cadence®

Virtuoso Floorplanner User Guide

Product Version ICADVM20.1
October 2020

© 2020 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

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

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Contents

Preface . . . 13
Scope . . . 14
Licensing Requirements . . . 14
Related Documentation . . . 14
What's New and KPNS . . . 14
Installation, Environment, and Infrastructure . . . 15
Technology Information . . . 15
Virtuoso Tools . . . 15
Additional Learning Resources . . . 16
Video Library . . . 16
Virtuoso Videos Book . . . 16
Rapid Adoption Kits . . . 17
Help and Support Facilities . . . 17
Customer Support . . . 17
Feedback about Documentation . . . 18
Typographic and Syntax Conventions . . . 19

1
Floorplanning: An Overview . . . 21
Introduction to Virtuoso Floorplanner . . . 21
Floorplanner Prerequisites . . . 24
Constraint Group Lookup Precedence . . . 25
Layer-Purpose Pair and Voltage Dependent Rules in Floorplanning . . . 28
Support for Net PROs in Virtuoso Floorplanner . . . 31
Preparing Data for Floorplanning . . . 36
Setting the Cell Type . . . 36
Setting the Block Type . . . 37
Setting Site Information for Filler Cells . . . 38
Floorplanning Design Flow . . . 40
Floorplan Commands . . . 44

© 2020 All Rights Reserved.

Using the Floorplanner Command-Line Flow 47

2
Generating a Physical Hierarchy 55

Physical Hierarchy Generation 55
Auto-Generate Hierarchy 62

3
Reinitializing the Design 69

Reinitializing a Floorplan 69
Initialization Options 71
Preserve Objects 76
Boundary Parameters 76

4
I/O Planning and Placement 81

Introduction to IO Planning 81
Database Preparation 82
Row Creation 83
PAD Placement 87
Style 88
Spacing 89
Supported Constraints 91
Filler Cells 92
Region Selection Mode 93
Select Filler Cells 94
Filler Instance Attribute 94
Corner Cells 96

5
Placing Blocks 99

Place Blocks 101
Load Solutions 127

© 2020 All Rights Reserved.

Report Placement Statistics 129

6
Updating Soft Blocks 131

Push Pre-Routes 131
Adjust Blocks 141
Remove Block Overlap 143
Snap Soft Blocks to Grid 143
Pull Soft Blocks Inside PR Boundary 144
Edit Soft Blocks 145

7
Creating Pin Group Guides 147

Introduction to Pin Group Guides 147
Creating a Pin Group Guide 150
Comparing Pin Group Guides 169
Placing Pin Group Guides as per Schematic or Symbol 170

8
Pin Placement 177

Pin Planning 179
Using the Pin Placement Form 179
Interleaving the Bits of Iterated Pins 197
Pin Optimization 199
Using the Pin Optimization Tab 201
Optimizing Pins that are Partially Outside the PR Boundary 221
Synchronizing Pin Optimization Options between Command Line and GUI 222
Alignment Issues During Pin Optimization 235
WSP/SP Support (ICADVM20.1 Only) 238
Using a Pin Constraints File 250
Running the Pin Checker (ICADVM20.1 Only) 254
Using the Pin Checker 255

© 2020 All Rights Reserved.

9
Using the Pin Tool 259
Understanding the Pin Tool Interface 260
Menu Bar 261
The Pin Tool Browser Windows 262
Creating Pins 272
Creating Pins 273
Creating Pins from Labels 273
Creating Pins from Net Shapes 279
Creating Pins Automatically 282
Promoting Pins 290
Resizing Pins 297
Making Pins Electrically Aware 303
Merging Pins 310
Planning and Optimizing Pins 313

10
Setting Floorplan Options 315
Floorplan Global Options 316
Color Coding Hard and Soft Blocks 317
Controlling Re-Generation of Instances, Pins, and PR Boundary 318
Governing Pin Resize During Pin Snapping 319
Respecting Layer Priority for Valid Routing Layers 319
Specifying the Default Block Type 320
Updating Label Layer Purpose Pairs 321
Block Annotations 323

11
Updating Pin Labels and Text Displays 327
Label Update GUI Options 328

© 2020 All Rights Reserved.

12
Other Floorplanning Commands 333
Edit Soft Blocks 333
Load Physical View 334
Create Soft Pins 334
Snap Pins 335
Analyze Connectivity 336
Create Feedthrough Terminal Pins 343
Creating Boundary Pins on Selected Shapes 343
Level-1 Editing 343
Pin Connectivity Setting 343
Set Celltype 344
Pin Alignments 344
Aligning Unconnected Bus Pins 358

A
Area Estimation Framework 361
Register Area Estimators 361
Other Utility Functions 373
Sample Area Estimation Procedures 375

B
Block Override Support 377
Concept of Overriding Terminals and Instance Terminals 378

C
Floorplan Property File 381
Master Properties 383
Binding Properties 384
Global Variables for Soft Block Generation 386
Boundary Tab Properties 387
Terminal Properties 392
Halo Properties 394

© 2020 All Rights Reserved.

D
Pin Constraints File . . . 397
Optimized Creation of Pin Constraints . . . 397
Pin Name . . . 398
Edge Constraint . . . 402
Pitch Constraint . . . 403
Status Constraint . . . 405
Sanity Checker . . . 405

E
Constraint Groups . . . 407

F
Migration from Preview to Floorplanner . . . 413
Preview-Floorplanner Menu and Feature Differences . . . 413
Configuring the Physical Hierarchy . . . 414
Setting the Cell Type . . . 415
Generating the Physical Hierarchy . . . 415
Editing Soft Blocks . . . 416
Enabling Floorplanner . . . 416
Generating Design Objects . . . 417
Controlling the Visibility and Selectability of Objects and Layers . . . 418
Using the Menu Commands . . . 419

G
Virtuoso Flooplanner Forms . . . 439
Analyze Connectivity . . . 441
Auto-Create Pins . . . 442
Auto Generate Hierarchy . . . 445
Block Annotations . . . 448
Block Placer . . . 449
Compare Pin Group Guides . . . 452
Corner Pad Placement . . . 453

© 2020 All Rights Reserved.

Create I/O Row 454
Create Pin From Label 455
Create Pin From Net Shapes 457
Electrically Aware Pin Sizing 459
Floorplan Global Options 460
Generate Physical Hierarchy 462
Insert Filler 463
I/O Pad Placer 464
Label Layer Purpose 465
Label Update 466
Load Solutions 467
Merge and Must Connect Pins 468
Pin Checker (ICADVM20.1 Only) 469
Pin Group Guide 470
Pin Group Guide - Place as Schematic/Symbol 472
Pin Placement 473
Pin Spacing Options 480
Pin Spacing Advanced Options 481
Pin Offset Options 482

H
Layout XL Floorplan Environment Variables 483

List of Floorplanner Environment Variables 483
Pin Tool Environment Variables 486
autoGenHierCreateView 488
autoGenHierPhysConfig 489
autoGenHierDepth 490
autoGenHierDefaultArea 491
autoGenHierPinOptMode 492
autoGenHierRunPinOpt 493
blockAnnotationArea 494
blockAnnotationAreaUnits 495
blockAnnotationInstName 496
blockAnnotationRotation 497
blockAnnotationSelMode 498

© 2020 All Rights Reserved.

blockAnnotationShrinkFactor 499
blockAnnotationTextLPP 500
blockAnnotationTextSize 501
bottomSidePinLayersConstraints 502
customPinGrid 503
defaultBlockTypeWhenUnknown 504
enableCongestion 505
enableNoPinsBetweenBusPins 506
enablePinLayerSideConstraints 507
enableTrackSkipping 508
honorCurrentPinLocation 509
ignoreConnectivityBusPinsAlign 510
labelLayerPurposePair 511
layerChangeResize 512
layerPriority 513
leftSidePinLayersConstraints 514
matchLabelLayerWithPin 515
numberOfSkipTracks 516
offSetByGrids 517
offSetBySpacing 518
offSetMode 519
optimizeInstancesPartiallyOutsidePRB 520
perLayerAdjustmentFactorList 521
perLayerCustomPinGridList 522
perLayerOffsetByMfgGridList 523
perLayerOffsetByRoutingGridList 524
perLayerOffsetBySpacingList 525
perLayerSpacingList 526
pinGridAdjustmentFactor 527
pinOptMode 528
rectangularPinEdgeOnPRB 529
retainLongPin 530
rightSidePinLayersConstraints 531
sameLabelPurpose 532
selectedOnly 533
spacingMode 534

© 2020 All Rights Reserved.

spacingType 535
spacingValue 536
tableSpacingRule 537
targetPinSide 538
topSidePinLayersConstraints 539
widthForPinGrid 540
Pin Tool Environment Variables 541
boundaryMultipleMode 541
closePinToolSubForms 542
autoPinCreateFigGroup 543
autoPinPathSpineAware 544
boundarySquareShapePin 545
buriedMultipleMode 546
buriedWholeShapePin 547
customLPPField 548
customPurposeField 549
deletePinCheckbox 550
deletePinSelection 551
modeOption 552
pinConnectivityMode 553
pinLayers 554
pinSizeSelection 555
purposeField 556
mergePinsConnMode 557
mergePinsSelMode 558
mergePinsSuffix 559
pinResizeMode 560
pinResizeAllEstimators 561
pinResizeCurrentEstimator 562
pinResizeFiles 563
promoteCreateLabelsOnPins 564
promoteCustomPurpose 565
promoteCustomLPP 566
promoteDeleteOldPins 567
promoteLayerChoice 568
promoteOddEvenChoice 569

© 2020 All Rights Reserved.

promotePinConnectivity 570
promotePinNetChoice 571
promotePinNetList 572
promotePinOnNetForSelInsts 573
promotePinOnNetPickLevel 574
promotePrefixChoice 575
promotePrefixString 576
promoteSchematicAware 577
promoteSelectiveBusBits 578
promoteSelectInCanvas 579
promoteSuffixChoice 580
promoteSuffixString 581
promoteToChoice 582
promoteTopLayerFig 583
promoteToLevel 584
promoteVisibleLPPs 585

© 2020 All Rights Reserved.

Preface

Virtuoso® Floorplanner contains automatic and interactive floorplanning tools and technologies that help mixed-signal designers develop a layout from a schematic in a methodical manner.

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

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

This preface contains the following topics:

- Scope
- Licensing Requirements
- Related Documentation
- Additional Learning Resources
- Customer Support
- Feedback about Documentation
- Typographic and Syntax Conventions

© 2020 All Rights Reserved.

Scope

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

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

Licensing Requirements

Virtuoso Floorplanner requires the following licenses:

In IC6.1.8, the floorplanning commands require one of the following licenses combined with four GXL tokens:

- Virtuoso_Layout_Suite_XL
- Virtuoso_Layout_Suite_GXL
- Virtuoso_Layout_Suite_EAD

In ICADVM20.1, the floorplanning commands require one of the following licenses combined with four GXL tokens:

- A Virtuoso_Layout_Suite_XL license combined with 4 GXL flexible license tokens
- A Virtuoso_Layout_Suite_EXL license

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

Related Documentation

What's New and KPNS

- *Virtuoso Floorplanner What's New*

© 2020 All Rights Reserved.

- *Virtuoso Floorplanner Known Problems and Solutions*

Installation, Environment, and Infrastructure

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

Technology Information

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

Virtuoso Tools

IC6.1.8 Only

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

ICADVM20.1 Only

- *Virtuoso Layout Viewer User Guide*
- *Virtuoso Layout Suite XL: Basic Editing User Guide*
- *Virtuoso Layout Suite XL: Connectivity Driven Editing Guide*
- *Virtuoso Layout Suite EXL Reference*
- *Virtuoso Concurrent Layout Editing User Guide*
- *Virtuoso Design Planner User Guide*

© 2020 All Rights Reserved.

- *Virtuoso Multi-Patterning Technology User Guide*
- *Virtuoso Width Spacing Patterns User Guide*
- *Virtuoso RF Flow Guide*

IC6.1.8 and ICADVM20.1

- *Virtuoso Abstract Generator User Guide*
- *Virtuoso Custom Digital Placer User Guide*
- *Virtuoso Design Rule Driven Editing User Guide*
- *Virtuoso Layout Suite SKILL Reference*
- *Virtuoso Module Generator User Guide*
- *Virtuoso Parameterized Cell Reference*
- *Virtuoso Symbolic Placement of Devices User Guide*

Additional Learning Resources

Video Library

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

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

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

Virtuoso Videos Book

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

© 2020 All Rights Reserved.

Rapid Adoption Kits

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

In addition, Cadence offers the following training course on Virtuoso Floorplanner:

- Virtuoso Floorplanner Flow

To explore the full range of training courses provided by Cadence in your region, visit Cadence Training or write to training_enroll@cadence.com.

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

Help and Support Facilities

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

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

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

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

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

© 2020 All Rights Reserved.

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

- Log on to Cadence Online Support

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

Feedback about Documentation

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

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

You can also submit feedback by using the following methods:

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

© 2020 All Rights Reserved.

Typographic and Syntax Conventions

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

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

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

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

1

Floorplanning: An Overview

This chapter contains the following sections:

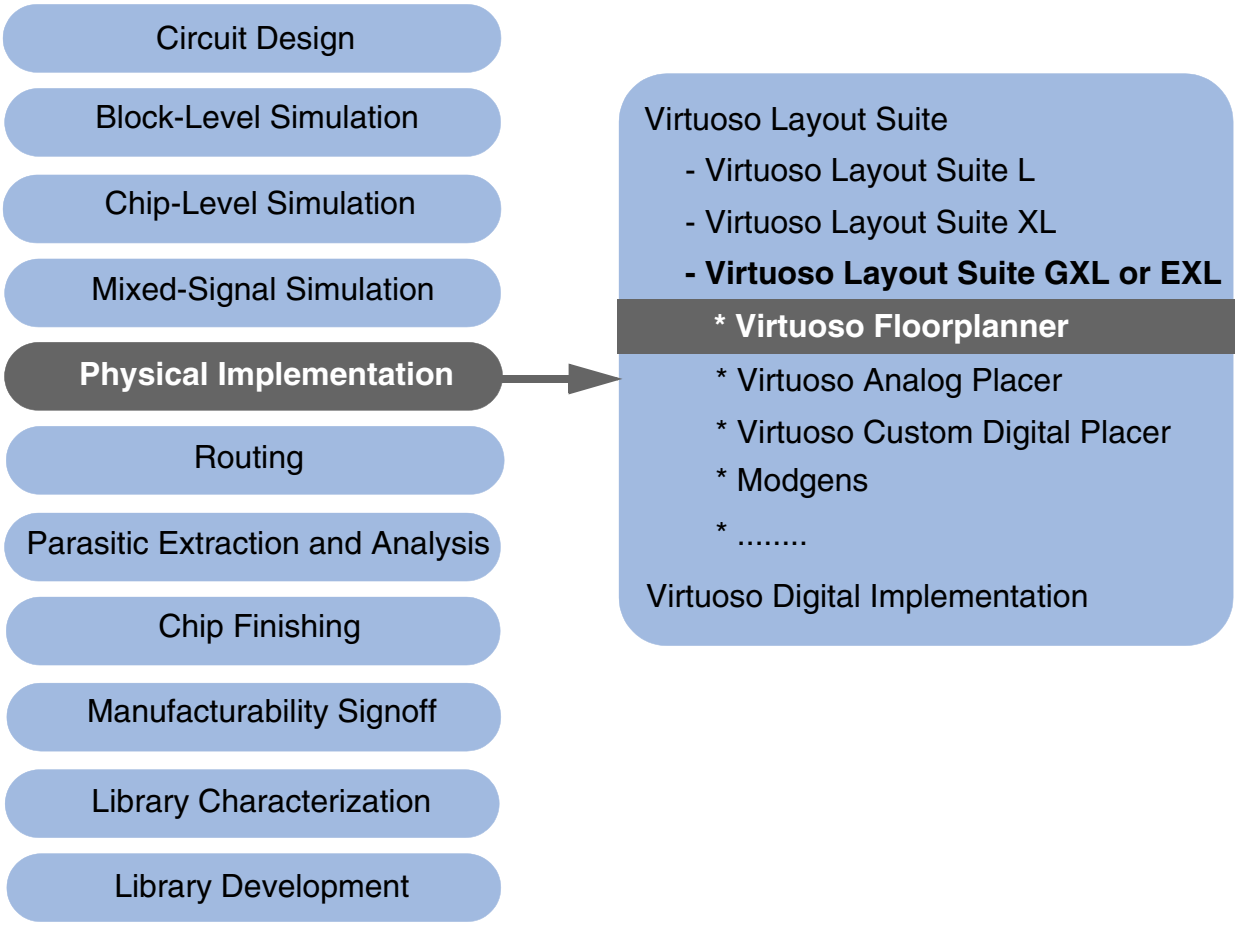
- Introduction to Virtuoso Floorplanner
- Floorplanner Prerequisites
- Constraint Group Lookup Precedence
- Layer-Purpose Pair and Voltage Dependent Rules in Floorplanning
- Support for Net PROs in Virtuoso Floorplanner
- Preparing Data for Floorplanning
- Floorplanning Design Flow
- Floorplan Commands
- Using the Floorplanner Command-Line Flow

Introduction to Virtuoso Floorplanner

The Virtuoso Layout Suite includes the *Floorplan* menu that lets you develop a layout from a schematic in a hierarchical manner. Floorplanner enables you to estimate area and to work with hierarchy early in the design cycle, thus enabling you to adopt a top-down approach to design, a bottom-up approach, or a mix of both methods.

© 2020 All Rights Reserved.

The following diagram shows how Virtuoso Floorplanner fits in the overall Virtuoso design flow:



Some of the powerful Floorplanner features include:

- **Pre-Partitioning:** Configures the hierarchy, that is creates soft blocks at any level in the logical hierarchy. You can use custom area estimators to determine the block size and classify blocks as digital or custom right at the entry stage.
- **Design Initialization:** Categorizes blocks as I/O, cores, macros, and custom cells, and provides an intuitive interface to start floorplanning. It also creates grids used for pin and block edge snapping.
- **Automatic I/O Placement:** Provides a connectivity and constraint-driven I/O placer.
- **Automatic Pin Optimizer:** A connectivity and constraint-driven feature that honors the edge, order and layer constraints.

Additionally, you can manually customize the layout using the following options:

© 2020 All Rights Reserved.

- **Rectilinear Block Support:** Includes options to create and edit the PR boundary at the top level (level0) and at the block level (level1). You can apply Chop, Re-shape, Stretch and Move operations at level0 and level1 on multiple PR boundaries with dynamic measurement and automatic pin and edge snapping features.
- **Pin Adjustment:** Provides options to align, re-layer, and resize pins.
- **Net Bundling:** Lets you create and edit bundles from nets to keep them together during pin optimization and placement.
- **Blockage Support:** Lets you to create and edit blockages that are respected by Pin Optimizer and placer.

© 2020 All Rights Reserved.

Floorplanner Prerequisites

Before using Floorplanner, ensure the following:

- Technology Requirements

  Virtuoso® Floorplanner requires technology-specific information about your design to be stored in a technology file for the design library. For information on creating and editing technology files, see:

  - *Virtuoso Technology Data User Guide*
  - *Virtuoso Technology Data Constraints Reference*

  You can use also use Cadence SKILL functions to query or update the technology file. For more information, see:

  - *Virtuoso Technology Data SKILL Reference*
  - *Virtuoso Layout Suite SKILL Reference*

- Constraint Group Lookup Precedence

- Licensing Requirements

  In IC6.1.8, floorplanning commands require one of the following licenses combined with 4 GXL tokens:

  - Virtuoso_Layout_Suite_XL
  - Virtuoso_Layout_Suite_GXL
  - Virtuoso_Layout_Suite_EAD

  In ICADVM20.1 floorplanning commands require either:

  - A Virtuoso_Layout_Suite_XL license combined with 4 GXL flexible license tokens
  - A Virtuoso_Layout_Suite_EXL license

© 2020 All Rights Reserved.

Constraint Group Lookup Precedence

A constraint group lets you specify different sets of design constraints to be applied under different circumstances. This allows the flexibility to experiment with less or more stringent process rules at different stages of the design process. For details about constraint groups and how to create them, refer to the Constraint Groups chapter.

In Virtuoso, you typically have the following constraint groups as a required minimum.

- Foundry constraint group

  The `Foundry` constraints group has the absolute minimum rules that must be followed to create the design.

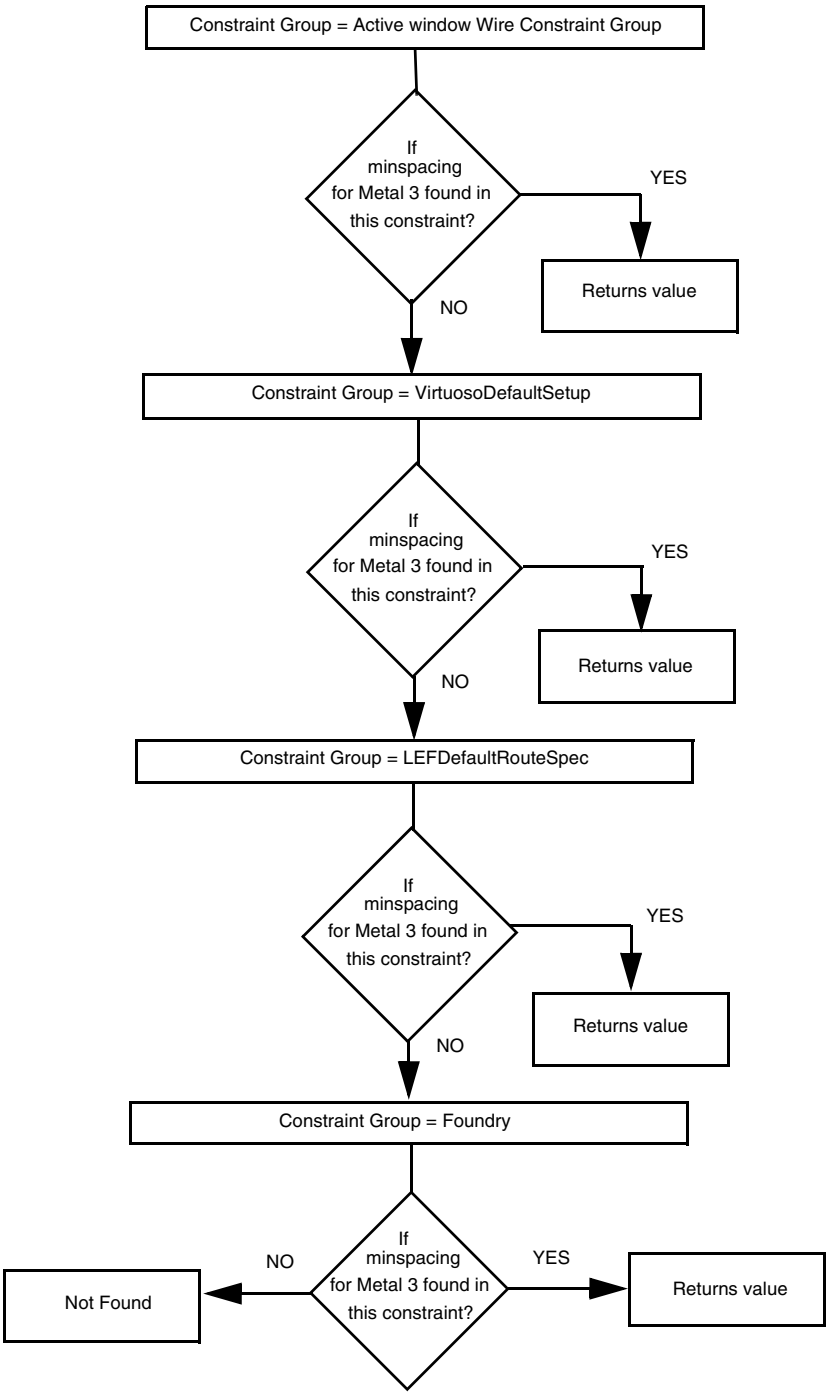
- Active constraint group

  The `Active` constraints group typically has `virtuosoDefaultSetup` as the default value.

For example during reinitialization, if the `minSpacing` value for *metal3* is required, the floorplanner tool searches the `minSpacing` value in the Active Window Wire constraint group. If the minSpacing value is found in the Active Window Wire constraint group, it is used. The floorplanner tool then searches for the `minSpacing` value in `virtuosoDefaultSetup` and then `LEFDefaultRouteSpec` and if still not found then in `foundry`.

© 2020 All Rights Reserved.

The following figure illustrates the flow of how to look at the information from the techfile or design constraints.



Note: Currently, floorplanning commands do not read the constraints defined on objects.

© 2020 All Rights Reserved.

The following section lists the technology file rules that Floorplanning tools require in order to work correctly. For more information about how to define these rules, refer to the *Virtuoso Technology Data Constraints Reference* user guide.

- `minSpacing` required for all the valid layers and layer purpose pairs
- `minWidth` for all the valid layers
- `minVoltageSpacing` for all valid layer purpose pairs
- Grid information for all the routing metal layers for creating placement and routing grids
- `mfgGridResolution` defined in technology file for creating manufacturing grid
- Preferred routing direction for all the metal layer
- Site definition for creating I/O rows
- Standard and custom via defs and standard via variants defined in the technology file to define the spacing type.
- `minEnclosure(minExtensionDistance)` and `minOppExtension` between the via layer and cut layer to define the spacing type
- Routing grid information from technology file for pin snapping
- Placement grid information for instance snapping
- All valid layers defined in the technology file

© 2020 All Rights Reserved.

Layer-Purpose Pair and Voltage Dependent Rules in Floorplanning

While reading the rules from the technology file, Floorplanner now reads layer-purpose-based and voltage-dependent rules, such as `minVoltageSpacing`, in addition to the layer-only rules. The maximum spacing value available from the voltage-dependent, layer-purpose-based, and layer-based spacing rules is used by Floorplanner as the minimum spacing value. A section of the technology file that defines the `minSpacing` rules is given below.

```
("virtuosoDefaultExtractorSetup"    nil
    spacingTables(
     ( minSpacing                ("metal1"    "0.8v")
       (( "width"     nil      nil )     )
       (
         0.3       0.45
         0.4       0.55
       )
     )

     ( minVoltageSpacing         "metal1"
       (( "voltage"     nil      nil ))
       (
         0.7    0.60
         1.0    0.75
      )
    )

    ) ;spacingTables

spacings(
     ( minWidth                  "metal1"    0.35 )
     ( minSpacing                ("metal1"    "0.5v")    0.30 )
     ( minSpacing                "metal1"    0.25 )
    ) ;spacings
) ;virtuosoDefaultExtractorSetup
```

The following table summarizes the `minSpacing` values derived from the technology file given above. The values in the table are derived based on the lookup order defined for the `minSpacing` rule.

Layer Name	Layer Purpose	Constraint Name	Value/Range

© 2020 All Rights Reserved.

Metal1		minVoltageSpacing	0.6, 0.75
Metal1	0.8v	minSpacing (depending on the *Metal1* width)	0.45, 0.55
Metal1		minSpacing	0.25
Metal1	0.5v	minSpacing	0.30

The following examples illustrate how minimum spacing value is determined. The examples are based on the section of the technology file given above.

