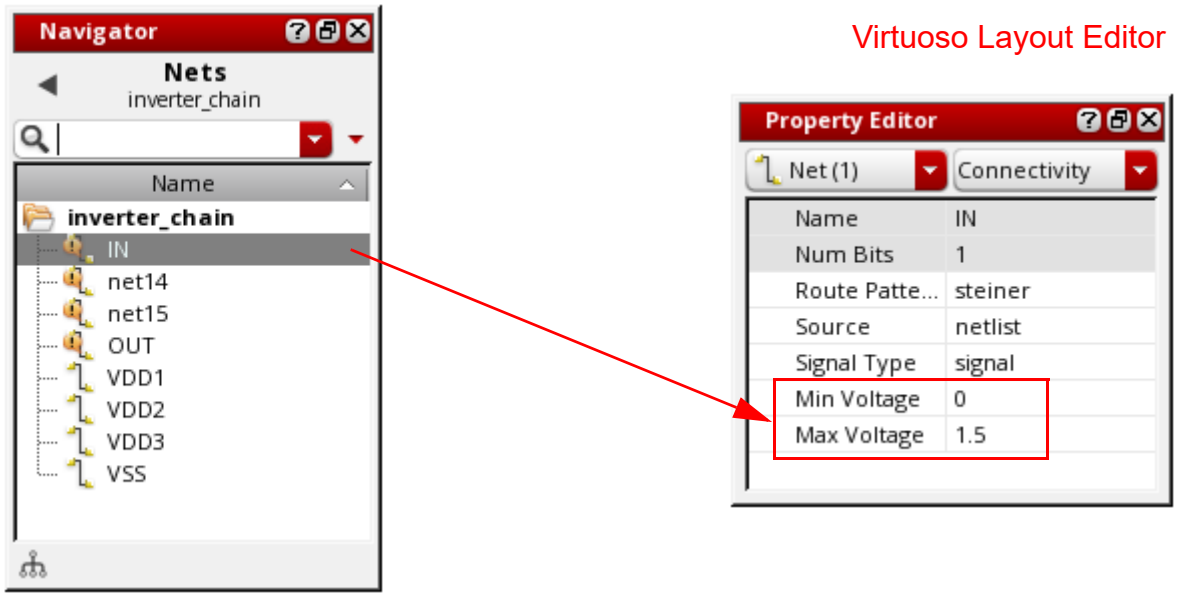
Layout-centric VDR Flow

You can also enter voltage values directly in the layout editor Property Editor assistant. The voltage values are made available to layout features such as DRD editing, the interactive wire editor, and the Virtuoso space-based router. You can then generate voltage labels or markers for all the nets at the current level of layout hierarchy, or for a set of nets selected in the Navigator assistant.

Entering Voltage Values in the Property Editor Assistant

To enter minimum and maximum voltage values for a net directly in the Property Editor assistant:

1. Open the layout design in Layout XL and open the Navigator and Property Editor assistants.
2. Select the net you require in the Navigator assistant.
3. Type the *Min Voltage* and *Max Voltage* values into the Property Editor assistant.



Note: If there are already voltage labels or markers for that net visible on the canvas, the values are automatically updated. If there are no labels or markers for that net, see Generating Voltage Labels for Manually Entered Voltages or Generating Voltage Markers for Manually Entered Voltages for information on how to create them.

4. Choose *File – Save* to save the layout view.

Layout editor tools such as DRD, the interactive wire editor, and VSR consider the voltages when checking that minimum voltage spacing constraints defined in the technology file are not being violated.

See Specifying a Minimum Voltage Spacing Constraint for more information.

Generating Voltage Labels for Manually Entered Voltages

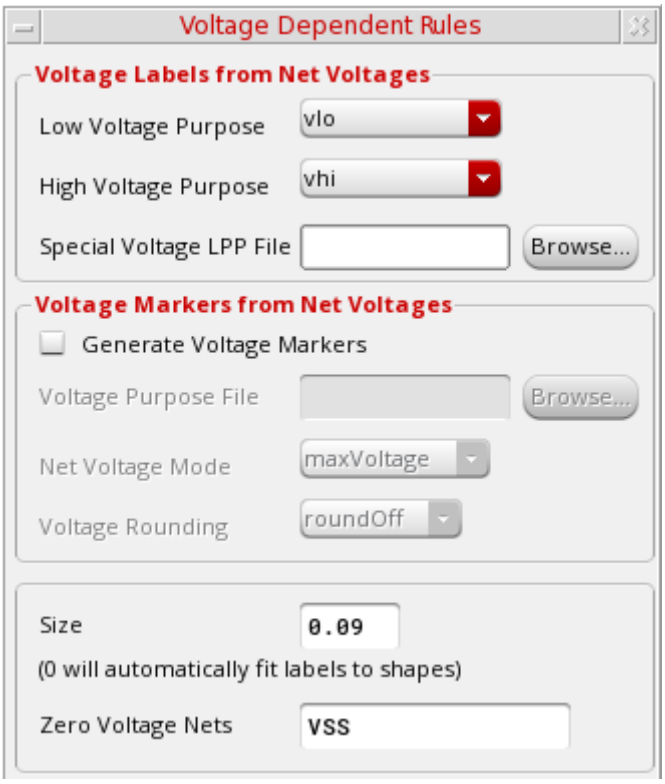
- Generating Labels on All Nets
- Generating Labels on Selected Nets

Generating Labels on All Nets

To generate labels for manually entered voltage values on all the nets at the current level of layout hierarchy:

1. Choose *Tools – Voltage Dependent Rules – Create Labels/Markers From Net Voltages* from the layout window menu bar.

   The condensed Voltage Dependent Rules form appears containing only the options required for label and marker generation.



2. Set the layer purpose and size options as required.

   Specify a *Special Voltage LPP File* if you require greater control over the purposes on which labels are drawn.

3. List the *Zero Voltage Nets* for which labels should be generated.

   By default, the field lists the nets in the design that have (0,0) voltage values and signal type `ground`.

4. Click *OK* to generate voltage labels for all the nets at the current level of layout hierarchy.

   The labels are generated on the geometry of the net in question. Where there is no geometry for that net on the canvas, the labels are generated on all the Pcell and instance terminals connected to the net (where such connectivity information exists).

Important

If you subsequently generate labels from simulation data, you must switch off the *Override Manually Entered Voltages* option to prevent these labels from being overwritten.

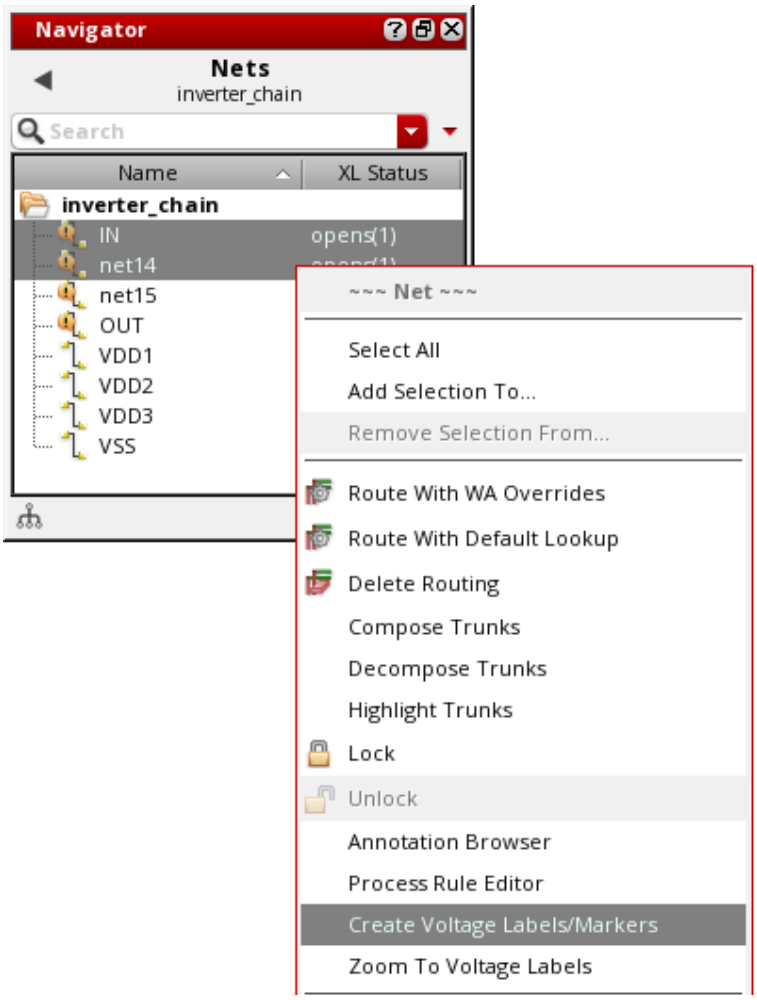
Note, however, that if you manually set both minimum and maximum voltages to 0 in the Property Editor assistant and then generate labels using these values, those labels *will* always be overwritten if you subsequently run the simulation-driven flow (provided the dataset you use contains values for the nets in question).

See Populating Voltages and Generating Labels or Markers for more information on the *Override Manually Entered Voltages* option.

Generating Labels on Selected Nets

To generate labels for manually entered voltage values on selected nets in the design:

1. In the Navigator assistant, select the nets for which you want to generate voltage labels.
2. Right-click, and choose *Create Voltage Labels/Markers*.



3. The truncated version of the Voltage Dependent Rules form appears.
4. Set the layer purpose and size options as required.

   Specify a *Special Voltage LPP File* if you require greater control over the purposes on which labels are drawn.
5. List any *Zero Voltage Nets* for which labels should be generated.

   By default, the field lists the nets in the selected set that have (0,0) voltage values and signal type `ground`.

6. Click *OK* to create voltage labels for the selected nets in the layout canvas in line with the values shown in the Property Editor assistant.

   The labels are generated on the geometry of the net in question. Where there is no geometry for that net on the canvas, the labels are generated on all the Pcell and instance terminals connected to the net (where such connectivity information exists).

Important

If you subsequently generate labels from simulation data, you must switch off the *Override Manually Entered Voltages* option to prevent these labels from being overwritten.

Note, however, that if you manually set both minimum and maximum voltages to 0 in the Property Editor assistant and then generate labels using these values, those labels *will* always be overwritten if you subsequently run the simulation-driven flow (provided the dataset you use contains values for the nets in question).

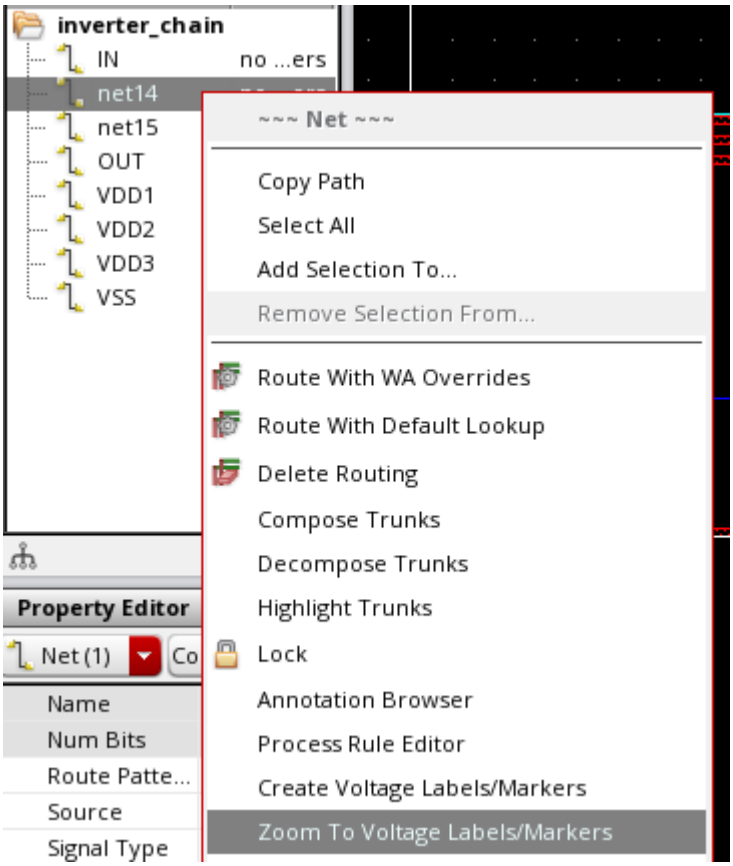
See Populating Voltages and Generating Labels or Markers for more information on the *Override Manually Entered Voltages* option.

You can also perform the same task programmatically by using the vdrCreateVoltageLabelOnNets SKILL function.

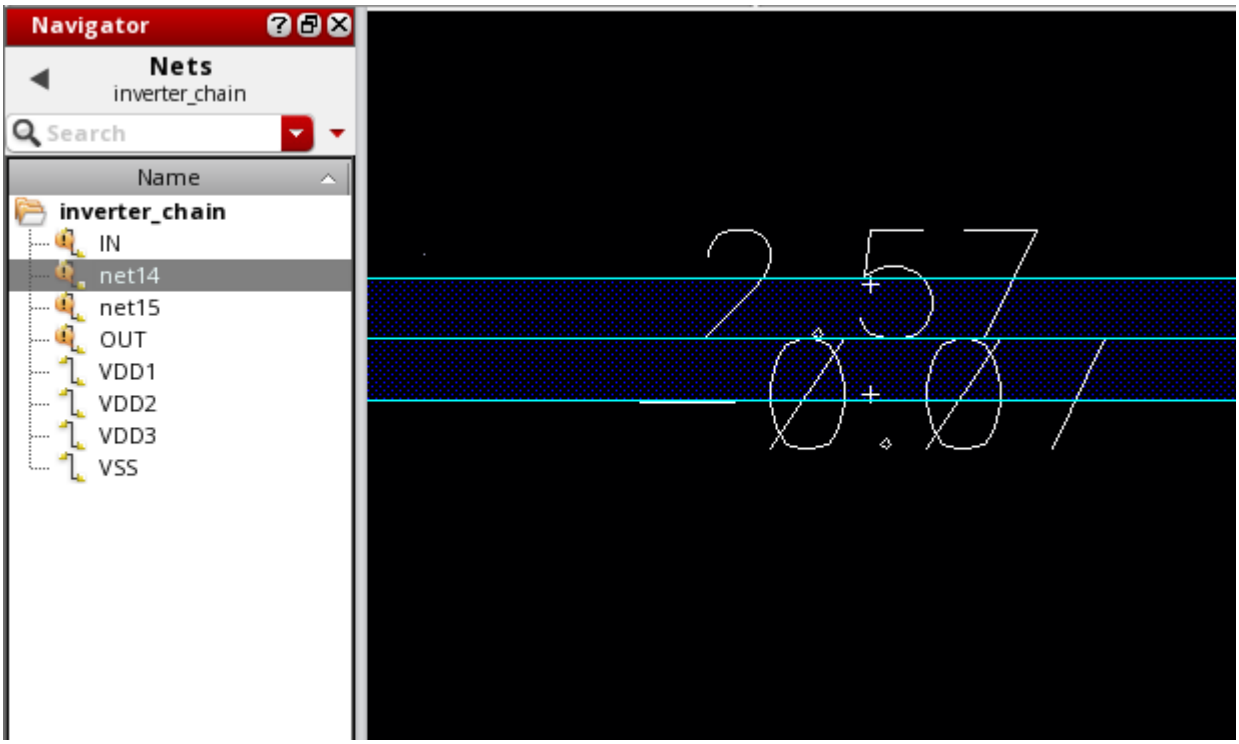
Viewing Voltage Labels in the Layout View

To view the voltage labels on a top-level net:

1. In the Navigator assistant, right-click to select the net and choose *Zoom To Voltage Labels/Markers*.



Layout XL zooms to the voltage labels on the selected net.



Note: *Zoom to Voltage Labels/Markers* works only for labels and markers on top-level nets. You cannot zoom to labels that have been generated on Pcell or instance terminals further down the hierarchy.

Checking Voltage Labels in the Layout View

You can use the vdrCheckVoltageLabels SKILL function to verify that all top-level nets in the specified layout cellview are correctly labeled on the canvas.

Note: There is no GUI equivalent for this function.

Generating Voltage Markers for Manually Entered Voltages

Some processes, especially at advanced nodes, require voltage markers to be present on predefined layers in the design. You can use the VDR layout-centric flow to create markers for all the nets at the current level of layout hierarchy, or for a set of nets selected in the Navigator assistant.

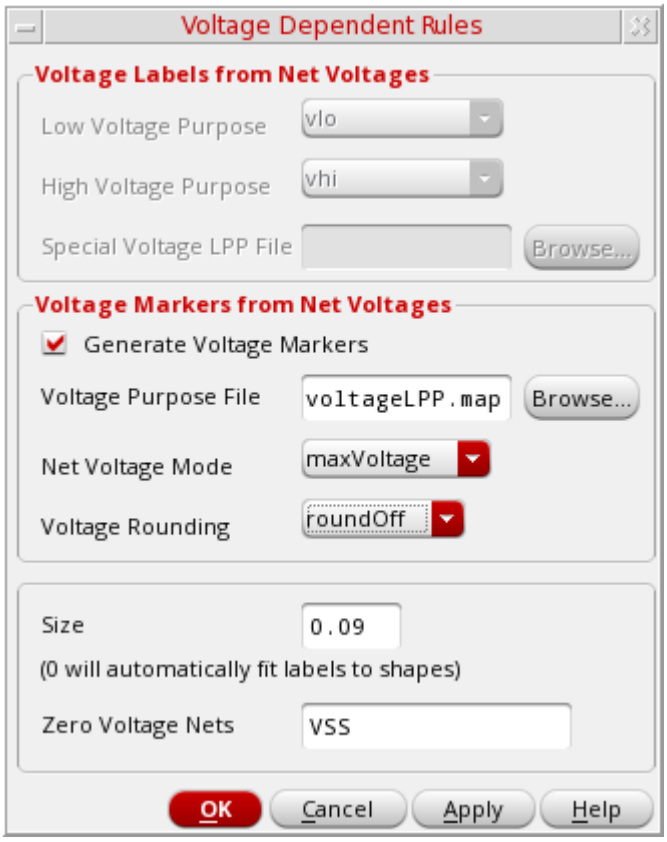
- Generating Markers on All Nets
- Generating Markers on Selected Nets

Generating Markers on All Nets

To generate markers for manually entered voltage values on all the nets at the current level of layout hierarchy:

1. Choose *Tools – Voltage Dependent Rules – Create Labels/Markers From Net Voltages* from the layout window menu bar.

   The truncated version of the Voltage Dependent Rules form appears.



2. Check the *Generate Voltage Markers* box to enable the controls in the *Voltage Markers from Net Voltages* group box.

   The *Voltage Labels* options are automatically disabled.

3. Specify the *Voltage Purpose File*, which lists the layer-purpose pairs on which markers for different voltage values are to be created.

   See Specifying Layers and Purposes for Voltage Markers for more information.

4. Choose whether markers are to be created for maximum, minimum, or all voltage values.

5. Choose the rounding rule to follow for voltage values.

   - *roundOff* rounds the voltage value to the nearest 0.01 (the default)
   - *ceiling* rounds up the voltage value to the nearest 0.01
   - *floor* rounds down the voltage value to the nearest 0.01

6. Change the *Size* setting if required.

   Note: If you set the size to 0.0, no marker is generated.

7. List the *Zero Voltage Nets* for which markers should be generated.

   By default, the field lists the nets in the design that have (0,0) voltage values and signal type `ground`.

8. Click *OK* to generate voltage markers for all the nets at the current level of layout hierarchy.

   The markers are generated on the geometry of the net in question.

Generating Markers on Selected Nets

To generate markers for manually entered voltage values on selected nets in the design:

1. In the Navigator assistant, select the nets for which you want to generate voltage labels.
2. Right-click, and choose *Create Voltage Labels/Markers*.

   The truncated version of the Voltage Dependent Rules form appears.

3. Check the *Generate Voltage Markers* box to enable the controls in the *Voltage Markers from Net Voltages* group box.
4. Specify the *Voltage Purpose File*, which lists the layer-purpose pairs on which markers for different voltage values are to be created. See Specifying Layers and Purposes for Voltage Markers for more information.

5. Choose whether markers are to be created for maximum, minimum, or all voltage values.
6. Choose the rounding rule to follow for voltage values.
   - *roundOff* rounds the voltage value to the nearest 0.01 (the default)
   - *ceiling* rounds up the voltage value to the nearest 0.01
   - *floor* rounds down the voltage value to the nearest 0.01
7. Change the *Size* setting if required.

   Note: If you set the size to 0.0, no marker is generated.
8. List the *Zero Voltage Nets* for which markers should be generated.

   By default, the field lists the nets in the design that have (0,0) voltage values and signal type `ground`.
9. Click *OK* to generate voltage markers for the selected nets at the current level of layout hierarchy.

   The markers are generated on the geometry of the net in question.