Example 1

Consider the following values to know the minimum spacing for Metal1 layer:

```
Layer Name = Metal1
Purpose = 0.8v
minWidth = 0.35
```

The spacing values obtained from the table above are as follows:

```
minSpacing value = 0.45 (snapdown minWidth from 0.35 to 0.3)
minVoltageSpacing value = 0.75 (most conservative value for minVoltageSpacing)
```

Therefore, minimum spacing for the *Metal1* layer is as follows:

```
MinSpacing Value for (Metal1 0.8v) & width 0.35  = Max of minSpacing &
minVoltageSpacing
max (0.45 & 0.75) = 0.75
```

In this example, consider a scenario where the minimum width for the layer is not specified.

```
Layer Name = Metal1
Purpose = 0.8v
```

The spacing values obtained from the table above are as follows:

```
minSpacing value = 0.55 (max value of the tableSpacing rule)
minVoltageSpacing value = 0.75
```

Therefore, minimum spacing for the *Metal1* layer when minWidth is not specified is as follows:

```
MinSpacing Value for (Metal1 0.8v)= Max of minSpacing & minVoltageSpacing
max (0.55 & 0.75) = 0.75
```

Example 2

Consider the following values for a layer:

```
Layer Name = Metal1
```


© 2020 All Rights Reserved.

```
Purpose = 0.7v
```

The spacing values obtained from the table above are as follows:

```
minSpacing value = 0.25 (as purpose '0.7v' does not exists in techfile
                        minSpacing rule, defaults to Metal1 layer minSpacing)
minVoltageSpacing value = 0.75
```

Therefore, minimum spacing for the *Metal1* layer is as follows:

```
MinSpacing Value for (Metal1 0.7v)= Max of minSpacing & minVoltageSpacing
max (0.25 & 0.75) = 0.75
```

In the example above, if we have width also defined but the layer-purpose pair is not found in the technology file, then the default layer rule is applied. The value derived from the default layer rule is then compared with the `minVoltageSpacing` rule to get the maximum of the `minSpacing` value.

The layer-purpose-based and voltage-dependent rule support has been added for the following Floorplan commands.

- Configure Physical Hierarchy
- Pin Optimization
- Block Placer
- Create Soft Pins
- Snap Pins
- Snap Pins
- Pin Alignments
- Push Pre-Routes

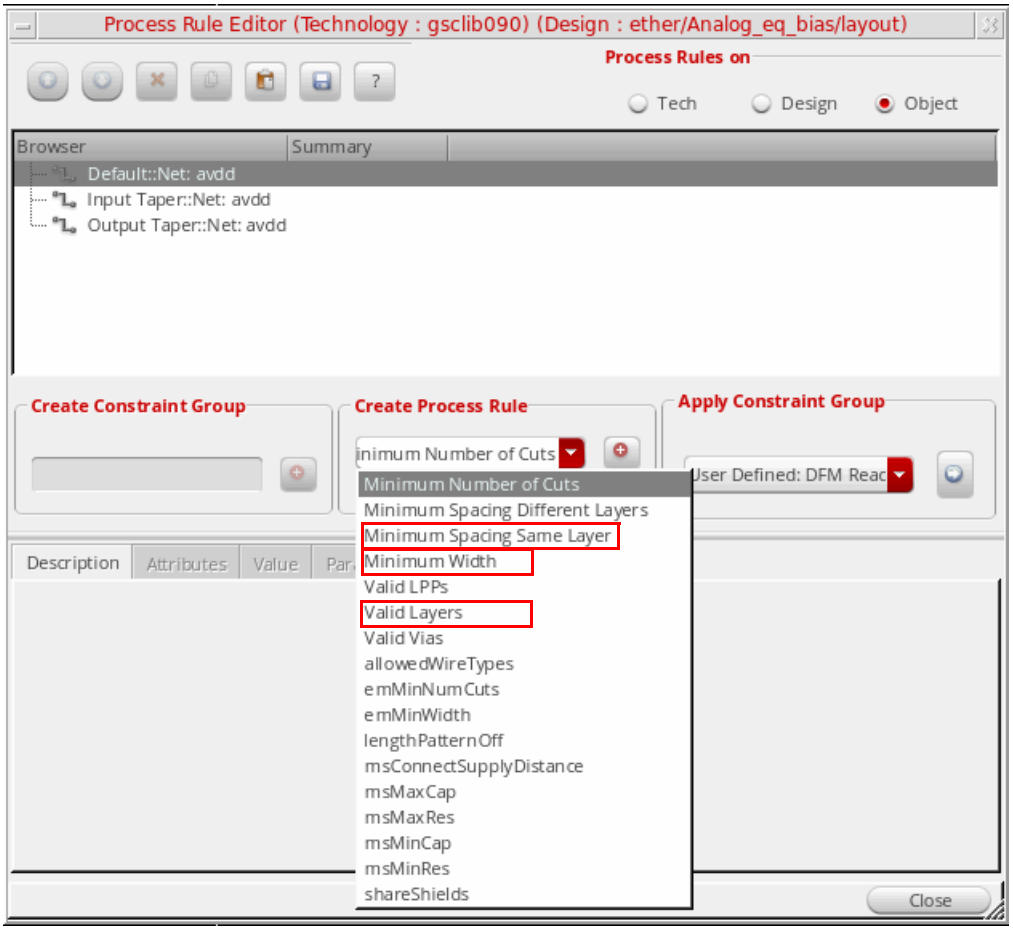
© 2020 All Rights Reserved.

Support for Net PROs in Virtuoso Floorplanner

A net process rule override (PRO) is a constraint defined on a specific net to override the generic foundry and design process rules on that net. Use either the Process Rule Editor or the Constraint Manager to define net PROs.

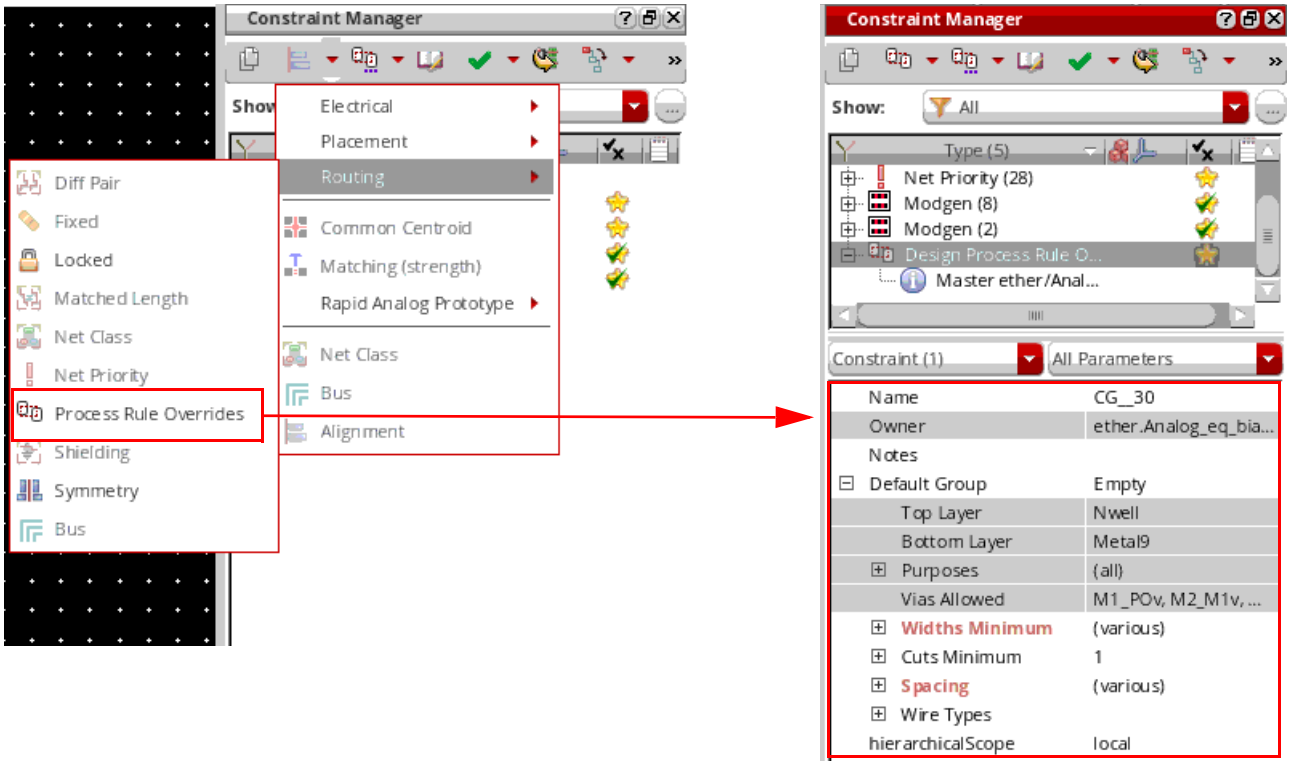
Video

For quick reference of this feature, refer to the Net Process Rule Overrides (PRO) in VFP demonstration.



Defining a net PRO using the Process Rule Editor

© 2020
All Rights Reserved.



Defining a net PRO using the Constraint Manager

The following process rule overrides are supported on nets:

- Minimum Spacing Same Layer override on the pin net, which defines the minimum space around each pin.Minimum Width override on the pin net, which defines the minimum width of pins.
- Valid Layers override on the pin net, which specifies the layers on which the pins belonging to the net can be placed. There can either be a single or a list of valid layers.

  Note: A net PRO can be defined for both top-level (level-0) and soft block (level-1) nets.

  - Top-level nets: If a net PRO is defined for a connected top-level net, then the pins are driven by these net PRO values.
  - Soft block nets: A net PRO defined for a connected soft block net has a higher priority than that defined for a connected top-level net. Therefore, the net PRO values of the connected top-level net are applied only when there are no net PROs definitions for the soft block net.

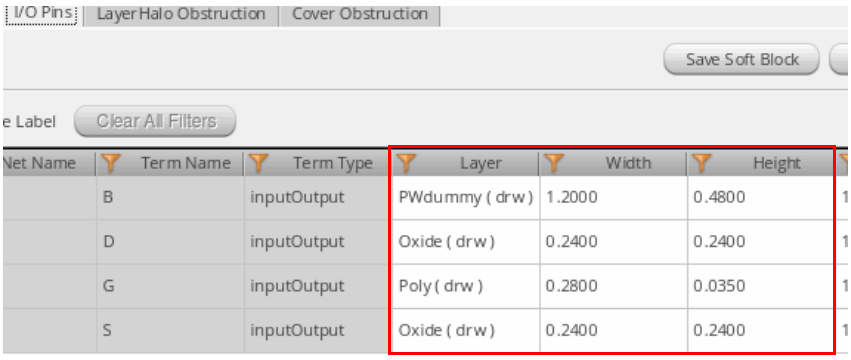
Video

© 2020
All Rights Reserved.

For quick reference of this feature, refer to the Net Process Rule Overrides (PRO) in VFP demonstration.

Net PRO definitions are honored by the following applications:

- Configure Physical Hierarchy (CPH):
  - Recognizes the net PROs defined in the schematic and displays corresponding values on the *I/O Pins* tab of the Soft Block *Attributes* pane.



Net Name	Term Name	Term Type	Layer	Width	Height
	B	inputOutput	PWdummy (drw)	1.2000	0.4800
	D	inputOutput	Oxide (drw)	0.2400	0.2400
	G	inputOutput	Poly (drw)	0.2800	0.0350
	S	inputOutput	Oxide (drw)	0.2400	0.2400

  The net PRO values that are read from the schematic are used for initial population of the respective fields. You can override these values manually; but you must ensure that the net PRO constraint values are not violated.

  - The Generate Physical Hierarchy (GPH) command generates pins in the layout as per the final values of *Layer*, *Width*, and *Height* specified on the *I/O Pin* tab.

- Pin Planning**:**

© 2020 All Rights Reserved.

- When a pin is selected in the pin table, the Pin Planner recognizes the corresponding net PRO and displays the values in the *Layer*, *Width*, and *Height* fields.

- You can edit these values manually, while ensuring that the net PRO constraint values are not violated. If the specified pin size is less than the `minWidth` defined in the net PRO, then the pin size is reverted to its original value.

  Example: The current pin width is 0.2 and net PRO specifies a `minWidth` of 0.3. If you change the pin width to 0.25, then the value rolls back to 0.2, which is the original value.

  If the specified pin *Width* or *Height* is equal to or greater than the net PRO value, then the new value is applied.

  Example: The current pin width is 0.2, and the net PRO specifies a `minWidth` of 0.3. If you change the pin width value to 0.4, then new value is accepted and applied.

■ Pin Optimization:

- Resizes the pins connected to the net to the minimum width specified in the net PRO.
- Moves pins that are connected to the net to the valid layers specified in the net PRO.
  - If there is a single valid layer, then all pins on the net are re-layered to the valid layer, and the pins are placed at feasible locations to ensure minimum net length.
  - If there is a list of valid layers, then the behavior depends on the state of the *Respect layer priority for valid routing layers constraint on net* option in

© 2020 All Rights Reserved.

the Floorplan Global Options form (Layout GXL Options form in IC6.1.8). For more information about this option, see Respecting Layer Priority for Valid Routing Layers.

- Applies the minimum spacing value mentioned in the net PRO to all pins in the selected net. Examples:
  - If the current pin width is 0.3, and net PRO specifies a minWidth of 0.4, then the pin width is increased to 0.4.
  - If the current pin width is 0.4, and net PRO specifies a minWidth of 0.3, then the pin width is not altered. This is because the net PRO constraint defines the minimum width. The current width is adhering to the minWidth requirement, and so the value remains unaltered.

© 2020 All Rights Reserved.

Preparing Data for Floorplanning

To prepare data for floorplanning, you need to set the cell type and the block type. You also need to set the site information for filler cells. You also need to set the site information for filler cells, if you plan to use IO Placer. For more information, see I/O Planning and Placement.

Setting the Cell Type

Floorplanner requires information about the celltype to distinguish the cells at the floorplanning level. Floorplanner behaves differently depending on celltype. For example, IO Placer will work on pads only and not on other cells.

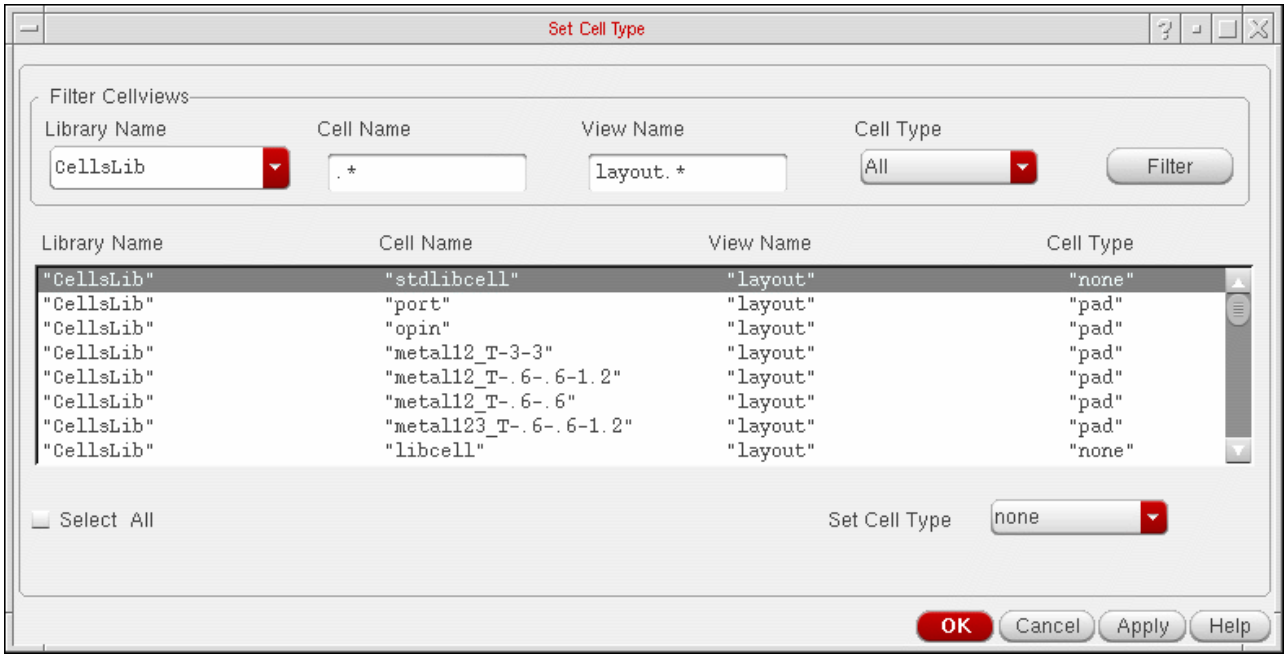
The following table lists the various celltypes supported by floorplanner and the corresponding Virtuoso Design Environment objects.

Floorplanner	Virtuoso Design Environment Objects	OACellType
IO (Pads)	pads, padSpacer, padAreaIO, corner	PadCellType, PadSpacerCellType, CornerCellType, PadAreaIOCellType
Core (Standard Cells)	core, coreSpacer, coreAntenna, coreWellTap	CoreCellType, CoreSpacerCellType, CoreAntennaCellType,CoreWellTapCellType
Hard Macro	coverBump, cover, blockRing, block	BlockCellType, BlockRingCellType, CoverCellType, CoverBumpCellType
Soft Layout, Soft Abstract	softMacro, blockBlackBox	SoftMacroCellType, BlockBlackBoxCellType
Cells (Pcells and custom cells)	None	NoCellType

© 2020 All Rights Reserved.

To set the celltype, click *Tools -> Set Cell Type* in the CIW. The Set CellType window is displayed as shown below

.



To set the celltype of cell, select the cell name in the list and then select the celltype from the Set Cell Type drop down list.

Setting the Block Type

Floorplanner supports two types of blocks “custom” and “digital”. By default, the blocktype is “custom”. You need to specify the blocktype attribute for the blocks in the design because floorplanner behaves differently for “custom” and “digital” blocktype. For example, block placer snaps all “digital” blocks to the placement grid and all “custom” blocks to the manufacturing grid.

The following table describes how setting the block type will affect the hard block and soft block origin, prBoundary, and pin snapping.

macroType	blockType	Object	Snapped With Respect To	Snapping Grid
Hard Block	Digital		Lower left point	Placement grid
	Custom		Lower left point	Manufacturing grid

© 2020 All Rights Reserved.

	Dummy / Not specified		Lower left point	User grid
Soft Block	Digital	prBoundary	All boundary points	Placement grid
	Custom	prBoundary	All boundary points	Manufacturing grid
	Dummy / Not specified	prBoundary	All boundary points	User grid

You can define the blocktype parameter using the *Soft Blocks* tab in the *Configure Physical Hierarchy* form.

In addition to “custom” and “digital” blocks, a design may contain blocks for which the `blockType` parameter is not defined. The *Default blockType (when unknown)* option in the Floorplan Global Options form defines the block type for such blocks. For more information, see Specifying the Default Block Type.

You can also use SKILL to change the blocktype attribute. For example, the following command issued in the CIW modifies the blocktype to “digital”:

```
id~>master~>blockType = “digital”
```

You can change the blocktype by descending in the master cellview and set the property to name blocktype, string, value custom or digital.

Alternatively, you can select the required block, right-click to display the shortcut menu, select *Block*, and then select the required block type - *Digital*, *Custom*, *Any*, or *As is*. When you select multiple blocks of different block types, and then display the shortcut menu, option *As is* selected by default. *As is* is not a block type. It signifies that there are multiple block with different types in the selection set. If you select *Any*, then you can use the Floorplan Global Options form (Layout GXL Options form in IC6.1.8) to specify the pin grid to be used for placing pins during pin optimization. For more information, see Specifying the Default Block Type.

Setting Site Information for Filler Cells

Before running the floorplanning commands, set the site information for filler cells. To do this, in the layout window, click the *File – Properties* command. The Edit Cellview Properties form opens. Use the options in the *SitePattern* group box to set the site information for filler cells.

© 2020 All Rights Reserved.

For more information about editing cellviews, see Edit Cellview Properties.

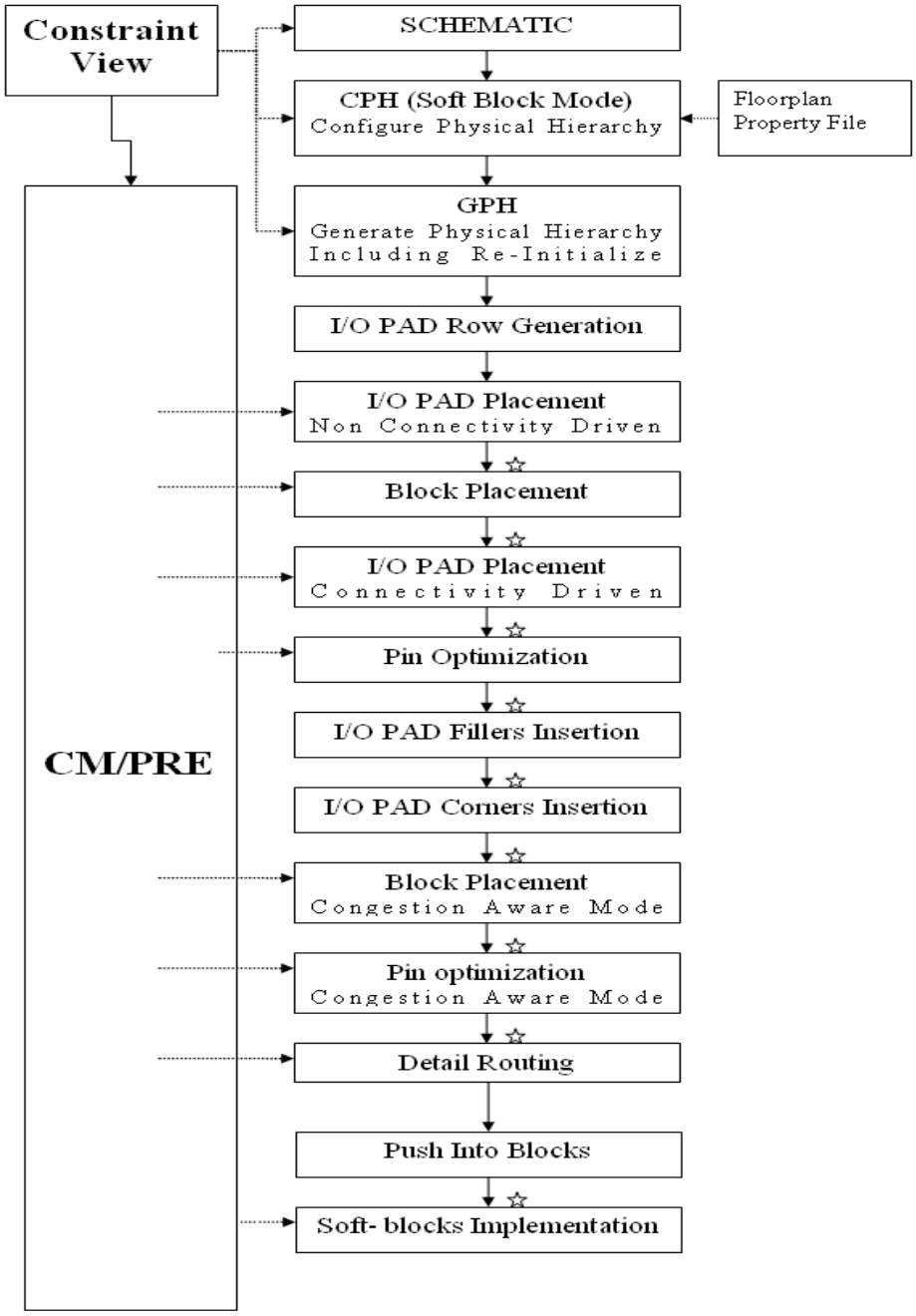
For more information about inserting filler cells, see Filler Cells.

© 2020 All Rights Reserved.

Floorplanning Design Flow

The following flow diagram shows the primary design flow in Floorplanner.

© 2020 All Rights Reserved.



☆Optional Manual Editing like:
- ⇒ Top Level prBoundary and Pins manipulation
- ⇒ Soft Blocks prBoundary and Pins manipulation
- ⇒ Pin Placer
- ⇒ Create Feedthrough Pins and Soft Pins
- ⇒ Pin Alignment
- ⇒ Blockage creation
- ⇒ Wire Editing

© 2020 All Rights Reserved.

The floorplanning design flow consists of the following steps. Each of these steps is described later in this guide in detail.

- Configure the physical hierarchy

  In this step, you specify how a layout implementation is generated from a schematic design. You use the *Soft Block* mode of the *Configure Physical Hierarchy* utility to define the soft blocks that will be created by the *Generate Physical Hierarchy* utility. For details about *Configure Physical Hierarchy* utility, refer to Configuring the Physical Hierarchy in *Virtuoso Layout Suite XL User Guide*.

- Generate the physical hierarchy

  In this step, you generate a physical hierarchy from a logical hierarchy in a schematic. The *Generate Physical Hierarchy* command creates soft layouts and soft abstracts in the layout canvas corresponding to the configuration view generated by the *Configure Physical Hierarchy* utility in the previous step. For details, refer to Generating a Physical Hierarchy.

- Reinitialize the design

  In this step, you reinitialize the design to place the cell instances according to their celltypes. When you reinitialize a design, the I/O pads are placed inside the top edge of the design boundary, the standard cells are placed outside the right edge of the design boundary, and the blocks are placed outside the left edge of the design boundary. This improves the legibility of the design. For details, refer to Reinitializing the Design.

  Note: The Reinitialize command can be run at any step during the design flow if you are not satisfied with the results and want to go back to the previous step.

- Place I/O Pads

  In this step, you place the I/O pads around the design boundary. The *IO Placer* utility enables you to place the I/O pads based on connectivity or side and alignment constraints. For details, refer to I/O Planning and Placement.

- Analyze connectivity

  In this step, you analyze connectivity, that is, the number of connection a block has with other blocks, so that during manual floorplanning, you can keep the blocks with more connectivity between them closer to each other. For details, refer to Placing Blocks

- Place blocks

  In this step, you place the blocks in the design either manually or automatically. After placing the blocks you can descend into individual blocks and place and route them. You can then return to the top level to place and route the blocks at the top level. For details, refer to Placing Blocks and Updating Soft Blocks.

© 2020 All Rights Reserved.

- Optimize pins

  In this step, you optimize the pins. Blocks placed together do not guarantee that the net length between the blocks or pins is optimal or the shortest. This is true even for blocks placed by the block placer. Therefore, Pin Optimizer is required to position pins of these blocks to get the shortest possible net lengths at a particular level.
  However, before you optimize pins, you must set the edge constraint on the top-level pins or the level1 pins by using Pin Planner. For details about pin planning, refer to Pin Planner in the *Virtuoso Custom Digital Placer User Guide*. For details on optimizing pins, refer to Pin Placement.

- Align pins

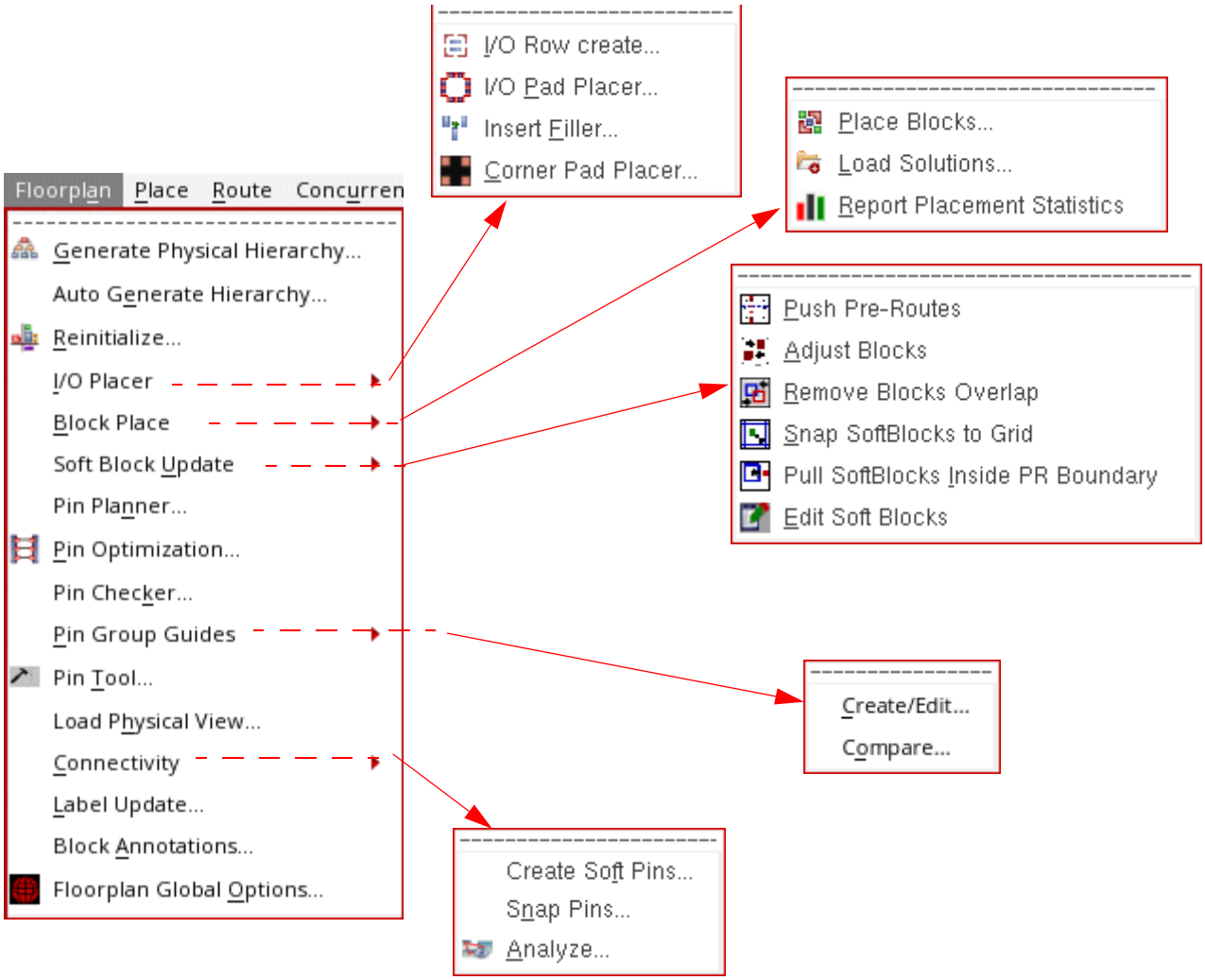
  In this step, you align the pins. For details, refer to Pin Alignments.

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

© 2020 All Rights Reserved.

Floorplan Commands

The *Floorplan* commands are available in Virtuoso Layout Suite on the *Floorplan* menu. The *Floorplan* menu is available by default in the layout window. To view the Floorplan commands, click the *Floorplan* menu in the layout window. The *Floorplan* commands are displayed as shown in the graphic.



Important

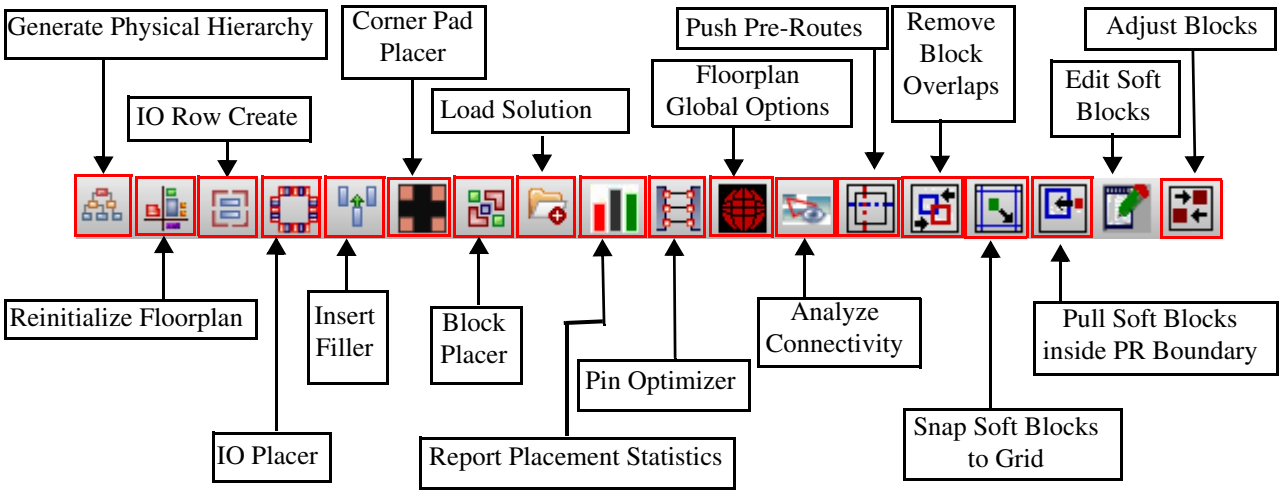
(ICADVM20.1 Only) In Layout EXL, all Floorplan commands are available under the *Plan* menu.

The *Floorplan* menu provides commands to manipulate hierarchy and define floorplans. Using the Floorplan commands, you can create a rough topology of the design rather than

© 2020 All Rights Reserved.

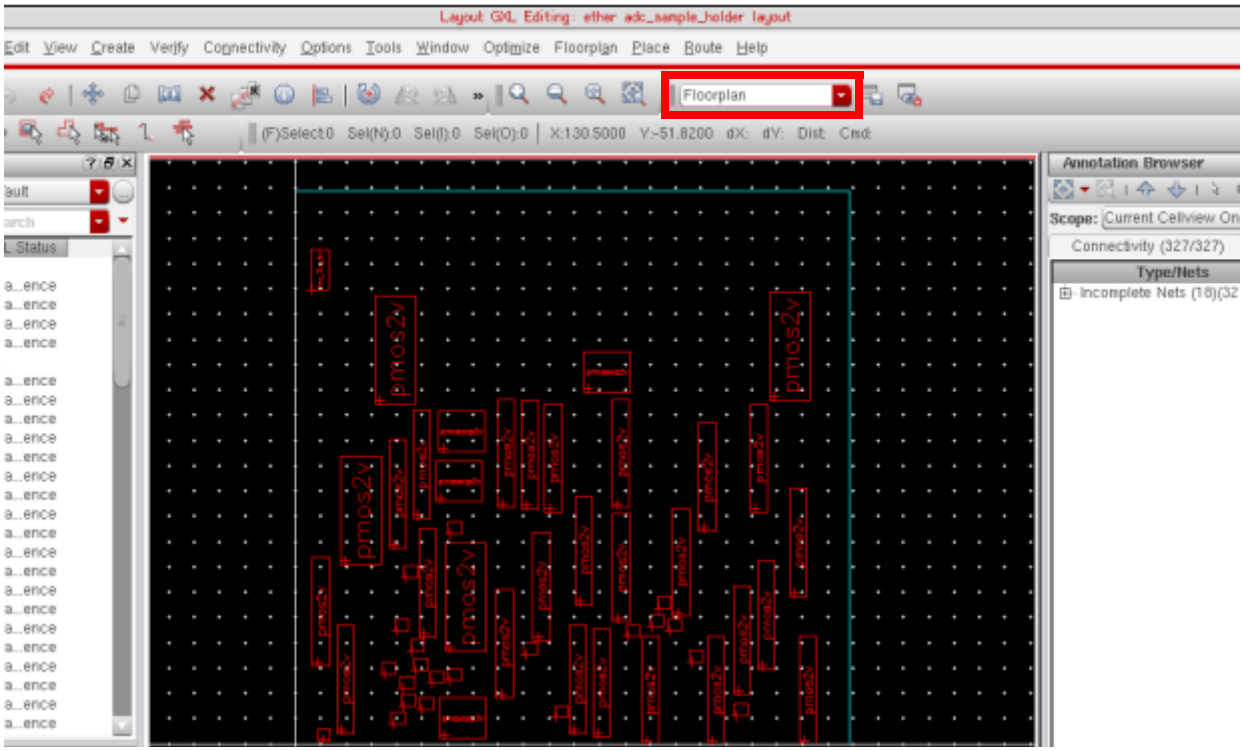
the final layout solution. You can use these commands to perform floorplanning tasks such as reinitializing a layout, placing I/Os and macros, optimizing pins, and update labels. The *Floorplan* menu also includes additional commands to update soft blocks, import predefined information from source physical cellviews, create soft pins, snap pins, analyze connectivity, and set floorplanning global options.

In addition to the menu commands, there is a Floorplan toolbar that you can use to invoke the floorplan commands. The Floorplan toolbar is available by default as shown in the graphic.



© 2020
All Rights Reserved.

Floorplanner has its own workspace that shows the canvas, the toolbars, the Navigator and Property Editor assistants, and Annotation Browser.



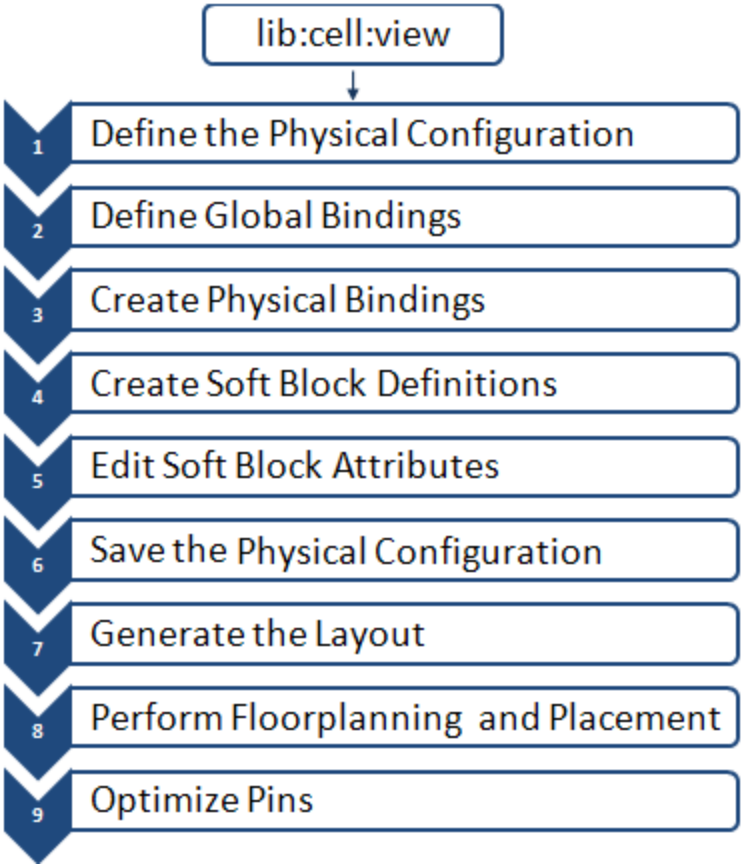
You use the *Floorplan* commands for:

- Generating a Physical Hierarchy
- Reinitializing the Design
- I/O Planning and Placement
- Placing Blocks
- Updating Soft Blocks
- Pin Optimization

© 2020 All Rights Reserved.

Using the Floorplanner Command-Line Flow

Instead of using the GUI commands, you can use the floorplan command-line SKILL APIs to generate a layout from a schematic in a hierarchical manner. These APIs are entered in the CIW. The following figure illustrates the overall design flow for developing a layout from a schematic in a hierarchical manner.



Each step in the design flow uses a set of SKILL APIs. The `lib:cell:view`, which represents the name of the library, cell, and view to be referenced in the schematic, is the input for the process. The description of the APIs used in each step is as follows.

1. Define the Physical Configuration (physConfig)

   Use the `cphCreatePhysConfig` API to create a physical configuration (physConfig) corresponding to the specified top-level `lib:cell:view`. A physConfig comprises multiple files that store design-specific overrides such as the schematic expansion rules and the instance mapping rules.

2. Define Global Bindings

© 2020 All Rights Reserved.

Use the `cphSetViewList` API to set the logical switch view list for the physical configuration view associated with the specified ID. The list specifies the view names used to descend into a hierarchical design to find schematic views.

Use the `cphSetStopList` API to specify the view names to be used to determine the corresponding physical view for a given logical view.

3. Create Physical Bindings

Use the `cphSetCellPhysicalBinding` API to set the physical binding for a specified logical cell in the physical configuration associated with the given ID.

4. Create Soft Block Definitions

Use the `cphSbDefineSoftBlock` API to define a soft block in the physConfig.This API creates CPH data for the soft block to be generated in the layout by using the logical `lib:cell:view`. The initial I/O pins are automatically generated in the layout and their attributes are configured based on the environment variables.

5. Edit Soft Block Attributes

After defining soft blocks in the CPH, use the following APIs to further customize the soft blocks as per your requirements.

- Viewing Soft Blocks and Pin Attributes

  Use the following APIs to view soft block and pin attributes.

  - `cphSbGetSoftBlocks`: Returns a list of defined soft blocks that correspond to the specified g_physConfigId
  - `cphSbGetSoftBlockId`: Returns the ID of the soft block corresponding to the specified physConfig and physical `lib:cell:view`
  - `cphSbGetFilteredIOPins`: Filters and displays pin IDs based on the specified arguments
  - `cphSbGetAllIOPins`: Returns a list of all pin IDs associated with a specific soft block
  - `cphSbDisplayAllIOPinsInfo`: Returns the pin information for all pins corresponding to the specified Physical configuration ID and soft block ID
  - `cphSbDisplayBoundaryInfo`: Displays valid boundary attributes such as the boundary type (rectangle or polygon), area, polygon points, aspect ratio, width, height, and utilization
  - `cphSbGetIOPinId`: Returns the pin ID for a specific soft block

© 2020 All Rights Reserved.

- `cphSbGetIOPinName`: Returns pin name and soft block name; helps verify whether a pin exists
- `cphSbDisplayIOPinInfo`: Returns all information related to the specified pin
- `cphSbDisplayObstruction`: Displays information related to the specified obstruction
- `cphSbDisplayAttributes`: Displays the block type, inherited view list, and stop view list of the specified block

- Customizing Soft blocks

  After viewing the soft block and pin attributes, you may want to customize them. All updates to the soft blocks and pin attributes are stored in the physConfig.

  The following environment variables are honored when editing soft blocks and pins.

  - initIOPinLayer
  - initPinHeight
  - initPinWidth
  - initPinMultiplicity
  - initCreatePadPins
  - createPinLabel

  Depending on the shape and structure of the soft block that you want to edit, choose one of the following APIs:

  - Polygon mode: `cphSbSetPolygonalBoundary`
  - Rectangle mode: `cphSbSetRectangularBoundary`
  - Rectangle mode with utilization input: `cphSbSetRectangularBoundaryUtil`

  The following are some additional APIs that you can use to edit soft blocks.

  - `cphSbEditIOPin`: Edits attributes of an I/O pin located at a specific physical location, which is specified as an argument
  - `cphSbEditIOPinById`: Edits an I/O pin with a specific pin ID
  - `cphSbAddIOPin`: Adds a new I/O pin to the soft block
  - `cphSbDefineIOPinLabelFlag`: Creates I/O pin labels.

© 2020 All Rights Reserved.

- `cphSbDelIOPin`: Deletes an I/O pin located at a specific physical location
- `cphSbDelIOPinById`: Deletes the I/O pin with a specific pin ID
- `cphSbGetAllIOPins`: Returns a list of all pin IDs associated with the specified soft block
- `cphSbGetIOPinId`: Returns the required pin ID. After retrieving the required pin IDs, you can use the `cphSbEditIOPinById` or `cphSbDelIOPinById` to edit or delete the retrieved pins.
- `cphSbIsValidIOPin`: Verifies whether a pin exists in a soft block
- `cphSbDefineObstruction`: Adds an obstruction or blockage within a soft block
- `cphSbDelObstruction`: Deletes an obstruction from a soft block
- `cphSbEditSoftBlockAttributes`: Edits the block type, inherited view list, and stop view list
- `cphSbLoadSoftBlocks`: Loads soft block data from the specified floorplan property file. When soft blocks are loaded, the instance and cell bindings between schematic and physical views of blocks are re-established based on the specifications in the floorplan property file.
- `cphSbSaveSoftBlocks`: Saves soft block data to the specified floorplan property file
- `cphSbRemoveSoftBlock`: Removes the soft block that is associated with the specified ID
- `cphSbDefineCovObstruction`: Defines cover obstructions
- `cphSbDelCovObstruction`: Deletes cover obstructions
- `cphSbDisplayCovObstructionInfo`: Displays information about cover obstructions
- `cphSbHasCovObstruction`: Checks for the presence of cover obstructions in a soft block

6. Save the Physical Configuration

The configuration settings remain as the transient data until you save them to the disk. Use the `cphSaveConfig` API to save the current physical configuration settings. You can save the settings at any point when configuring the physical hierarchy. The saved settings take effect the next time you generate or update the layout view.

© 2020 All Rights Reserved.

7. Generate the Layout

   After customizing soft blocks, use the `vfpCPHGenPhysicalHierNoPropFile` API to generate the physical hierarchy. Soft layouts and soft abstracts are created in the layout canvas corresponding to the configuration that you defined.

8. Perform Floorplanning and Placement

   After generating the physical configuration in the layout canvas, you can perform the following floorplanning and placement tasks:

   - I/O Placement: I/O Placement is a row-based placement methodology in which you perform the following tasks in sequence:
     - Create I/O rows.
     - Place pads inside the I/O rows.
     - Insert filler cells in the gaps between the pads and rows.
     - Place corner cells in the four corners of the design.
   - Floorplanning and Placement: Floorplanning and placement involves placing all instances in a design and minimizing the overall wire length and chip area. Depending on the type of instances that you want to place, choose from the following automatic placement tools:
     - Block Placer (under Virtuoso® Floorplanner): For placing blocks
     - Virtuoso® Custom Placer: For placing standard cells
     - Virtuoso® Analog Placer: For placing Pcells and other custom cells

   During floorplanning and placement, you may need to further edit soft block attributes, such as the boundary, pins, and obstructions.

   In Edit Soft Block Attributes, updates to the soft block and pin attributes were stored in physConfig. At that point, the instances were not available on the layout canvas. However, in Perform Floorplanning and Placement, modifications are made to instances after they are generated on the layout canvas.

   Instead of using the GUI to run the floorplanning commands, you can use the following command-line APIs to perform the corresponding editing tasks.

   Soft Block Boundaries

   - `vfpSbSetPolygonalBoundary`: Modifies the co-ordinates of polygonal soft blocks

© 2020 All Rights Reserved.

- `vfpSbSetRectangularBoundary`: Modifies the co-ordinates of rectangular soft blocks

Editing I/O Pins

- `vfpSbEditIOPin`: Edits attributes of an existing I/O pin

Editing Obstructions

- `vfpSbDefineObstruction`: Adds obstructions
- `vfpSbDeleteObstruction`: Deletes obstructions

Editing Soft Block Attributes

- `vfpSbEditSoftBlockType`: Modifies the block type of soft blocks

9. Optimize Pins

In this step, you optimize the pins. Pin Optimizer optimizes the positions of top-level and soft block pins to achieve the shortest possible aggregated net length.

Use the `vpaOptimizePins` command-line API to run pin optimization without opening the Pin Optimization GUI.

Example

This example uses APIs to define and edit soft blocks and pins before generating the physical hierarchy.

In this example, the following environment variables are set before starting the physical configuration. These environment variables provide default values for layer, pin height, and pin width to be applied when defining individual blocks and pins.

```
envSetVal("layoutXL" "initIOPinLayer" 'string "metal1 pin")
envSetVal("layoutXL" "initPinHeight" 'float 2.0)
envSetVal("layoutXL" "initPinWidth" 'float 2.0)
```

Important

Setting environment variables is an optional step. If not set, however, you need to specify values for these parameters when defining individual the blocks and pins.

The first step in the design flow is to define a physConfig by using the `cphCreatePhysConfig` API, with the `lib:cell:view` of the layout and schematic as parameters. A unique physConfig ID is returned. In the following code, the physConfig ID for the top-level soft block is assigned to variable `cphId`.

© 2020 All Rights Reserved.

```
cphId = cphCreatePhysConfig("design" "top_level" "physConfig_example" "design"
"top_viewtest" "schematic")
```

Next, you define soft blocks by using the `cphSbDefineSoftBlock` API. This API creates CPH data for the soft block to be generated in the layout by using the specified logical `lib:cell:view`. Here, `sbID` is the top-level block and `sbB1` and `sbB2` are other design-level soft blocks.

```
sbId = cphSbDefineSoftBlock(cphId
                            "design"
                            "top_level"
                            "layout_example"
                            ?logLib "design"
                            ?logCell "top_level"
                            ?logView "schematic"
                            )
sbB1 = cphSbDefineSoftBlock(cphId
                            "design"
                            "block1"
                            "layout_example"
                            ?logLib "design"
                            ?logCell "block1"
                            ?logView "schematic"
                            )
sbB3 = cphSbDefineSoftBlock(cphId
                            "design"
                            "block3"
                            "layout_example"
                            ?logLib "design"
                            ?logCell "block3"
                            ?logView "schematic"
                            )
```

After defining soft blocks, their boundaries are defined.

```
foreach(sb list(sbB1 sbB3)
        cphSbSetRectangularBoundary(cphId sb 50 50)
        )
cphSbSetRectangularBoundary(cphId sbId 400 400)
```

Here, the `foreach(sb list(sbB1 sbB2)` function includes the IDs of all the soft blocks for which boundaries need to be created. The `cphSbSetRectangualarBoundary` specifies that the boundary should be rectangular in shape. The parameters of this API are the physConfig ID, the soft lock ID, and the dimensions of the rectangular boundary.

Next, the pins are defined.

```
foreach(pinId cphSbGetAllIOPins(cphId sbId)
```

The `cphSbGetAllIOPins` API retrieves the IDs of all the pin located at the top level. The parameters for this API are the physConfig ID (`cphId`) and the corresponding soft block ID (`sbId`). The dimensions of the pins are based on the values you specified when defining the environment variables. If required, you can edit the dimensions of the pins by using the

© 2020 All Rights Reserved.

cphSbEditIOPinById API. You can also use this API to you can create and edit pins for the remaining soft blocks.

```
cphSbEditIOPinById(cphId pinId ?lpp "metal3 drawing" ?width 10.0 ?height 10.0
?sigType "signal")
foreach(pinId cphSbGetAllIOPins(cphId sbB1)
cphSbEditIOPinById(cphId pinId ?lpp "metal1 drawing" ?width 5.0 ?height 5.0
?sigType "signal")
)
foreach(pinId cphSbGetAllIOPins(cphId sbB3)
cphSbEditIOPinById(cphId pinId ?lpp "metal1 drawing" ?width 3.0 ?height 3.0
?sigType "signal")
)
```

After specifying all required configuration settings, the configuration settings are saved to the disk.

```
cphSaveConfig(cphId)
```

The final step is to generate the physical hierarchy in the layout canvas without specifying the floorplan property file using the vfpCPHGenPhysicalHierNoPropFile API.

```
vfpCPHGenPhysicalHierNoPropFile("design" "top_level" "schematic" "design"
"top_level" "layout_example")
```

The `lib:cell:view` of the schematic and layout are specified as the parameters.

You can now perform floorplanning, placement, and pin optimization.

© 2020 All Rights Reserved.

2

Generating a Physical Hierarchy

This chapter covers the following topics:

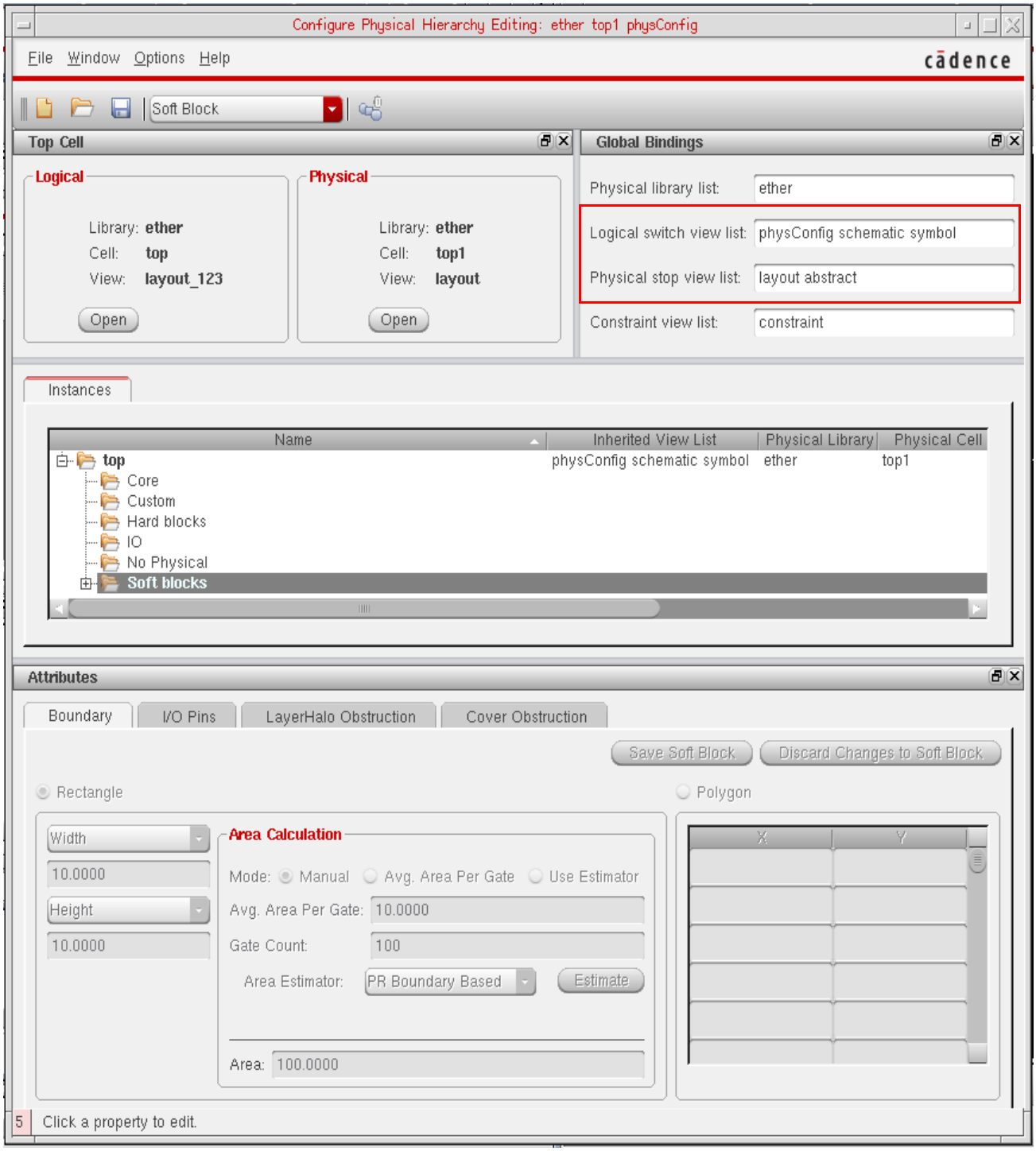
- Physical Hierarchy Generation
- Auto-Generate Hierarchy

Physical Hierarchy Generation

The Generate Physical Hierarchy generates the physical hierarchy based on the configuration defined in Configure Physical Hierarchy (CPH). It allows you to generate a physical hierarchy from the logical hierarchy in a schematic. It also creates soft layouts and soft abstracts in the layout canvas corresponding to the configuration view generated by CPH. The Generate Physical Hierarchy uses two main engines, Generate All From Source and Generate Abstract From Symbol, to generate blocks.

© 2020 All Rights Reserved.

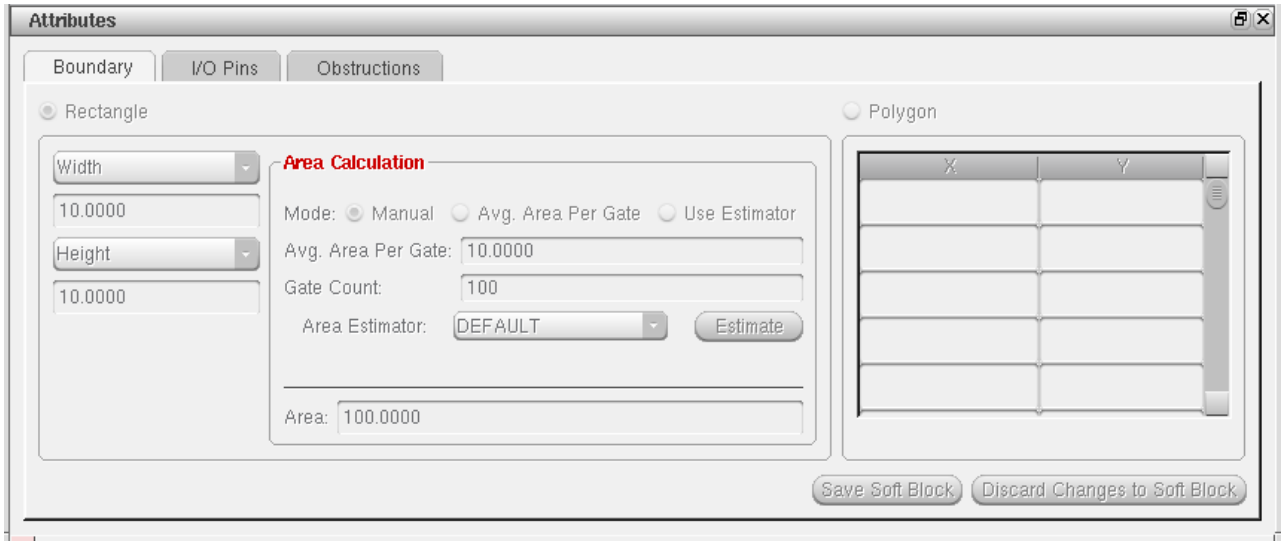
By default, Generate Physical Hierarchy looks for layout views and is guided by the configuration you specify in the *Logical Switch View List* and *Physical Stop View List* fields in the *Global Bindings* section of the CPH interface.