You can also perform the same task programmatically by using the vdrCreateVoltageMarkersOnNets SKILL function.

Post-Processing Voltage Labels and Markers

You can use the vdrPostLabelCreationCallback environment variable to specify a custom SKILL callback that can be set to perform any required post-processing tasks on VDR-generated labels. The specified callback must accept a cellview ID as an argument and is run automatically after label generation is complete.

For example, the callback shown below collects all the labels generated by the VDR flow in the specified cellview and creates a property to link them to a dataset called `dataset1`.

```
procedure(_myPostVdrCB(cv)
    let((shape)
    foreach(shape cv~>shapes
        if(shape~>objType == "label" then
          if(prop = dbFindProp(shape "CDNS_VDR_LABEL") then
            dbCreateProp(shape "DataSetName" 'string "dataset1")
          )
        )
    )
    )
)
```

To modify the callback to post-process marker objects instead of labels, change the `objType` to `"rect"`.

You can also specify the callback function name as an optional argument when using the vdrCreateVoltageLabel and vdrCreateVoltageMarkers SKILL functions.

Note: There is no GUI equivalent for this feature.

Deleting Voltage Labels and Markers

To delete all the labels and markers generated by the voltage dependent rules flows, do one of the following:

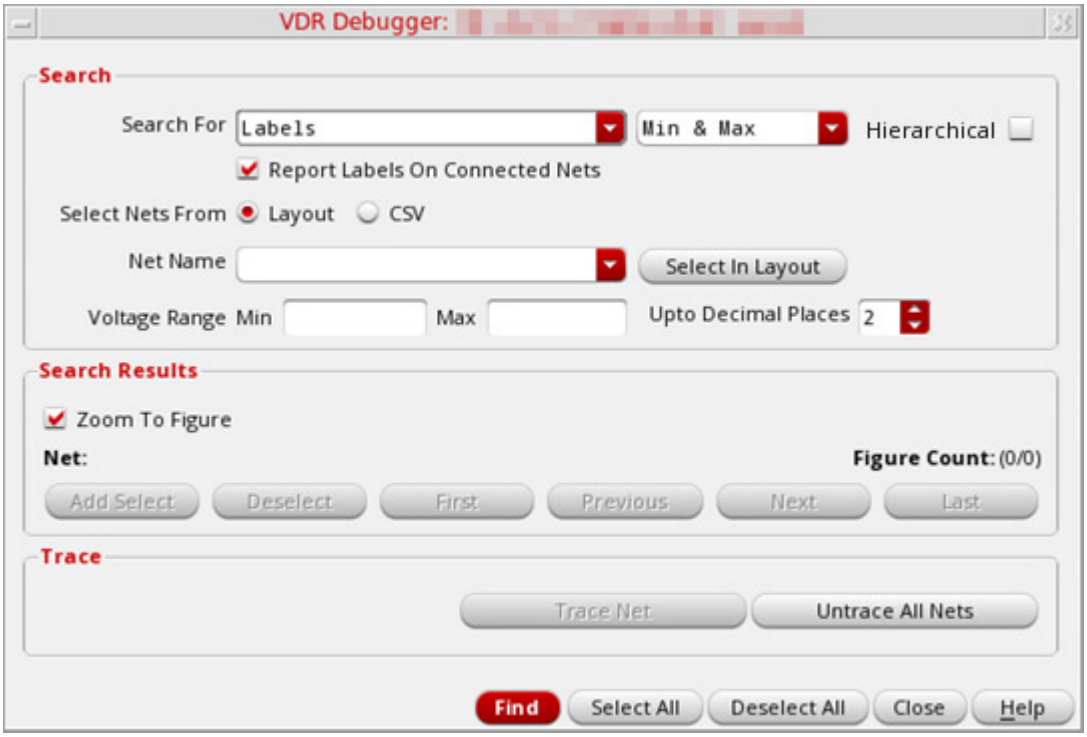
- Choose *Tools – Voltage Dependent Rules – Delete Labels/Markers* from the layout window menu bar
- Call the vdrDeleteGUI SKILL function

Searching Voltage Labels for Debugging

To search and display voltage labels for debugging:

1. Open the layout design in Layout XL.
2. Choose *Tools – Voltage Dependent Rules – VDR Debugger* from the layout window menu bar.

   The VDR Debugger form opens.



3. Select *Labels* from the *Search For* drop-down list and specify one of the following:
   - *Min & Max*: To search for both *Vmin* and *Vmax* labels.
   - *Min*: To search for only *Vmin* labels.
   - *Max*: To search for only *Vmax* labels.
4. Select the *Hierarchical* check box to search for labels across complete hierarchy.
5. Select the *Report Labels On Connected Nets* check box to list labels on the connected nets.
6. From the *Select Nets From* options, specify whether you want to select nets from the layout or from a CSV file.

7. If you selected layout in step 5, specify the name of the net in the *Net Name* field manually or select the net from layout. If you leave this field blank, voltage labels are searched for all available nets in the layout or the CSV file.

   Note: The *Net Name* field supports adding regular expressions. For example, if you type `VDD*` in the *Net Name* field, voltage labels are searched for all nets whose name starts with `VDD`, such as `VDD1`, `VDD2`, `VDD3`, and so on.

8. If you selected *CSV* in step 5, specify the path to the CSV file in the *File Name* field or click *Browse* to select the CSV file. Click *Load* to load the CSV file.

   Clicking *Load* automatically updates the Net Name field with the first net available in the CSV file.

9. In the *Voltage Range* fields, specify the minimum and maximum values of voltages.

10. In the *Decimal Places* field, specify the number of decimal places for net voltage labels.

    You can specify any integer value between `0–5`.

11. Click *Find* to search for the voltage labels.

    An information message is printed in the CIW to indicate total number of VDR labels found for all nets or for the specified net.

    The *Search Results* section in the VDR Debugger form is also updated with the total count of VDR labels found.

12. In the *Search Results* section, click the following commands:

    a. *Zoom To Figure*: To enable zooming on the selected voltage label.

    b. *Next*: To move to the next voltage label.

    c. *Previous*: To move to the previous voltage label.

    d. *First*: To directly move to the first voltage label.

    e. *Last*: To directly move to the last voltage label.

    f. *Add Selec*t: To add the voltage label to the currently selected set.

g. *Deselect*: To remove the voltage label from the currently selected set.

13. Click *Close* to close the form.

Related Topics

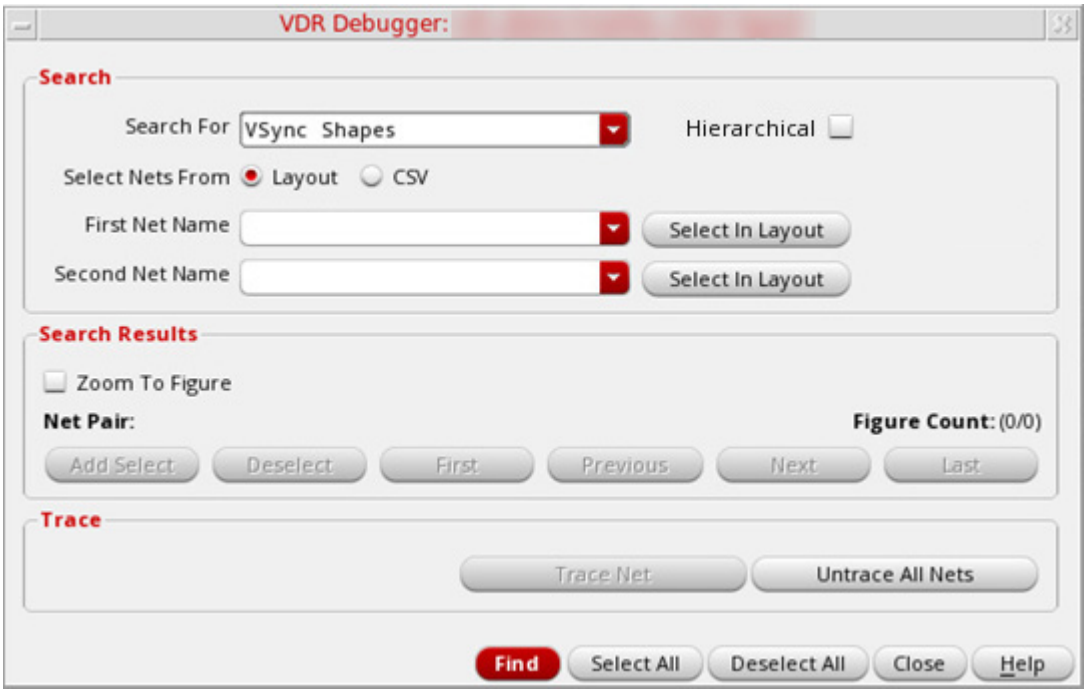
VDR Debugger

Searching VSync Shapes for Debugging

To search and display VSync shapes for debugging:

1. Open the layout design in Layout XL.
2. Choose *Tools – Voltage Dependent Rules – VDR Debugger* from the layout window menu bar.

   The VDR Debugger form opens.



3. Select *VSync Shapes* from the *Search For* drop-down list.
4. Select the *Hierarchical* check box to search for VSync shapes across complete hierarchy.
5. From the *Select Nets From* options, specify whether you want to select nets from the layout or from a CSV file.
6. If you selected *Layout* in step 4, specify the name of the nets in the *First Net Name* and *Second Net Name* fields manually or select them from layout.

   - If you leave both the fields blank, VSync shapes are searched for all available nets in the layout.

- If you specify both the nets, only VSync shapes that are associated with both the nets are searched.
- If you specify only one net, VSync shapes associated with the specified net are searched.

7. If you selected *CSV* in step 4, specify the path to the CSV file in the *File Name* field or click *Browse* to select the CSV file. Click *Load* to load the CSV file.

   Clicking *Load* automatically updates the *First Net Name* and *Second Net Name* fields with the nets available in the CSV file.

8. Click *Find* to search for the vsync shapes.

   An information message is printed in the CIW to indicate total number of VSync shapes found for all nets or for the specified nets.

   The *Search Results* section in the VDR Debugger form is also updated with the total count of VSync shapes found.

9. In the *Search Results* section, click the following commands:
   a. *Zoom To Figure*: To enable zooming on the selected VSync shape.
   b. *Next*: To move to the next VSync shape.
   c. *Previous*: To move to the previous VSync shape.
   d. *First*: To directly move to the first VSync shape.
   e. *Last*: To directly move to the last VSync shape.
   f. *Add Selec*t: To add the VSync shape to the currently selected set.
   g. *Deselect*: To remove the VSync shape from the currently selected set.
10. In the *Trace* section, click *Trace Net* to display visualization of the connected nets.
11. Click *Untrace All Nets* to remove visualization.
12. Click *Close* to close the form.

Related Topics

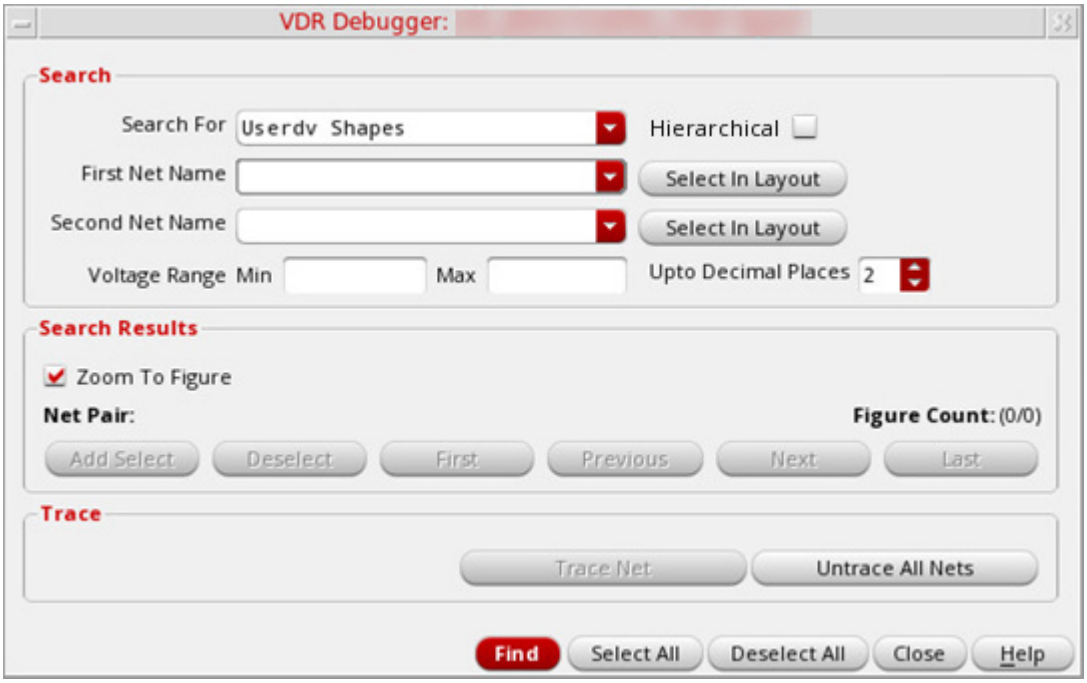
VDR Debugger

Searching Userdv Shapes for Debugging

To search and display Userdv shapes for debugging:

1. Open the layout design in Layout XL.
2. Choose *Tools – Voltage Dependent Rules – VDR Debugger* from the layout window menu bar.

   The VDR Debugger form opens.



3. Select *Userdv Shapes* from the *Search For* drop-down list.
4. Select the *Hierarchical* check box to search for Userdv shapes across complete hierarchy.
5. Specify the name of the nets in the *First Net Name* and *Second Net Name* fields manually or select them from layout.
   - If you leave both the fields blank, Userdv shapes are searched for all available nets in the layout.
   - If you specify both the nets, only Userdv shapes that are associated with both the nets are searched.
   - If you specify only one net, Userdv shapes that are associated with the specified net are searched.

6. In the *Voltage Range* fields, specify the minimum and maximum values of voltages.
7. In the *Decimal Places* field, specify the number of decimal places for net voltage.

   You can specify any integer value between `0–5`.
8. Click *Find* to search for Userdv shapes.

   An information message is printed in the CIW to indicate total number of Userdv shapes found for all nets or for the specified net.

   The *Search Results* section in the VDR Debugger form is also updated with the total count of Userdv shapes found.

9. In the *Search Results* section, click the following commands:
   a. *Zoom To Figure*: To enable zooming on the selected Userdv shape.
   b. *Next*: To move to the next Userdv shape.
   c. *Previous*: To move to the previous Userdv shape.
   d. *First*: To directly move to the first Userdv shape.
   e. *Last*: To directly move to the last Userdv shape.
   f. *Add Selec*t: To add the Userdv shape to the currently selected set.
   g. *Deselect*: To remove the Userdv shape from the currently selected set.
10. Click *Close* to close the form.

Related Topics

VDR Debugger

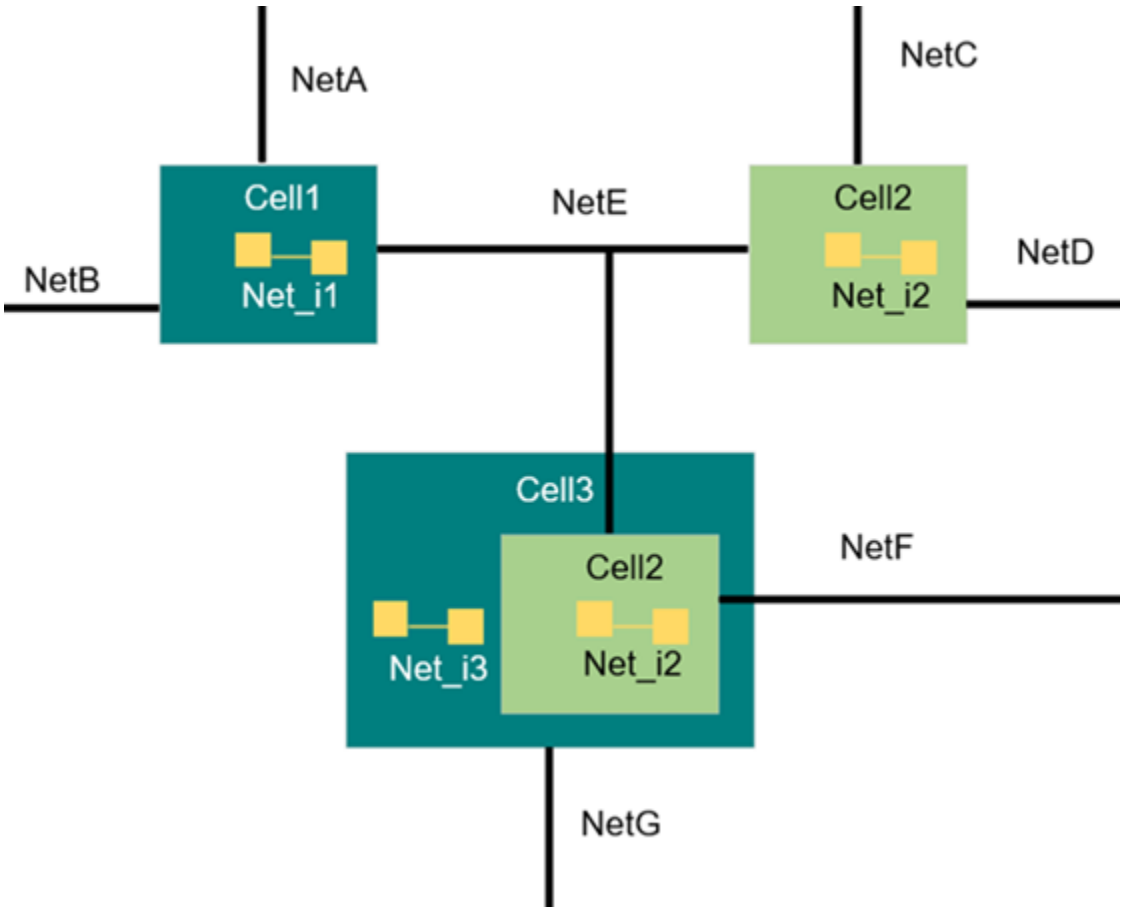
Rules for Creating Voltage Labels in Shared Cells

The rules for creating voltage labels inside the shared cells are as follows:

- For a shared cell, voltage labels are created for all internal nets across 32 hierarchy levels.
- If a cell is not specified as the shared cell, voltage labels for nets are created as per the specified *Hierarchy Stop Level*. For example, if the *Hierarchy Stop Level* is set to `0`, labels are created only for top-level nets.
- Shared cells are considered only when they are instantiated on the current top level.

Example 1

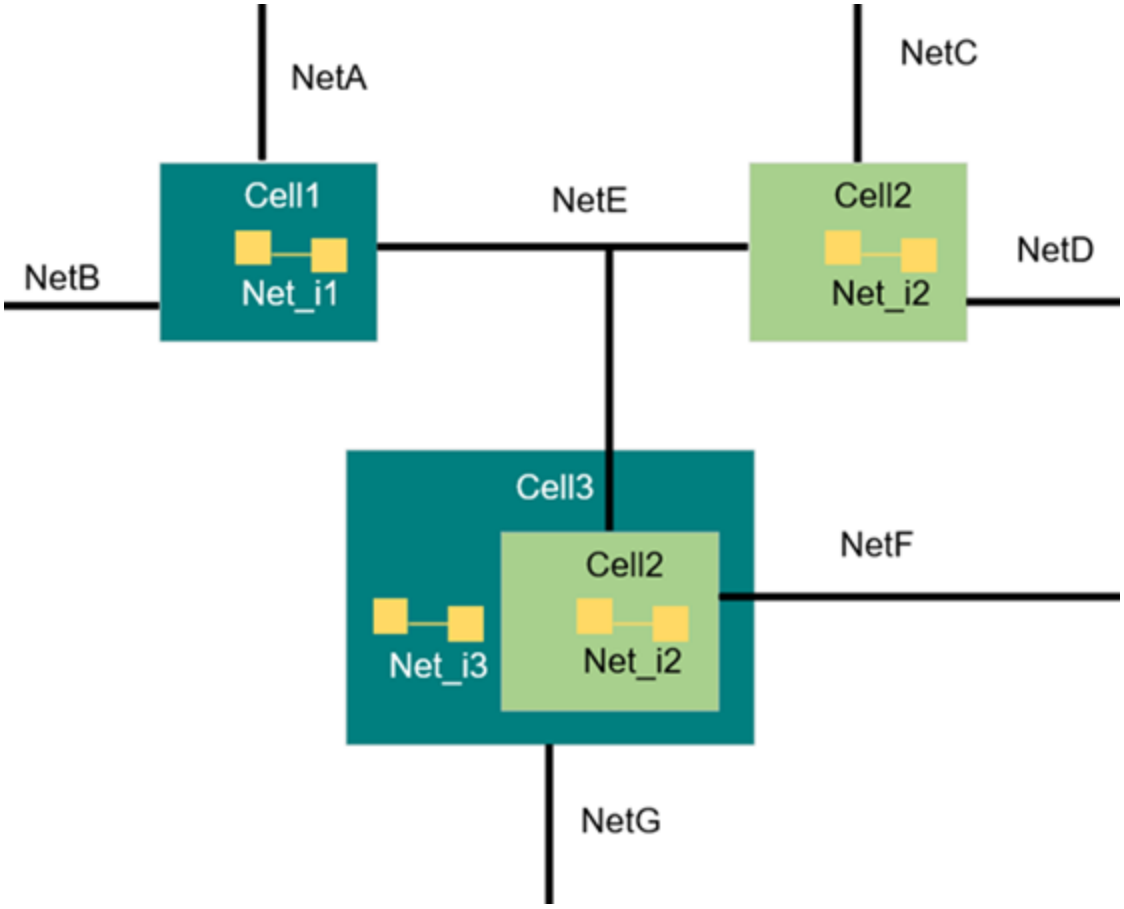
If no shared cells are specified, voltage labels are created only top-level nets: `NetA`, `NetB`, `NetC`, `NetD`, `NetE`, `NetF`, and `NetG`.