© 2020 All Rights Reserved.

You can also specify the attributes, such as boundary, IO pins, and obstructions, for each soft block and soft abstract in the *Attributes* section of the Soft Block mode. While defining soft blocks in the CPH form, the pin layer will be read from the `intIOPinLayer` environment variable. Following is the syntax to set the environment variable.

```
envSetVal("Layout XL", "initIOPinLayer", "string", "METAL3 pin")
```



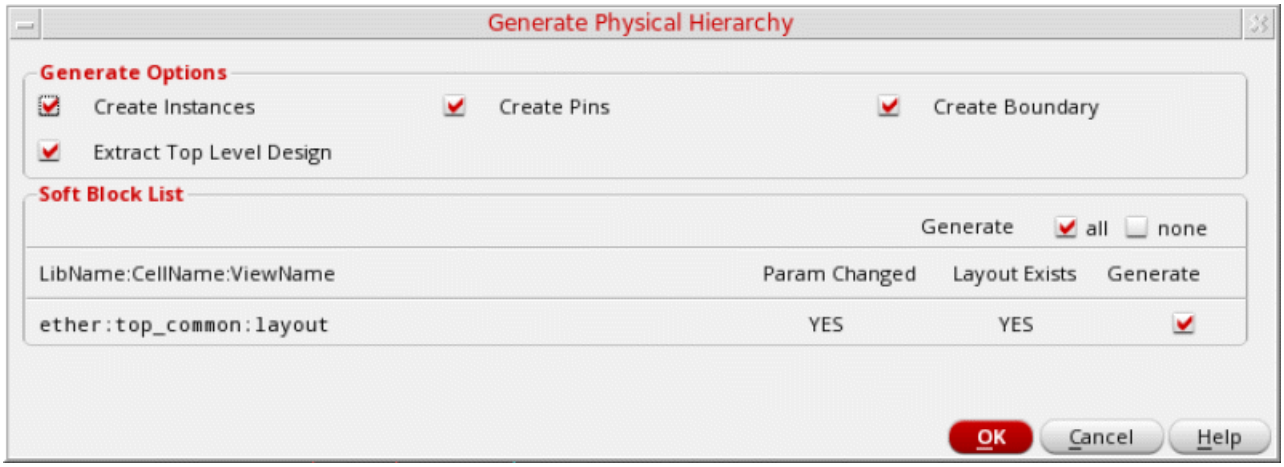
For details on the parameters used by the Generate Physical Hierarchy command, refer the Soft Block Mode section in the Virtuoso Layout Suite XL User Guide.

The Generate Physical Hierarchy command allows you to generate the components in the design while maintaining the hierarchy levels defined in the CPH Soft Block mode. To generate the physical hierarchy:

1. Choose *Floorplan – Generate Physical Hierarchy*. The Generate Physical Hierarchy form is displayed.

© 2020 All Rights Reserved.

Note: In Layout EXL, choose *Plan – Generate – Generate Physical Hierarchy*.



2. (ICADVM20.1 EXL Only) In the *Generate Options* section, select the objects to be generated when the physical hierarchy is generated. All options are selected by default. Clear the check boxes for the objects that you do not want to generate

   - *Create Instances*: Generates instances in the layout canvas.
   - *Create Pins*: Generates pins below the PR boundary.
   - *Create Boundary*: Generates the PR boundary.

   Related Environment Variables: `initCreateInstances`, `initCreatePins` and `initCreateBoundary`

3. Select *Extract top level design* checkbox. In this state, the value from the `extractAfterGenerateAll` environment variable is honored while opening the GPH form. It allows you to selectively enable or disable the connectivity extraction for top block only and control the extraction for the remaining of the soft block through the environment variable.

4. Review the *Soft Block List*. The tool reads all the soft attributes and checks for the existence of cellviews defined in Configure Physical Hierarchy. The following information is displayed in the Soft Block List section:

   - `LibName:CellName:ViewName`: Specifies the names of soft blocks defined in Configure Physical Hierarchy.
   - `Param Changed`: Specifies whether the soft block parameters have been updated.
   - `Layout Exists`: Specifies whether a layout exists for the given soft block.
   - `Generate`: Allows you to specify whether or not you want to generate the soft block. The check box is selected by default in the following conditions:

© 2020 All Rights Reserved.

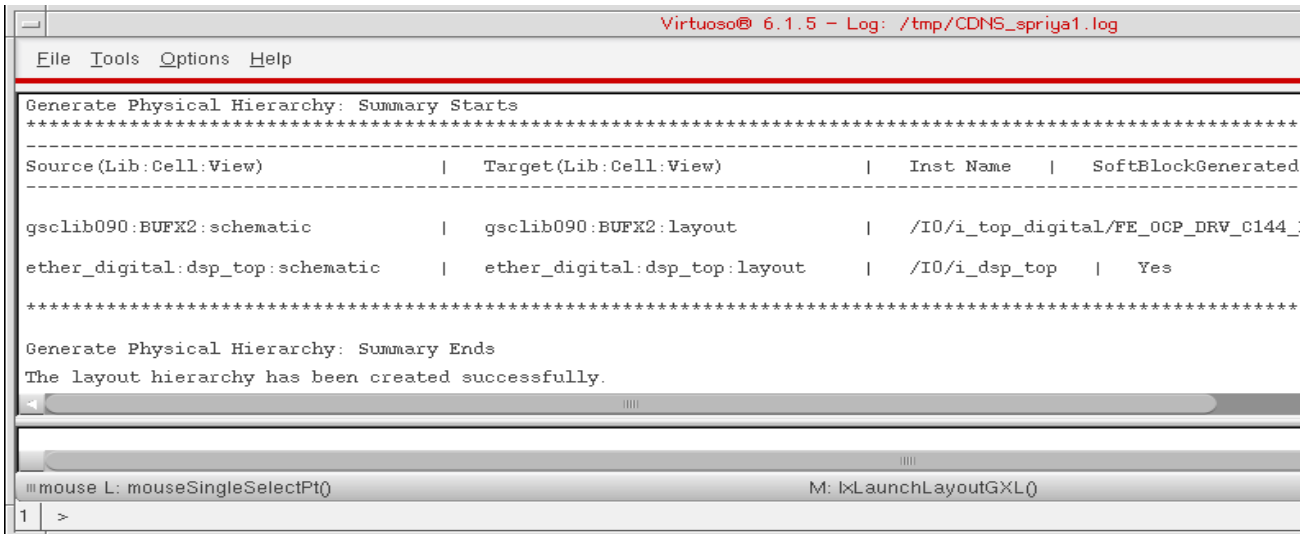
- The soft block parameters have changed and a layout exists.
- The soft block parameters has changed but the layout does not exists.
- Neither the soft block parameters have changed nor the layout exists.

The check box is deselected only when the soft block parameters have not changed, but the layout for the soft block exists.

The pin purpose is read from the `initIOPinLayer` environment variable. Pins are created on this purpose while generating the physical hierarchy.

5. In the *Generate* option, select *all* to generate all soft blocks or *none* to generate none of the soft blocks configured.
6. Click *OK* to generate the physical hierarchy in the layout window.

A summary report of the soft blocks generated by Generate Physical Hierarchy is displayed. The summary report displays a table that lists the source `lib:cell:view` name, the target `lib:cell:view` name, the instance name, and the status of whether the soft block has been generated.



After running Generate Physical Hierarchy, the connectivity extractor is not run automatically. To get the opens and shorts marks, run the extractor by choosing *Connectivity – Update – Extract – Layout*.

Important

If there are incremental updates in the schematic, use the *Connectivity – Update – Components and Nets* command to update the connectivity.

© 2020 All Rights Reserved.

Consider a top-level design layout that contains soft blocks and has been generated by using Configure Physical Hierarchy and Generate Physical Hierarchy and subsequently you have added a block in the schematic.

To generate the layout for the new block, you can use Configure Physical Hierarchy to define the soft block parameters in the schematic and then use Generate Physical Hierarchy to generate the corresponding layout for the new block. In this case, only the new block will be generated because the top-level and existing soft block parameters have not been changed in Configure Physical Hierarchy.

To add the new block to the top-level design layout, use the *Connectivity – Update – Components and Nets* command. The newly generated layout of the block are added to the top-level design layout and connectivity is updated.

After generating the physical hierarchy, for a few soft blocks, a few extra pins may be generated in the layout view. Usually, these are power and ground pins that were not visible in the logical hierarchy of the design. Generate Physical Hierarchy promotes these pins to the top of each soft block while generating their layout views. Use the options in the *Edit – Soft Blocks* form to update the parameters for these pins. To know more about these options, see Changing I/O Pin Attributes.

While generating a physical hierarchy using the Generate Physical Hierarchy command, the following environment variables are honored.

- initCreatePins: Generates pins in the layout canvas.
- initCreateInstances: Generates instances in the layout canvas.
- initCreateBoundary: Generates a PR boundary in the layout canvas.

  If the initCreateInstances, initCreatePins, and initCreateBoundary environment variables are all set to `nil`, then the hierarchy is not generated. Instead, an error message is displayed.

  If one or two of these environment variables are set to `nil`, then the hierarchy is generated. However a warning message is displayed.

  Alternatively, use the *Create Instances*, *Create Pins*, and *Create Boundary* check boxes in the Floorplan Global Options form (Layout GXL Options form in IC6.1.8) to specify whether instances, pins, and PR Boundary need to be generated. For more information, see Controlling Re-Generation of Instances, Pins, and PR Boundary.
- initGlobalNetPins: Generates layout pins for the global nets in the schematic.
- initCreatePadPins: Generates pins and pads for schematic pins that are connected to I/O pads (cells of type `pad`, `padSpacer`, or `padAreaIO`).

© 2020 All Rights Reserved.

- initCreateSnapBoundary: Creates a rectangular snap boundary enclosing the generated PR boundary.
- initCreateMTM: Preserves user-defined bindings.
- initDoStacking: Abuts MOS transistors in a specific order.
- initDoFolding: Divides the prepared transistors into two or more folds.
- extractAfterGenerateAll: Runs connectivity extraction as part of the layout generation command, and lists all incomplete nets in the design after generating the layout.
- initIOPinLayer: Specifies the layer on which pins must be generated.
- initIOLabelType: Specifies the type of label to be generated for I/O pins. Valid values are: `Label` and `Text Display`.
- pinTextSameLayer: Specifies whether the textDisplay or label layer must be the same as pin layer.
- pinTextSamePurpose: Specifies whether the textDisplay or label purpose must be the same as pin layer purpose.

© 2020 All Rights Reserved.

Auto-Generate Hierarchy

(ICADVM20.1 Only) Virtuoso® Floorplanner provides the Auto-Generate Hierarchy functionality, which lets you configure, specify bindings, and generate top blocks and soft blocks in the layout canvas. The Auto-Generate Hierarchy lets you quickly generate a physical hierarchy by applying common parameters. You can define common boundary and pin parameters for the blocks you want to generate. You can specify the shape and size of the boundary that is created when the soft block is generated.

By default, layout view components are generated in the same library as the source schematic library. If the source library is the read-only mode, then the Auto-Generate Hierarchy command fails to generate layout views in the schematic library.

In such a scenario, use `schLayLibraryPair` to map the read-only schematic library to a different layout library for generation. In the following example, a schematic design library `design_sch` is mapped to a layout design library `design_lay`:

```
envSetVal("layoutXL" "schLayLibraryPair" 'string "((\"design_sch\" \"design_lay\")
```

In the following example, regular expressions are used to define mappings:

```
envSetVal("layoutXL" "schLayLibraryPair" 'string "((\"*_sch\" \"*_lay\")
(\"*_sch2\" \"*_lay2\") (\"sch_*\" \"lay_*\") (\"lib1_*_sch2\" \"lib1_*_lay2\"))")
```

Note: Ensure that you set the `schLayLibraryPair` before running the Auto Generate Hierarchy command.

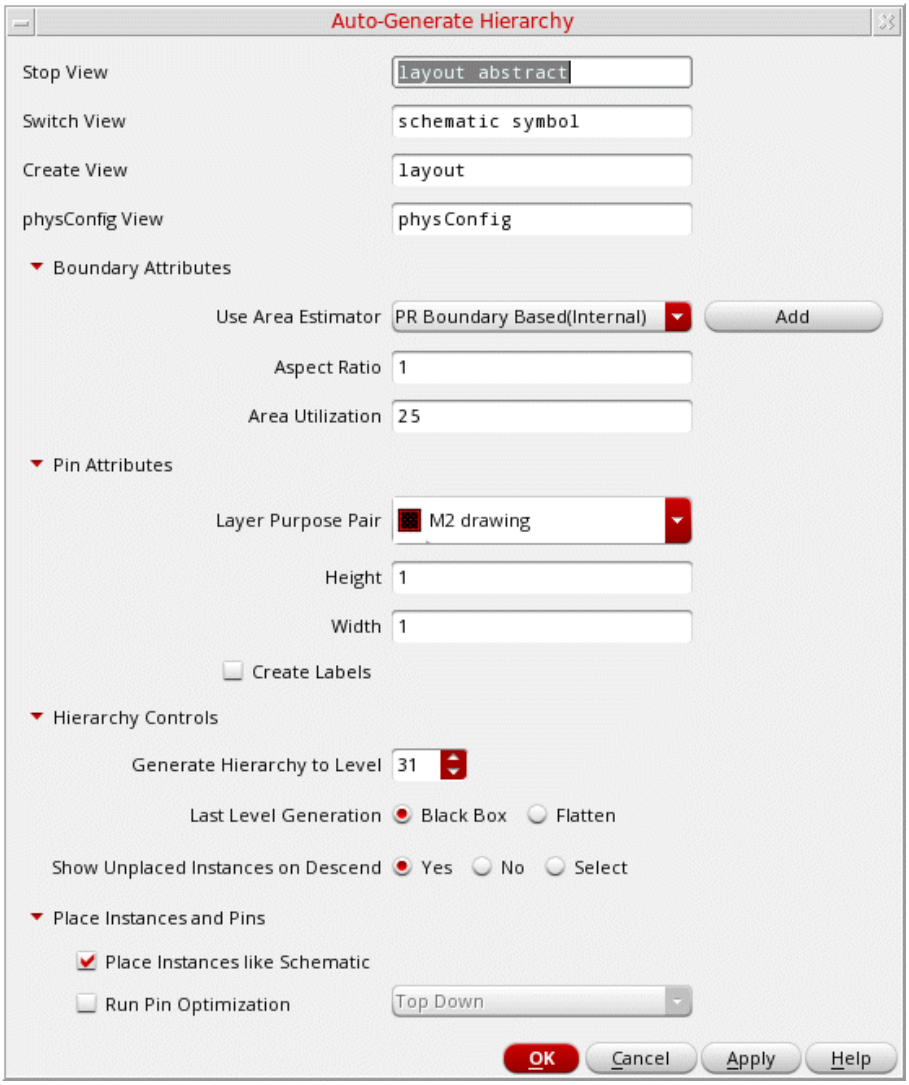
To generate a physical hierarchy:

1. Select *Floorplan – Auto Generate Hierarchy*.

   Note: In Layout EXL, choose *Plan – Generate – Auto Generate Hierarchy*.

© 2020 All Rights Reserved.

The Auto-Generate Hierarchy form appears.



2. In *Stop View*, specify the required physical view names to be used to determine the corresponding physical views. When traversing a hierarchy, the tool instantiates blocks using one of the specified stop views. If no matching stop view names are found, a new view is created with the specified *Create View* field name.

3. In *Switch View* names, specify the names to be used to create a physical view for blocks if the stop views are absent in the hierarchy.

4. In *Create View*, specify the name to be used for the new layout view, which will be created when no matching stop views are found.

5. In *physconfig View,* specify the name of the physical configuration view that provides information about the schematic-layout pairs.

© 2020 All Rights Reserved.

6. Expand *Boundary Attributes* to display the options in this section. You can use the following methods to define the boundary:
   - The *Use Area Estimator* option lets you use one of the following area estimator functions:
     - *PR Boundary Based*: Sums up the polygonal area of the individual instances to derive the overall area estimation.
     - *BBox Based*: Sums up the minimum bounding box area of the individual instances to derive the overall area estimation.
     - *Add*: Lets you register your own area estimation functions and use them. For more information, see Registering and De-Registering Area Estimation Functions.
   - In *Aspect Ratio*, set the width-to-length ratio of the boundary. The default value is `1`, which specifies a square boundary. An aspect ratio of `.5` specifies a boundary twice as high as it is wide. A value of `2` specifies a boundary twice as wide as it is high.
   - In *Area Utilization*, specify the percentage of area within the boundary that must be filled. The default is `25%`.
7. Expand *Pin Attributes* to specify the following common settings for all the soft block pins:
   - *Layer Purpose Pair*: Select the LPP on which the pins must be generated.
   - *Height*: Specify the pin height.
   - *Width*: Specify the pin width.
   - *Create Labels*: Specify whether labels are to be generated along with the soft block pins. When this option is selected, the following environment variables are honored:
     - createPinLabel
     - pinTextSameLayer
     - pinTextSamePurpose
     - labelLayer
     - labelPurpose
     - labelFont
     - labelOverbar

© 2020 All Rights Reserved.

- ❍ labelDrafting

8. Select *Initialize Instances Outside Boundary* to generate all the instances and pins outside the PR boundary, without any overlap. The placement statuses of the instances are set to none. When the check box is cleared, all instances are placed inside the PR boundary.

9. (ICADVM20.1 EXL Only) Expand *Hierarchy Controls* to define settings for hierarchical instances.

10. Specify a value in *Generate Hierarchy to Level* to define the level up to which the hierarchy must be generated. The default (maximum) value is `31`.

11. (ICADVM20.1 EXL Only) Set *Last Level Generation* to one of the following to specify the state in which the last level of hierarchy is to be generated.

    - ❑ *Black Box*: The last hierarchical level is generated as a black box, which contains only the PR boundary and pins.
    - ❑ *Flatten*: Generates the PR boundary and pins in the last level. If any additional stop views are found inside the hierarchy, those levels are flattened and displayed.

12. Set *Show Unplaced Instances on Descend* to one of the following values to specify whether unplaced instances are to be displayed when you descend into the hierarchy:

    - ❑ *Yes* (default): Shows all unplaced instances.
    - ❑ *No*: Hides all unplaced instances.
    - ❑ *Select*: Displays a popup message each time you descend into a hierarchy. You can specify whether you want to show or hide all unplaced instances.

    Environment variable: `showUnplacedInstances`

13. (ICADVM20.1 EXL Only) Expand *Place Instances and Pins*.

14. (ICADVM20.1 EXL Only) Select *Place Instances like Schematic* to place the generated soft block instances inside the PR boundary as in the corresponding schematic view. If not selected, the soft block instances are placed outside the PR boundary.

15. (ICADVM20.1 EXL Only) Select *Run Pin Optimization* to run Pin Optimizer hierarchically on the generated blocks. You can select one of the following modes to run Pin Optimizer:

    - ❑ *Top Down*: Pin Optimizer is first run on the level-0 and level-1 soft blocks to get the best possible net length. Any existing pins, for example those of any level-1 hard blocks, are recognized and the soft block pins are aligned to these pins.

© 2020 All Rights Reserved.

As the next step, the pin positions of the level-1 soft blocks, determined in the previous step, are used to derive the pin positions inside them. This process continues until the leaf level is reached.

- *Bottom Up*: Follows the same behavior as described in *Top Down*, but runs in the opposite direction. Pin Optimizer is first run on pins of the leaf-level layouts that have been realized. The soft block pin positions are then derived.

  Environment variables: autoGenHierRunPinOpt, autoGenHierPinOptMode

16. Click *OK*.

The generated layout is opened in a new window.

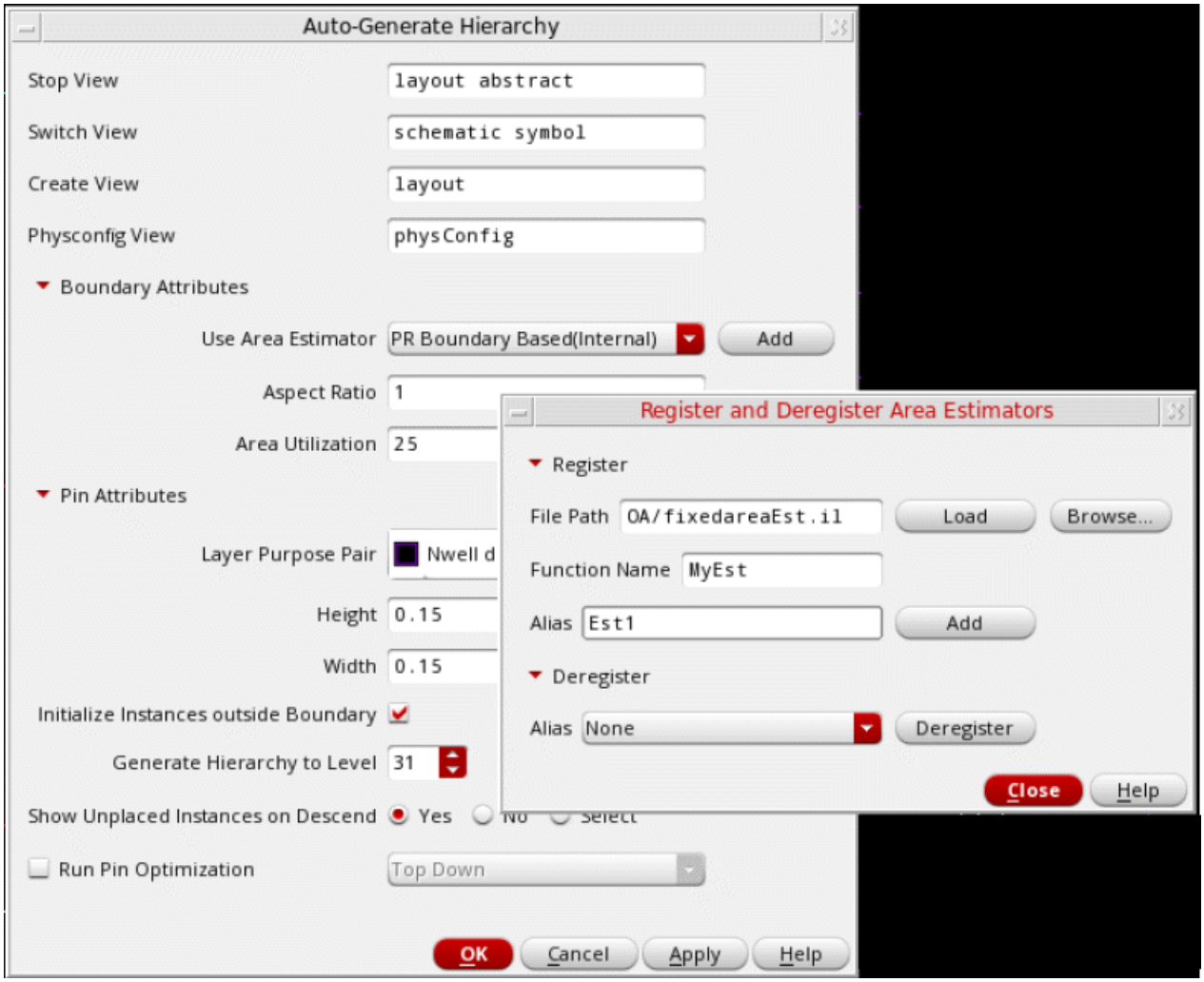
Registering and De-Registering Area Estimation Functions

To register an area estimator function:

1. Click *Add* beside the *Use Area Estimator* list.

© 2020 All Rights Reserved.

The Add/Delete Area Estimators form appears.



2. Expand the *Register* section.
3. Specify the *File Path* of the area estimator file that contains the boundary definitions. Alternatively, click *Browse* to select the required file and close the window.
4. Click *Load* to load the area estimator function.
5. Specify a *Function Name* and *Alias*.
6. Click *Add* to register the area estimator function.
7. Click *Close*.
8. Select the area estimator function that you registered from the *Use Area Estimator* list.
9. Click *Apply* to run the area estimator function.

© 2020 All Rights Reserved.

To de-register an area estimator function:

1. In the Add/Delete Area Estimators form, select the area estimator function from the *Deregister – Alias* list.
2. Click *Deregister*.
3. Click *Close*.

© 2020 All Rights Reserved.

3

Reinitializing the Design

Reinitializing a Floorplan

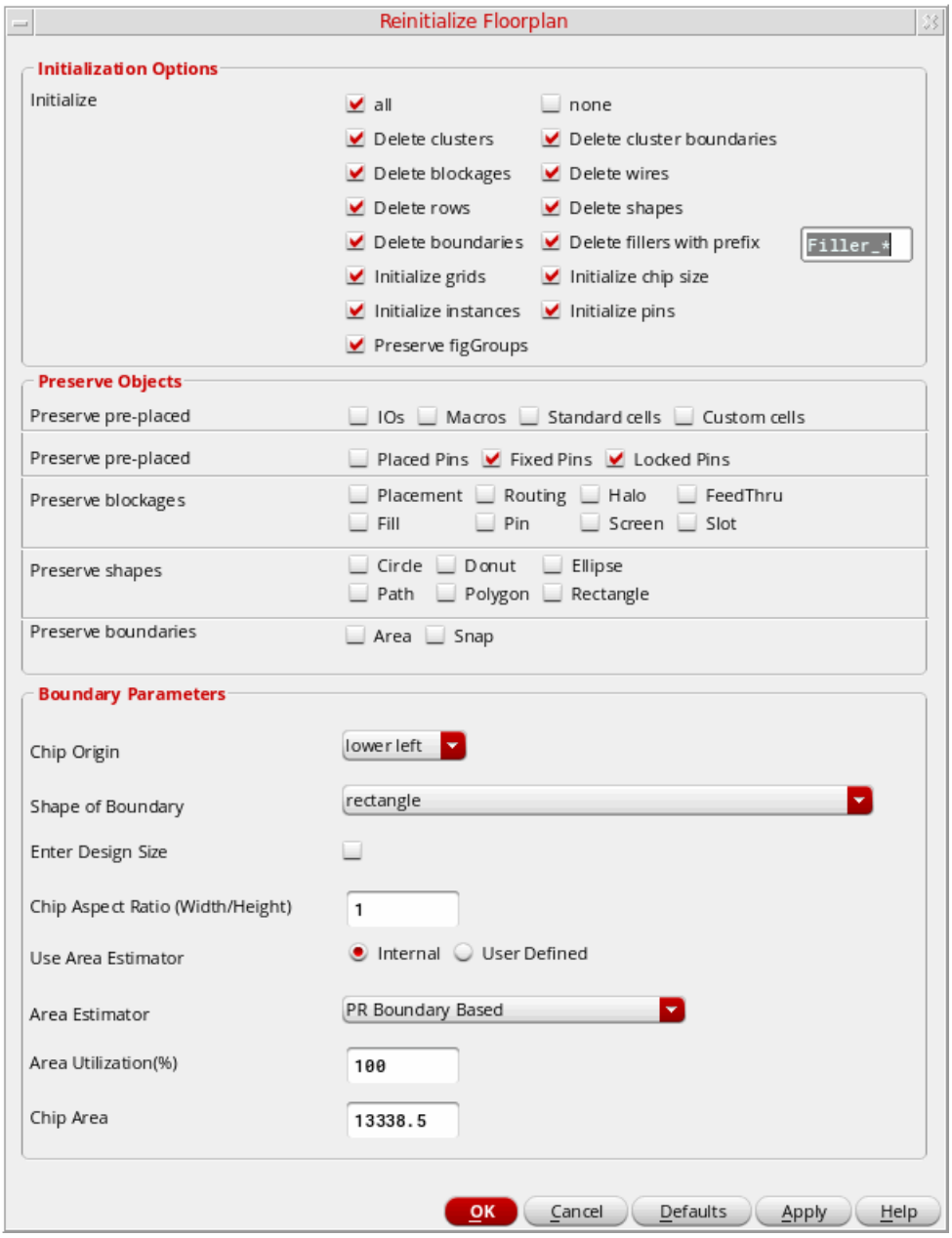
The *Floorplan – Reinitialize* command makes a collection of design objects based on their cellType and places them around the design boundary. You can use the command to alter the state of different design objects simultaneously. By default, the Reinitialize command is run just after Generate Physical Hierarchy, however, it is not mandatory.

Important

(ICADVM20.1 Only) In Layout EXL, choose *Plan – Generate – Reinitialize*.

© 2020 All Rights Reserved.

The *Floorplan – Reinitialize* command displays the *Reinitialize Floorplan* form shown in the graphic.



The form has the following three sections:

- Initialization Options
- Preserve Objects
- Boundary Parameters

© 2020 All Rights Reserved.

Initialization Options

The *Initialization Options* section includes the check boxes that you can use to reinitialize or retain the existing values of the various design objects.

- *all*: Select this option to reinitialize all the design objects.
- *none*: Select this option to deselect all the other options.
- *Delete clusters*: Select this option to delete all the existing clusters and its corresponding cluster boundaries in the design as a cluster boundary could not exist without the associated cluster.
- *Delete cluster boundaries*: Select this option to delete all the cluster boundaries in the design.
- *Delete blockages*: Select this option to delete all the blockages from the design. On selecting this option it activates the *Preserve blockages* option in the *Preserve Objects* section. You can use the *Preserve blockages* field to selectively preserve the **placement**, **routing**, **halo**, **fill**, **pin**, **screen**, **slot** and **wiring** blockages while deleting other blockages.
- *Delete wires*: Select this option to delete all existing geometric routes — vias and pathSegs. The geometric routes is basically the routing made by autorouter (Nanoroute, VCAR (IC6.1.8 only), VSR, or CSR) and with the Wire Editor with the option Segment in Route.
- *Delete rows*: Select this option to delete existing rows in the design. An existing row can either be a standard row or a custom row.

  (Advanced Node Layout EAD Only) Any existing row regions are also deleted. For more information about row regions, see Layout Support for Row Regions in *Virtuoso Placer User Guide*.
- *Delete shapes*: Select this option to delete existing shapes in the design. On selecting this option it activates the *Preserve shapes* option in the *Preserve Objects* section. You can use the *Preserve Objects* section to selectively preserve **circle, donut ellipse, path, polygon**, and **rectangle** shapes while deleting other shapes.
- *Delete boundaries*: Select this option to delete snap and area boundaries in the design. Selecting this option activates the *Preserve boundaries* option in the *Preserve Objects* section. You can use the *Preserve boundaries* field to selectively preserve the **area** and **snap** boundaries.
- *Delete fillers with prefix* - Select this option to delete the filler instances in the design with the filler prefix name specified in the string field next to the *Delete fillers* with prefix option. However, if the filler prefix is not specified in the string filed and the *Delete fillers*

© 2020 All Rights Reserved.

with prefix option is selected, a warning message is displayed. Also, if the characters such as "*" , ".*", etc. are specified in the string field, all the physical only instances in the design are deleted and a warning is displayed.

- *Initialize grids*: Select this option to initialize the grids in a design.

  The grids are initialized in the following order:

  - Detailed routing grid: Each detailed routing grid is defined by the following:
    - Direction (horizontal or vertical)
    - Routing layer
    - Starting point
    - Routing pitch or step
    - Number of grid lines.

    All the routing layer information is read from the following section of the technology file:

```
routingGrids(
;    (                            layer     pitch)
;    ( -------                    -----     -----)
     ( horizontalPitch           "Metal1"   0.29 )
     ( verticalPitch             "Metal1"   0.29 )
     ( horizontalOffset          "Metal1"   0.145 )
     ( verticalOffset            "Metal1"   0.145 )
     ( horizontalPitch           "Metal2"   0.58 )
     ( verticalPitch             "Metal2"   0.58 )
     ( horizontalOffset          "Metal2"   0.145 )
     ( verticalOffset            "Metal2"   0.145 )
    ) ;routingGrids
```

    Using these value and the design boundary, Floorplanner automatically calculates the starting point and the number of grid lines required to cover the entire design area.

    If you do not define a value for `pitch`, the command uses a default value derived from the line-to-via spacing rule defined in the technology file, and displays a warning message.

© 2020 All Rights Reserved.

If the routing offset is not defined for a routing layer in the technology file, then its value is considered by default as half of the routing pitch.

Floorplanner calculates the starting point of the routing grid on each layer by adding the routing offset to the left most coordinate of the bBox of the design boundary.

Floorplanner calculates the count (number of grids) from the step (routing pitch) and the bBox of the design boundary.

If the routing grid cannot be created because the required information is not specified in the technology file, then the routing grid is created by using the values of the manufacturing grid for that layer.

Routing grids in the non-preferred direction are created by using the pitch information. If a routing layer has horizontal as the preferred direction, then the grid in the non-preferred direction, which is vertical for the layer, is created by using the horizontal pitch. Similarly, if the preferred direction is vertical then the routing grid in the non-preferred direction, which is horizontal for the layer, is created by using the vertical pitch.

For example, consider the following entries in the routingGrids and routingDirections section in the technology file:

```
routingDirections(
 ;( layer                        direction     )
 ;( -----                        ---------     )
  ( Poly                         "none"       )
  ( Metal1                       "horizontal" )
  ( Metal2                       "vertical"   )
  ( Metal3                       "horizontal" )
routingGrids(
;    (                              layer     pitch)
;    ( -------                      -----     -----)
     ( horizontalPitch             "Metal1"   0.29 )
     ( verticalPitch               "Metal1"   0.29 )
     ( horizontalOffset            "Metal1"   0.145 )
     ( verticalOffset              "Metal1"   0.145 )
     ( horizontalPitch             "Metal2"   0.58 )
     ( verticalPitch               "Metal2"   0.58 )
     ( horizontalOffset            "Metal2"   0.145 )
     ( verticalOffset              "Metal2"   0.145 )
    ) ;routingGrids
```


© 2020 All Rights Reserved.

In the above, the preferred direction for Metal1 is horizontal. For Metal1, grids in the non-preferred direction, which is vertical, are created by using the horizontal pitch for Metal1 (0.29).

Unlike in the above example, if the horizontal pitch for Metal1 is not specified in the technology file, then the grids in the non-preferred direction is created by using the horizontal pitch of Metal2 layer (0.58).

Note: Routing grid is necessary for the routing and placement of pins.

- Placement grid: The placement grid information does not come from the technology file. It is calculated from the detailed routing grid itself. A placement grid is defined by the following:
  - Direction (horizontal or vertical)
  - Starting Point
  - Step
  - Number of grid lines

  Placement grids always begin at (0.0 0.0), and have unique step value for horizontal direction and vertical direction.

  The placement grid step is the same as the step for the parallel detailed routing grid on the lowest routing layer, i.e. step of horizontal placement grid is equal to the step of the detailed routing grid of the lowermost horizontal routing layer. Similarly step of vertical placement grid is equal to the step of the detailed routing grid of the lowermost vertical routing layer.

  The distance between any two parallel placement grid and detailed routing grid lines is defined by `offset`, a required property defined in the technology file. If you do not define a value for this property, by default the value is set to half the pitch for the detailed routing grid.

- Manufacturing grid: This grid gives the minimum possible resolution and is used for the placement of the analog blocks and their pins. All the grids are in multiples of manufacturing grid. The manufacturing grid value is read from the mfgGridResolutionSection section, which is a sub-section of the controls section in the technology file:

```
mfgGridResolution(( 0.001000 ))

;mfgGridResolution
```

  So, the manufacturing grid step is equal to the mfgGridResolution value and the start point is the lower-left of the design boundary.

© 2020 All Rights Reserved.

You must initialize grids before performing any action on a block.

- *Initialize chip size*: Select this option to create a design boundary. On selecting this option, the Boundary Parameters section in the *Reinitialize Floorplan* form appears. You can create rectangle or rectilinear boundary using the options available in the *Boundary Parameters* section.When the design boundary gets initialized then the halo blockage associated with its prBoundary, if any, is deleted. Also, while initializing the design boundary, if the top level PR boundary is missing and *Initialize chip size* option is off, a warning message will be displayed.

- *Initialize instances*: Select this option to move all *unplaced* and *placed* instances to a position outside the PR boundary without any overlaps. Their instance placement status is reset to *none*. This option does not move *locked* and *fixed* instances. The instances are moved to different quadrants depending on the type of the instance master (oaCellType). Use the CIW – Tools – Set Cell Type form to set the type for a big number of cells at the same time. You can also edit the Master Properties and change the cell type accordingly. The following rules describe the position of the various instance types:

  - Standard cells are placed on the right boundary of the chip.
  - I/Os are placed at the upper boundary of the chip.
  - Macros are placed on the left boundary of the chip.
  - Custom cells are placed at the lower boundary of the chip. If there are any chained custom cells in the design, reinitialization does not break the chaining between the custom cells. However, reinitialization does not create chained custom cells.

  Selecting the *Initialize instances* option activates the *Preserve pre-placed* option in the Preserve Objects section that you can use to selectively preserve the placement of *I/Os*, *Macros*, *Standard cells*, and *Custom cells*.

- *Initialize pins*: Select this option to move all the pins inside the PR boundary of the design and change the pin placement status to *unplaced*.

- *Preserve figGroups*: Enables you to preserve figGroups present in the design. Set this option to ON if you want to preserve the figGroups and the state of the figures associated with it.

  - When the *Preserve figGroups* option is ON, then the figGroups are preserved and placed at the lower boundary of the chip right adjacent to the custom cells. However, when the *Preserve figGroups* option is OFF, then all the figGroups are deleted from the design.

© 2020 All Rights Reserved.

Preserve Objects

The *Preserve Objects* section has the following options:

- *Preserve pre-placed*: This option enables you to selectively preserve the locations of the pre-placed I/Os, macros, standard cells, and custom cells.
- *Preserve pre-placed:* This option enables you to selectively preserve the location of pre-placed pins, fixed pins, and locked pins. By default, fixed and locked pins are preserved.
- *Preserve blockages*: This option enables you to selectively preserve the *Placement*, *Routing*, *Halo*, *Feedthru*, *Fill*, *Pin*, *Screen*, and *Slot* blockages.
- *Preserve shapes*: This option enables you to selectively preserve the *Circle*, *Donut*, *Ellipse*, *Path*, *Polygon*, and/or *Rectangle* shapes.
- *Preserve boundaries*: This option enables you to selectively preserve the *Area* and/or *Snap* boundaries in a design.

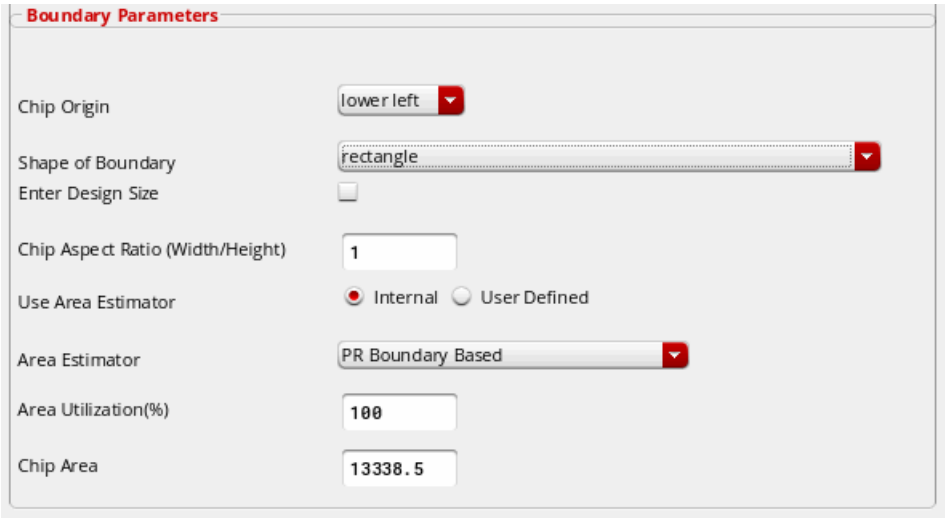
Boundary Parameters

The *Boundary Parameters* section has the following options:

- *Chip Origin*: Select this option to specify the point of origin of the bounding box of the rectangle's boundary as *lower left* or *center*. The default value is *lower left*. If there is no PR boundary, by default, the PR boundary will be created with the lower left of the PR boundary at the origin.

© 2020 All Rights Reserved.

- *Shape of Boundary*: Select this option to specify the boundary as *rectangle* or *polygon*. By default, the shape of the boundary is *rectangle* and the form appears as shown in the graphic.



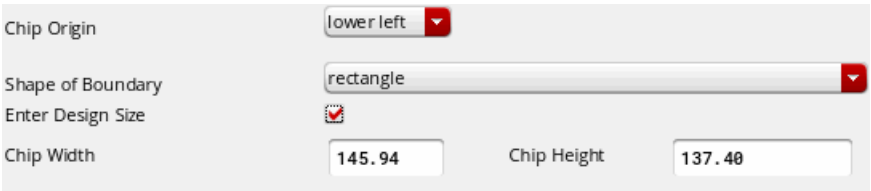
The final coordinates of the new rectangle design boundary depend on the value of *Chip Origin*. If you specify *Chip Origin* as *lower left*, then the lower left of the bounding box of the rectangle's boundary should be at (0.0, 0.0). If you specify it as *center*, then the center of the bounding box of the rectangle boundary should be at (0.0, 0.0).

You can create a rectangular design boundary by:

- Explicitly specifying the **width** and **height** of the boundary.
- Automatically calculate the **width** and **height** using desired aspect ratio (width/height) and estimated required **area** by the boundary.

The following fields are available when the value of *Shape of Boundary* field is **rectangle**:

- *Enter Design Size*: Select the *Enter Design Size* option to specify the width and height of the rectangle. On selecting the *Enter Design Size* option the *Chip Width* and *Chip Height* fields are enabled.

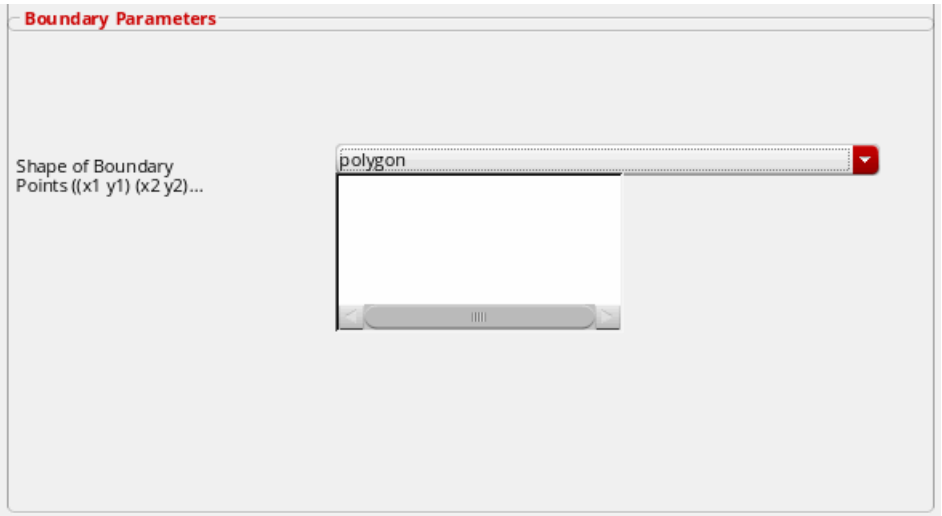


You use the *Chip Width* and the *Chip Height* fields to specify the width and the height of the rectangle.

© 2020
All Rights Reserved.

- *Chip Aspect Ratio (Width/Height):* Use this field to specify the width-to-height ratio for the chip. The default value is 1.0. This option is available only when *Enter Design Size* is deselected.
- *Use Area Estimator*: You can specify the area estimator that you want to use.
  - *Internal*: Choose this option when you want to use internal area estimator. There are two internal estimators “PR Boundary Based” and “BBox Based” are already present.
  - *User Defined*: Choose this option to use your own registered area estimators.
- *Area Estimator*: Select the area estimator you want to use to estimate the area required by the chip. For more information, see Area Estimation Framework.
- *Chip Area*: This field displays the value of the area calculated by the area estimator. You can manually specify the area in this field. The value of this field will be used to calculate the final chip width and height.

You can also create a rectilinear design boundary by selecting *Polygon* in the *Shape of Boundary* field. The *Boundary Parameters* section changes as shown in the graphic.



To create a rectilinear polygon, specify the vertices of rectilinear polygon in the *Points (x1 y1) (x2 y2)*... field and click *OK*.

Use the following rules to enter the co-ordinates (points):

- Specify the points as (x1 y1) (x2 y2)... or ((x1 y1) (x2 y2)).

  Example1: (0 0) (0 100) (100 100) (100 50) (50 50) (50 0)

  Example 2: ((0 0) (0 100) (100 100) (100 50) (50 50) (50 0))

© 2020 All Rights Reserved.

- Points should be rectilinear.

  Example: (0 0) (0 100) (100 100)....

  Non example: (0 0) (0 100) (100 150)....

- Points should be continuous.

  Example: (0 0) (0 100) (100 100).....

  Non example: (0 0) (100 100) (0 100).....

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

4

I/O Planning and Placement

Introduction to IO Planning

The IO placement engine consists of set of commands by which user can optimize IO position. It is a row based placement methodology. The first step in IO planning and placement is to create IO rows. Once the IO rows are created, then place PADs inside the IO rows. After PADs are placed inside the IO rows, you can insert filler cells in the gaps between the pads and the rows. Thereafter, you can place corner cells on the 4 corners of the design.

The following commands have been provided for IO placement engine in the Virtuoso platform:

- Row Creation

  Creates IO rows of PAD type siteDefs available in technology.

- PAD Placement

  Places celltype of PAD instances in IO rows.

- Filler Cells

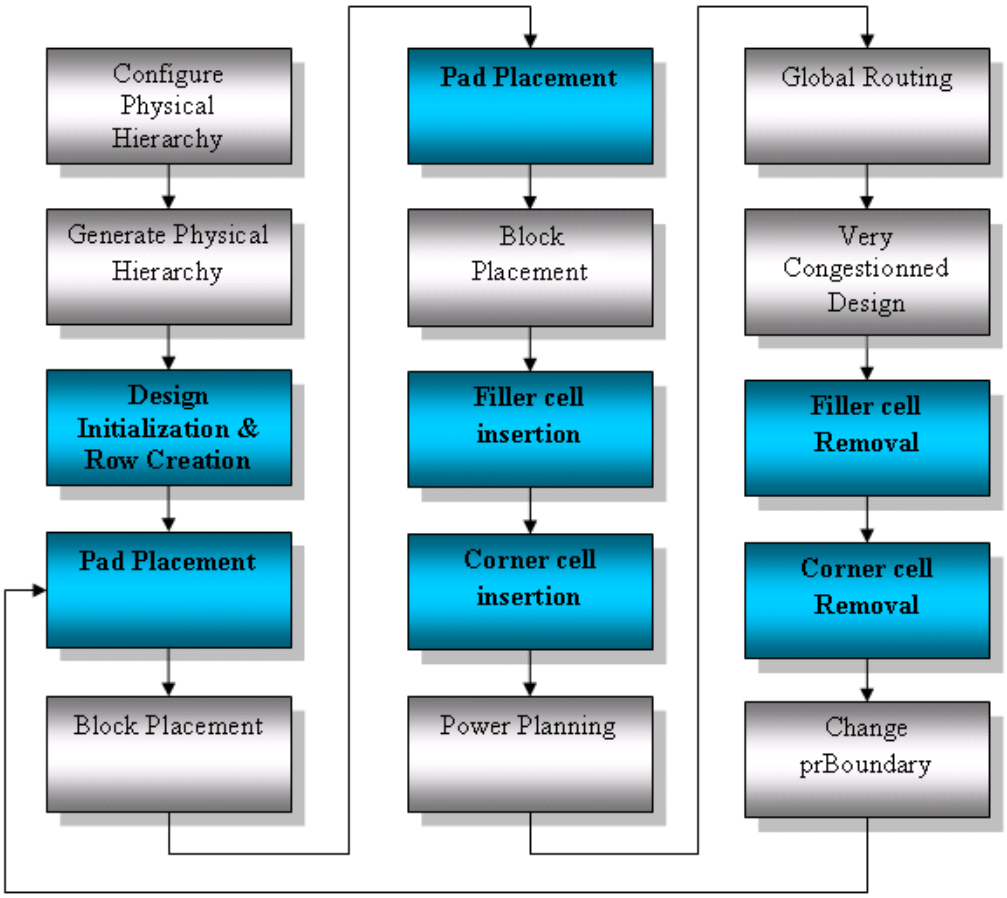
  Inserts filler cells once Pads instances are placed to maintain signal continuity in the IO row.

- Corner Cells

  Inserts corner cells not in the row but in between the rows to maintain signal continuity between IO ring.

The following diagram describes the flow of IO placement engine.

© 2020 All Rights Reserved.



Database Preparation

In database preparation, IO Placer requires information about the technology file, Site pattern, and cell type for pads, fillers, and corners.

In technology file, ensure that at least one SiteDefs exists and is of PAD type.

Site pattern - for each instance, that is, pad, corner, filler, you can define a site pattern. A site pattern is a set of sitedefs. The site definition is read from the technology file. For more information on how to set site definition, refer to the *Virtuoso Technology Data ASCII Files Reference* user guide.

© 2020 All Rights Reserved.

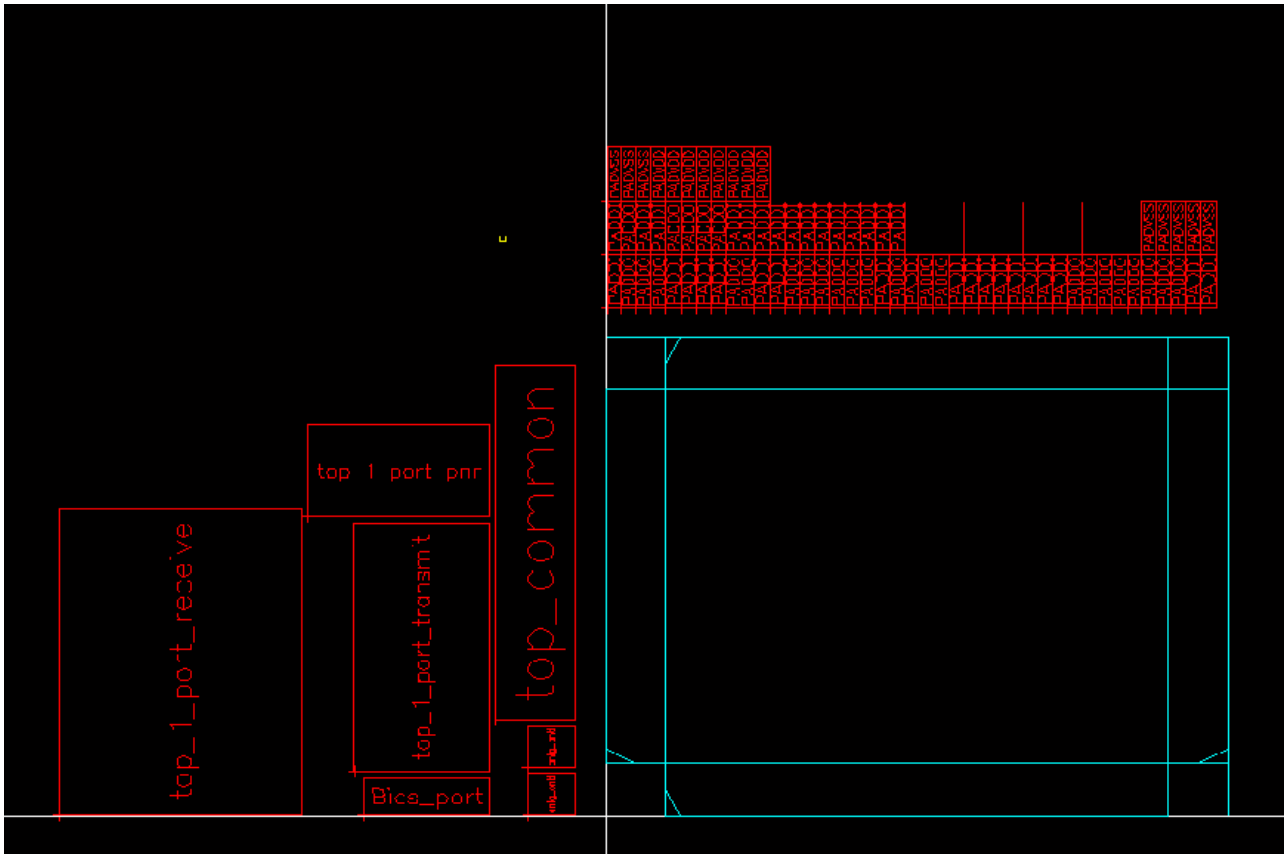
Setting the Cell Type - The various CellTypes for the I/O planning and placement are shown in the table below:

I/O	OpenAccess Object	Cell Type	Supported By
I/O	PadCellType	pad padAreaIO	PAD placement
	PadSpacerCellType	padSpacer	Filler Cell
	CornerCellType	corner	Corner Cell
	PadAreaIOCellType	padAreaIO	Not Supported
	CoverBumpCellType	coverBump	Not Supported

Row Creation

In Floorplanner, the first step that IO planning and placement has is to create IO rows. The rows determine the legal placement area for pads, where to place the PADs and the type of PADs that are to be placed. You can manually create rows and use IO Placer with manually created rows. However, the manually created rows need to have the right settings. There is no restriction on where should the rows be created. Also, it is not mandatory to create rows on the periphery.

© 2020 All Rights Reserved.



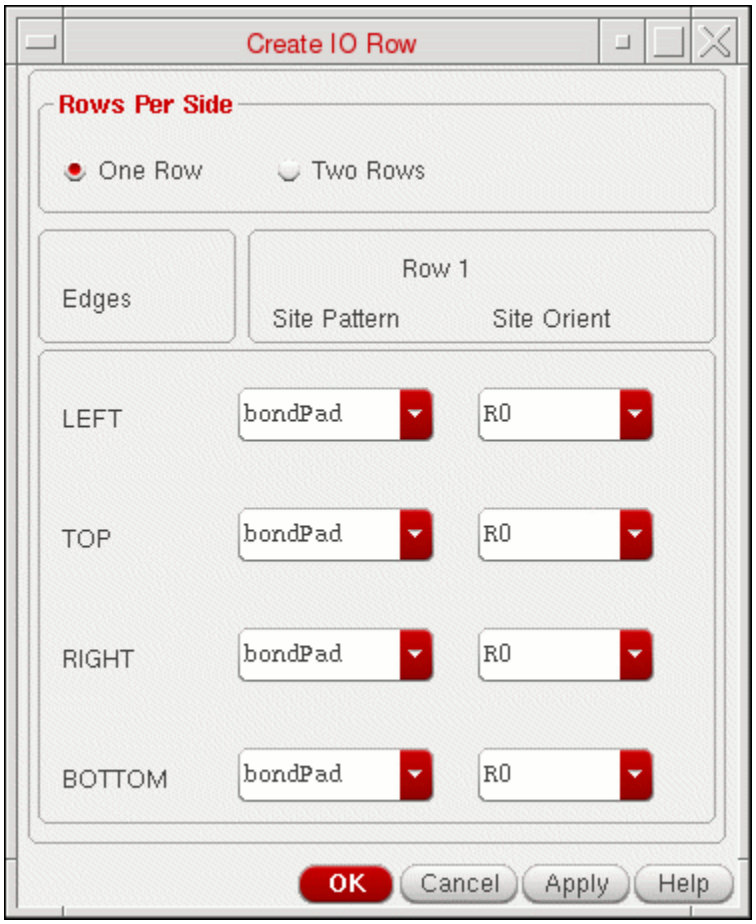
To create rows, you can click the *Floorplan – I/O Placer - I/O Row Create* command. The Create IO Row form is displayed.