Example 2

If `Cell2` is specified as a shared cell and *Hierarchy Stop Level* is set to `0`, voltage labels are created only top-level nets: `NetA`, `NetB`, `NetC`, `NetD`, `NetE`, `NetF`, `NetG`, and `Cell2/Net_i2`.

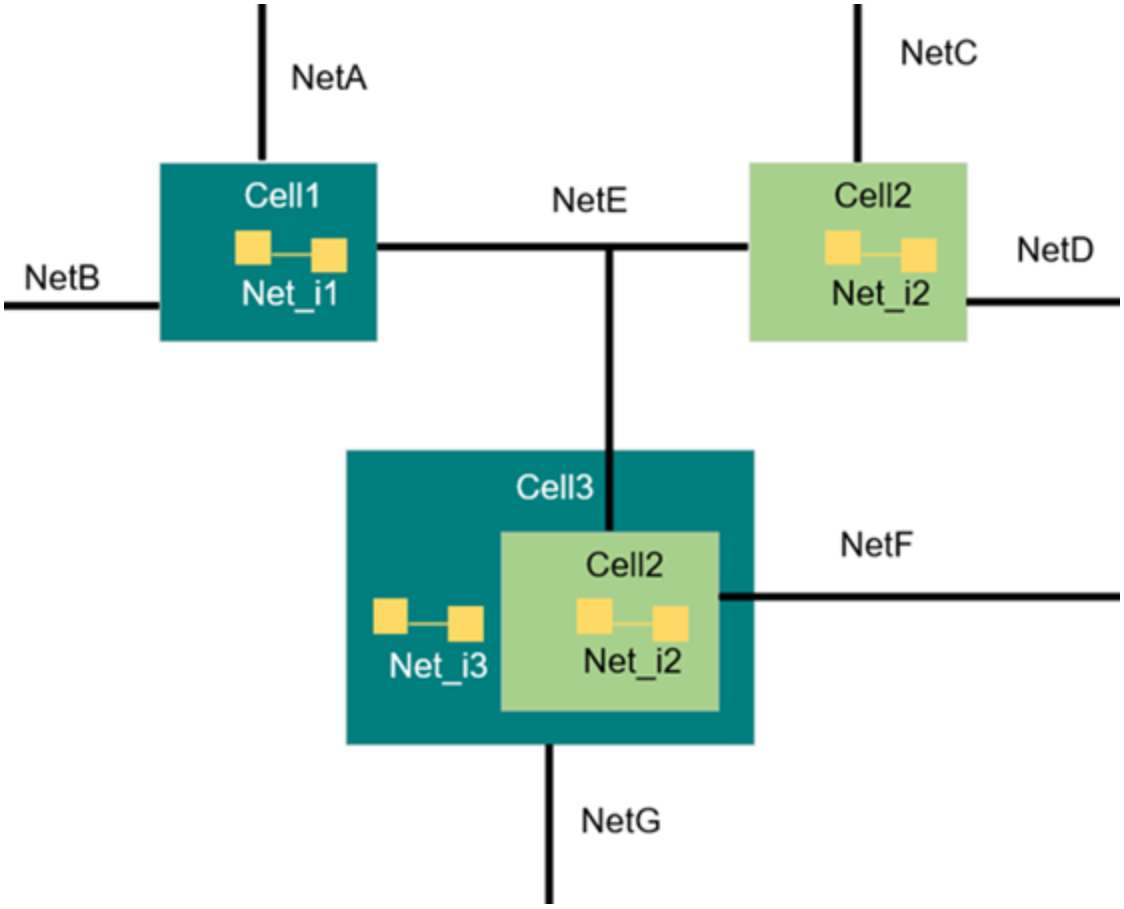
Note that voltage label is not created for the `Cell3/Cell2/Net_i2`. because `Cell3` is not specified as a shared cell.



Example 3

If `Cell3` is specified as a shared cell and *Hierarchy Stop Level* is set to `0`, voltage labels are created only top-level nets: `NetA`, `NetB`, `NetC`, `NetD`, `NetE`, `NetF`, `NetG`, and `Cell3/Net_i3`, and `Cell3/Cell2/Net_i2`.

Note that voltage label is not created for the `Cell2/Net_i2`. because `Cell2` is not specified as a shared cell.



Related Topics

Voltage Dependent Rules

vdrHierarchyStopLevel

vdrSharedCellList

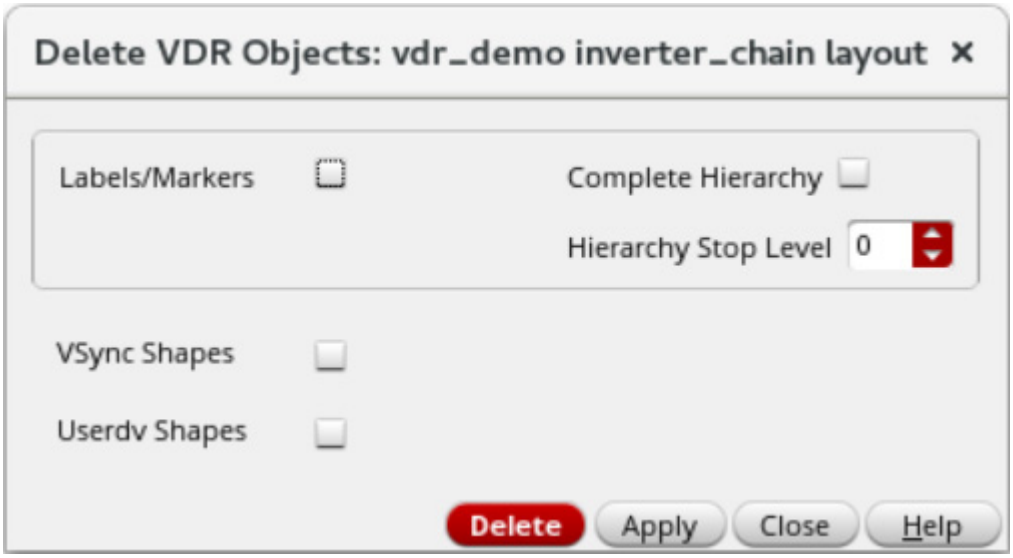
vdrSharedCellListForInternalNetsOnly

Deleting VDR Objects

To delete VDR objects:

1. From the menu bar of Virtuoso Studio Layout Suite EXL or MXL, choose *Tools – Voltage Dependent Rules – Delete VDR Objects*.

   The Delete VDR Objects form opens.



2. Select the *Labels/Markers* check box if you want to delete voltage labels and voltage markers.
3. Select the *Complete Hierarchy* check box if you want to delete voltage label and voltage markers across the complete hierarchy.
4. In the *Hierarchy Stop Level* field, specify the number of hierarchy levels up to which voltage labels and markers are deleted. The default value is `0`, which indicates that voltage label and markers are deleted only for top-level nets. You can specify any integer value between `0`–`32` in this spin box. The maximum value `32` indicates that voltage labels and markers are deleted for all nets across `32` hierarchy levels.
5. Select the *VSync Shapes* check box if you want to delete VSync shapes.
6. Select the *Userdv Shapes* check box if you want to delete the Userdv shapes.
7. Click *Delete* to delete the selected VDR objects.

2

Environment Variables

The list below provides information on the names, descriptions, and graphical user interface equivalents for the Virtuoso® Layout Suite L layout editor environment variables used in the voltage dependent rules flows.

Important

Only the environment variables listed below 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.

List of VDR-related Environment Variables

vdrConstraintGroupName

vdrCreateLabelsForOpenNet

vdrCreateLabelOnHighestMetal

vdrDebugHierSharedCellsOnly

vdrEnableHierStopField

vdrEnableHierStopField

vdrGenerateLabelsOn

vdrGenerateMarkers

vdrHierarchyStopLevel

vdrHighVoltagePurpose

vdrHighVoltagePurposes

vdrLabelHeight

vdrVoltagePurposeFile

vdrVoltageRounding

vdrVSyncCreateCheckLayer

vdrVSyncIgnorePurposes

vdrVSyncSanityCheckLayer

vdrZeroShapeNets

vdrZeroVoltageNets

vdrConstraintGroupName

```
layout vdrConstraintGroupName string "cgName"
```

Description

Specifies the name of a constraint group that contains layer-purpose pair (LPP) information for VDR labels and markers. The specified constraint group must in turn contain `voltageLabelMapping` or `voltageMarkerMapping` constraints, which define the LPPs on which labels or markers will be drawn.

When set, the environment variable automatically enables the constrained label generation flow, which lets you create and edit labels manually in the layout view.

It also enables the Voltage Synced Nets flow in Layout EXL, which lets you mark and check the voltages of nets that must always transition in phase with each other.

The default is an empty string, meaning that LPP information is not read from the technology file and the constrained label flow is not enabled.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrConstraintGroupName")
envSetVal("layout" "vdrConstraintGroupName" 'string "myVDRLppCg")
```

Related Topics

Types of VDR Labels

Defining and Checking Voltage Synced Nets

vdrCreateLabelsForOpenNet

```
layout vdrCreateLabelsForOpenNet cyclic { "none" | "virtual" | "all" }
```

Description

Specifies whether to create labels for disjoint nets.

- `none`: Allows only one label to be created per net, regardless of how many disjoint sections the net has. This is the default value.
- `virtual`: Creates labels for each disjoint section of a net that are marked by common label netname:
- `all`: Creates a label for each disjoint section of a net

This value is stored between two consecutive simulation runs and it is used to decide whether to delete labels. If mode of label creation is changed in the next run, labels are deleted.

GUI Equivalent

The *Label for Open Nets* field in the Voltage Dependent Rules form.

Examples

```
envGetVal("layout" "vdrCreateLabelsForOpenNet")
envSetVal("layout" "vdrCreateLabelsForOpenNet" 'cyclic "none")
envSetVal("layout" "vdrCreateLabelsForOpenNet" 'cyclic "virtual")
envSetVal("layout" "vdrCreateLabelsForOpenNet" 'cyclic "all")
```

Related Topics

Types of VDR Labels

Specifying Layers and Purposes for Generic Voltage Labels

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Labels for Manually Entered Voltages

Simulation Driven VDR Flow

Voltage Dependent Rules

vdrCreateLabelOnHighestMetal

```
layout vdrCreateLabelOnHighestMetal boolean { t | nil }
```

Description

Specifies whether to create labels for nets on the topmost metal shape having the highest mask.

The default value is `nil`, which indicates that labels are created on the lowest mask shape.

When this environment variable is set to `t`, in addition to `metal`, layer functions `poly` and `li` are also considered to get the highest mask shape for nets.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrCreateLabelOnHighestMetal")
envSetVal("layout" "vdrCreateLabelOnHighestMetal" 'boolean t)
```

vdrDebugHierSharedCellsOnly

```
layout vdrDebugHierSharedCellsOnly boolean { t | nil }
```

Description

Specifies whether to search hierarchically only the shared cells for voltage labels, VSync shapes, and Userdv shapes.

The default value is `nil`, which indicates that all cells are searched hierarchically for voltage labels, VSync shapes, and Userdv shapes.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrDebugHierSharedCellsOnly")
envSetVal("layout" "vdrDebugHierSharedCellsOnly" 'boolean t)
```

vdrDeltaVIgnorePurposes

```
layout vdrDeltaVIgnorePurposes string "listOfPurposes"
```

Description

Specifies a list of purposes on which shapes in the design are ignored when creating Userdv shapes or performing checks on Userdv shapes.

The default value is an empty string, which means that no shapes in the design are ignored when creating Userdv shapes or performing sanity checks.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrDeltaVIgnorePurposes")
envSetVal("layout" "vdrDeltaVIgnorePurposes" 'string "fill pin")
```

vdrEnableHierStopField

```
layout vdrEnableHierStopField boolean { t | nil }
```

Description

Controls the availability of the *Hierarchy Stop Level* field in the Voltage Dependent Rules form. The default is `t`, which means that the form field can be edited by the end user. When set to `nil`, the field is disabled, meaning that the end user cannot change the value.

GUI Equivalent

Command:	*Tools – Voltage Dependent Rules*
Field:	*Hierarchy Stop Level*

Examples

```
envGetVal("layout" "vdrEnableHierStopField")
envSetVal("layout" "vdrEnableHierStopField" 'boolean t)
envSetVal("layout" "vdrEnableHierStopField" 'boolean nil)
```

Related Topics

Voltage Dependent Rules (form)

vdrHierarchyStopLevel

Specifying Layers and Purposes for Generic Voltage Labels

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Labels for Manually Entered Voltages

Generating Voltage Markers from Simulation Data for All Nets

Generating Voltage Markers for Manually Entered Voltages

vdrGenerateLabels

```
layout vdrGenerateLabels boolean { t | nil }
```

Description

Specifies that the VDR flow is to be run in label generation mode. The default is `nil`.

GUI Equivalent

Command: *Tools – Voltage Dependent Rules –*

- *Create Labels/Markers From Simulation Voltages*
- *Create Labels/Markers From Net Voltages*

Field: *Generate Voltage Labels*

Examples

```
envGetVal("layout" "vdrGenerateLabels")
envSetVal("layout" "vdrGenerateLabels" 'boolean t)
envSetVal("layout" "vdrGenerateLabels" 'boolean nil)
```

Related Topics

Voltage Dependent Rules (form)

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Labels for Manually Entered Voltages

vdrGenerateLabelsOn

```
layout vdrGenerateLabelsOn cyclic "netType"
```

Description

Specifies the scope of nets for which labels are to be generated.

- `External and Internal Nets` (the default) specifies that labels are generated for all nets in the design.
- `External Nets Only` specifies that labels are generated only for external nets. A net is considered external if it is connected to any of the terminals of the cellview to which it belongs. When the cell is instantiated at a higher level, these nets are propagated up the hierarchy, allowing you to make connections to the terminals to which they are connected.
- `Internal Nets Only` specifies that labels are to be generated only for nets that are wholly internal to the cellview in which they belong. When the cellview is instantiated at a higher level, internal nets are not propagated up the hierarchy.

GUI Equivalent

Command:	*Tools – Voltage Dependent Rules*
Field:	*Generate Labels On*

Examples

```
envGetVal("layout" "vdrGenerateLabelsOn")
envSetVal("layout" "vdrGenerateLabelsOn" 'cyclic "External Nets Only")
envSetVal("layout" "vdrGenerateLabelsOn" 'cyclic "Internal Nets Only")
```

Related Topics

Voltage Dependent Rules (form)

Specifying Layers and Purposes for Generic Voltage Labels

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Labels for Manually Entered Voltages

vdrGenerateMarkers

```
layout vdrGenerateMarkers boolean { t | nil }
```

Description

Specifies that the VDR flow is to be run in marker generation mode. The default is `nil`.

GUI Equivalent

Command: *Tools – Voltage Dependent Rules –*

- *Create Labels/Markers From Simulation Voltages*
- *Create Labels/Markers From Net Voltages*

Field: *Generate Voltage Markers*

Examples

```
envGetVal("layout" "vdrGenerateMarkers")
envSetVal("layout" "vdrGenerateMarkers" 'boolean t)
envSetVal("layout" "vdrGenerateMarkers" 'boolean nil)
```

Related Topics

Voltage Dependent Rules (form)

Generating Voltage Markers from Simulation Data for All Nets

Generating Voltage Markers for Manually Entered Voltages

vdrHierarchyStopLevel

```
layout vdrHierarchyStopLevel int stopLevel
```

Description

Specifies how many hierarchy levels the software searches to find the nets on which to generate labels. Specify an integer value between 0 and 32, where for example:

- `0` means that labels are generated only for top-level nets (this is the default)
- `1` means top-level nets and nets located one level below in the hierarchy
- `2` means top-level nets and nets located one and two levels below in the hierarchy

GUI Equivalent

Command:	*Tools – Voltage Dependent Rules*
Field:	*Hierarchy Stop Level*

Examples

```
envGetVal("layout" "vdrHierarchyStopLevel")
envSetVal("layout" "vdrHierarchyStopLevel" 'int 2)
```

Related Topics

Voltage Dependent Rules (form)

vdrEnableHierStopField

Specifying Layers and Purposes for Generic Voltage Labels

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Labels for Manually Entered Voltages

Generating Voltage Markers from Simulation Data for All Nets

Generating Voltage Markers for Manually Entered Voltages

vdrHighVoltagePurpose

```
layout vdrHighVoltagePurpose string "purposeName"
```

Description

Specifies the layer purpose on which to draw the maximum voltage label for a net when generating labels using the VDR flow.

The default is `"vhi"`.

GUI Equivalent

Command:	*Tools – Voltage Dependent Rules*
Field:	*High Voltage Purpose*

Examples

```
envGetVal("layout" "vdrHighVoltagePurpose")
envSetVal("layout" "vdrHighVoltagePurpose" 'string "vhi")
```

Related Topics

vdrLowVoltagePurpose

Voltage Dependent Rules (form)

Specifying Layers and Purposes for Generic Voltage Labels

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Labels for Manually Entered Voltages

vdrHighVoltagePurposes

```
layout vdrHighVoltagePurposes string "listOfpurposeNames"
```

Description

Specifies a list of the high voltage layer purposes in the order of increasing priority. Sanity Checker validates the voltage labels of a net that are created on the layer purpose with the highest priority while ignoring the voltage labels on other payer purposes with lower priority.

The default is `""`.

Sanity Checker first retrieves the LPP (layer-purpose pair) information from the constraint group specified by the environment variable `vdrConstraintGroupName`. If `vdrConstraintGroupName` is set to `nil`, it extracts the LPP information from voltage purpose file specified by the environment variable `vdrVoltagePurposeFile`. If `vdrVoltagePurposeFile` is also set to `nil`, it retrieves the layer-purpose information from the environment variables, `vdrHighVoltagePurposes` and `vdrLowVoltagePurposes`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrHighVoltagePurposes")
envSetVal("layout" "vdrHighVoltagePurposes" 'string "net slot vhi")
```

Note that the layer purpose `vhi` has the highest priority and `net` has the lowest priority.

Related Topics

vdrHighVoltagePurpose

vdrLowVoltagePurpose

vdrLowVoltagePurposes

Voltage Dependent Rules (form)

vdrConstraintGroupName

vdrVoltagePurposeFile

vdrLabelHeight

```
layout vdrLabelHeight float heightValue
```

Description

Specifies the default height for labels or markers generated by the VDR flow.

The default is `0.0`.

- For labels, this means that the label is automatically sized to match the height of the shape with which it is associated.
- For markers, it prevents the marker from being created at all.

GUI Equivalent

Command:	*Tools – Voltage Dependent Rules*
Field:	*Size*

Examples

```
envGetVal("layout" "vdrLabelHeight")
envSetVal("layout" "vdrLabelHeight" 'float 0.05)
```

Related Topics

Voltage Dependent Rules (form)

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Labels for Manually Entered Voltages

Generating Voltage Markers for Manually Entered Voltages

vdrLayerPurposeFile

```
layout vdrLayerPurposeFile string "fileName"
```

Description

Specifies the name of a special layer purpose file, which lets you override the default vdrHighVoltagePurpose and vdrLowVoltagePurpose settings if your process requires it.

The default is `""` (an empty string).

GUI Equivalent

Command: *Tools – Voltage Dependent Rules –*

- *Create Labels/Markers From Simulation Voltages*

Create Labels/Markers From Net Voltages

Field: *Special Voltage LPP File*

Examples

```
envGetVal("layout" "vdrLayerPurposeFile")
envSetVal("layout" "vdrLayerPurposeFile" 'string "vdrLayerPurpose.map")
```

Related Topics

Voltage Dependent Rules (form)

Specifying Layers and Purposes for Generic Voltage Labels

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Labels for Manually Entered Voltages

vdrLogFile

```
layout vdrLogFile string "fileName"
```

Description

Specifies the name of a log file in which messages are saved when generating labels from a voltage information file in verbose mode.

The default is `""` (an empty string).

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrLogFile")
envSetVal("layout" "vdrLogFile" 'string "messageLog")
```

Related Topics

VSync Constraints Visualizer

vdrLowVoltagePurpose

```
layout vdrLowVoltagePurpose string "purposeName"
```

Description

Specifies the layer purpose on which to draw the minimum voltage label for a net when generating labels using the VDR flow.