Important

(ICADVM20.1 Only) In Layout EXL, choose *Plan – I/O Planning – I/O Row Create*.

Note: The site definition of the type ‘PAD’ will be read from the technology file and displayed in the *Site Pattern* fields. If there is no site definition of type PAD available, the row creation UI is not displayed.

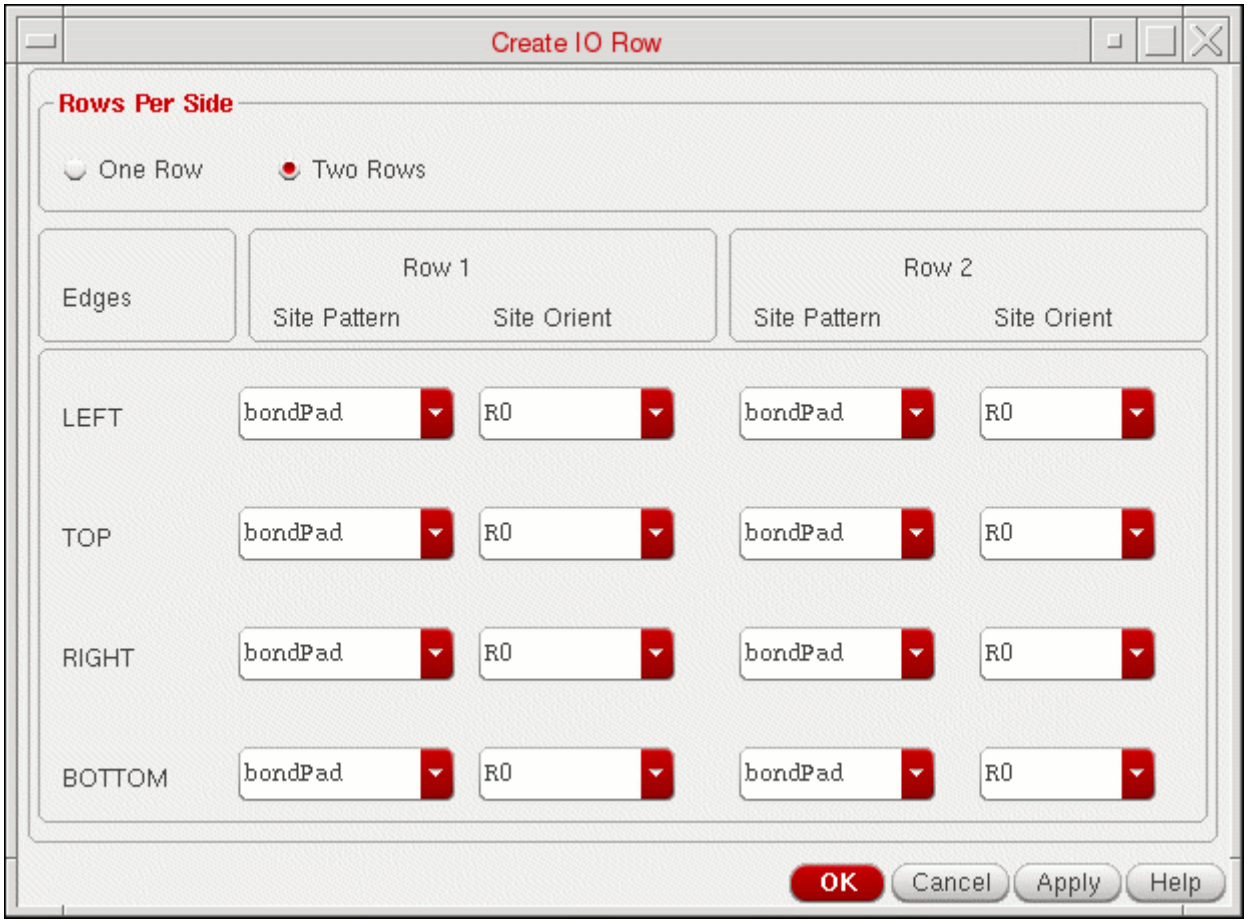
© 2020 All Rights Reserved.



In the Create IO Row form, the *Rows Per Side* section has two options, *One Row* and *Two Rows*. *One Row* is selected by default and the options for Row 1 are displayed. A maximum of two rows can be created on each side (i.e. TOP, LEFT, BOTTOM, and RIGHT) of the top

© 2020 All Rights Reserved.

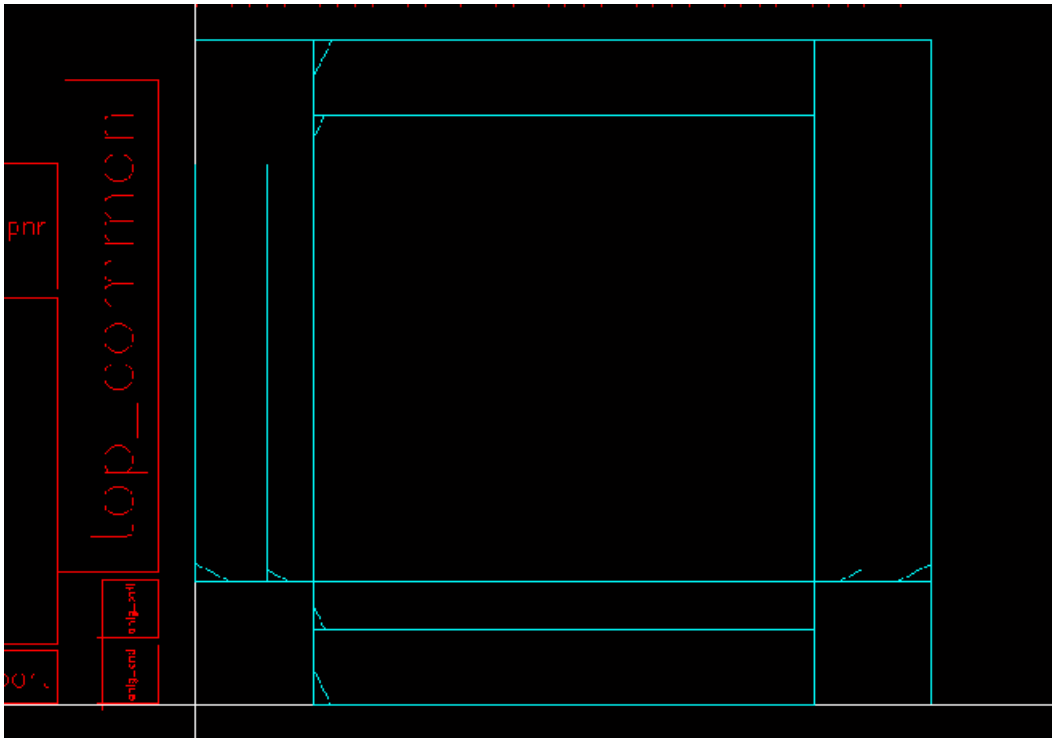
level boundary. If you want to create two rows on each side, select *Two Rows*. This will insert the UI for the second row, as displayed in the following figure



To create a row, you need to specify the row site patterns and their orientation for each side from the *Site Pattern* and *Site Orient* combo fields, respectively and then click *OK*. This will create the row at the appropriate position. If the row already exists at the position where new row is to be created, the existing pad rows will be replaced by the new row. If you run this command again and some rows already exists, they will be deleted automatically.

© 2020 All Rights Reserved.

Note: The site pattern is extracted from the technology file. The site orientation is the relative orientation of the pad inside the row and not the final pad orientation.



PAD Placement

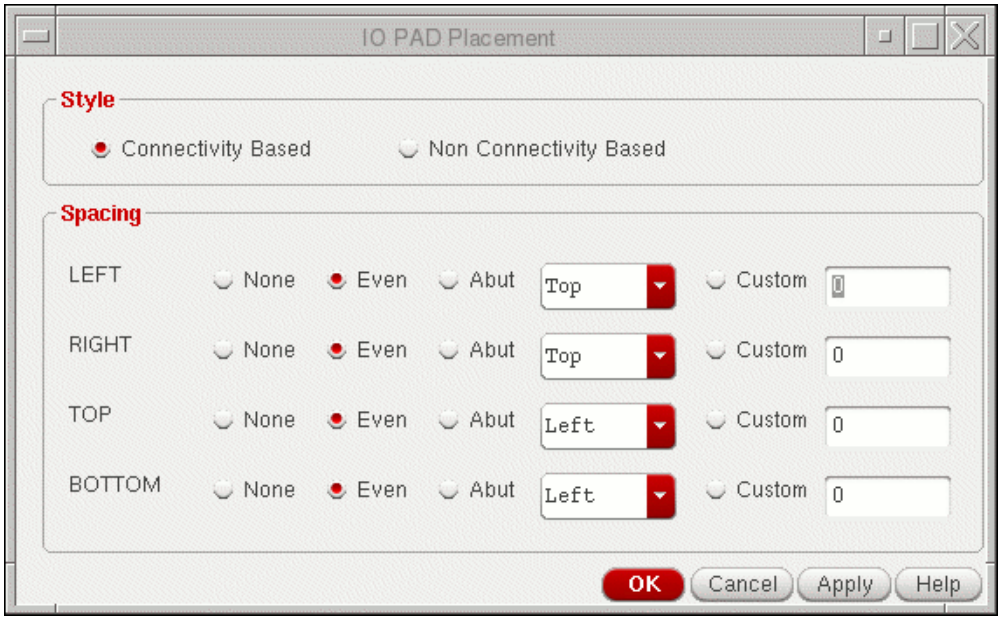
PAD placement is an essential part of the IO placement methodology. It is the next step after row creation in IO Placer. IO rows should be present before running PAD placement. The PADs are placed on the periphery of the design in the IO rows.

To place PADS on IO rows, click the *Floorplan – I/O Placer – I/O PAD Placer* command. The IO PAD Placer form is displayed.

Important

(ICADVM20.1 Only) In Layout EXL, choose *Plan – I/O Planning – I/O Pad Placer*.

© 2020 All Rights Reserved.



The IO PAD Placement form has two sections — *Style* and *Spacing*. When the IO Placer form is invoked, you can select the placement style and put the placement constraints if required.

Style

There are two types of placement styles that are supported, Connectivity Based and Non-Connectivity Based. The Style section allows you to select either ‘Connectivity Based’ or ‘Non Connectivity Based’ style of placement. By default, Connectivity Based style is selected.

- *Connectivity Based*

  You can use this option to place the PADs in the rows minimizing the net length between the PADs and the placed blocks in the design.

  Note: The assumption is that in Connectivity mode, the core instances re placed inside the prBoundary.

- *Non Connectivity Based*

  This option enables you to distribute the PADs around the periphery of the design boundary without considering the connectivity between PADs and blocks placed in the design.

© 2020 All Rights Reserved.

Spacing

Spacing constraints defines the inter spacing between the PADs in a given row. This is a post processing step for the PAD placement engine. To define the inter spacing between PADs on per row basis, you can select any one of the following four spacing constraints

- *None*

  This option is selected by default. This means that all the placement constraints are OFF by default and pads will be placed with spacing value 0.

- *Even*

  You can select this option to distribute PADs evenly (with same spacing between PADs) in the designated row.

- *Abut*

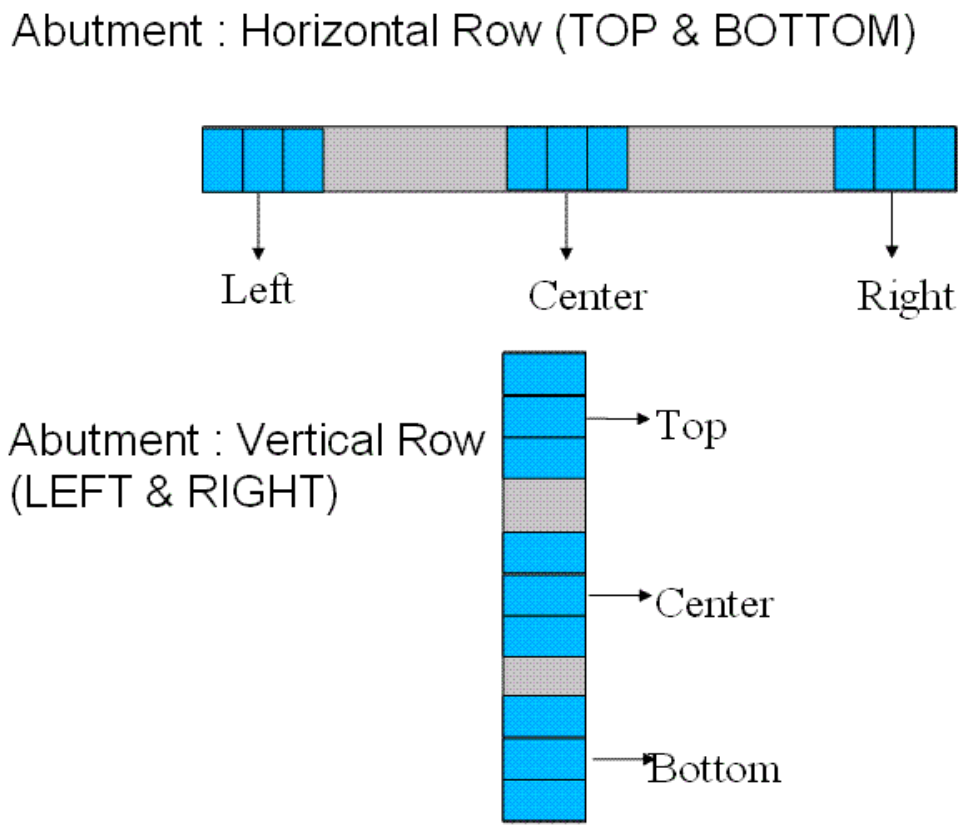
  This option guides the placement engine to abut the PADs in the designated row. The selection of abutment will depend upon the row side.

  - If the row is horizontal (TOP or BOTTOM), then the choices are ‘LEFT’, ‘RIGHT’, and ‘CENTER’.
  - If the row is vertical (LEFT or RIGHT), then the choices are ‘TOP’, ‘BOTTOM’, and ‘CENTER’.

  For example you may specify area as *Left*, *Right*, and *Center* for the top and bottom rows and *Top*, *Bottom*, and *Center* for the left and right rows.

© 2020 All Rights Reserved.

The following figure describes the various areas for each row.



The following table lists the various abutment options.

Rows	Area
Top	Center, Left, Right
Bottom	Center, Left, Right
Left	Center, Top, Bottom
Right	Center, Top, Bottom

- *Custom*

  This option guides the placement engine to place the PAD with the specified inter spacing value.

© 2020 All Rights Reserved.

Supported Constraints

The IO Pad Placer supports the following constraints:

- Fixed/Locked Constraint

  IOPlacer does not place PADS that have the fixed or locked constraints. The following table lists the placement status of I/Os before and after I/O placement.

Placement status before I/O Placement	Placement status after I/O Placement
Placed	Placed
Firm	Firm
Locked	Locked

- Alignment Constraint

  You need to specify the side (TOP, BOTTOM, LEFT, RIGHT) for pad instances.

  IO Placer preserves the order in which the pad instances are placed in the design. In some cases, you may need to set the *PreserveOrder* parameter of the alignment constraint to `TRUE` to get the required results.

- Orientation Constraint

  You need to specify the orientation for pad instances.

- Spacing constraints

  The spacing constraint is defined in the IO Pad Placement form (Even, Abut, and Custom).

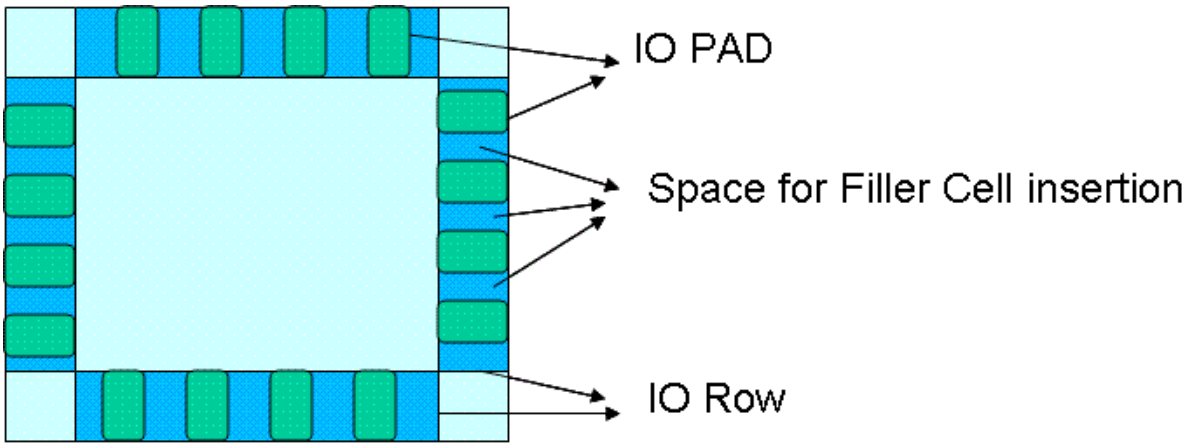
- Placement Blockage

  I/O Pad Placer respects placement blockages and pre-placed instances and does not allow you to place an instance in the design. This means while placing a PAD, if an instance is found on a location where the PAD is to be placed, the placement of the PAD is accordingly adjusted leaving the instance as is.

© 2020 All Rights Reserved.

Filler Cells

The Filler cells insertion is needed to maintain signal continuity in the IO ring for power and ground distribution and for ElectroStatics Discharge (ESD) protection reasons.



Filler cells insertion is done after PAD placement is done in the IO rows of the design. The filler cells are inserted in the empty space between PADs in an IO row. In order to fill the gap, the filler instances with the exact height matching the row site height are chosen. This is done to minimize the number of filler cells that are inserted by the insert filler engine. The filler cells that are to be inserted will be used from the Filler Cells table in the Insert Filler Cells form.

To insert filler cells, you first need to define the design area. It could either be an entire design or a selected region of a design. After defining the design area, select the filler cells from the libraries that are to be used.

To open the Insert Filler Cells form click the *Floorplan – I/O Placer* – Insert Filler command. The Insert Filler Cells form is displayed.

Important

(ICADVM20.1 Only) In Layout EXL, all Floorplan commands are available under the *Plan* menu.

The Insert Filler Cells form has three sections.

- Region Selection Mode
- Select Filler Cells

© 2020 All Rights Reserved.

- Filler Instance Attribute

Region Selection Mode

In this section, you can select either of the two modes, i.e, work on entire design or work on a selected region.

- *Entire Design*

  Enables you to select the entire design for inserting filler cells. This means that filler cells will be inserted in all the IO rows of the design.

- *Selected Region*

  Enables you to select a region for inserting filler cells. This means that only few IO rows need to be used for filler insertions.

  - *Draw*

© 2020 All Rights Reserved.

Clicking *Draw* will enable you to select the area of the design in the canvas, on which operation needs to be performed. This will automatically populate the *Coordinates* field accordingly.

- *Coordinates*

  Defines the coordinates of the selected region. You can manually edit the value in the coordinates.

- *Zoom to*

  Clicking the *Zoom to* button will zoom in the selected region.

Select Filler Cells

The *Select Filler Cells* section facilitates you in selection of few cells from entire gamut of libraries/cells/views. The few selected cells will be used by the engine as filler cells.

To select a lib:cell:view for the filler cell, do one of the following:

- Type the library, cell, and view names in corresponding fields.
- Click *Browse* to display the Library Browser window. Select the lib:cell:view name from the window. The value of the selected lib:cell:view name will be displayed in the *Library Name*, *Cell Name*, and *View Name* fields, respectively. You can keep the Library Manager window open and click *Add Filler* as soon as a new cell is selected.

  - *Add Filler*

    To add the selected lib:cell:view name as a filler, click *Add Filler*. The selected lib:cell:view name will be displayed in the filler table.

    Note: Invalid lib:cell:view values are not added to the filler table. An appropriate warning message is displayed in the CIW.

  - *Remove Filler*

    To delete the selected lib:cell:view name as a filler, click *Remove Filler*. It will just remove the selected rows from the table.

Filler Instance Attribute

In this section, you can define the instance name prefix and the placement status of the filler cell.

- *Instance Name Prefix*

© 2020 All Rights Reserved.

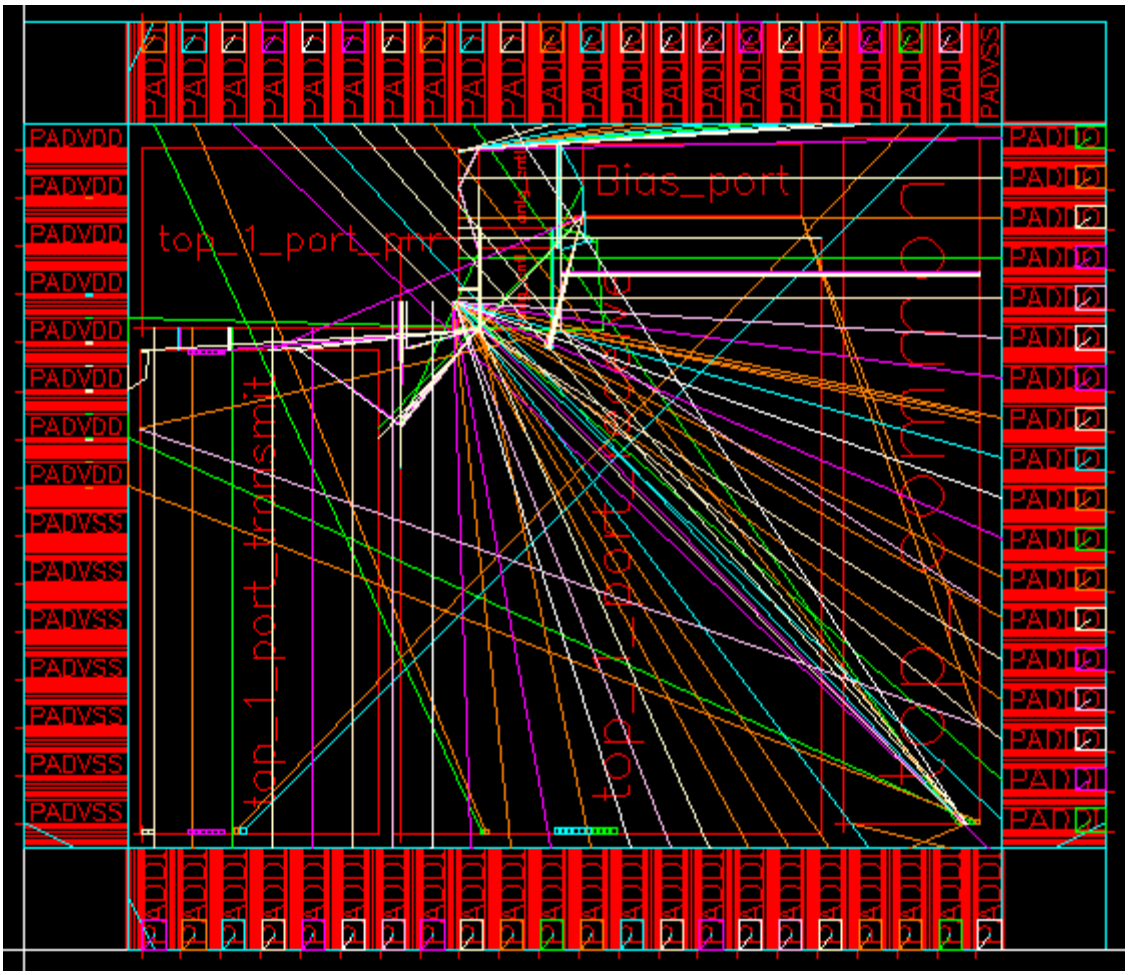
Contains the instance name prefix, which will be pre-pended to the inserted physical only filler instances. This will help you recognize the filler inserted in a specific row or location for deleting them or doing some edits.

Note: The physicalOnly attribute is automatically added to the new cells that are inserted.

- *Mark Fillers as Fixed*

  Select this field to specify the placement status of inserted cells as *Fixed*. By default, the check box is not selected. When the check box is deselected, the placement status of the inserted cells is set to *Placed*.

After clicking *OK* or *Apply* in the Insert Filler Cells form, the filler will be inserted according to their site pattern (to choose the correct row to place them) and then filler cells will be placed from the biggest to the smallest in order to minimize the number of inserted filler cells.

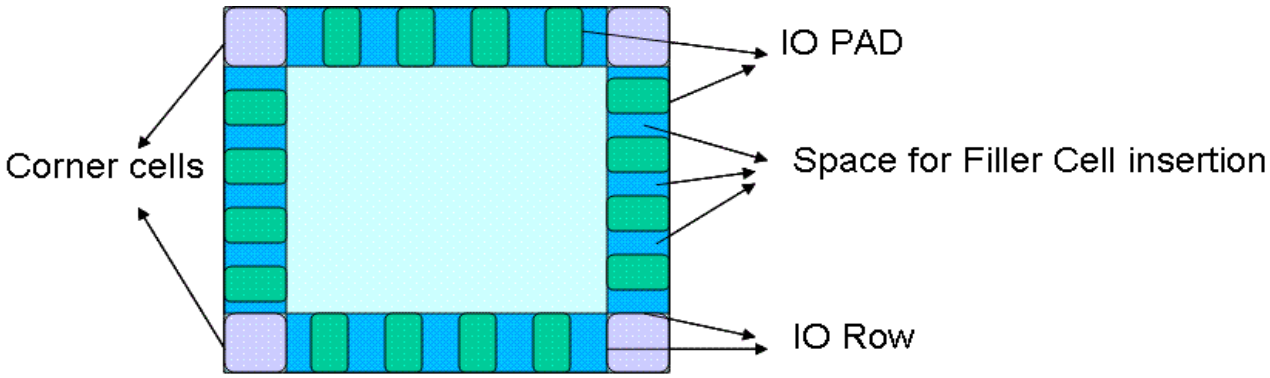


© 2020
All Rights Reserved.

Note: Filler cells with height less than row site height will not be considered. This is because if we have filler cells lesser than the row site there will be discontinuity in the rails.

Corner Cells

Corner cells insertion is needed to maintain signal continuity in the IO ring between the IO row power and ground distribution. The four corner cells are required to be placed on the four corners of the design.



Corner cells insertion and deletion is done after PAD and filler cell placement is done in the IO rows of the design.

The high level algorithm for corner cells is as follows:

- The list of the selected instances are filled in the corner UI.
- Engine takes the inputs instance name and its orientation for each corner from the UI.
- The corner row is created of appropriate site(s) in the design before placing the corner cells.
- Corner cells will be placed in the appropriate row as per the orientation specified.

Before inserting corner cells, ensure the following:

- The library has at least four instances with their `cellType` set to corner. If not, a suitable message is displayed in the CIW.
- The row in which the corner cells must be placed has the same `sitePattern` as the corner cells. Corner cells can be placed only in rows with matching `sitePattern`.
- The technology file contains the `sitePattern` under the `siteDefs` section. If not, a suitable message is displayed in the CIW.

© 2020 All Rights Reserved.

If all the above conditions are met, a corner row of the given `sitePattern` is created, and cells with `cellType` corner are inserted in these rows.