The default is `"vlo"`.

GUI Equivalent

Command:	*Tools – Voltage Dependent Rules*
Field:	*Low Voltage Purpose*

Examples

```
envGetVal("layout" "vdrLowVoltagePurpose")
envSetVal("layout" "vdrLowVoltagePurpose" 'string "vlo")
```

Related Topics

vdrHighVoltagePurpose

Voltage Dependent Rules (form)

Specifying Layers and Purposes for Generic Voltage Labels

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Labels for Manually Entered Voltages

vdrLowVoltagePurposes

```
layout vdrLowVoltagePurposes string "listOfPurposeNames"
```

Description

Specifies a list of the low voltage layer purposes in the order of increasing priority. Sanity Checker validates the voltage labels of a net that are created on the layer purpose with the highest priority while ignoring the voltage labels on other payer purposes with lower priority.

The default is `""`.

Sanity Checker first retrieves the LPP (layer-purpose pair) information from the constraint group specified by the environment variable `vdrConstraintGroupName`. If `vdrConstraintGroupName` is set to `nil`, it extracts the LPP information from voltage purpose file specified by the environment variable `vdrVoltagePurposeFile`. If `vdrVoltagePurposeFile` is also set to `nil`, it retrieves the layer-purpose information from the environment variables, `vdrHighVoltagePurposes` and `vdrLowVoltagePurposes`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrLowVoltagePurposes")
envSetVal("layout" "vdrLowVoltagePurposes" 'string "pin fill vlo")
```

Note that the layer purpose `vlo` has the highest priority and `pin` has the lowest priority.

Related Topics

vdrLowVoltagePurposes

vdrHighVoltagePurpose

vdrHighVoltagePurposes

vdrConstraintGroupName

vdrVoltagePurposeFile

Voltage Dependent Rules (form)

vdrNetVoltageMode

```
layout vdrNetVoltageMode cyclic { "maxVoltage" | "minVoltage" | "bothVoltage" }
```

Description

Specifies whether markers are to be created for maximum, minimum, or all voltage values.

- `maxVoltage` creates markers for maximum voltage values only
- `minVoltage` creates markers for minimum voltage values only
- `bothVoltage` creates markers for all voltage values

The default is `"maxVoltage"`.

GUI Equivalent

Command: *Tools – Voltage Dependent Rules –*
- *Create Labels/Markers From Simulation Voltages*
- *Create Labels/Markers From Net Voltages*

Field: *Net Voltage Mode*

Examples

```
envGetVal("layout" "vdrNetVoltageMode")
envSetVal("layout" "vdrNetVoltageMode" 'cyclic "maxVoltage")
```

Related Topics

Voltage Dependent Rules (form)

Generating Voltage Markers from Simulation Data for All Nets

Generating Voltage Markers for Manually Entered Voltages

vdrOverrideMode

```
layout vdrOverrideMode boolean { t | nil }
```

Description

Specifies that any manually entered voltage values on nets are overridden by the values from the simulation datasets. The default is `nil`.

GUI Equivalent

Command:	*Tools – Voltage Dependent Rules – Create Labels/Markers From Simulation Voltages*
Field:	*Override Manually Entered Voltages*

Examples

```
envGetVal("layout" "vdrOverrideMode")
envSetVal("layout" "vdrOverrideMode" 'boolean t)
```

Related Topics

Voltage Dependent Rules

Populating Voltages and Generating Labels or Markers

vdrPostLabelCreationCallback

```
layout vdrPostLabelCreationCallback string "procedureName"
```

Description

Specifies a user-defined SKILL procedure that can be used to perform any required post-processing tasks on the generated labels. The specified callback must accept a cellview ID as an argument and is run automatically after label generation is complete.

For example, the callback shown below collects all the labels generated by the VDR flow in the specified cellview and creates a property to link them to a dataset called `dataset1`.

```
procedure(_myPostVdrCB(cv)
    let((shape)
    foreach(shape cv~>shapes
        if(shape~>objType == "label" then
          if(prop = dbFindProp(shape "CDNS_VDR_LABEL") then
            dbCreateProp(shape "DataSetName" 'string "dataset1")
          )
        )
    )
    )
)
```

To modify the callback to post-process marker objects instead of labels, change the `objType` to `"rect"`.

You can also specify the callback function name as an optional argument when using the vdrCreateVoltageLabel SKILL function.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrPostLabelCreationCallback")
envSetVal("layout" "vdrPostLabelCreationCallback" 'string "_myPostVdrCB")
```

Related Topics

Post-Processing Voltage Labels and Markers

vdrPrecision

```
layout vdrPrecision int num_decimal_places
```

Description

Specifies the number of decimal places for net voltage labels or markers being created. Specify an integer value between 0 and 5. If an integer value outside this range is specified, the value is reset to the default value. The default is `2`.

Note: Existing labels are not changed unless they are regenerated.

GUI Equivalent

Command:	*Tools – Voltage Dependent Rules – Create Labels/Markers From Simulation Voltages*
Field:	*Decimal Places*

Examples

```
envGetVal("layout" "vdrPrecision")
envSetVal("layout" "vdrPrecision" 'int 3)
```

Related Topics

vdrVoltageRounding

Specifying Layers and Purposes for Generic Voltage Labels

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Labels for Manually Entered Voltages

vdrReportLowerHierMarkerTouchingTopLevelNet

```
layout vdrReportLowerHierMarkerTouchingTopLevelNet boolean { t | nil }
```

Description

Specifies whether to report a lower-level marker as invalid if it touches any top-level shapes. If a marker touches a shape present in the lower hierarchy, it will not be reported as invalid even if touches a top-level net.

The default value is `nil`, which indicates that Sanity checker reports violations for all invalid markers that are present in the lower hierarchy.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrReportLowerHierMarkerTouchingTopLevelNet")
envSetVal("layout" "vdrReportLowerHierMarkerTouchingTopLevelNet" 'boolean t)
```

vdrSanityCheckerCheckAgainst

```
layout vdrSanityCheckerCheckAgainst cyclic { "Schematic" | "Layout" | "CSV" |
     "Datasets" }
```

Description

Specifies whether voltage values in labels are checked against schematic net properties, layout net properties or a CSV file.

- `Schematic` checks values against schematic net properties (this is the default)
- `Layout` checks values against layout net properties
- `CSV` checks values against the voltage values in the CSV file specified in vdrSanityCheckerCsvFileName
- `Datasets` checks values against datasets specified in vdrSanityCheckerDatasets

GUI Equivalent

Command: *Tools – Voltage Dependent Rules – Sanity Checker*

Field:
- *Check Against Schematic*
- *Check Against Layout*
- *Check Against CSV*
- *Check Against Datasets*

Examples

```
envGetVal("layout" "vdrSanityCheckerCheckAgainst")
envSetVal("layout" "vdrSanityCheckerCheckAgainst" 'cyclic "Layout")
envSetVal("layout" "vdrSanityCheckerCheckAgainst" 'cyclic "CSV")
envSetVal("layout" "vdrSanityCheckerCheckAgainst" 'cyclic "Datasets")
```

Related Topics

Sanity Checking Voltage Values in Constrained Labels

vdrSanityCheckerCompleteHierarchy

```
layout vdrSanityCheckerCompleteHierarchy boolean { t | nil }
```

Description

Specifies whether to perform sanity checks on complete hierarchy. The default value is `nil`, which indicates that sanity checks are performed only on the current hierarchy.

GUI Equivalent

The *Complete Hierarchy* check box in the VDR Sanity Checker form.

Examples

```
envGetVal("layout" "vdrSanityCheckerCompleteHierarchy")
envSetVal("layout" "vdrSanityCheckerCompleteHierarchy" 'boolean t)
```

vdrSanityCheckerCsvFileName

```
layout vdrSanityCheckerCsvFileName string "fileName"
```

Description

Specifies a CSV file containing voltage values against which labels are checked when vdrSanityCheckerCheckAgainst is set to `CSV`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrSanityCheckerCsvFileName")
envSetVal("layout" "vdrSanityCheckerCsvFileName" 'string "voltages.csv")
```

Related Topics

Sanity Checking Voltage Values in Constrained Labels

vdrSanityCheckerDatasets

```
layout vdrSanityCheckerDatasets string "list_of_datasets"
```

Description

Specifies one or more datasets against which labels are checked when vdrSanityCheckerCheckAgainst is set to `Datasets`. If several datasets are specified, the labels are checked against the worst case derived from the specified datasets. You can specify several datasets by adding their names to the string in a space-separated format.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrSanityCheckerDatasets")
envSetVal("layout" "vdrSanityCheckerDatasets" 'string "vdr_dataset_0")
envSetVal("layout" "vdrSanityCheckerDatasets" 'string "vdr_dataset_0
vdr_dataset_1")
```

Related Topics

Sanity Checking Voltage Values in Constrained Labels

vdrSanityCheckerGenLogFile

```
layout vdrSanityCheckerGenLogFile boolean { t | nil }
```

Description

Specifies that the sanity checker comparison report is to be captured in a log file. You specify the log filename and location using vdrSanityCheckerLogFile.

When you specify a filename, only a summary message is printed in the CIW. If you do not specify a filename, discrepancies are reported in a table printed in the CIW.

GUI Equivalent

Command:	*Tools – Voltage Dependent Rules – Sanity Checker*
Field:	*Generate report in text file*

Examples

```
envGetVal("layout" "vdrSanityCheckerGenLogFile")
envSetVal("layout" "vdrSanityCheckerGenLogFile" 'boolean t)
```

Related Topics

Sanity Checking Voltage Values in Constrained Labels

Defining and Checking Voltage Synced Nets

vdrSanityCheckerLogFile

```
layout vdrSanityCheckerLogFile string "fileName"
```

Description

Specifies the path and name of the sanity checker report log file.

When you specify a filename, only a summary message is printed in the CIW. If you do not specify a filename, discrepancies are reported in a table printed in the CIW.

GUI Equivalent

Command:	*Tools – Voltage Dependent Rules – Sanity Checker*
Field:	*File Name*

Examples

```
envGetVal("layout" "vdrSanityCheckerLogFile")
envSetVal("layout" "vdrSanityCheckerLogFile" 'string "vdrReport.log")
```

Related Topics

Sanity Checking Voltage Values in Constrained Labels

Defining and Checking Voltage Synced Nets

vdrSanityCheckerObjectType

```
layout vdrSanityCheckerObjectType cyclic { "Labels" | "VSync" | "Userdv"}
```

Description

Specifies what is to be checked by the VDR Sanity Checker.

- `Labels` performs the sanity check on voltage values that are printed as labels in the layout (this is the default)
- `VSync` performs the sanity check on vsync shapes in the layout view (ICADVM20.1 EXL Only)
- `Userdv` performs the sanity check on Userdv shapes in the layout view (ICADVM20.1 EXL Only)

GUI Equivalent

Command: *Tools – Voltage Dependent Rules – Sanity Checker*

Field:
- *Check Labels*
- *Check VSync Shapes*
- *Check Userdv Shapes*

Examples

```
envGetVal("layout" "vdrSanityCheckerObjectType")
envSetVal("layout" "vdrSanityCheckerObjectType" 'cyclic "Labels")
envSetVal("layout" "vdrSanityCheckerObjectType" 'cyclic "VSync")
envSetVal("layout" "vdrSanityCheckerObjectType" 'cyclic "Userdv")
```

Related Topics

Sanity Checking Voltage Values in Constrained Labels

Defining and Checking Voltage Synced Nets

vdrSanityCheckerTolerance

```
layout vdrSanityCheckerTolerance float toleranceValue
```

Description

Specifies a threshold beyond which voltage mismatches are to be reported by the sanity checker. The default is 0.00.

GUI Equivalent

Command:	*Tools – Voltage Dependent Rules – Sanity Checker*
Field:	*Tolerance*

Examples

```
envGetVal("layout" "vdrSanityCheckerTolerance")
envSetVal("layout" "vdrSanityCheckerTolerance" 'float 0.05)
```

Related Topics

Sanity Checking Voltage Values in Constrained Labels

vdrSanityCheckerToleranceType

```
layout vdrSanityCheckerToleranceType cyclic { "Absolute" | "Relative" }
```

Description

Specifies whether the sanity checker tolerance value is considered an absolute value or a relative percentage based on the net voltage.

GUI Equivalent

Command:	*Tools – Voltage Dependent Rules – Sanity Checker*
Field:	*Absolute* and *Relative*

Examples

```
envGetVal("layout" "vdrSanityCheckerToleranceType")
envSetVal("layout" "vdrSanityCheckerToleranceType" 'cyclic "Relative")
```

Related Topics

Sanity Checking Voltage Values in Constrained Labels

vdrSharedCellList

```
layout vdrSharedCellList string list_of_shared_cells
```

Description

Specifies a list of shared cells. In addition, cells with their design intent profile names as `VDR Shared Cell` are also considered as shared cells. For more information, see *Virtuoso Design Intent User Guide*.

You can use regular expressions to specify a list of cells that you want to set as the shared cells. For example:

The following example sets all cells inside the cellview that have library names starting with `vdr_` and cell names starting with `inv` as the shared cells.

```
envSetVal("layout" "vdrSharedCellList" 'string "(vdr_* inv*)")
```

The following example sets all cells inside the cellview that have library names starting with `v` and ending with `r`, and cell names starting with `in` and ending with `er` as the shared cells.

```
envSetVal("layout" "vdrSharedCellList" 'string "(v*r in*er)")
```

While evaluating regular expressions, VDR considers instances only in current cellview and checks whether their library and cell names match with the patterns specified in the regular expression.

The default is `""` (an empty string), which indicates that no shared cells are defined.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrSharedCellList")
envSetVal("layout" "vdrSharedCellList" 'string "(library1 cell1) (library2
cell2)")
envSetVal("layout" "vdrSharedCellList" 'string "(vdr_* inv*)")
envSetVal("layout" "vdrSharedCellList" 'string "(v*r in*er)")
```

Related Topics

Rules for Creating Voltage Labels in Shared Cells

vdrSharedCellListForInternalNetsOnly

vdrSharedCellListForInternalNetsOnly

```
layout vdrSharedCellListForInternalNetsOnly string list_of_shared_cells
```

Description

Specifies a list of shared cells for which voltage labels are created only on the internal nets. No voltage labels are created on external nets for the cells specified in this environment variable.

If a cell is specified in both the environment variables, `vdrSharedCellList` and `vdrSharedCellListForInternalNetsOnly`, the latter takes precedence.

You can use regular expressions to specify a list of cells for which you do not want to create voltage labels on the external nets.

The following example specifies all cells inside the cellview that have library names starting with `vdr_` and cell names starting with `inv` as the shared cells.

```
envSetVal("layout" "vdrSharedCellListForInternalNetsOnly" 'string "(vdr_* inv*)")
```

The following example sets all cells inside the cellview that have library names starting with `v` and ending with `r`, and cell names starting with `in` and ending with `er` as the shared cells.

```
envSetVal("layout" "vdrSharedCellListForInternalNetsOnly" 'string "(v*r in*er)")
```

While evaluating regular expressions, VDR considers instances only in current cellview and checks whether their library and cell names match with the patterns specified in the regular expression.

The default is "" (an empty string).

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrSharedCellListForInternalNetsOnly")
envSetVal("layout" "vdrSharedCellListForInternalNetsOnly" 'string "(library1
cell1) (library2 cell2)")
envSetVal("layout" "vdrSharedCellListForInternalNetsOnly" 'string "(vdr_* inv*)")
envSetVal("layout" "vdrSharedCellListForInternalNetsOnly" 'string "(v*r in*er)")
```

Related Topics

Rules for Creating Voltage Labels in Shared Cells

vdrSharedCellList

vdrSnapLabelMfgGrid

```
layout vdrSnapLabelMfgGrid boolean { t | nil }
```

Description

Specifies whether the newly created voltage labels are aligned with the manufacturing grid.

The default value is `nil`, which indicates that the voltages labels are aligned with the database grid, instead of the manufacturing grid.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrSnapLabelMfgGrid")
envSetVal("layout" "vdrSnapLabelMfgGrid" 'boolean t)
```

vdrUseDatasetsOnlyForLabelCreation

```
layout vdrUseDatasetsOnlyForLabelCreation boolean { t | nil }
```

Description

When creating voltage labels using datasets, voltage values captured in the dataset are used. Net voltages are used to create voltage labels for nets that are not found in the dataset

This environment variable specifies whether to use only datasets for creating voltage labels for nets.

The default value is `nil`, which indicates that net voltages are used for creating voltage labels for nets which are not found in the dataset.

If you set this environment variable to `t`, voltage labels for nets are created only using datasets. This means that voltage labels are not created for nets that are not found in the dataset.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrUseDatasetsOnlyForLabelCreation")
envSetVal("layout" "vdrUseDatasetsOnlyForLabelCreation" 'boolean t)
```

vdrValidLayersList

```
layout vdrValidLayersList string "list_of_layerNames"
```

Description

Specifies the valid layers for generating labels and markers. The default is `""` (an empty string), which means that all layers are considered valid for generating labels and markers.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrValidLayersList")
envSetVal("layout" "vdrValidLayersList" 'string "M1 M2 M3 M4 M5")
```

Related Topics

Specifying Layers and Purposes for Generic Voltage Labels

vdrGetValidLayers

vdrSetValidLayers

vdrVerbose

```
layout vdrVerbose boolean { t | nil }
```

Description

Enables verbose mode when generating labels based on a voltage information file. The software prints one message per entry in the file confirming the action taken for that entry.

The default is `nil`, which means that no messages are issued. When set to `t`, messages are printed in the CIW.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrVerbose")
envSetVal("layout" "vdrVerbose" 'boolean t)
```

Related Topics

VSync Constraints Visualizer

vdrVoltagePurposeFile

```
layout vdrVoltagePurposeFile string "fileName"
```

Description

Specifies the name of the voltage purpose file, which determines the layer-purpose pairs on which markers for different voltage values are to be created. The default is `""` (an empty string).

GUI Equivalent

Command: *Tools – Voltage Dependent Rules –*

- *Create Labels/Markers From Simulation Voltages*
- *Create Labels/Markers From Net Voltages*

Field: *Voltage Purpose File*

Examples

```
envGetVal("layout" "vdrVoltagePurposeFile")
envSetVal("layout" "vdrVoltagePurposeFile" 'string "volt_LPP.map")
```

Related Topics

Voltage Dependent Rules (form)

Specifying Layers and Purposes for Voltage Markers

Generating Voltage Markers from Simulation Data for All Nets

Generating Voltage Markers for Manually Entered Voltages

vdrVoltageRounding

```
layout vdrVoltageRounding cyclic { "roundOff" | "floor" | "ceiling" }
```

Description

Specifies the rounding rule to follow for voltage values in marker-based VDR flows.

- `roundOff` rounds the voltage value to the nearest 0.01
- `ceiling` rounds up the voltage value to the nearest 0.01
- `floor` rounds down the voltage value to the nearest 0.01

The default is `"roundOff"`.

GUI Equivalent

Command: *Tools – Voltage Dependent Rules –*

- *Create Labels/Markers From Simulation Voltages*
- *Create Labels/Markers From Net Voltages*

Field: *Voltage Rounding*

Examples

```
envGetVal("layout" "vdrVoltageRounding")
envSetVal("layout" "vdrVoltageRounding" 'cyclic "floor")
```

Related Topics

Voltage Dependent Rules (form)

Generating Voltage Markers from Simulation Data for All Nets

Generating Voltage Markers for Manually Entered Voltages

vdrVSyncCreateCheckLayer

```
layout vdrVSyncCreateCheckLayer string "vSyncSpec"
```

Description

Overrides the vsync shape creation specification defined in the process technology file to improve the routability of the design.

The *vSyncSpec* comprises a list of one or more strings, each separated by a space. Each string specifies one or two input layers and an output layer and purpose on which the vsync shape is to be drawn. The input and output layer specifications are separated by a comma.

t_inputLayer1[:*t_inputLayer2*],*t_outputLayer*:*t_outputPurpose*

For example,

- `"Metal1,Metal1:v_sync"`

  Specifies a single layer check on layer `Metal1` with the vsync shape created on LPP `Metal1 vsync`.

- `"Metal1:Metal2,Metal1:v_sync"`

  Specifies a layer-pair check on layers `Metal1` and `Metal2` with the vsync shape created on LPP `Metal1 vsync`.

The default is `""` (empty string), which means that the vsync layer specification in the technology file is used when creating vsync shapes.

Note: When using the vdrVSyncSanityCheckLayer environment variable, ensure that you set it to the same value as `vdrVSyncCreateCheckLayer`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrVSyncCreateCheckLayer")
envSetVal("layout" "vdrVSyncCreateCheckLayer" 'string "Metal1,Metal1:v_sync
Metal2,Metal2:v_sync Metal1:Metal2,Metal1:v_sync")
```

Related Topics

Defining and Checking Voltage Synced Nets

Specifying Layers and Purposes for Synced Nets

vdrVSyncIgnorePurposes

```
layout vdrDeltaVIgnorePurposes string "listOfPurposes"
```

Description

Specifies a list of purposes on which shapes in the design are ignored when creating VSync markers or performing sanity checks on those markers.

The default value is an empty string, which means that no shapes in the design are ignored when creating VSync markers or performing sanity checks.

The following examples show the differences between when you run `vdrRunVSyncSanityChecker` with and without any `vdrVSyncIgnorePurposes` set.

- Without `vdrVSyncIgnorePurposes` set:

```
envSetVal("layout" "vdrVSyncIgnorePurposes" 'string "")
=> t


vdrRunVSyncSanityChecker(geGetEditCellView())
================================================================================
Layout Design:
Lib/Cell/View:  vsyncExample/pathSegIgnorePurpose/layout
================================================================================
VSync Shape on ('Metal1' 'vsync') and bBox ((5.084 0.6905) (5.516 9.4845)) is
incorrect and is touching a non-net shape.
VSync Shape missing between 'A' and 'B' because no empty area has been found
between the net shapes.
================================================================================
=> t
```

- With `vdrVSyncIgnorePurposes` set:

```
envSetVal("layout" "vdrVSyncIgnorePurposes" 'string "fill pin")
=> t


vdrRunVSyncSanityChecker(geGetEditCellView())
INFO (VDR-2010): All the VSync nets in the cellview are correctly marked with
the VSync shapes.
=> t
```

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrVSyncIgnorePurposes")
envSetVal("layout" "vdrVSyncIgnorePurposes" 'string "fill pin")
```

vdrVSyncSanityCheckLayer

```
layout vdrVSyncSanityCheckLayer string "vSyncSpec"
```

Description

Modifies the vsync layer definition in the process technology file to specify which vsync shapes are to be checked by the VDR Sanity Checker.

The *vSyncSpec* comprises a list of one or more strings, each separated by a space. Each string specifies one or two input layers and an output layer and purpose on which shapes are to be checked. The input and output layer specifications are separated by a comma.

t_inputLayer1[:*t_inputLayer2*],*t_outputLayer*:*t_outputPurpose*

For example,

- `"Metal1,Metal1:v_sync"`

  Specifies a single layer check on layer `Metal1` with the vsync shape created on LPP `Metal1 vsync`.

- `"Metal1:Metal2,Metal1:v_sync"`

  Specifies a layer-pair check on layers `Metal1` and `Metal2` with the vsync shape created on LPP `Metal1 vsync`.

The default is `""` (empty string), which means that the vsync layer specification in the technology file is used by the sanity checker.

Note: When using the vdrVSyncCreateCheckLayer environment variable, ensure that you set it to the same value as `vdrVSyncSanityCheckLayer`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrVSyncSanityCheckLayer")
```

```
envSetVal("layout" "vdrVSyncSanityCheckLayer" 'string "Metal1,Metal1:v_sync
Metal2,Metal2:v_sync Metal1:Metal2,Metal1:v_sync")
```

Related Topics

Defining and Checking Voltage Synced Nets

Specifying Layers and Purposes for Synced Nets

vdrZeroShapeNets

```
layout vdrZeroShapeNets boolean { t | nil }
```

Description

Allows the generation of labels or markers for nets on which no geometry exists. The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "vdrZeroShapeNets")
envSetVal("layout" "vdrZeroShapeNets" 'boolean t)
envSetVal("layout" "vdrZeroShapeNets" 'boolean nil)
```

Related Topics

Voltage Dependent Rules (form)

Generating Voltage Labels from Simulation Data for All Nets

vdrZeroVoltageNets

```
layout vdrZeroVoltageNets string "list_of_netNames"
```

Description

Lists the names of nets that have voltage values of (0,0) but for which labels or markers should be generated anyway.

List the net names, with each name separated by a space or a comma and the whole list enclosed by double quotes. The asterisk (*) is supported as a wildcard character.

The default is `""` (an empty string), which means that the corresponding form field is seeded with the names of nets that have (0,0) voltage values and signal type `ground`.

Voltage labels are created for zero-voltage nets specified by the environment variable `vdrZeroVoltageNets`, irrespective of the value set for the environment variable `vdrUseDatasetsOnlyForLabelCreation`.

GUI Equivalent

Command:	*Tools – Voltage Dependent Rules – Create Labels/Markers From Net Voltages*
	Navigator – RMB Net Name – Create Voltage Labels/Markers
Field:	*Zero Voltage Nets*

Examples

```
envGetVal("layout" "vdrZeroVoltageNets")
envSetVal("layout" "vdrZeroVoltageNets" 'string "VSS VSS1")
envSetVal("layout" "vdrZeroVoltageNets" 'string "VSS VSS*")
```

Related Topics

Voltage Dependent Rules (from net voltages) (form)

Voltage Dependent Rules (form)

Generating Voltage Labels for Manually Entered Voltages

Layout-centric VDR Flow

vdrUseDatasetsOnlyForLabelCreation

A

Voltage Dependent Rules Forms

The following forms are used in the voltage dependent rules flows.

- Delete VDR Objects
- EAD Setup
- VDR Dataset
- VDR Debugger
- VDR Sanity Checker
- Voltage Dependent Rules
- Voltage Dependent Rules (from net voltages)
- VSync Constraints Visualizer

Delete VDR Objects

Use this form to delete specified VDR objects.

You can open this form by selecting *Tools — Voltage Dependent Rules — Delete VDR Objects* from the menu bar of Virtuoso Layout Suite EXL or MXL.

Labels/Markers deletes voltage labels and voltage markers.

Complete Hierarchy deletes voltage labels and voltage markers for all nets across the complete hierarchy.

Hierarchy Stop Level specifies the number of hierarchy levels up to which voltage labels and markers are deleted. the default value is `0`, which indicates that voltage label and markers are deleted only for top-level nets. you can specify any integer value between `0`–`32` in this spin box. the maximum value `32` indicates that voltage labels and markers are deleted for all nets across `32` hierarchy levels.

VSync Shapes deletes VSync shapes.

Userdv Shapes deletes Userdv shapes.

Related Topics

vdrDeleteGUI

EAD Setup

Use the **EAD Setup** form to enable EAD mode, specify the design under test, and enable voltage capture in Virtuoso ADE.

Note: The other controls on the form relate to the Virtuoso Electrically Aware Design flow. For more information, see The EAD Setup Form in the *Virtuoso Electrically Aware Design Flow Guide*.

Related Topics

Enabling Voltage Capture in Virtuoso ADE

Simulation Driven VDR Flow

VDR Dataset

Use the **VDR Dataset** form to view, refresh, and delete voltage information derived from simulation datasets and CSV datasets. The form lists the nets in each dataset and the minimum and maximum voltages allowed for each net. You can also create a new CSV dataset from an existing CSV file.

For more information, see Working with VDR Datasets in the *Virtuoso Electrically Aware Design Flow Guide*.

Related Topics

Simulation Driven VDR Flow

VDR Debugger

(ICADVM20.1 EXL Only) Use this form to search for and display label and shape information for debugging.

Search lets you search for objects based on specific criteria.

Search For lets you select what type of object to search for.

- *Labels* lets you search for labels.
- *VSync Shapes* lets you search for VSync shapes.
- *Userdv Shapes* lets you search for Userdv shapes.
- *Min & Max* lets you search for both Vmin and Vmax labels (*Labels* only).
- *Min* lets you search for Vmin labels only (*Labels* only).
- *Max* lets you search for Vmax labels only (*Labels* only).
- *Hierarchical* lets you search labels, VSync, and Userdv shapes across complete hierarchy.

Report Labels On Connected Nets lists labels on connected nets (*Labels* only).

Select Nets From lets you select the nets from either the *Layout* itself or a *CSV* file.

Net Name lets you specify the net to be searched by name (*Labels* only).

First Net Name lets you specify the name of the first net that is used for the search.

Second Net Name lets you specify the name of the second net that is used for the search.

Voltage Range lets you define a voltage range for searching for labels and shapes.

Search Results lists the search results.

Zoom To Figure enables zooming on the object.

Net displays the net name for the current searched label. Clicking *Next* and *Previous* changes this value according to the net name corresponding to the searched label (*Labels* only).

Net Pair displays the net pair name for the current searched label. Clicking *Next* and *Previous* changes this value according to the net pair name corresponding to the searched label.

Trace displays a visualization of the connected nets.

Trace Net displays the visualization.

Untrace All Nets removes all visualization.

Related Topics

Searching Voltage Labels for Debugging

Searching VSync Shapes for Debugging

Searching Userdv Shapes for Debugging

VDR Sanity Checker

Use the **VDR Sanity Checker** form to check constrained VDR voltage labels in the layout view and report any labels that are missing or which have values different from the values specified for the net in the schematic or layout design. In ICADVM20.1 Layout EXL, you can also check that vsync shapes in the layout canvas all have a corresponding *Voltage Synced Nets* constraint in the layout view. You can also check that Userdv shapes in the layout canvas all have a corresponding *User Delta Voltage* constraint in the layout view.

Note: The form is available only if the vdrConstraintGroupName environment variable is set. This environment variable also enables the constrained VDR label flow.

Check Labels runs the sanity checker on voltage values printed as labels in the layout view.
Environment variable: vdrSanityCheckerObjectType

Check VSync Shapes runs the sanity check on vsync shapes in the layout view (ICADVM20.1 EXL Only).
Environment variable: vdrSanityCheckerObjectType

Check Userdv Shapes runs the sanity check on Userdv shapes in the layout view (ICADVM20.1 EXL Only).
Environment variable: vdrSanityCheckerObjectType

Check Against Schematic checks the voltage values against layout net properties.
Environment variable: vdrSanityCheckerCheckAgainst

Check Against Layout checks the voltage values against schematic net properties.
Environment variable: vdrSanityCheckerCheckAgainst

Check Against CSV checks the voltage values against the values in a CSV file.
Environment variable: vdrSanityCheckerCheckAgainst

Tolerance specifies the threshold beyond which voltage mismatches are to be reported.

- *Absolute* specifies that the tolerance value is an absolute value
- *Relative* specifies that the value is a relative percentage based on the net voltage.

Environment variables: vdrSanityCheckerTolerance and vdrSanityCheckerToleranceType

Generate report in text file Specifies that the sanity checker comparison report is to be captured in a log file. Use *File Name* to specify the path and name of the log file.
Environment variables: vdrSanityCheckerGenLogFile and vdrSanityCheckerLogFile

Related Topics

Sanity Checking Voltage Values in Constrained Labels

Performing a Label Sanity Check

Defining and Checking Voltage Synced Nets

Voltage Dependent Rules

Use the **Voltage Dependent Rules** form to bring voltage data from Virtuoso ADE into the OpenAccess layout view and generate labels or markers for the minimum and maximum voltages in the layout canvas.

Simulation Datasets lets you choose one or more voltage datasets containing the maximum and minimum values you want to reference in your layout design. If required, click *Update* to retrieve the latest versions of the listed voltage datasets from Virtuoso ADE.

Mode

Replace specifies that the voltage values for the nets in the selected datasets are to be replaced in the layout design. This is the default. Voltages are replaced **only** for the nets listed in the selected datasets.

Update specifies that the voltage values for the nets in the selected datasets are to be updated in the layout design, but only if

- The minimum voltage value specified for a net is lower than the voltage value (database or label) currently in the layout design
- The maximum voltage value specified for a net is higher than the voltage value (database or label) currently in the layout design

Override Manually Entered Voltages specifies that any manually entered voltage values on nets are overridden by the values from the simulation datasets.

Important

Manually-entered values are values entered on nets using the Property Editor assistant in VLS (or in VSE and then propagated to the layout using *Generate All From Source*). By default, these values are not overridden. The only exception is if you set both minimum and maximum voltages to 0 manually in the Property Editor assistant and then generate labels directly from the Navigator using these values. Those labels *will* be overwritten if you subsequently run the simulation-driven flow (provided the dataset you use contains values for the nets in question and even if *Override Manually Entered Voltages* is switched off).

Voltage Labels

Generate Voltage Labels creates labels on nets based on the settings specified in the group box at the bottom of the form. When you check the box, *Generate Voltage Markers* is automatically disabled.
Environment variable: vdrEnableHierStopField

In *Replace* mode, all voltage labels are replaced for the nets in the selected datasets. In *Update* mode:

- Minimum voltages are updated only if an existing label shows a higher voltage than the corresponding value in the selected datasets
- Maximum voltages are updated only if the existing label shows a lower voltage than the corresponding value in the selected datasets

Labels for lower-level cells are generated at the current level of hierarchy. Layout XL updates only system-generated labels; user-generated labels are left untouched.

Low Voltage Purpose specifies the layer purpose to use for minimum voltage labels. The default is `vlo`.
Environment variable: vdrLowVoltagePurpose

High Voltage Purpose specifies the layer purpose to use for maximum voltage labels. The default is `vhi`.
Environment variable: vdrHighVoltagePurpose

Special Voltage LPP File is a text file that can be used to override the default *High Voltage Purpose* and *Low Voltage Purpose* settings if your process requires it. See Specifying Layers and Purposes for Generic Voltage Labels for more information.
Environment variable: vdrLayerPurposeFile

Labels for Open Nets specifies whether to create labels for disjoint nets.
Valid values are:

- *None*: Creates only one label per net, regardless of how many disjoint sections the net has. This is the default value.
- *Virtual*: Creates labels for each disjoint section of a net that are marked by common label netname:
- *All*: Creates a label for each disjoint section of a net.

Environment variable: vdrCreateLabelsForOpenNet

Voltage Markers

Generate Voltage Markers creates markers on nets based on the settings specified in the group box at the bottom of the form. When you check the box, *Generate Voltage Labels* is automatically disabled.
Environment variable: vdrGenerateMarkers

The behavior in *Replace* and *Update* modes is the same as described in the Voltage Labels section.

Voltage Purpose File specifies the name of a voltage purpose file, which lists the layer-purpose pairs on which markers for different voltage values are to be created. Click *Browse* to locate the file in your file system.
Environment variable: vdrVoltagePurposeFile

Net Voltage Mode specifies whether markers are to be created for maximum, minimum, or all voltage values.
Environment variable: vdrNetVoltageMode

- *maxVoltage* creates markers only for maximum voltage values (the default)
- *minVoltage* creates markers only for minimum voltage values
- *bothVoltage* creates markers for all voltage values

Voltage Rounding specifies the rounding rule to follow for voltage values.
Environment variable: vdrVoltageRounding

- *roundOff* rounds the voltage value to the nearest 0.01 (the default)
- *ceiling* rounds up the voltage value to the nearest 0.01
- *floor* rounds down the voltage value to the nearest 0.01

The common options at the bottom of the form let you specify for which nets labels or markers are to be generated and how far down the hierarchy to search for those nets. The *Size* option lets you control the size of the labels and markers that are created. The *Decimal Places* option lets you control the number of decimal places for net voltage labels or markers that are created.

Net Selection specifies whether labels or markers are to be generated for external nets, internal nets, or all nets.
Environment variable: vdrGenerateLabelsOn

- *External and Internal Nets* (the default) specifies that labels or markers are generated for all nets in the design.
- *External Nets Only* specifies that labels or markers are generated only for external nets. A net is considered external if it is connected to any of the terminals of the cellview to which it belongs. When the cell is instantiated at a higher level, these nets are propagated up the hierarchy, allowing you to make connections to the terminals to which they are connected.
- *Internal Nets Only* specifies that labels or markers are to be generated only for nets that are wholly internal to the cellview in which they belong. When the cellview is instantiated at a higher level, internal nets are not propagated up the hierarchy.

Hierarchy Stop Level specifies how many hierarchy levels the software searches to find nets on which to generate labels or markers.
Environment variables: vdrHierarchyStopLevel, vdrEnableHierStopField

For example,

- 0 means that labels/markers are generated for top-level nets only (the default)
- 1 means top-level nets and nets located one level below in the hierarchy
- 2 means top-level nets and nets located one and two levels below in the hierarchy

Size specifies the height of the labels or markers created. The default is 0.0. For labels, this means that the label is automatically sized to match the height of the shape with which it is associated. For markers, it prevents the marker from being created at all.
Environment variable: vdrLabelHeight

Decimal Places specifies the number of decimal places for net voltage labels or markers being created between 0 and 5. If a value outside this range is specified, the value is reset to the default value. The default is 2.
Environment variable: vdrPrecision

Related Topics

Specifying Layers and Purposes for Generic Voltage Labels

Specifying Layers and Purposes for Voltage Markers

Storing Voltages in the OpenAccess Database

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Markers from Simulation Data for All Nets

Simulation Driven VDR Flow

Voltage Dependent Rules (from net voltages)

Use this form to generate voltage labels or markers for nets selected directly from the Navigator assistant.

Voltage Labels from Net Voltages

Low Voltage Purpose specifies the layer purpose to use for minimum voltage labels. The default is `vlo`.
Environment variable: vdrLowVoltagePurpose

High Voltage Purpose specifies the layer purpose to use for maximum voltage labels. The default is `vhi`.
Environment variable: vdrHighVoltagePurpose

Special Voltage LPP File is a text file that can be used to override the default *High Voltage Purpose* and *Low Voltage Purpose* settings if your process requires it. See Specifying Layers and Purposes for Generic Voltage Labels for more information.
Environment variable: vdrLayerPurposeFile

Voltage Markers

Generate Voltage Markers creates markers on nets based on the settings specified in the group box at the bottom of the form. When you check the box, the voltage purpose options above are automatically disabled.
Environment variable: vdrGenerateMarkers

Voltage Purpose File specifies the name of a voltage purpose file, which lists the layer-purpose pairs on which markers for different voltage values are to be created. Click *Browse* to locate the file in your file system.
Environment variable: vdrVoltagePurposeFile

Net Voltage Mode specifies whether markers are to be created for maximum, minimum, or all voltage values.
Environment variable: vdrNetVoltageMode

- *maxVoltage* creates markers only for maximum voltage values (the default)
- *minVoltage* creates markers only for minimum voltage values
- *bothVoltage* creates markers for all voltage values

Voltage Rounding specifies the rounding rule to follow for voltage values.
Environment variable: vdrVoltageRounding

- *roundOff* rounds the voltage value to the nearest 0.01 (the default)
- *ceiling* rounds up the voltage value to the nearest 0.01

- *floor* rounds down the voltage value to the nearest 0.01

The options at the bottom of the form let you specify the size of the labels or markers that are created and enter a list of nets for which labels or markers are to be generated even if their voltage values are 0.

Size specifies the height of the labels or markers created. The default is 0.0. For labels, this means that the label is automatically sized to match the height of the shape with which it is associated. For markers, it prevents the marker from being created at all.
Environment variable: vdrLabelHeight

Zero Voltage Nets lists the nets which have voltage values of (0,0) but for which labels should be generated anyway.
Environment variable: vdrZeroVoltageNets

The field lists the net names specified by the environment variable. If the environment variable is set to its default value (an empty string), the field lists the nets that have voltage values of (0,0) and signal type `ground`.

Note: Zero voltage nets are ignored by the sanity checker provided there are either no labels at all on the specified net or both Vmin and Vmax values are set to 0. If one or both values is nonzero, or the number of labels is other than 0 or 2, the sanity checker reports an error.

Related Topics

Generating Voltage Labels for Manually Entered Voltages

Generating Voltage Markers for Manually Entered Voltages

Layout-centric VDR Flow

VSync Constraints Visualizer

(ICADVM20.1 EXL Only) Use this form to create *Voltage Synced Nets* (vsync) constraints from the contents of a CSV file, list the vsync constraints currently present in the layout view, and delete those that are no longer required.

Create Constraints From CSV File lets you choose a CSV file from your file system and use it as a source from which to create vsync constraints in your design.

Browse helps you locate the CSV file you require.

Create lets you generate vsync constraints in the layout view based on the entries in the selected file.

Voltage Synced Nets lists the vsync constraints currently present in the design.

Refresh lets you update the list to reflect any changes made since the last update.

Delete removes the selected constraints from the list and the design.

Related Topics

Defining and Checking Voltage Synced Nets

B

Voltage Dependent Rules Functions

The list below lets you access information about syntax, descriptions, and examples for the Cadence® SKILL functions associated with the Virtuoso® voltage dependent rules flows. Only the functions listed here are supported for public use. Any other functions, and undocumented aspects of the functions described below, are private and subject to change or removal at any time.

VDR-related SKILL Functions

The functions listed below let you create and remove voltage labels for the nets in your design:

- vdrCheckVoltageLabels
- vdrCreateUserdvConstraintsFromFile
- vdrCreateVoltageLabel
- vdrCreateVoltageLabelEx
- vdrCreateVoltageLabelOnNets
- vdrCreateVoltageMarkers
- vdrCreateVoltageMarkersOnNets
- vdrCreateVSyncConstraintsFromFile
- vdrDebuggerGUI
- vdrDeleteGUI
- vdrDeleteLabels
- vdrDisplaySchematicGUI
- vdrGenerateLabelsGUI
- vdrGenerateUserdvShapes

- vdrGenerateVSyncShapes
- vdrGetValidLayers
- vdrResetNetVoltages
- vdrRunSanityChecker
- vdrRunUserdvSanityChecker
- vdrRunVoltageConflictChecker
- vdrRunVSyncSanityChecker
- vdrSanityCheckerGUI
- vdrSetNetVoltageRange
- vdrSetValidLayers
- vdrTransferVSyncConstraints
- vdrVsyncVisualizerGUI

VDR-related Database Access SKILL Functions

The functions listed below let you manipulate voltage values for nets directly in the design database.

- To set, get, and unset the minimum and maximum voltages for a net, use:
  - dbGetNetVoltageRange
  - dbSetNetVoltageRange
  - dbUnsetNetVoltageRange
- To set, get, and unset the voltage source for a net, use:
  - dbGetNetVoltageRangeSource
  - dbSetNetVoltageRangeSource
  - dbUnsetNetVoltageRangeSource
- To set and get the default minimum and maximum voltages for nets in a cellview, use:
  - dbGetCellViewNetVoltageRange
  - dbSetCellViewNetVoltageRange

For detailed information, click any of the links above or see Chapter 2, “Database Access,” in the *Virtuoso Design Environment SKILL Reference*.

vdrCheckVoltageLabels