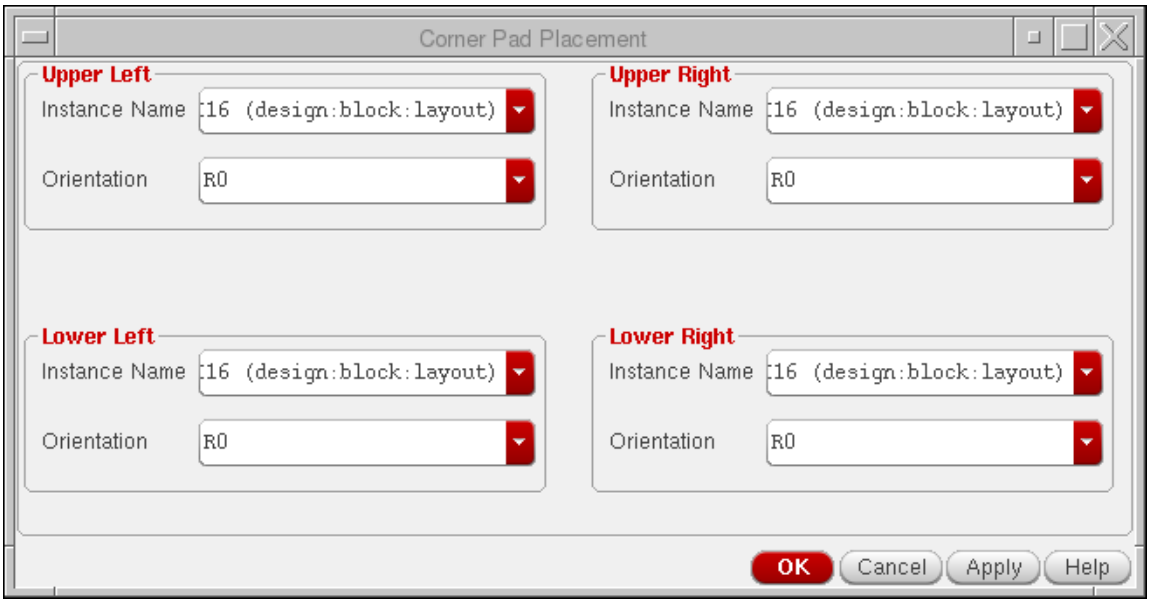
The procedure to insert corner cells is as follows

- Selecting corner instances

  Ensure that the corner instances are already present in the design. The corner instances can be selected in the following ways:

  - Navigator
  - Search Engine
  - Canvas Selection

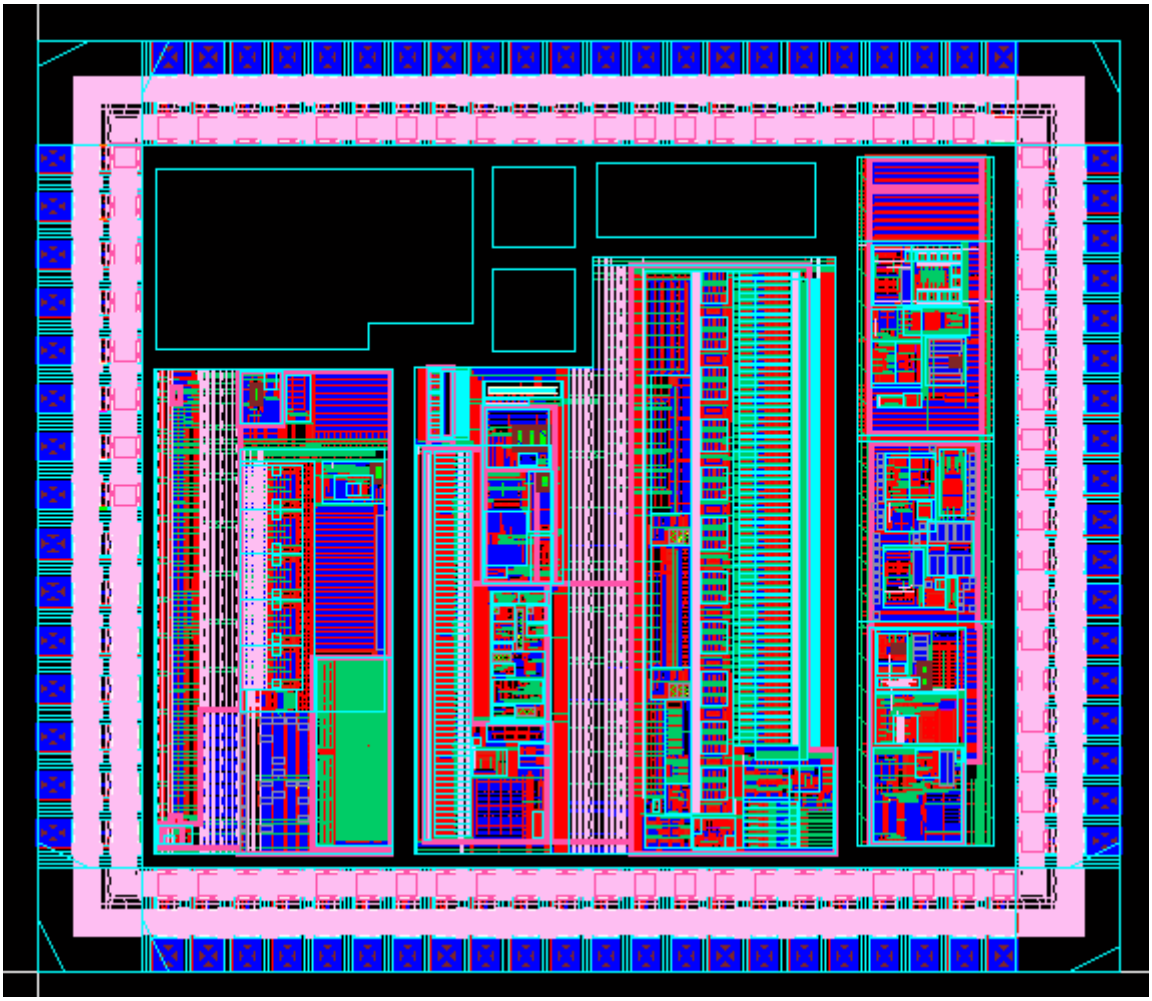
Once the corner instances are selected in the design, invoke the Corner Pad Placer command by clicking *Floorplan – I/O Placer - Corner Pad Placer*. The Corner Pad Placement form is displayed. The form is by default populated with the selected instances. However, if the corner instances are not selected, the form will not be displayed and a corresponding message is displayed in the status bar.



- Placing the corner - In the Corner Pad Placement form, you can specify another instance for each corner from the *Instance Name* list. You can also set the orientation for each corner from the *Orientation* drop-down list box. Once the instance is selected and the

© 2020 All Rights Reserved.

orientation is set, click the OK button. A row will be created automatically. Simultaneously, a corner cell is placed in that row.



© 2020 All Rights Reserved.

5

Placing Blocks

The *Floorplan -> Block Placer* command enables you to automatically place all hard and soft blocks in a design and minimize the wire length and chip area.

Important

In Layout EXL, choose *Plan – Block Planning – Place Blocks*.

Block placer runs on the entire design. Before you run block placer, ensure that the design has a PRBoundary and at least one soft or hard block. The command can adjust soft blocks within the specified width and height range to maximize chip density.

Block placer supports both top-level and block-level rectilinear design boundaries.

The following table lists the cellTypes on which block placer operates. For information on how to set the cellType, refer to the Setting the Cell Type section.

Blocks	Celltype
hard blocks	block
	blockRing
	cover
Soft blocks	softMacro
	blockBlackBox

The following table mentions block snapping with respect to the block type.

Block Type		Edge Snapping
Hard Block/Soft Block	CUSTOM	Manufacturing grid
	DIGITAL	Placement grid

© 2020 All Rights Reserved.

Note: Block placer does not place standard cells and I/Os.

Block placer moves the macros (hard blocks and soft blocks) with the following placement status:

- None
- Placed

Block placer does not alter the position and dimensions of macros with the following status:

- Fixed
- Locked

The following table lists the placement status of blocks before and after block placement.

Placement status before Block Placement	Placement status after Block Placement
None	Placed
Placed	Placed
Firm	Firm
Locked	Locked
Unplaced	Placed

When block placer is invoked on a design, it reads the constraints specified in the Constraint Manager. You can also specify additional constraints in the *Block Placer* form. Block placer tries to respect these constraints at all times.

Block placer also takes into account the blockages in a design and does not place blocks on blockages. Block placer treats pre-routes as blockages, and therefore does not place blocks on pre-routes. Also, the top-level pins having the status as Fixed, Placed, and Locked are treated as blockages, and therefore does not place blocks on such pins.

During block placement, Block placer ignores the connectivity information of standard cells that are outside the PRBoundary. Block placer does not check for overlaps between blocks and pins during placement.

Note: The log file is available in the bplogs/bp_report.log file. You can modify the log file location using an environment variable BPDIR. If this variable is set, a directory is created in the testcase directory with the value of BPDIR variable. However, if the variable is not set, then the default directory *bpLogs* is created in testcase directory.

© 2020 All Rights Reserved.

The block placer menu has the following options:

- Place Blocks
- Load Solutions
- Report Placement Statistics

Place Blocks

To invoke block placer, click *Floorplan -> Block Placer -> Place Blocks*. The *Block Placer* form is displayed.

© 2020 All Rights Reserved.

Important

(ICADVM20.1 Only) In Layout EXL, choose *Plan – Block Planning – Place Blocks*.

© 2020
All Rights Reserved.

The *Block Placer* form has the following tabs:

- General
- Constraints

General

The general tab has the following sections that you can use to place blocks:

- Chip Dimensions
- Cost Order
- Options

Chip Dimensions

The Chip Dimensions section has the following options:

- *Fixed Die* — Select this option to specify that block placer cannot alter the chip dimensions. This option is selected by default if Boundary Area constraint is unavailable in the design. However, if Boundary Area constraint is available, the *Fixed Die* option is deselected. The block placer runs in the variable die mode and can alter the chip dimensions, if required, to place the blocks.

  Block placer will always run in *Fixed Die* mode if any of the following is true:

  - Design boundary is rectilinear in shape
  - The coreBoxSpec or IOBox attributes of PRBoundary are present
  - Design contains IO rows

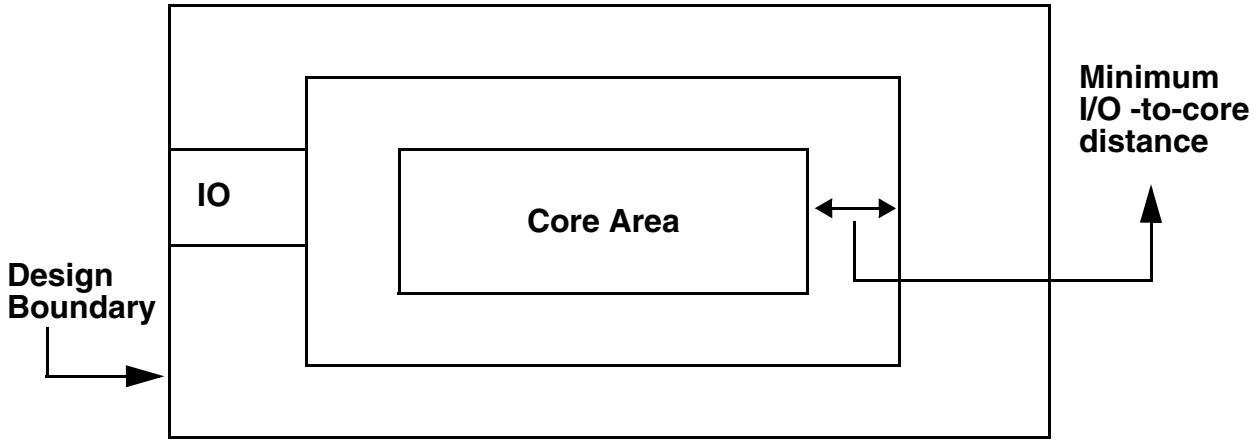
  If the *Fixed Die* option is selected, the *Specify Chip Dimension Range* option is not available.

  The value you specify in this field is retained when you restart block placer.

  Minimum I/O to Core Distance — Specify the distance between the IOBox and the coreBox (the spacing between the pads and the blocks). The default value is the maximum value of minSpacings of routing layers across all layer purposes. This option

© 2020 All Rights Reserved.

is available only when there are IO pads in the design. The value you specify in this field is retained when you restart block placer.



Minimum Boundary to Instance Distance — This option is available only when there are no IO pads in the design and takes the top-level rectilinear boundary into account. *S*pecify the minimum distance between the design boundary and instance edge in the *Minimum Boundary to Instance Distance field*. The default value is the sum of the maximum value of minSpacings of routing layers across all layer purposes and the largest pin size in the design. The block placer retains the value you specify in this field when you restart block placer in the same Virtuoso session.

- *Specify Chip Dimension Range* — This option is available only if the *Fixed Die* option is not selected. You can specify the method that you want to use to define the dimensions of the chip by using the following options:
  - *Width and Height*
  - *Area and Aspect Ratio*

If you want to specify the width and the height range of the chip, select the *Width and Height* option button. Selecting the *Width and Height* option, activates the following options:

PRBoundary Width Range — You specify the minimum and maximum width of the chip in this field. The initial value is populated from the values specified in the areaBoundary constraint in Constraint Manager. This field is available only if the *Fixed Die* option is not selected. The values you specify in this field are retained when you restart block placer.

PRBoundary Height Range — You specify the minimum and maximum height of the chip in this field. The initial value is populated from the values specified in the areaBoundary constraint in Constraint Manager. This field is available only if the *Fixed*

© 2020 All Rights Reserved.

Die option is not selected. The values you specify in this field are retained when you restart block placer.

If you want to specify the area and aspect ratio of the chip, select the *Area and Aspect Ratio* option button. Selecting the *Area and Aspect Ratio* option button changes the form to display the following options:

- *PRBoundary Area Range* — Specify the minimum and maximum area for the design boundary. The initial value is populated from the values specified in the areaBoundary constraint in Constraint Manager. This field is available only if the *Fixed Die* option is not selected.

  The area range is calculated from the boundaryArea constraint as follows:

  minimum area = minimum width * minimum height

  maximum area = maximum width * maximum height

  The values you specify in this field are retained when you restart block placer.

- *PRBoundary Aspect Ratio (W/H) Range* — Specify the minimum and maximum aspect ratio for the design boundary. The initial value is populated from the values specified in the areaBoundary constraint in Constraint Manager.This field is available only if the *Fixed Die* option is not selected.

  The aspect ratio range is calculated from the boundaryArea constraint as follows:

  minimum aspect ratio = minimum width / maximum height

  maximum aspect ratio = maximum width / minimum height

  The values you specify in this field are retained when you restart block placer.

Cost Order

Block placer uses the cost order field while minimizing cost. The cost with the highest priority is minimized first. The following table lists the default order from highest to lowest priority and describes each priority.

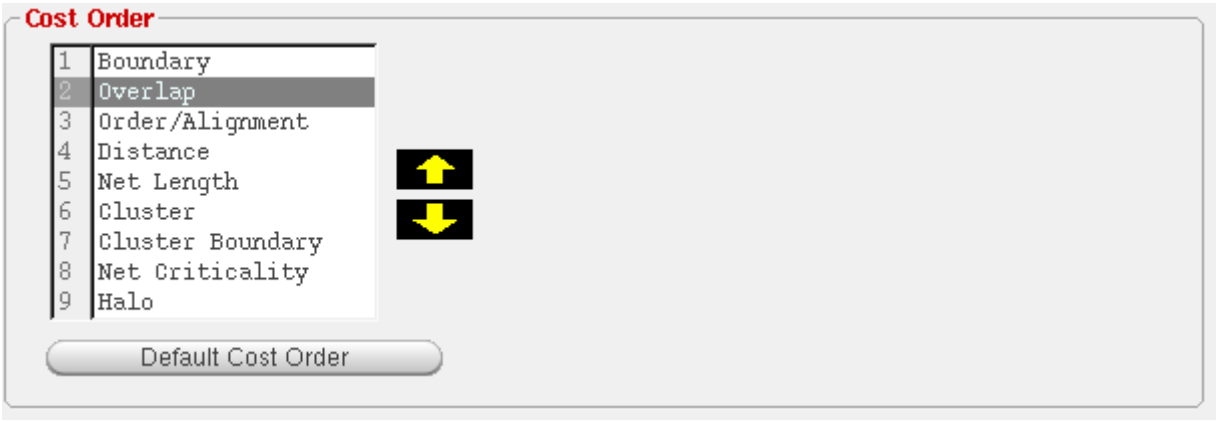
Priority	Description
Overlap	This cost is used to minimize overlap between the blocks placed by block placer. If block placer is unable to place blocks without overlap, the information is displayed in the log file and the CIW.

© 2020 All Rights Reserved.

Priority	Description
Boundary	This cost is used to place all the blocks within the design boundary. If the design core area is insufficient to accommodate all the blocks, block placer will generate a solution with some overlap.
Order	This cost is used by block placer to align the blocks.
Distance	When this cost is used, block placer ensures that the minimum distance between the blocks and minimum distance between boundary and instance is maintained.
Net Length	This cost is used by block placer to minimize net length. The net length is calculated based on the steiner net length estimation.
Cluster	This cost is used by block placer to place together all the blocks belonging to a particular cluster.
Cluster Boundary	When this cost is used, block placer places all the instances belonging to the cluster (corresponding to the cluster boundary) within the cluster boundary. Block placer treats the cluster boundary as inclusive cluster boundary i.e. it can contain instances that do not belong to the cluster.
Net Criticality	This cost ensures proximity of blocks connected by nets of high criticality. It also takes into account the connectivity between the placed, fixed, and locked pins.
Halo	This cost is used to specify that no block should be placed on the placement halo. Halos are treated as placement blockages. If there are no area halos around the blocks and there is no minimum distance constraint, then the blocks are abutted.

© 2020 All Rights Reserved.

You can modify the cost order priority by using the arrow keys. You can also reload the default cost order priority by clicking the *Default Cost Order* button as shown in the graphic.



Options

The Options section offers the following options:

- *Default Minimum Distance between Instances* — You can specify the default minimum distance between the instances in this field by specifying the value in the *User Defined* field. You can also override this value for any instance pair through the constraints tab (described later). The default value, which is used when you select the *Pins per Side* option, is calculated by using the following formula:

  spacing = [total number of pins/number of sides of block] X [max of `minWidth` of layer across all layer purposes + max of `minSpacing` across all layer purposes]

  The value you specify in this field is retained when you restart block placer.

- *Optimize Top-Level Pins* — Select this option if you want block placer to optimize the top-level pins after block placement. However, the soft block pins will not be optimized. The value you specify in this field is retained when you restart block placer.

  When block placer is run on variable die mode, the top-level pins are made floating. However, if the *Optimize Top-Level Pins* option is selected, the pins are placed accordingly.

  Note: When the *Optimize Top-Level Pins* option is selected, Block placer will not consider the top-level pins with *Placed* status as blockages. However, if the *Optimize Top-Level Pins* option is not selected, then the top-level pins with *Placed* status are considered as routing blockages and allows the block placer to place blocks and avoid shorts with already placed pins.

© 2020 All Rights Reserved.

- *Effort* — You can specify the effort that block placer should make before arriving at the possible solutions. The minimum allowed value is 1 (least effort) and maximum allowed value is 5 (maximum effort). The default value is 3.
- *Number of Solutions* — Block placer can generate multiple solutions in a single execution. You use this field to specify the number of solutions that you want block placer to generate. The minimum allowed value is 1 and maximum value is 5, which also provides the best results. The default value is 3. By default, block placer loads the best solution. For more information, refer to the Load Solutions section.
- *Enable Multi Threading* — Selecting this option reduces the time taken by the block placer to place a large number of macros.

  Note: In addition to the number of macros, the placement time depends on the design connectivity, constraints, and machine configuration, such as the number of cores and CPUs. If these parameters are kept constant, then enabling multi threading should result in a 2x performance improvement.
- *Enable Congestion Based Refinement* — In Virtuoso Space-based Router, Global Route replaces all opens with global routes and re-routes to reduce congestion. The *Enable Congestion Based* Refinement checkbox is enabled only if congestion data is available. When this field is enabled, the congestion value is taken into account during block placement. You can specify the congestion threshold value in the *Congestion Value* field. For example, if you specify the threshold value as 80, block placer will try to reduce the congestion in all areas in the design that have a congestion above 80 percent.

  Note: If you are placing blocks in the congestion based refinement mode, you should also optimize the top-level pins for optimum results. If you do not optimize the top-level pins while running block placer in the congestion based refinement mode, a message is displayed in the CIW. Fixed pins at the top level are not moved when you optimize the top-level pins.

  Note: After congestion-aware block placement, if you click *Undo*, the congestion and global route data may still remain. In such a situation, run the global route and congestion analysis on the design to retrieve the congestion data.
- *Number of Iterations* — You use the *Number of Iterations* field to specify the maximum number of iterations that can be performed between the congestion-based refinement block placer and the Global Router to bring congestion within the limits. The congestion limit is specified in the *Congestion Value* field. Specifying the number of iterations will improve the quality of congestion based refinement block placer and reach satisfactory congestion level. By default, the *Number of Iterations* option is disabled. This option is only enabled when *Enable Congestion Based* Refinement field is selected.

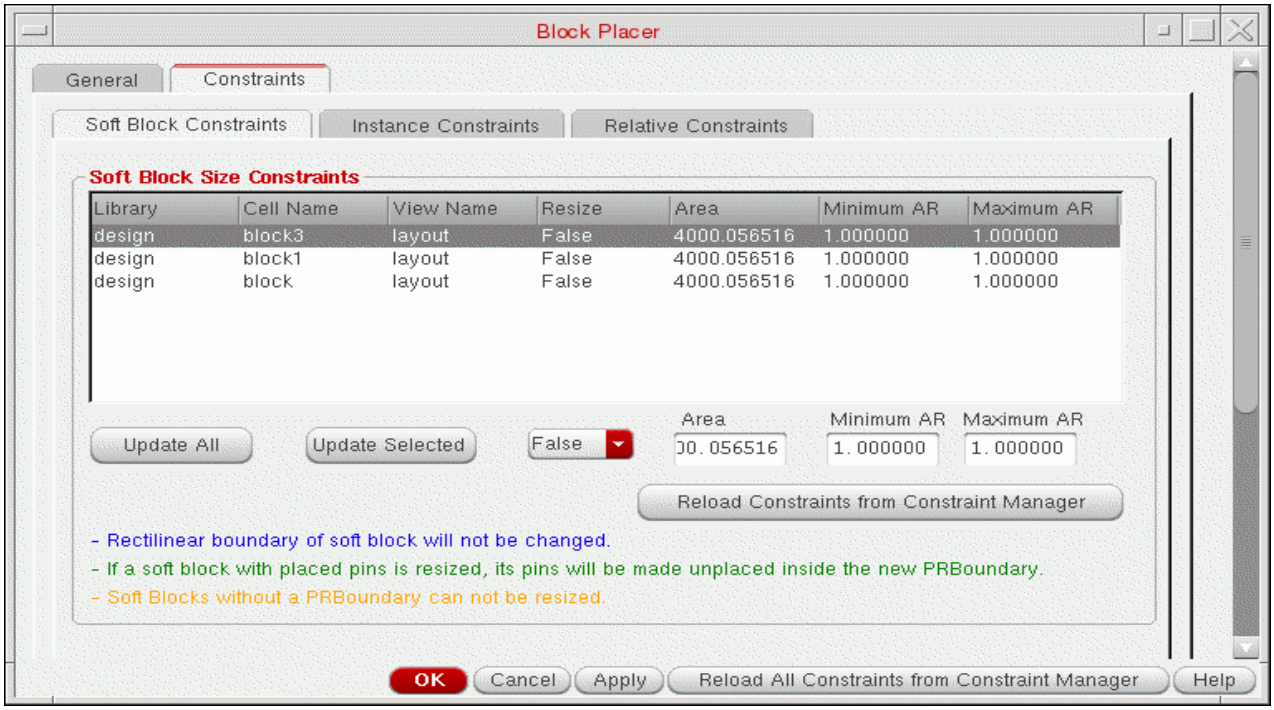
© 2020 All Rights Reserved.

Constraints

You use the Constraints tab of the Block Placer form to specify the constraints that you want block placer to use to converge on a solution. As described previously, the initial values in the Constraints tab are populated from the values specified in the Constraint Manager. You use the Constraints tab to modify the displayed values. However, the values you specify must not conflict with the constraint values in the Constraint Manager. In addition, you can add constraints in this tab.

The Constraints tab has the following sub-tabs, as shown in the graphic:

- Soft Block Constraints
- Instance Constraints
- Relative Constraints



Soft Block Constraints

The table in the *Soft Block Size Constraints* section lists all the soft blocks — rectangular and rectilinear — in the design cellview. The various fields in the table are:

- *Library, Cell Name, and View Name* — These are read-only fields that depict the library, cell name and view name of the soft block.

© 2020 All Rights Reserved.

- *Resize* — The value in the Resize field for a soft block indicates if block placer can modify the boundary of the soft block. The default value is False.

  If you modify the *Resize* field for a rectilinear soft block to *True*, block placer resizes the soft block, converts it to a rectangular soft block, and displays an appropriate message for soft block resize in the log file.

- *Area, Minimum AR,* and *Maximum AR* — You use these fields to specify the area, minimum aspect ratio, and maximum aspect ratio for a soft block. Block placer uses the aspect ratio range to resize a block only if the *Resize* field is set to *True*.

  If the *Area, Minimum AR,* and *Maximum AR* fields for a soft block are empty, it means no corresponding constraints exist. In this case, block placer resizes the soft block without any aspect ratio constraint to optimize space.

To specify the side constraints for soft blocks, perform the following steps:

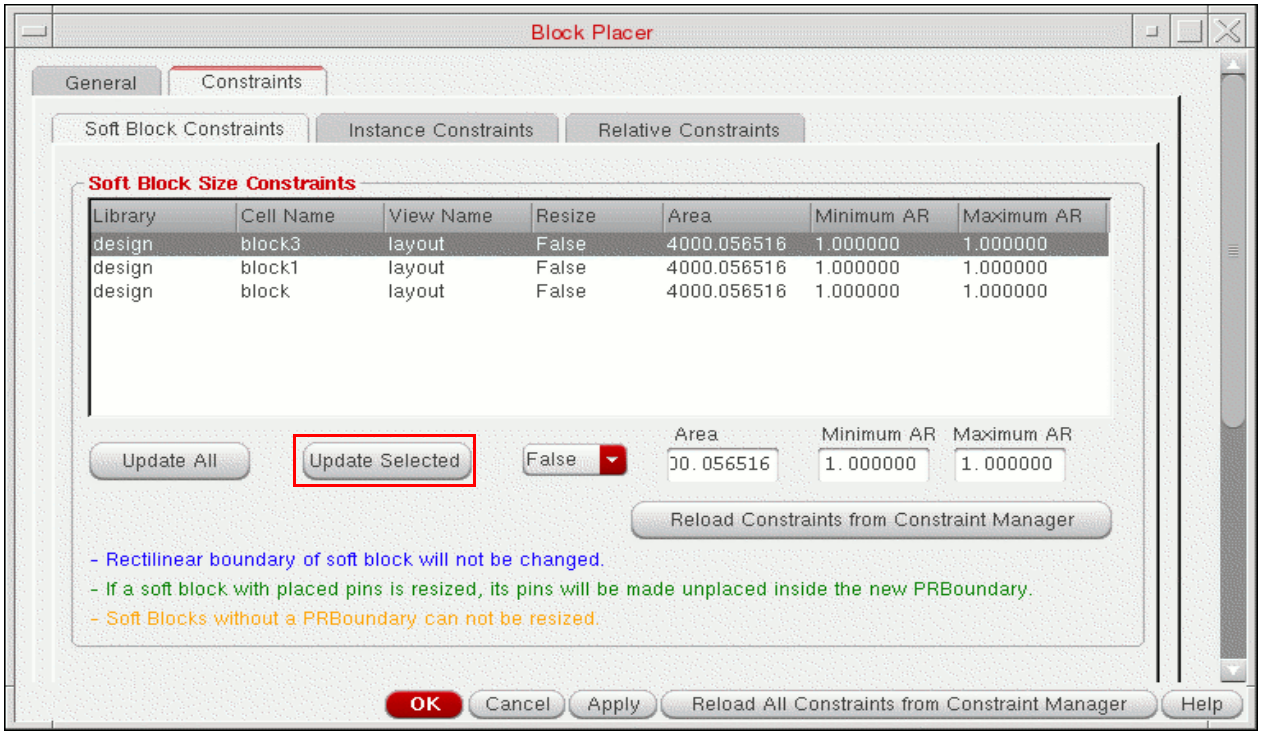
1. Select the row for the soft block that you want to modify.