```
vdrCheckVoltageLabels(
     d_cellviewID
     [ t_reportFilename ]
     [ g_enablePopup ]
     )
     => t / nil
```

Description

Verifies that all top-level nets in the specified layout cellview are correctly labeled on the canvas.

Arguments

d_cellviewID	Database ID of the layout cellview in which labels are to be checked.
t_reportFilename	Name of the report file to which information about missing or incorrect labels is appended. Enclose the filename in double quotes. If you do not specify a filename, the information is printed to the CIW instead.
g_enablePopup	Displays the messages issued by the command in a pop-up window when the command is run (the default). When set to `nil`, no pop-up window appears and the messages are printed to the CIW instead.

Value Returned

`t`	All the top-level nets in the design are correctly labeled on the canvas.
`nil`	Some of the top-level nets in the design do not have correct labels. Check the report file or CIW for details.

Example

```
returnVal = true
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
    when(instHeaders = cellView~>instHeaders
       if(vdrCheckVoltageLabels(instHeader~>master "vdrReport.log" nil) == nil)
         returnVal = nil
       )
    )
  )
)
```

Checks that all the top-level nets in the current design are correctly labeled on the canvas and appends information to a file called `vdrReport.log` in the current working directory.

Related Topics

Checking Voltage Labels in the Layout View

vdrCreateUserdvConstraintsFromFile

```
vdrCreateUserdvConstraintsFromFile(
     t_libName
     t_cellName
     t_viewName
     t_fileName
     )
     => nil
```

Description

Creates User Delta Voltage (userdv) constraints in the specified source schematic cellview based on information from the given CSV file. The schematic view is opened in read-only mode.

Arguments

t_libName	Name of the library in which the schematic cellview resides.
t_cellName	Name of the schematic cell in which constraints are to be created.
t_viewName	Name of the view in which userdv constraints are to be created.
t_fileName	Name of the CSV file specifying the userdv constraints to be created.

Value Returned

None

Example

Creates the userdv constraint specified in file `Data.csv` in `cellview lib1/cell1/schematic` and writes appropriate messages to the log file.

```
scv = dbOpenCellViewByType("lib1" "cell1" "schematic" "" "r")
srcCache = ciCacheGet(scv) ;gives constraint cache of cellview
listOfCons = nil
foreach(elm srcCache~>constraints
          when(elm~>type == 'deltaVoltage listOfCons = cons(elm listOfCons)))
printf("LOG: Constraints in target layout: %L\n" listOfCons~>name)
```

```
output => LOG: Constraints in target layout: nil
vdrCreateUserdvConstraintsFromFile("lib1" "cell1" "schematic" "./Data.csv")
output =>
INFO (CMGR-5020): Created constraint of type 'deltaVoltage' in cache 'lib1 cell1
schematic' with name 'Constr_1'.
INFO (CMGR-5020): Created constraint of type 'deltaVoltage' in cache 'lib1 cell1
schematic' with name 'Constr_2'
srcCache = ciCacheGet(scv)
listOfCons = nil
foreach(elm srcCache~>constraints
          when(elm~>type == 'deltaVoltage listOfCons = cons(elm listOfCons)))
printf("LOG: Constraints in target layout: %L\n" listOfCons~>name)
output=> LOG: Constraints in target layout: ("Constr_2" "Constr_1")
```

vdrCreateVoltageLabel

```
vdrCreateVoltageLabel(
     d_cellviewID
     lt_datasetName
     t_lowVPurposeName
     t_highVPurposeName
     [ f_labelHeight ]
     [ g_externalNets ]
     [ g_internalNets ]
     [ g_update ]
     [ g_generateLabels ]
     [ g_overrideMode ]
     [ x_hierStopLevel ]
     [ u_customFunc ]
     [ u_postLabelCreationCB ]
     [ t_voltageInfoFile ]
     [ t_layerPurposeFile ]
     [ g_verbose ]
     [ t_logFile ]
     [ g_propagateNetVoltages ]
     [ lt_sourceName ]
     [ g_schematicCSV ]
     )
     => t / nil
```

Description

Creates labels on the nets in a layout design reflecting the minimum and maximum voltages from simulation data or a specified list of sources. Labels can also be generated for nets on which no geometry exists if the vdrZeroShapeNets environment variable is set to `t`.

Note: The same functionality is provided by the vdrCreateVoltageLabelEx API, which features keyed optional arguments for added convenience.

Arguments

d_cellviewID	Database ID of the layout cellview for which labels are to be created.

lt_datasetName	One or more strings representing the names of the simulation datasets that contain the minimum and maximum voltage data for the nets in the specified design.
	
	To specify a single dataset name, use this format:
	`"voltages_0"`
	
	To specify more than one dataset name, use this format:
	`("voltages_0" "voltages_1" "voltages_2")`

t_lowVPurposeName	Name of the layer purpose on which to draw minimum voltage labels. This argument is mandatory; the function does not consider the setting of the vdrLowVoltagePurpose environment variable.
t_highVPurposeName	Name of the layer purpose on which to draw maximum voltage labels. This argument is mandatory; the function does not consider the setting of the vdrHighVoltagePurpose environment variable.
f_labelHeight	Height of the labels created. The default is 0.0 microns, which means that the label is automatically sized to match the height of the shape with which it is associated.
g_externalNets	Creates labels for external nets, that is, nets that are connected to the terminals of the cellview to which they belong. The default is `t`. To create labels only for internal nets, specify `nil` for this argument.
g_internalNets	Creates labels for internal nets. The default is `t`. To create labels only for external nets, specify `nil` for this argument.
g_update	Runs label creation in *Update* mode. The default is `nil`, which means that label creation runs in *Replace* mode.

g_generateLabels

Specifies whether labels are to be created on the canvas or not.

The default is `t`.

Use in conjunction with the *propagateNetVoltages* option to control whether only net properties, only labels, both properties and labels, or neither properties nor labels are updated.

g_overrideMode

Specifies that any manually entered voltage values on nets are to be overridden by the values from the simulation datasets.

The default is `nil`, which means that manually entered values are not overridden.

x_hierStopLevel

Number of hierarchy levels the software searches to find the nets on which to create labels. For example:

- `0` means that labels are created only for top-level nets (this is the default)
- `1` means top-level nets and nets located one level below in the hierarchy
- `2` means top-level nets and nets located one and two levels below in the hierarchy

The default is `32`.

u_customFunc

User-defined SKILL procedure that can be called to create or instantiate objects other than regular text labels.

The default is `nil`, which means that no custom SKILL function is specified.

u_postLabelCreationCB

User-defined SKILL procedure that can be used to perform any required post-processing tasks on the created labels.

The default is `nil`, which means that no user-defined post-processing is performed.

For more information, see Post-Processing Voltage Labels and Markers.

t_voltageInfoFile

Specifies the name of a comma-separated value (CSV) file containing voltage information, which can be used as an input for the VDR flow instead of a simulation dataset. The CSV file specifies net names and corresponding minimum and maximum voltage values. The asterisk (*) is supported as a wildcard character. The default is `nil`.

For more information, see Generating Voltage Labels from a Voltage Information File.

t_layerPurposeFile

Special layer purpose file that lets you override the default *t_highVPurposeName* and *t_lowVPurposeName* arguments when a process requires it.

The default is `nil`.

See Specifying Layers and Purposes for Generic Voltage Labels for more information.

g_verbose

Enables verbose mode when generating labels based on a voltage information file. The software prints one message per entry in the file confirming the action taken for that entry.

The default is `nil`, which means that no messages are issued. When set to `t`, messages are printed in the CIW. Use *t_logFile* to save the messages to a separate log file.

The creation of voltage labels in non-verbose mode is faster than that in verbose mode.

t_logFile

Name of a log file in which messages are saved when generating labels from a voltage information file in verbose mode.

g_propagateNetVoltages

Specifies whether the voltage values are to be updated for the specified nets in the database. The default is `t`.

Use in conjunction with the *generateLabel* argument to control whether only properties, only labels, both properties and labels, or neither properties nor labels are updated.

lt_sourceName

	Specifies a list of alternative sources from which voltages values are read and the order in which they are considered. Along with view names, the list may also reference a voltage information file specified by the *voltageInfoFile* argument. For example: `?source list("csv" "schematic" "layout") ?dataFile "csv")` The list of sources is considered only if you have not specified datasets as the source of voltage values. If no dataset, data file, or source list is specified, the voltages in the layout view are used to create labels.
g_schematicCSV	Specifies whether net names in the CSV file are assumed to be in the schematic or layout name space. The default is `nil`, which means that net names are assumed to be in the layout name space. When set to `t`, net names are assumed to be in the schematic name space.

Value Returned

`t`	Labels were created with no errors.
`nil`	Labels could not be created due to errors.

Example

```
vdrCreateVoltageLabel(geGetEditCellView() "voltages_0" "drawing" "drawing" 0.0 t
nil nil t nil 0 nil '_myPostVdrCB)
```

Runs label creation in *Replace* mode for only the top-level nets in the current cellview. Voltage values are taken from a dataset called `voltages_0`, and labels are drawn on layer purpose `drawing`. The code then automatically executes a user-defined callback named `_myPostVdrCB` to perform some post-processing tasks.

```
vdrCreateVoltageLabel(geGetEditCellView() nil "drawing" "drawing" 0.0 t nil nil t
nil 0 nil '_myPostVdrCB "./voltages.csv")
```

Performs the same operation as above but uses the information in the file called `voltages.csv` instead of datasets as input.

```
vdrCreateVoltageLabel(geGetEditCellView() nil "drawing" "drawing" 0.0 t nil nil t
nil 0 nil '_myPostVdrCB "./voltages.csv")
INFO (VDR-2001): VDR annotation deleted 12 voltage labels.
INFO (VDR-2004): VDR annotation created 12 voltage labels.
```

```
INFO (VDR-2005): VDR annotation updated 0 voltage labels.
```

Related Topics

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Labels from a Voltage Information File

vdrCreateVoltageLabelEx

```
vdrCreateVoltageLabelEx(
     d_cellviewID
     lt_datasetName
     t_lowVPurposeName
     t_highVPurposeName
     [ ?labelHeight f_labelHeight ]
     [ ?externalNet { t | nil } ]
     [ ?internalNet { t | nil } ]
     [ ?update { t | nil } ]
     [ ?generateLabel { t | nil } ]
     [ ?overrideMode { t | nil } ]
     [ ?hierStopLevel x_hierStopLevel ]
     [ ?customFunc u_customFunc ]
     [ ?postLabelCreationCB u_postLabelCreationCB ]
     [ ?dataFile t_voltageInfoFile ]
     [ ?lppFile t_layerPurposeFile ]
     [ ?verbose { t | nil } ]
     [ ?logFile t_logFile ]
     [ ?propagateNetVoltages { t | nil } ]
     [ ?source lt_sourceName ]
     [ ?schematicCSV { t | nil } ]
     )
     => t / nil
```

Description

Creates labels on the nets in a layout design reflecting the minimum and maximum voltages from simulation data or a specified list of sources. Labels can also be generated for nets on which no geometry exists if the vdrZeroShapeNets environment variable is set to `t`.

Arguments

d_cellviewID	Database ID of the layout cellview for which labels are to be created.

lt_datasetName

One or more strings representing the names of the simulation datasets that contain the minimum and maximum voltage data for the nets in the specified design.

To specify a single dataset name, use this format:

```
"voltages_0"
```

To specify more than one dataset name, use this format:

```
("voltages_0" "voltages_1" "voltages_2")
```

t_lowVPurposeName

Name of the layer purpose on which to draw minimum voltage labels. This argument is mandatory; the function does not consider the setting of the vdrLowVoltagePurpose environment variable.

t_highVPurposeName

Name of the layer purpose on which to draw maximum voltage labels. This argument is mandatory; the function does not consider the setting of the vdrHighVoltagePurpose environment variable.

?labelHeight *f_labelHeight*

Height of the labels created.

The default is 0.0 microns, which means that the label is automatically sized to match the height of the shape with which it is associated.

?externalNet { t | nil }

Creates labels for external nets, that is, nets that are connected to the terminals of the cellview to which they belong.

The default is `t`. To create labels only for internal nets, specify `nil` for this argument.

?internalNet { t | nil }

Creates labels for internal nets.

The default is `t`. To create labels only for external nets, specify `nil` for this argument.

`?update { t | nil }`

Runs label creation in *Update* mode.

The default is `nil`, which means that label creation runs in *Replace* mode.

`?generateLabel { t | nil }`

Specifies whether labels are to be created on the canvas or not.

The default is `t`.

Use in conjunction with the `?propagateNetVoltages` option to control whether only net properties, only labels, both properties and labels, or neither properties nor labels are updated.

`?overrideMode { t | nil }`

Specifies that any manually entered voltage values on nets are to be overridden by the values from the simulation datasets.

The default is `nil`, which means that manually entered values are not overridden.

`?hierStopLevel` *`x_hierStopLevel`*

Number of hierarchy levels the software searches to find the nets on which to create labels. For example:

- `0` means that labels are created only for top-level nets (this is the default)
- `1` means top-level nets and nets located one level below in the hierarchy
- `2` means top-level nets and nets located one and two levels below in the hierarchy

The default is `32`.

`?customFunc` *`u_customFunc`*

User-defined SKILL procedure that can be called to create or instantiate objects other than regular text labels.

The default is `nil`, which means that no custom SKILL function is specified.

`?postLabelCreationCB` *`u_postLabelCreationCB`*

User-defined SKILL procedure used to perform any required post-processing tasks on the created labels. The default is `nil`, which means that no post-processing is performed. For more information, see Post-Processing Voltage Labels and Markers.

`?dataFile` *`t_voltageInfoFile`*

Specifies the name of a comma-separated value (CSV) file containing voltage information, which can be used as an input for the VDR flow instead of a simulation dataset. The CSV file specifies net names and corresponding minimum and maximum voltage values. The asterisk (*) is supported as a wildcard character.

The default is `nil`.

For more information, see Generating Voltage Labels from a Voltage Information File.

`?lppFile` *`t_layerPurposeFile`*

Special layer purpose file that lets you override the default *`t_highVPurposeName`* and *`t_lowVPurposeName`* arguments when your process requires it.

The default is `nil`.

See Specifying Layers and Purposes for Generic Voltage Labels for more information.

`?verbose { t | nil }`

Enables verbose mode when generating labels based on a voltage information file. The software prints one message per entry in the file confirming the action taken for that entry.

The default is `nil`, which means no messages are issued. When set to `t`, messages are printed in the CIW. Use *`t_logFile`* to save the messages to a separate log file.

`?logFile` *`t_logFile`*

Name of a log file in which messages are saved when generating labels from a voltage information file in verbose mode.

`?propagateNetVoltages { t | nil }`

Specifies whether the voltage values are to be updated for the specified nets in the database. The default is `t`.

Use in conjunction with the `?generateLabel` option to control whether only net properties, only labels, both properties and labels, or neither properties nor labels are updated.

`?source` *`lt_sourceName`*

Specifies a list of alternative sources from which voltages values are read and the order in which they are considered. Along with view names, the list may also reference a voltage information file specified by the `?dataFile` argument. For example:

```
?source list("csv" "schematic" "layout")
?dataFile "voltageInfoFile.csv"
```

The list of sources is considered only if you have not specified datasets as the source of voltage values. If no dataset, data file, or source list is specified, the voltages in the layout view are used to create labels.

`?schematicCSV { t | nil }`

Specifies whether net names in the CSV file are assumed to be in the schematic or layout name space.

The default is `nil`, which means that net names are assumed to be in the layout name space. When set to `t`, net names are assumed to be in the schematic name space.

Value Returned

`t`	Labels were created with no errors.
`nil`	Labels could not be created due to errors.

Examples

```
cv = geGetEditCellView()
=> db:0x2675871a
vdrCreateVoltageLabelEx(cv "voltages_0" "drawing" "drawing" ?internalNet nil
?hierStopLevel 0)
=> t
```

Creates labels from the voltage values from dataset `voltages_0` for the top-level external nets in *Replace* mode.

Related Topics

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Labels from a Voltage Information File

vdrCreateVoltageLabelOnNets

```
vdrCreateVoltageLabelOnNets(
     d_cellviewID
     ld_netIDs
     t_lowVPurposeName
     t_highVPurposeName
     [ f_labelHeight ]
     [ u_customFunc ]
     [ lt_ignoreZeroVoltNets ]
     [ u_postLabelCreationCB ]
     [ t_layerPurposeFile ]
     )
     => t / nil
```

Description

Creates labels from the voltage values entered manually in the Property Editor assistant for the specified top-level nets in the given cellview. The labels are created on the geometry of the net in question. Where there is no geometry, the labels are created on all the Pcell and instance terminals connected to the net. The command works only in Layout XL and higher tiers.

Arguments

d_cellviewID	Database ID of the layout cellview containing the nets for which labels are to be created.
ld_netIDs	List of database IDs identifying the nets for which labels are to be created.
t_lowVPurposeName	Name of the layer purpose on which to draw minimum voltage labels. This argument is mandatory; the function does not consider the setting of the vdrLowVoltagePurpose environment variable.
t_highVPurposeName	Name of the layer purpose on which to draw maximum voltage labels. This argument is mandatory; the function does not consider the setting of the vdrHighVoltagePurpose environment variable.

f_labelHeight	Height of the labels created. The default is 0.0 microns, which means that the label is automatically sized to match the height of the shape with which it is associated.
u_customFunc	User-defined SKILL procedure that can be called to create or instantiate objects other than regular text labels. The default is `nil`, which means that no custom SKILL function is specified.
lt_ignoreZeroVoltNets	List of nets that have voltage values of (0,0) but for which labels should be created anyway. Names must each be enclosed in double quotes and separated by a space. The asterisk (*) is supported as a wildcard character.
t_layerPurposeFile	Special layer purpose file that lets you override the default *t_highVPurposeName* and *t_lowVPurposeName* arguments when your process requires it. See Specifying Layers and Purposes for Generic Voltage Labels for more information.

Value Returned

`t`	Label creation ran without any errors.
`nil`	Label creation encountered errors.

Example

```
vdrCreateVoltageLabelOnNets(geGetEditCellView() netIDs "vlo" "vhi" 0.2)
t
```

Creates labels for the list of `netIDs` in the currently edited cellview. The labels are 0.2 microns high with the minimum voltage label drawn on layer purpose `vlo` and the maximum voltage label on purpose `vhi`.

Related Topics

Generating Voltage Labels for Manually Entered Voltages

vdrCreateVoltageMarkers

```
vdrCreateVoltageMarkers(
     d_cellviewID
     t_voltagePurposeFile
     [ ?dataset t_datasetName ]
     [ ?update { t | nil } ]
     [ ?override_mode { t | nil } ]
     [ ?size f_markerHeight ]
     [ ?externalNet { t | nil } ]
     [ ?internalNet { t | nil } ]
     [ ?hierStopLevel x_hierStopLevel ]
     [ ?postMarkerCreationCB u_postMarkerCreationCB ]
     [ ?csvFile t_voltageInfoFile ]
     [ ?voltageRounding { roundOff | floor | ceiling } ]
     [ ?mode { maxVoltage | minVoltage | bothVoltage } ]
     )
     => t / nil
```

Description

Creates markers for minimum and/or maximum voltages on the nets in the specified design. Voltage information can be taken from simulation datasets, from user-defined voltages on nets, or from a voltage information file. The voltage purpose file specifies on which layer-purpose pair a marker is created depending on the voltage value in question.

Arguments

d_cellviewID
: Database ID of the layout cellview containing the nets for which markers are to be created.

t_voltagePurposeFile
: Lists the layer-purpose pairs on which markers for different voltage values are to be created.

?dataset *lt_datasetName*
: One or more strings representing the names of the simulation datasets containing minimum and maximum voltage data for the nets in the specified design.

  For example, to specify a single dataset name, use this format: `"voltages_0"`. To specify more than one dataset name, type `("voltages_0" "voltages_1" "voltages_2")`

?update { t | nil }

Runs marker creation in *Update* mode, which means that values are based on the last voltages set on net; that is

- The lower of the current minimum and the last minimum
- The higher of the current maximum and the last maximum

The default is `nil`, which means that marker creation is run in *Replace* mode.

`?override_mode { t | nil }`

Specifies that any manually entered voltage values on nets are to be overridden by the values from the simulation datasets.

The default is `nil`, which means that manually entered values are not overridden.

`?size` *`f_markerHeight`*

Height of the markers created.

The default is 0.0 microns, which means that no marker is created.

`?externalNet { t | nil }`

Creates markers for external nets; that is, nets that are connected to the terminals of the cellview to which they belong.

The default is `t`.

`?internalNet { t | nil }`

Creates markers for internal nets.

The default is `t`.

`?hierStopLevel` *`x_hierStopLevel`*

Specifies how many hierarchy levels the software searches to find the nets on which to create markers. For example, 0 means that markers are created for top-level nets only; 1 means top-level nets and nets located one level below in the hierarchy; 2 means top-level nets and nets located one and two levels below in the hierarchy; and so on. The default is 32.

`?postMarkerCreationCB` *`u_postMarkerCreationCB`*

Specifies a user-defined SKILL procedure that can be used to perform any required post-processing tasks on the created labels. For more information, see Post-Processing Voltage Labels and Markers.

`?csvFile` *`t_voltageInfoFile`*

Specifies the name of a comma-separated value (CSV) file containing voltage information, which can be used instead of a simulation dataset as an input for the VDR flow. The CSV file specifies net names and corresponding minimum and maximum voltage values. Wildcard (*) is supported.

For more information, see Generating Voltage Labels from a Voltage Information File.

`?voltageRounding { "roundOff" | "floor" | "ceiling" }`

Specifies the rounding rule to follow for voltage values.

- `roundOff` rounds the voltage value to the nearest 0.01
- `ceiling` rounds up the voltage value to the nearest 0.01
- `floor` rounds down the voltage value to the nearest 0.01

The default is "`roundOff`".

`?mode { "maxVoltage" | "minVoltage" | "bothVoltage" }`

Specifies whether markers are to be created for maximum, minimum, or all voltage values.

The default is `"maxVoltage"`.

Value Returned

`t`	Marker creation ran without any errors.
`nil`	Marker creation encountered errors.

Example

```
when(cellView = geGetEditCellView(window)
     voltageLPPFile = "volt_LPP.map"
     dataset = "VDR_dataset_0"
     vdrCreateVoltageMarkers(cellview voltageLPPFile ?dataset dataset
       ?update nil ?override_mode nil ?size 0.01 ?externalNet t
       ?internalNet nil ?hierStopLevel 3 ?postMarkerCreationCB nil
```

```
        ?csvFile nil ?voltageRounding "floor" ?mode "bothVoltage")
    )
```

Creates markers for the minimum and maximum voltages on external nets in the specified dataset down to hierarchy level 3 of the specified cellview. The markers are 0.01 high. Voltage values are rounded down to the nearest 0.01.

Related Topics

Generating Voltage Markers from Simulation Data for All Nets

Specifying Layers and Purposes for Voltage Markers

vdrCreateVoltageMarkersOnNets

```
vdrCreateVoltageMarkersOnNets(
     d_cellviewID
     ld_netIDs
     t_voltagePurposeFile
     [ ?size f_markerHeight ]
     [ ?postMarkerCreationCB t_postMarkerCreationCB ]
     [ ?voltageRounding { roundOff | floor | ceiling } ]
     [ ?voltageMode { maxVoltage | minVoltage | bothVoltage } ]
     [ ?ignoreZeroVoltNets lt_netNames ]
     )
     => t / nil
```

Description

Creates markers from the voltage values entered manually in the Property Editor assistant for the specified top-level nets in the given cellview.

Arguments

d_cellviewID	Database ID of the layout cellview containing the nets for which markers are to be created.
ld_netIDs	List of database IDs identifying the nets for which markers are to be created.
t_voltagePurposeFile	Lists the layer-purpose pairs on which markers for different voltage values are to be created.
?size *f_markerHeight*	Height of the markers created. The default is 0.0 microns, which means that no marker is created.
?postMarkerCreationCB *t_postMarkerCreationCB*	Specifies the name of a user-defined SKILL procedure that can be used to perform any required post-processing tasks on the generated markers. For more information, see Post-Processing Voltage Labels and Markers.

`?voltageRounding { "roundOff" | "floor" | "ceiling" }`

Specifies the rounding rule to follow.

- `roundOff` rounds the voltage value to the nearest 0.01
- `ceiling` rounds up the voltage value to the nearest 0.01
- `floor` rounds down the voltage value to the nearest 0.01

The default is "`roundOff`".

`?mode { "maxVoltage" | "minVoltage" | "bothVoltage" }`

Specifies whether markers are to be created for maximum, minimum, or all voltage values.

The default is `"maxVoltage"`.

`?ignoreZeroVoltNets` *`lt_netNames`*

Lists the names of nets which have voltage values of (0,0) but for which markers should be created anyway.

Names must each be enclosed in double quotes and separated by a space. The asterisk (*) is supported as a wildcard character.

Value Returned

`t`	Marker creation ran without any errors.
`nil`	Marker creation encountered errors.

Example

```
cv = geGetEditCellView()
netIds = cv~>nets
callback = stringToSymbol("vdrMarkerCB")
vdrCreateVoltageMarkersOnNets(cv netIds "voltagePurpose.map"
  ?size 0.01 ?voltageRounding "floor" ?voltageMode "bothVoltage"
  ?ignoreZeroVoltNets list("AVSS" "AVDD") ?postMarkerCreationCB callback)
```

Creates markers for minimum and maximum voltage values for the list of `netIDs` in the currently edited cellview. The markers are 0.01 high and are generated for nets `AVSS` and `AVDD` even if their values are 0. Voltage values are rounded down to the nearest 0.01. The code automatically executes a user-defined callback named `vdrMarkerCB` to perform some post-processing tasks.

Related Topics

Generating Voltage Markers for Manually Entered Voltages

vdrCreateVSyncConstraintsFromFile

```
vdrCreateVSyncConstraintsFromFile(
     t_libName
     t_cellName
     t_viewName
     t_fileName
     )
     => nil
```

Description

(ICADVM20.1 Only) Creates Voltage Synced Net (vsync) constraints in the specified source schematic cellview based on information from the given CSV file. The schematic view is opened in read-only mode.

Arguments

t_libName	Name of the library in which the schematic cellview resides.
t_cellName	Name of the schematic cell in which constraints are to be created.
t_viewName	Name of the view in which vsync constraints are to be created.
t_fileName	Name of the CSV file specifying the vsync constraints to be created.

Value Returned

None

Example

```
scv = dbOpenCellViewByType("lib1" "cell1" "schematic" "" "r")
srcCache = ciCacheGet(scv);gives constraint cache of cellview
listOfCons = nil
foreach(elm srcCache~>constraints
          when(elm~>type == 'voltageSyncedNets listOfCons = cons(elm listOfCons)))
printf("LOG: Constraints in target layout: %L\n" listOfCons~>name)
output => LOG: Constraints in target layout: nil


vdrCreateVSyncConstraintsFromFile("lib1" "cell1" "schematic" "./Data.csv")


output =>
INFO (CMGR-5020): Created constraint of type 'voltageSyncedNets' in cache 'lib1
```

```
cell1 schematic' with name 'Constr_1'.
INFO (CMGR-5020): Created constraint of type 'voltageSyncedNets' in cache 'lib1
cell1 schematic' with name 'Constr_2'


srcCache = ciCacheGet(scv)
listOfCons = nil
foreach(elm srcCache~>constraints
          when(elm~>type == 'voltageSyncedNets listOfCons = cons(elm listOfCons)))
printf("LOG: Constraints in target layout: %L\n" listOfCons~>name)
output=> LOG: Constraints in target layout: ("Constr_2" "Constr_1")
```

Creates the vsync constraint specified in file `Data.csv` in cellview `lib1/cell1/schematic` and writes appropriate messages to the log file.

Related Topics

VSync Constraints Visualizer

Defining and Checking Voltage Synced Nets

vdrDebuggerGUI

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

Description

Opens the VDR Debugger form.

Arguments

None

Value Returned

`t`	The VDR Debugger form is displayed.
`nil`	Failed to open the VDR Debugger form because an error occurred.

Example

```
vdrDebuggerGUI()
t
```

Displays the VDR Debugger form.

Related Topics

VDR Debugger

vdrDeleteGUI

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

Description

Opens the Delete VDR Objects form, where you can select *Labels/Markers*, *VSync Shapes*, and *Userdv Shapes* check boxes to delete voltage labels and markers, VSync shapes, and Userdv shapes, respectively.

Arguments

None

Value Returned

`t`	The Delete VDR Objects form is displayed.
`nil`	Failed to open the Delete VDR Objects form because an error occurred.

Example

```
vdrDeleteGUI()
=> t
```

Opens the Delete VDR Objects form.

Related Topics

VDR Debugger

vdrDeleteLabels

```
vdrDeleteLabels(
     d_cellviewID
     )
     => nil
```

Description

Deletes from the specified cellview all the labels and markers created using the VDR flows.

Arguments

d_cellviewID	Database ID of the top layout cellview from which labels and markers are to be deleted.

Value Returned

`nil`	All the VDR labels and markers in the cellview were deleted.

Example

```
vdrDeleteLabels(topCvId)
```

Deletes all the VDR labels and markers in the cellview with the given database ID.

Related Topics

Deleting Voltage Labels and Markers

vdrDisplaySchematicGUI

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

Description

Opens the Voltage Dependent Rules form, which you can use to transfer voltage data on schematic nets from the specified simulation voltage datasets. This function works only in Layout XL and higher tiers.

Arguments

None

Value Returned

`t`	The Voltage Dependent Rules form opens.
`nil`	The form cannot be opened, possibly because you are not suing Layout XL.

Examples

The following example opens the Voltage Dependent Rules form.

```
vdrDisplaySchematicGUI()
=> t
```

Related Topics

Generating Voltage Labels from Simulation Data for All Nets

Voltage Dependent Rules

vdrGenerateLabelsGUI

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

Description

Opens the Voltage Dependent Rules form, which you can use to create voltage labels or markers on nets. The command works only in Layout XL and higher tiers.

Arguments

None

Value Returned

`t`	The form was opened.
`nil`	The form could not be opened, possibly because you are not using Layout XL.

Example

```
vdrGenerateLabelsGUI()
=> t
```

Opens the Voltage Dependent Rules form.

Related Topics

Generating Voltage Labels from Simulation Data for All Nets

Generating Voltage Markers from Simulation Data for All Nets

vdrGenerateUserdvShapes

```
vdrGenerateUserdvShapes(
     d_cellviewID
     [ ?verbose { t | nil } ]
     [ ?logFile t_logFile ]
     )
     => t / nil
```

Description

(ICADVM20.1 EXL Only) Creates User Delta Voltage (Userdv) shapes between nets that are constrained by a User Delta Voltage constraint. The Userdv marker shape and associated text label that are created correspond to each User Delta Voltage constraint. You can optionally print messages relating to problems encountered during Userdv shape creation and save these messages to a specified log file if required.

Arguments

d_cellviewID	Database ID of the layout cellview in which the Userdv shapes are to be created.
?verbose { t | nil }	Creates verbose reports displaying debug messages during Userdv shape creation. The reports include reasons for unsuccessful attempts to create Userdv shapes. The default is `nil`, which means that no messages are issued. When set to `t`, messages are printed in the CIW. Use *t_logFile* to save the messages to a separate log file. The creation of Userdv shapes in non-verbose mode is faster than that in verbose mode.
?logFile *t_logFile*	Name of the debug message log file.

Value Returned

`t`	Userdv shapes were created.
`nil`	Userdv shapes could not be created.

Examples

```
vdrGenerateUserdvShapes(cv)
```

Generates a Userdv marker shape and associated text label corresponding to each User Delta Voltage constraint in the given cellview.

```
vdrGenerateUserdvShapes(cv ?verbose t ?logFile "userdv.log")
```

Generates Userdv marker shapes and prints messages relating to problems encountered during the creation. Those messages are saved in a file called `"userdv.log"`.

Related Topics

VDR Debugger

vdrGenerateVSyncShapes

```
vdrGenerateVSyncShapes(
     d_cellviewID
     [ ?verbose { t | nil } ]
     [ ?logFile t_logFile ]
     )
     => t / nil
```

Description

(ICADVM20.1 EXL Only) Creates voltage sync (vsync) shapes between nets that are constrained by a common voltage sync constraint. You can optionally print messages relating to problems encountered during vsync shape creation and save these messages to a specified log file if required.

Arguments

d_cellviewID	Database ID of the layout cellview in which the vsync shapes are to be created.
?verbose { t | nil }	Create verbose reports displaying debug messages during vsync shape creation. The reports include reasons for unsuccessful attempts to create vsync shapes. The default is `nil`, which means that no messages are issued. When set to `t`, messages are printed in the CIW. Use *t_logFile* to save the messages to a separate log file. Creation of vsync shapes in non-verbose mode is faster than that in verbose mode.
?logFile *t_logFile*	Debug message log file.

Value Returned

t	The vsync shapes were created.
nil	The vsync shapes could not be created.

Example

```
when(window = hiGetCurrentWindow()
  when(cellview = geGetEditCellView(window)
   vdrGenerateVSyncShapes(cellview ?verbose ?logFile "vsyncLog.txt")
  )
)
```

Generates vsync shapes for the cellview being edited in the current session and creates a verbose report with name `vsyncLog.txt`.

Related Topics

Defining and Checking Voltage Synced Nets

vdrGetValidLayers

```
vdrGetValidLayers(
     )
     => l_layers / nil
```

Description

Returns the list of valid layers on which labels or markers can be generated. The list was specified previously using the `vdrSetValidLayers` function.

Arguments

None

Value Returned

l_layers	List of layer names.
nil	No layer names have been registered.

Example

```
cv = geGetEditCellView()
if(cv then
    layers = list("Metal1" "Metal2" "Poly")
    vdrSetValidLayers(cv layers)
)

vdrGetValidLayers()
>("Metal1" "Metal2" "Poly")
```

Returns the list of layer names registered using the vdrSetValidLayers command.

Related Topics

vdrSetValidLayers

vdrValidLayersList

Specifying Layers and Purposes for Generic Voltage Labels

vdrResetNetVoltages

```
vdrResetNetVoltages(
     d_cellviewID
     [ ?hierDepth x_hierDepth ]
     )
     => t / nil
```

Description

Resets the minimum and maximum voltage values and voltage range source for all nets to zero in the specified layout or schematic view.

Arguments

d_cellviewID	ID of the layout or the schematic view.
?hierDepth *x_hierDepth*	Specifies the hierarchy levels. Default value is `0`, which indicates that voltage values are reset only for top-level nets. You can specify any integer value from `0`–`32`.

Value Returned

`t`	Net voltages are reset successfully.
`nil`	Net voltages cannot be reset because of an error.

Examples

The following example resets the top-level net voltages in the specified layout cellview.

```
cvID=dbOpenCellViewByType("vdr_demo" "inverter_chain" "layout")
;Opens the specified layout cellview.
vdrResetNetVoltages(cvID)
;Resets the top-level net voltages in the currently open layout cellview.
=> db:0x2bb8e89a
=> t
```

The following example resets voltage values for the nets up to two levels down from the top level.

```
cvID=dbOpenCellViewByType("vdr_demo" "inverter_chain" "layout")
;Opens the specified layout cellview.
vdrResetNetVoltages(cvID ?hierDepth 2)
;Resets the voltage values for the top-level nets and nets located one and two
levels below the hierarchy.
=> db:0x2bb8e89a
=> t
```

vdrRunSanityChecker

```
vdrRunSanityChecker(
     d_cellviewID
     [ g_checkMarkers ]
     [ g_checkAgainstLayout ]
     [ f_tolerance ]
     [ g_isToleranceAbsolute ]
     [ t_logFile ]
     [ t_csvName ]
     [ l_datasets ]
     [ t_checkAgainstSource ]
     [ g_completeHierarchy ]
     )
     => t / nil
```

Description

Checks constrained voltage labels in the layout against the voltage values stored in the schematic or layout net properties and reports any discrepancies between the values. You can optionally apply a threshold up to which mismatches are tolerated and not reported. You can also specify the location and name of a log file in which the report is captured.

Arguments

d_cellviewID	Database ID of the layout cellview in which labels and markers are to be checked.
g_checkMarkers	Specifies whether to check markers (`t`) or labels (`nil`) in the layout. Environment variable: vdrSanityCheckerObjectType
g_checkAgainstLayout	Specifies whether to check the voltage values in labels or markers against layout net properties (`t`) or schematic net properties (`nil`). Environment variable: vdrSanityCheckerCheckAgainst
f_tolerance	Threshold beyond which mismatches are to be reported. Environment variable: vdrSanityCheckerTolerance
g_isToleranceAbsolute	

Specifies whether the tolerance value is considered an absolute value (`t`) or a relative percentage based on the net voltage (`nil`).

Environment variable: vdrSanityCheckerToleranceType

t_logFile

Path and name of a log file in which the discrepancy report is to be captured.

When you specify a log filename, only a summary message is printed in the CIW. If you do not specify a log filename, discrepancies are reported in a table printed in the CIW.

Environment variable: vdrSanityCheckerLogFile

t_csvName

Specifies the path and name of the CSV file containing voltage values against which labels are checked.

This argument is considered only when the argument *t_checkAgainstSource* is set to `CSV`.

Environment Variable: vdrSanityCheckerCsvFileName

l_datasets

Specifies a list of space-separated datasets against which labels are checked.

This argument is considered only when the argument *t_checkAgainstSource* is set to `Datasets`.

Environment Variable: vdrSanityCheckerDatasets

t_checkAgainstSource

Specifies whether voltage values in labels are checked against schematic net properties, layout net properties, a CSV file, or specified datasets.

Valid values are: `Schematic`, `Layout`, `CSV`, and `Datasets`.

The default value is `Schematic`.

Environment Variable: vdrSanityCheckerCheckAgainst

g_completeHierarchy

Specifies whether to perform sanity checks on the complete hierarchy.

Valid values are `t` and `nil`.

The default value is `nil`.

Environment Variable: vdrSanityCheckerCompleteHierarchy

Value Returned

`t`	The comparison report was generated.
`nil`	The sanity check could not be run.

Example

The following example checks the voltage labels in the currently edited layout cellview against the values stored in the layout net properties and captures the report in a file called `vdrReport.log`. Only discrepancies of more than `0.04`V are reported.

```
checkMarkers = nil
checkAgainstLayout = t
tolerance = 0.04
isToleranceAbsolute = t
logFile = "vdrReport.log"
csvName = nil
datasets = "VDR_dataset_0 VDR_dataset_1"
checkAgainstSource = "Datasets"
completeHierarchy = t
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
     vdrRunSanityChecker(cellView checkMarkers checkAgainstLayout tolerance
     isToleranceAbsolute logFile csvName datasets checkAgainstSource
     completeHierarchy)
   )
)
=> t
```

The following example checks the voltage labels in the currently edited layout cellview against the values stored in the `myCSV.csv` file and captures the report in a file `vdrReportCSV.log`. Only discrepancies of more than `0.3`V are reported.

```
checkMarkers = nil
checkAgainstLayout = nil
tolerance = 0.3
isToleranceAbsolute = t
logFile = "vdrReportCSV.log"
csvName = "home/user/myCSV.csv"
datasets = ""
checkAgainstSource = "CSV"
completeHierarchy = nil
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
     vdrRunSanityChecker(cellView checkMarkers checkAgainstLayout tolerance
     isToleranceAbsolute logFile csvName datasets checkAgainstSource
     completeHierarchy)
   )
)
=> t
```

Related Topics

Sanity Checking Voltage Values in Constrained Labels

vdrRunUserdvSanityChecker

```
vdrRunUserdvSanityChecker(
     d_cellviewID
     [ t_logFile ]
     )
     => t / nil
```

Description

(ICADVM20.1 EXL Only) Checks whether there are valid User Delta Voltage (Userdv) shapes on nets that are constrained by a common Userdv constraint. The checker reports any Userdv shapes that are wrongly created and any nets that have a Userdv constraint defined but no Userdv shape between them.

Arguments

d_cellviewID	Database ID of the layout cellview to be checked.
t_logFile	Path and name of a log file in which the report is to be captured. When you specify a log filename, only a summary message is printed in the CIW. If you do not specify a log filename, discrepancies are reported in a table printed in the CIW. Environment variable: vdrSanityCheckerLogFile

Value Returned

t	The sanity checks on the userdv shapes are run successfully.
nil	The sanity checks on the userdv shapes cannot be run because of an error.

Examples

The following example runs Sanity Checker on the Userdv shapes in the currently edited layout cellview and saves the report in the log file `userdvCoherenceReport.log`.

```
window=hiGetCurrentWindow()
;Return the ID of the current window.
cellView=geGetWindowCellView(window)
;Returns the database ID of the cellview displayed in the current window.
```

```
scv = dbOpenCellViewByType("vdr_demo" "inverter_chain" "layout" "" "r")
;Opens the specified library cellview in read-only mode
srcCache = ciCacheGet(scv) ;Gives constraint cache of cellview
listOfCons = nil
foreach(elm srcCache~>constraints
          when(elm~>type == 'deltaVoltage listOfCons = cons(elm listOfCons)))
printf("LOG: Constraints in target layout: %L\n" listOfCons~>name)
output => LOG: Constraints in target layout: nil
vdrCreateUserdvConstraintsFromFile("vdr_demo" "inverter_chain" "layout" "./
myuserdvData.csv")
vdrGenerateUserdvShapes(
                        scv
                        ?verbose t
                        ?logFile "./userdvLogFile"
                        )
;Creates userdv shapes between nets constrainted by userdv constraints
vdrRunUserdvSanityChecker(
                             cellView
                             "./userdvCoherenceReport.log"
                             )
; Runs Sanity Checker on the userdv shapes created in the specified cellview.
=> window:2
=> db:0x2c3deb1a
=> db:0x2c3deb1a
=> ci:0x3b00a3f0
=> nil
=> nil
=>
LOG: Constraints in target layout: nil
=> t
=>
INFO (CMGR-5020): Created constraint of type 'deltaVoltage' in cache 'vdr_demo
inverter_chain layout' with name 'Constr_0'.
INFO (CMGR-5020): Created constraint of type 'deltaVoltage' in cache 'vdr_demo
inverter_chain layout' with name 'Constr_3'.
=> t
=>
INFO (VDR-2011): VDR annotation deleted 0 Userdv Shapes.
INFO (VDR-2000): Successfully created Userdv Shape between 'E' and 'F' on ('Metal1'
'vsync') for voltage 0.4 and bBox ((1.164 0.0415) (1.244 0.4845)).
```

```
INFO (VDR-2000): Successfully created Userdv Shape between 'A' and 'B' on ('Metal1'
'vsync') for voltage 0.4 and bBox ((1.734 0.0415) (1.814 0.4845)).


INFO (VDR-2012): VDR annotation created 2 Userdv Shapes.
=> t
=>
INFO (VDR-2010): All the Userdv nets in the cellview 'vdr_demo/inverter_chain/
layout' are correctly marked with the Userdv shapes.
=> t
```

Related Topics

Sanity Checking Voltage Values in Constrained Labels

vdrRunVoltageConflictChecker

```
vdrRunVoltageConflictChecker(
     d_cellviewID
     [ ?hierDepth x_hierDepth ]
     [ ?precision x_precision ]
     [ ?tolerance n_tolerance ]
     [ ?toleranceType t_toleranceType ]
     [ ?reportFile t_reportFile ]
     [ ?skipZeroVoltageNets g_skipZeroVoltageNets ]
     [ ?fixConflict g_fixConflict ]
     )
     => t / nil
```

Description

Checks voltage conflicts between top-level and hierarchical nets on the specified layout or schematic cellview. Conflict report is printed in the CIW and also saved in the file `conflict.rpt` by default.

Arguments

`d_cellviewID` ID of the layout or schematic cellview.

`?hierDepth` *`x_hierDepth`*

Hierarchy level upto which conflicts are to be reported.

Default value is `0`, which indicates that conflicts are reported only for top-level nets.

You can specify any integer value between `0`–`32`.

`?precision` *`x_precision`*

Specifies the precision value.

Default value is `2`.

`?tolerance` *`n_tolerance`*

Threshold beyond which voltage conflicts are to be reported.

Default tolerance value is `0.0`.

`?toleranceType` *`t_toleranceType`*

Specifies whether the tolerance value is to be considered as an absolute value or a relative percentage based on the net voltage.

Default value is `Absolute`.

Valid values are: `Absolute` and `Relative`.

`?reportFile` *`t_reportFile`*

Name of the report file in which conflicts are to be printed.

The default value is `conflict.rpt`.

`?skipZeroVoltageNets` *`g_skipZeroVoltageNets`*

Specifies whether to skip reporting conflicts for zero-voltage hierarchical nets.

The default value is `t`, which indicates that conflicts for zero-voltage hierarchical nets are not reported.

`?fixConflict` *`g_fixConflict`*

Specifies whether to fix the reported conflicts.

The default value is `nil`, which indicates that voltages for lower-level nets are reset, but are retained for top-level nets

Value Returned

`t`	Voltage conflicts between top-level and hierarchical nets are successfully reported.
`nil`	Voltage conflicts cannot be reported because of an error.

Examples

Opens the specified layout cellview.

```
cvID=dbOpenCellViewByType("vdr_demo" "inverter_chain" "layout")
```

Runs conflict checker for hierarchical nets upto level `32`.

```
vdrRunVoltageConflictChecker(cvID ?hierDepth 32)
=> t
```

Runs conflict checker for hierarchical nets upto level `32` and logs the report in the file `myreport.rpt`.

```
vdrRunVoltageConflictChecker(cvID ?hierDepth 32 ?reportFile "myreport.rpt")
=> t
```

Runs conflict checker for hierarchical nets upto level `32` and fixes those conflicts. (Resets hierarchical nets voltages to zero.)

```
vdrRunVoltageConflictChecker(cvID ?hierDepth 32 ?fixConflict t)
=> t
```

Reports conflicts for zero-voltage hierarchical nets.

```
vdrRunVoltageConflictChecker(cvID ?hierDepth 32 ?skipZeroVoltageNets nil)
=> t
```

vdrRunVSyncSanityChecker

```
vdrRunVSyncSanityChecker(
     d_cellviewID
     [ t_logFile ]
     )
     => t / nil
```

Description

(ICADVM20.1 EXL Only) Checks whether there are valid voltage sync (vsync) shapes on nets that are constrained by a common vsync constraint. The checker reports any vsync shapes that are wrongly created and any nets that have a vsync constraint defined but no vsync shape between them.

Arguments

d_cellviewID	Database ID of the layout cellview to be checked.
t_logFile	Path and name of a log file in which the report is to be captured. When you specify a log filename, only a summary message is printed in the CIW. If you do not specify a log filename, discrepancies are reported in a table printed in the CIW. Environment variable: vdrSanityCheckerLogFile

Value Returned

`t`	The sanity check was completed.
`nil`	The sanity check could not be run.

Example

```
when(window = hiGetCurrentWindow()
  when(cellView = geGetEditCellView(window)
    logFile = "report.log"
    vdrRunVSyncSanityChecker(cellView logFile)
  )
)
```

Checks vsync shapes in the currently edited layout cellview and saves the report in a file called `report.log`.

Related Topics

Sanity Checking Voltage Values in Constrained Labels

Defining and Checking Voltage Synced Nets

vdrSanityCheckerGUI

```
vdrSanityCheckerGUI(
     d_windowID
     )
     => t / nil
```

Description

Opens the VDR Sanity Checker form, which you can use to check constrained VDR voltage labels in the layout view and report any labels that are missing or which have values different from the values specified for the net in the schematic or layout design.

Arguments

d_windowID	ID of the layout window for which you want to open the VDR Sanity Checker form.

Value Returned

t	The VDR Sanity Checker form is displayed.
nil	The VDR Sanity Checker form cannot be displayed because of an error.

Examples

The following example opens the VDR Sanity Checker form for the specified layout window.

```
window=hiGetCurrentWindow()
;Returns the ID of the current window.
vdrSanityCheckerGUI(window)
;Opens the VDR Sanity Checker form for the specified layout window.
=> window:4
=> t
```

Related Topics

VDR Debugger

vdrSetNetVoltageRange

```
vdrSetNetVoltageRange(
     t_libName
     t_cellName
     l_viewList
     [ ?netVoltages l_netVoltages ]
     [ ?voltageDataFile t_voltageInfoFile ]
     [ ?verbose { t | nil } ]
     [ ?logFile t_logFile ]
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets net voltages in the specified list of cellviews. The voltages can be either specified directly as a list when the command is called or read from voltage information file.

Arguments

t_libName

Name of the library in which the cellviews to be updated reside.

t_cellName

Name of the cell whose views are to be updated.

l_viewList

List of view names for which net voltages are to be set.

?netVoltages *l_netVoltages*

List of net names and corresponding minimum and maximum voltage values to be applied to the specified list of cellviews. The asterisk (*) is supported as a wildcard character.

Examples:

```
list("net15" 0.11 9.09)
list("net*" 0.11 1.09)
```

The ?netVoltages and ?voltageDateFile arguments are mutually exclusive.

?voltageDataFile *t_voltageInfoFile*

Name of a comma-separated value (CSV) file that contains voltage information, including net names and corresponding minimum and maximum voltage values. The asterisk (*) is supported as a wildcard character.

The `?netVoltages` and `?voltageDateFile` arguments are mutually exclusive.

`?verbose { t | nil }`

Enables verbose mode when generating labels based on a voltage information file. The software prints one message per entry in the file confirming the action taken for that entry.

The default is `nil`, which means that no messages are issued. When set to `t`, messages are printed in the CIW. Use `?logFile` to save the messages to a separate log file.

`?logFile` *`t_logFile`*

Name of a log file in which messages are saved when generating labels from a voltage information file in verbose mode.

Value Returned

`t`	The specified net voltages were set in the cellviews given.
`nil`	The specified net voltages could not be set.

Examples

```
vdrSetNetVoltageRange(cv~>libName cv~>cellName list(list("layout" "r")
list("layout_org" "r") ) ?netVoltages list("net*" 0.11 1.09))
```

Sets the specified minimum and maximum voltages on all the nets in the `layout` and `layout_org` views in the specified cell in read mode.

```
vdrSetNetVoltageRange(cv~>libName cv~>cellName list(list("layout" "r")
list("layout_org" "r") ) ?netVoltages list("net15" 0.11 9.09))
```

Sets the specified minimum and maximum voltages on `net15` in the `layout` and `layout_org` views in the specified cell in read mode.

The following example sets the minimum and maximum voltage values for the nets whose names start with `VDD` to `0.8` and `2.8`, respectively.

```
cv=dbOpenCellViewByType(
```

```
                        "vdr_demo"
                        "inverter_chain"
                        "layout"
                        ""
                        "r"
                        )
;Opens the specified layout cellview in read-only mode.
cv~>nets~>name
;Prints names of nets available in the specified cellview.

vdd1=dbFindNetByName(
                        cv
                        "VDD1"
                    )
;Retrieves the database ID of the net VDD1.

vdd2=dbFindNetByName(
                        cv
                        "VDD2"
                    )
;Retrieves the database ID of the net VDD2.

vdd3=dbFindNetByName(
                        cv
                        "VDD3"
                    )
;Retrieves the database ID of the net VDD3.
=> db:0x2c4ee21a
=>
=> ("VSS" "VDD1" "net15" "IN" "VDD2"
"net14" "VDD3" "OUT"
)
=> db:0x2c4e9a1b
=>db:0x2c4e9a1e
=> db:0x2c4e9a20

printf("The minimum and maximum voltage values for the net VDD1 is %L \n"
        dbGetNetVoltageRange(
                                vdd1
                            )
        )
```

```
;Prints the minimum and maximum voltage values currently set for the net VDD1.
=> The minimum and maximum voltage values for the net VDD1 is (1.5 1.5)
=> t
printf("The minimum and maximum voltage values for the net VDD2 is %L \n"
        dbGetNetVoltageRange(
                                vdd2
                            )
        )
;Prints the minimum and maximum voltage values currently set for the net VDD2.
=> The minimum and maximum voltage values for the net VDD2 is (2.5 2.5)
=> t

printf("The minimum and maximum voltage values for the net VDD3 is %L \n"
        dbGetNetVoltageRange(
                                vdd3
                            )
        )
;Prints the minimum and maximum voltage values currently set for the net VDD3.
=> The minimum and maximum voltage values for the net VDD3 is (3.5 3.5)
=> t

vdrSetNetVoltageRange(
                        cv~>libName
                        cv~>cellName
                        list("layout" "r")
                        ?netVoltages list("VDD*" 0.8 2.8)
                    )
;Sets the minimum and maximum voltage values for all nets whose names start with
VDD to 0.8 and 2.8, respectively.
=> t
```

To verify whether the minimum and maximum voltage values for the nets, `VDD1`, `VDD2`, and `VDD3` have been set to `0.8` and `2.8`, respectively, run the function `dbGetNetVoltageRange` again.

```
printf("The minimum and maximum voltage values for the net VDD1 is %L \n"
        dbGetNetVoltageRange(
                                vdd1
                            )
        )
=> The minimum and maximum voltage values for the net VDD1 is (0.8 2.8)
=> t
```

```
>
printf("The minimum and maximum voltage values for the net VDD2 is %L \n"
        dbGetNetVoltageRange(
                                vdd2
                            )
        )
=> The minimum and maximum voltage values for the net VDD2 is (0.8 2.8)
=> t

printf("The minimum and maximum voltage values for the net VDD3 is %L \n"
        dbGetNetVoltageRange(
                                vdd3
                            )
        )
=> The minimum and maximum voltage values for the net VDD2 is (0.8 2.8)
=> t
```

Consider that you have a CSV file `netVoltages.csv` ,which contains information about nets and their minimum and maximum voltages. Contents of the CSV file are as follows:

```
#Net minV maxV
#All comments start with '#'
#Top-level Nets

net13,1,1.1
net17,1.2,1.3
IN,1.4,1.5
OUT,1.6,1.7
VSS,1.8,1.9
VDD1,2.0,2.1
VDD2,2.2,2.3
net14,2.4,2.5
net15,2.6,2.7
VDD3,2.8,2.9
```

The following example sets the minimum and maximum voltage values for the nets according to the information contained in the `netVoltages.csv` file.

```
vdrSetNetVoltageRange(
                        cv~>libName
                        cv~>cellName
                        list("layout" "r")
```

```
                        ?voltageDataFile "./netVoltages.csv"
                        ?verbose t
                    )
; Sets the minimum and maximum voltage values of the nets according to the values
specified in the file netVoltages.csv. The netVoltages.csv file contains
information about net names and their minimum and maximum voltages.
=>
INFO (VDR-2000): ./netVoltages.csv:5: Min/Max voltage values have been read for net
'net13'.
INFO (VDR-2000): ./netVoltages.csv:6: Min/Max voltage values have been read for net
'net17'.
INFO (VDR-2000): ./netVoltages.csv:7: Min/Max voltage values have been read for net
'IN'.
INFO (VDR-2000): ./netVoltages.csv:8: Min/Max voltage values have been read for net
'OUT'.
INFO (VDR-2000): ./netVoltages.csv:9: Min/Max voltage values have been read for net
'VSS'.
INFO (VDR-2000): ./netVoltages.csv:10: Min/Max voltage values have been read for
net 'VDD1'.
INFO (VDR-2000): ./netVoltages.csv:11: Min/Max voltage values have been read for
net 'VDD2'.
INFO (VDR-2000): ./netVoltages.csv:12: Min/Max voltage values have been read for
net 'net14'.
INFO (VDR-2000): ./netVoltages.csv:13: Min/Max voltage values have been read for
net 'net15'.
INFO (VDR-2000): ./netVoltages.csv:14: Min/Max voltage values have been read for
net 'VDD3'.
=> t
```

To verify whether the minimum and maximum voltage values for the nets have been set according to the voltage information contained in the specified CSV file `netVoltages.csv`, run the function `dbGetNetVoltageRange` again.

```
printf("The minimum and maximum voltage values for the net VDD1 is %L \n"
        dbGetNetVoltageRange(
                                vdd1
                            )
        )
=> The minimum and maximum voltage values for the net VDD1 is (2.0 2.1)
=> t

printf("The minimum and maximum voltage values for the net VDD2 is %L \n"
        dbGetNetVoltageRange(
                                vdd2
                            )
```

```
        )
=> The minimum and maximum voltage values for the net VDD2 is (2.2 2.3)
=> t

printf("The minimum and maximum voltage values for the net VDD3 is %L \n"
        dbGetNetVoltageRange(
                                vdd3
                            )
        )
=> The minimum and maximum voltage values for the net VDD3 is (2.8 2.9)
=> t
```

Related Topics

Generating Voltage Labels from a Voltage Information File

vdrSetValidLayers

```
vdrSetValidLayers(
     d_cellviewID
     { l_layers | nil }
     )
     => t / nil
```

Description

Specifies a list of valid layers on which labels or markers can be generated. The software creates labels or markers only for nets on one of the listed layers. Each subsequent call to the function overrides all the previously set layers.

Arguments

d_cellviewID	Database ID of the cellview for which the valid layers are to be set.
l_layers | nil	List of valid layer names, each enclosed in double quotes and separated by a space. Type `nil` to unset all previously registered layer names.

Value Returned

`t`	The specified list of layers has been successfully registered (or unset if you specified `nil`).
`nil`	An error occurred while registering the list of layers.

Example

```
cv = geGetEditCellView()
rValue = nil
if(cv then
    layers = list("Metal1" "Metal2" "Metal3" "Poly")
    rValue = vdrSetValidLayers(cv layers)
)
```

Registers `Metal1`, `Metal2`, `Metal3`, and `Poly` as valid layers for use in the voltage dependent rules flow. Labels or markers are created only for nets on those layers.

```
cv = geGetEditCellView()
rValue = nil
if(cv then
    layers = nil
```

```
    rValue = vdrSetValidLayers(cv layers)
)
```

Removes the valid layers specification for the current cellview.

Related Topics

vdrGetValidLayers

vdrValidLayersList

Specifying Layers and Purposes for Generic Voltage Labels

vdrTransferVSyncConstraints

```
vdrTransferVSyncConstraints(
     l_srcCellview
     [ l_tgtCellviews ]
     )
     =>
```

Description

Transfers Voltage Synced Net (vsync) constraints from a specified source cellview to the specified target cellviews without opening any of the cellviews involved. If there is no target cellview specified, then the constraints are transferred to all open cellviews.

Arguments

l_srcCellview	The cellview containing the constraints to be transferred. For example: `list("lib1" "cell1" "schematic")`
l_tgtCellviews	List of one or more cellviews to which the constraints are to be transferred. For example: `list(list("lib1" "cell1" "layout") list("lib2" "cell2" "layout_cv"))`

Value Returned

None

Example

```
fromInfo =list("lib1" "cell1" "schematic")
toInfo = list(list("lib1" "cell1" "layout") list("lib2" "cell2" "layout_cv"))

tcv = dbOpenCellViewByType("lib1" "cell1" "layout" "" "r")
tgtCache = ciCacheGet(tcv);gives constraint cache of cellview
listOfCons = nil
foreach(elm tgtCache~>constraints
          when(elm~>type == 'voltageSyncedNets listOfCons = cons(elm listOfCons)))
printf("LOG: Constraints in target layout: %L\n" listOfCons~>name)
output => LOG: Constraints in target layout: nil

vdrTransferVSyncConstraints(fromInfo toInfo)

output => INFO (LX-1107): Started Layout XL for cellview 'lib1/cell1/layout'.
INFO (CMGR-6068): Updated 2 of 2 constraints
Successfully transferred Voltage Synced Nets Constraints to '2' target cellviews.
```

```
tgtCache = ciCacheGet(tcv);gives constraint cache of cell view
listOfCons = nil
foreach(elm tgtCache~>constraints
          when(elm~>type == 'voltageSyncedNets listOfCons = cons(elm listOfCons)))
printf("LOG: Constraints in target layout: %L\n" listOfCons~>name)
output => LOG: Constraints in target layout: ("Constr_2" "Constr_1")
```

Transfers voltage synced net constraints from source cellview `lib1/cell1/schematic` to target cellviews `lib1/cell1/layout` and `lib2/cell2/layout_cv`.

Related Topics

VSync Constraints Visualizer

Defining and Checking Voltage Synced Nets

vdrVsyncVisualizerGUI

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

Description

(ICADVM20.1 EXL Only) Opens the VDR Constraints Visualizer form, which you can use to create *Voltage Synced Net* (vsync) constraints from the contents of a CSV file, list the vsync constraints currently present in the layout view, and delete those that are no longer required.

Arguments

None

Value Returned

`t`	The form was opened.
`nil`	The form could not be opened.

Example

```
vdrVsyncVisualizerGUI()
t
```

Opens the VDR Constraints Visualizer form.

Related Topics

VSync Constraints Visualizer

Defining and Checking Voltage Synced Nets