   The *Area, Minimum AR,* and *Maximum AR* fields below the table show the values for the specified soft block.

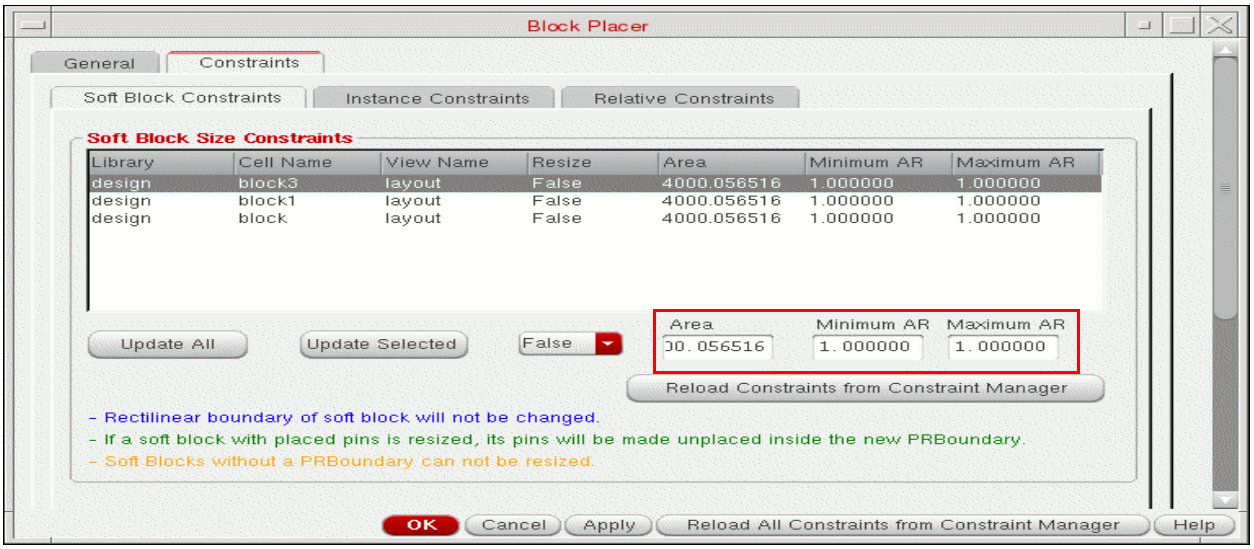
2. Modify the values in the *Area, Minimum AR,* and *Maximum AR* fields. The valid values are positive numbers, AS IS, and empty field. If a field value is AS IS, then the corresponding value in the table is not updated.

© 2020 All Rights Reserved.

3. Click the *Update Selected* button to apply the changes to the selected soft block as shown in the graphic.



You can also update multiple soft blocks simultaneously by selecting the corresponding rows. If you select multiple rows, the *Area, Minimum AR,* and *Maximum AR* fields display the values if they are same for all the selected soft blocks, else it displays AS IS as shown in the graphic.



© 2020 All Rights Reserved.

To update multiple rows, select the rows and perform the steps 2 and 3 given above. When you click the *Update Selected* button, all the selected rows are updated with the new values. To update all rows, click *Update All*.

You can modify the *Area, Minimum AR,* and *Maximum AR* values within the limits specified using the *minWidth*, *maxWidth*, *minHeight*, and *maxHeight* parameters of the Boundary Area constraint in Constraint Manager. If the values you specify violates the Boundary Area constraint, a warning message is displayed in the CIW.

For example, consider a soft block that has the following parameters of the Boundary Area constraint specified in the Constraint Manager:

```
minWidth = 50
maxWidth = 200
minHeight = 20
maxHeight = 100
```

In the above example, the minimum aspect ratio = *MinWidth*/*MaxHeight* = 50/100 = 0.5 and the maximum aspect ratio = *MaxWidth*/*MinHeight* = 200/20 = 10.

If the values you specify in the *Minimum AR* and *Maximum AR* fields do not fall with in the minimum and maximum aspect ratio as calculated above, the values specified in the Constraint Manager are loaded in the *Minimum AR* and *Maximum AR* fields.

Note: You can also reload the constraint values in the Constraint Manager by using the *Reload Constraints from Constraint Manager* button. Only the constraints specified in the Constraint Manager are reloaded. The other additional constraints added in the form remain.

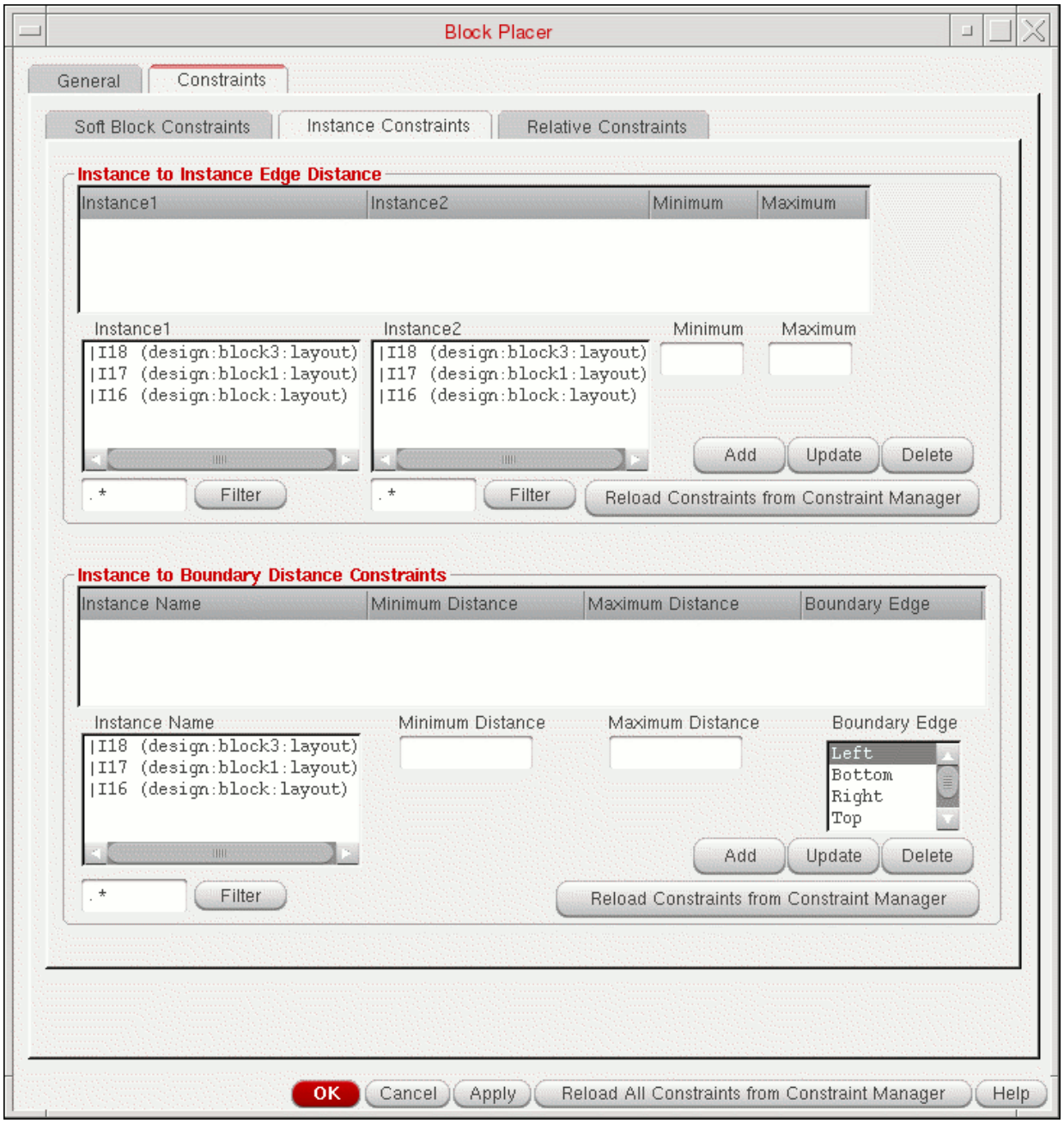
Instance Constraints

The Instance Constraints tab has the following sections as shown in the graphic:

- Instance to Instance Edge Distance

© 2020 All Rights Reserved.

- Instance to Boundary Distance Constraints



Instance to Instance Edge Distance

You use the *Instance To Instance Edge Distance* section to specify the distance constraints between any two instances. This section has the following fields:

© 2020 All Rights Reserved.

- *Instance to Instance Distance* — This table is populated from the Instance to Instance Distance constraints present in the Constraint Manager.
- *Instance1 and Instance2* — You use the Instance1 and Instance2 fields to specify the instance names on which the minimum and maximum distance constraint will be applied.

  Instance1 and *Instance2* fields list all the instances in the design.
- *Minimum and Maximum* — The *Minimum* and *Maximum* fields specify the edge-to-edge distance values.

If a distance constraint exists in the Constraint Manager, you can modify the minimum and maximum distance values between two instances within the limits specified in the Constraint Manager.

For example, if the distance constraint between two instances Instance1 and Instance2 in the Constraint Manager are specified as 20 and 60, you can specify the minimum and maximum distance values between 20 and 60. This means that the minimum distance that you specify should not be less than 20 units and the maximum distance should not be more than 60 units. The table below lists some of the possible incorrect values.

	Minimum	Maximum	Correct/ Incorrect
Value in Constraint Manager	20	60	
Modified Values	25	45	✓
Modified Values	15	50	✗
Modified Values	25	65	✗

If the values you specify violates the distance constraint in the Constraint Manager, a warning message is displayed in the CIW. In addition, if you specify only the minimum value, the maximum distance will be equal to the maximum width and height of the chip. Similarly, if you specify only the maximum value, the minimum distance will be zero.

Note: If an incorrect value is entered, the values specified in the Constraint Manager is loaded.

To add a constraint, perform the following steps:

1. Select the instances Instance1 and Instance2 between which you want to specify the distance constraint.

© 2020 All Rights Reserved.

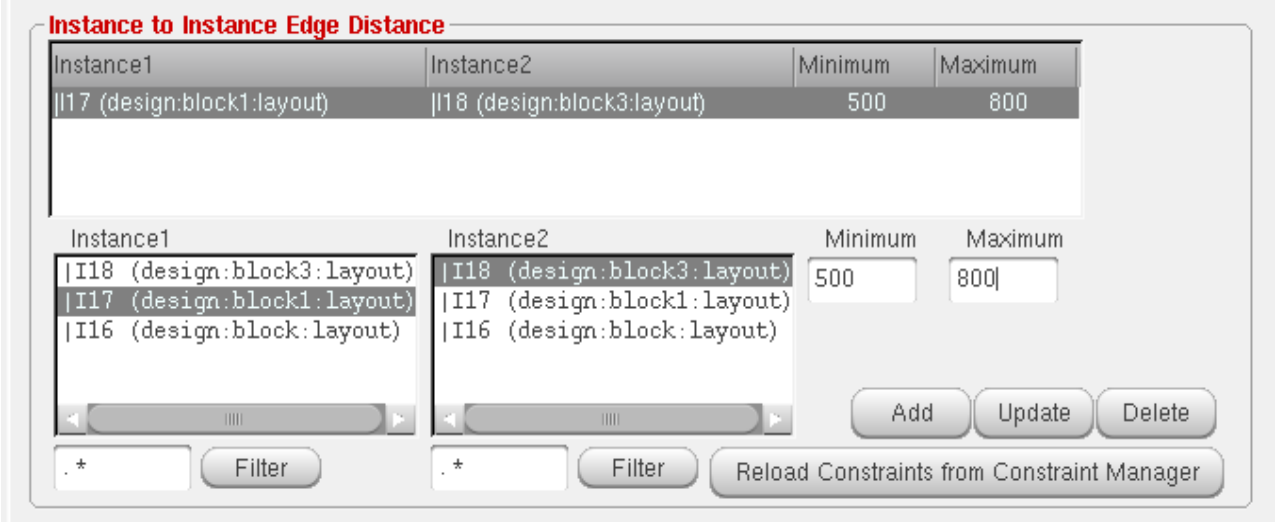
2. Specify the minimum and the maximum distance in the *Minimum* and *Maximum* fields.
3. Click the *Add* button.

   The new constraint gets added. However, if the *Minimum* and *Maximum* fields are empty, the new constraint is not added and an appropriate warning message will be displayed.

To modify a constraint, perform the following steps:

1. To update a constraint, you need to first select a constraint. However, if a constraint is not selected, an appropriate warning message will be displayed. Select the row that lists the constraint in the table.

   The *Minimum* and *Maximum* fields display the minimum and maximum distance values as shown in the graphic.

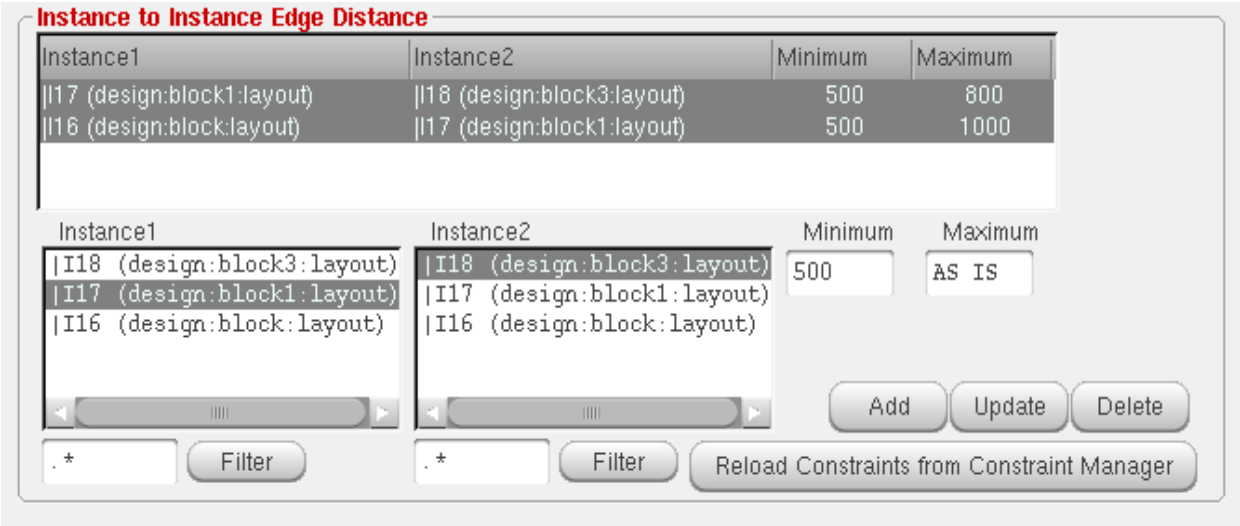


2. Enter the new values for the minimum and maximum distance in the *Minimum* and *Maximum* fields.
3. Click the *Update* button.

   The values are updated.

© 2020
All Rights Reserved.

You can also select multiple constraints and update them simultaneously. If you select multiple rows, *Minimum* and *Maximum* fields display the values if they are same, else it displays *AS IS* as shown in the graphic below.



To delete a constraint, perform the following steps:

1. Select the row(s) that lists the constraint in the table
2. Click the *Delete* button.

   You can also select multiple constraints and delete them simultaneously.

Note: Constraints that are populated from the Constraints Manager cannot be deleted.

You can also reload the constraint values in the Constraint Manager by using the *Reload Constraints from Constraint Manager* button. Only the constraints coming from the Constraint Manager are reloaded. The other additional constraints added in the GUI remain.

Instance to Boundary Distance Constraints

You use the *Instance To Boundary Distance Constraints* section to specify the distance constraints between an instance and the design boundary. This section has the following fields:

- *Instance to Boundary Distance Table* — This table lists all the Instance to Boundary Distance constraints present in the Constraint Manager.
- *Instance Name* — The *Instance Name* field shows the names of all the instances in the design.

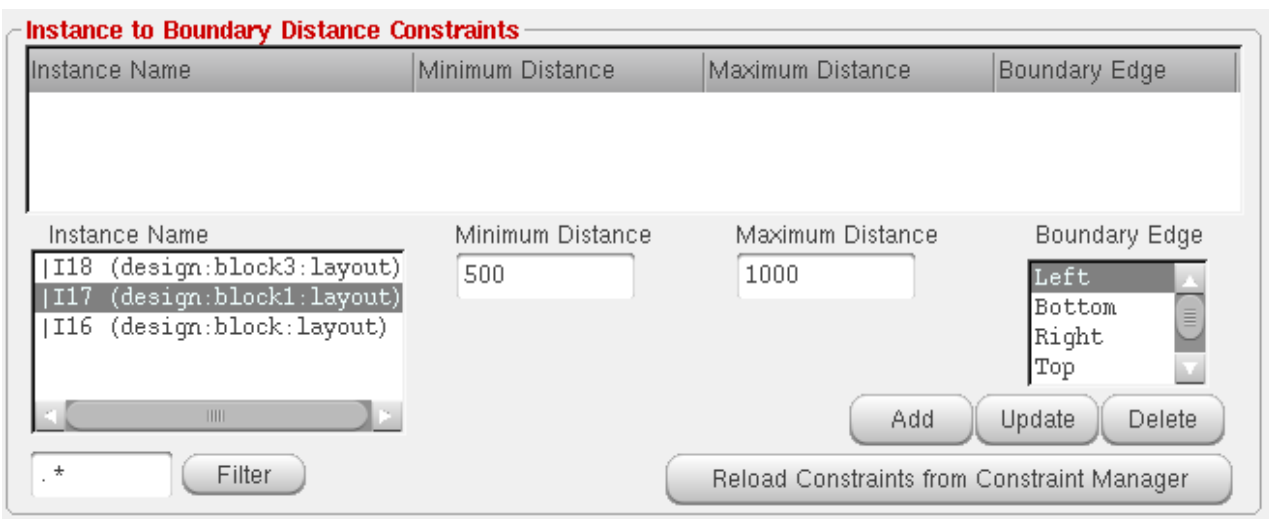
© 2020 All Rights Reserved.

- *Minimum Distance and Maximum Distance* — The *Minimum Distance* and *Maximum Distance* fields show the minimum and maximum distance of the instance from the design boundary. The initial values of *Minimum Distance* and *Maximum Distance* are populated from the *Distance* constraint in the Constraint Manager.
- *Boundary Edge* — The *Boundary Edge* field shows the edge of the boundary to which the distance is calculated. The valid values are *Left*, *Right*, *Top*, and *Bottom*.
- *Filter* — The *Filter* field enables you to filter the instances in the *Instance Name* list box.

You can add, modify, and delete the values of the instance to boundary constraints using this section. You can also reload the values in the Constraint Manager by using the *Reload Constraints from Constraint Manager* button. Only the constraints populated from the Constraint Manager are reloaded. The other additional constraints added in the form remain.

To add a constraint, perform the following steps:

1. Select the instance from the *Instance Name* list box.
2. Specify the minimum and maximum distance in the *Minimum Distance* and *Maximum Distance* fields as shown in the graphic.



3. Specify the edge to which the constraint should be applied in the *Boundary Edge* list box.
4. Click the *Add* button.

   The constraint is added. However, if the *Minimum* and *Maximum* fields are empty, the new constraint is not added and an appropriate warning message will be displayed.

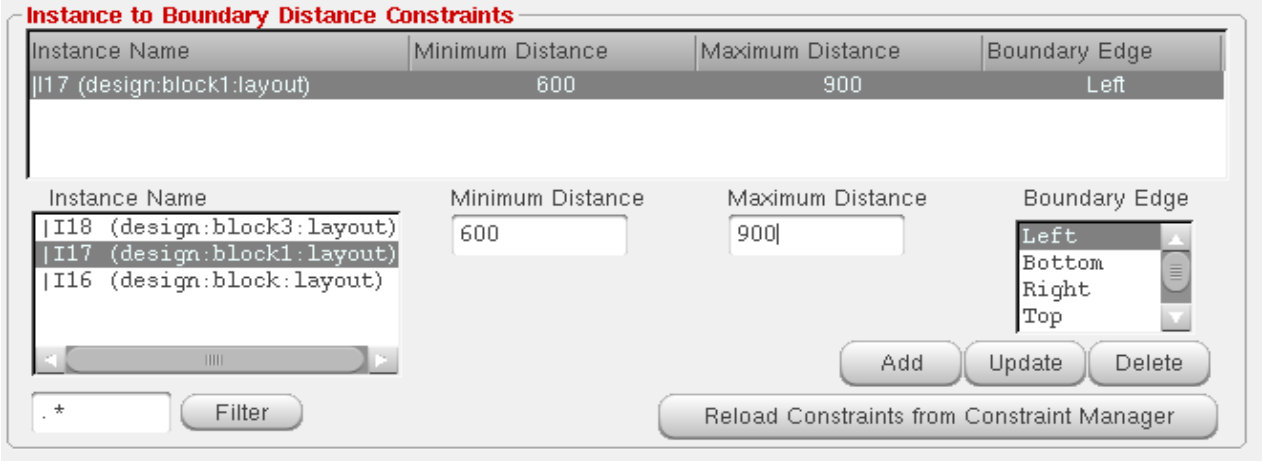
© 2020 All Rights Reserved.

To modify a constraint, perform the following steps:

1. To update a constraint, you need to first select a constraint. However, if a constraint is not selected, an appropriate warning message will be displayed.Select the instance from the table.

   The minimum and maximum distance from the design boundary, if specified in the Constraint Manager, is shown in the *Minimum Distance* and *Maximum Distance* fields.

2. Type the minimum and maximum distance in the *Minimum Distance* and *Maximum Distance* fields as shown in the graphic.

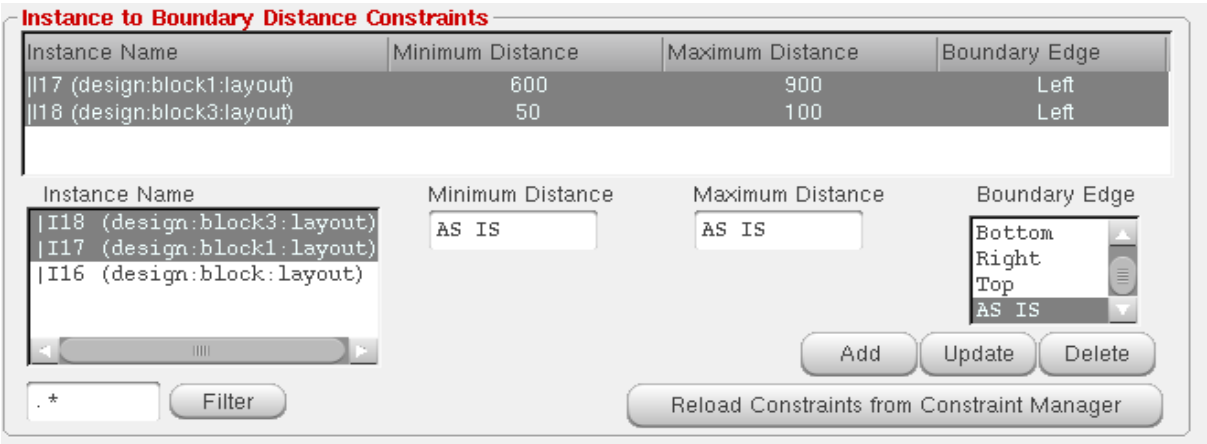


3. Click the *Update* button.

   The instance is updated.

© 2020
All Rights Reserved.

You can also select multiple constraints and update them simultaneously. If you select multiple rows, *Minimum Distance*, *Maximum Distance*, and *Boundary Edge* fields display the values if they are same, else it displays AS IS as shown in the graphic.



To delete a constraint, perform the following steps:

1. Select the constraint(s) from the table
2. Click the *Delete* button.

   You can also select multiple constraints and delete them simultaneously.

Note: You cannot edit the constraints specified in Constraint Manager.

Relative Constraints

Block placer respects the order and alignment constraints for blocks specified in the table in the *Order/Alignment* section.

The order and alignment constraints use the following parameters:

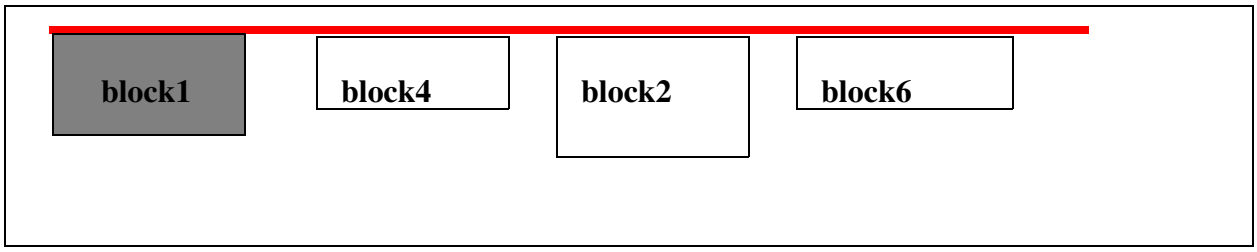
- Reference instance — The instance that is used as the reference while determining the alignment side.
- Target instance — The instance that is aligned with the reference instance.
- Alignment side — The edge of the reference instance on which the target instance will be aligned.
- Relative position — The relative position of the target instance with respect to the reference instance.

© 2020 All Rights Reserved.

For example, consider the following order and alignment constraint:

Parameter	Value
Reference instance	block1
Target instances	block4, block2, block6
Alignment side	Top
Relative position	Right

In this case, the placement of the blocks will be as follows:



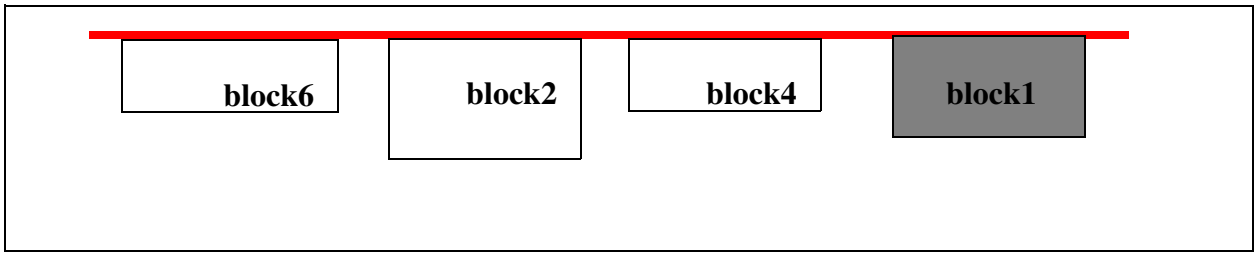
This is because block1 is the reference block and the top edge of block1 is aligned with the top edge of block4, block2, and block6. The target blocks are placed in the specified order on the right side of the reference block because the relative position parameter is set to right.

Consider another constraint as given in the following table:

Parameter	Value
Reference instance	block1
Target instances	block4, block2, block6
Alignment side	Top
Relative position	Left

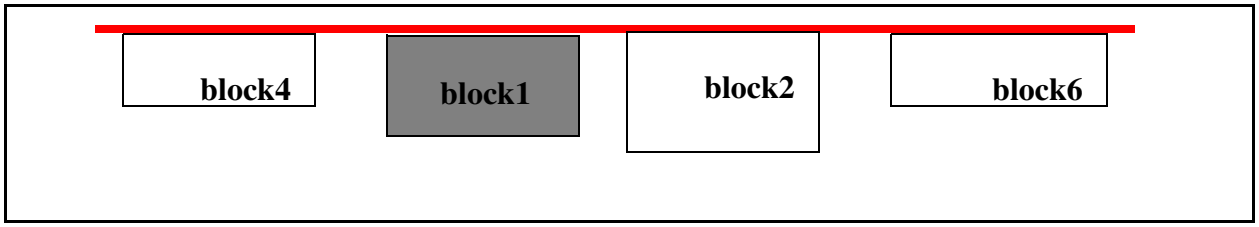
© 2020 All Rights Reserved.

In this case, the placement of the blocks will be as follows:



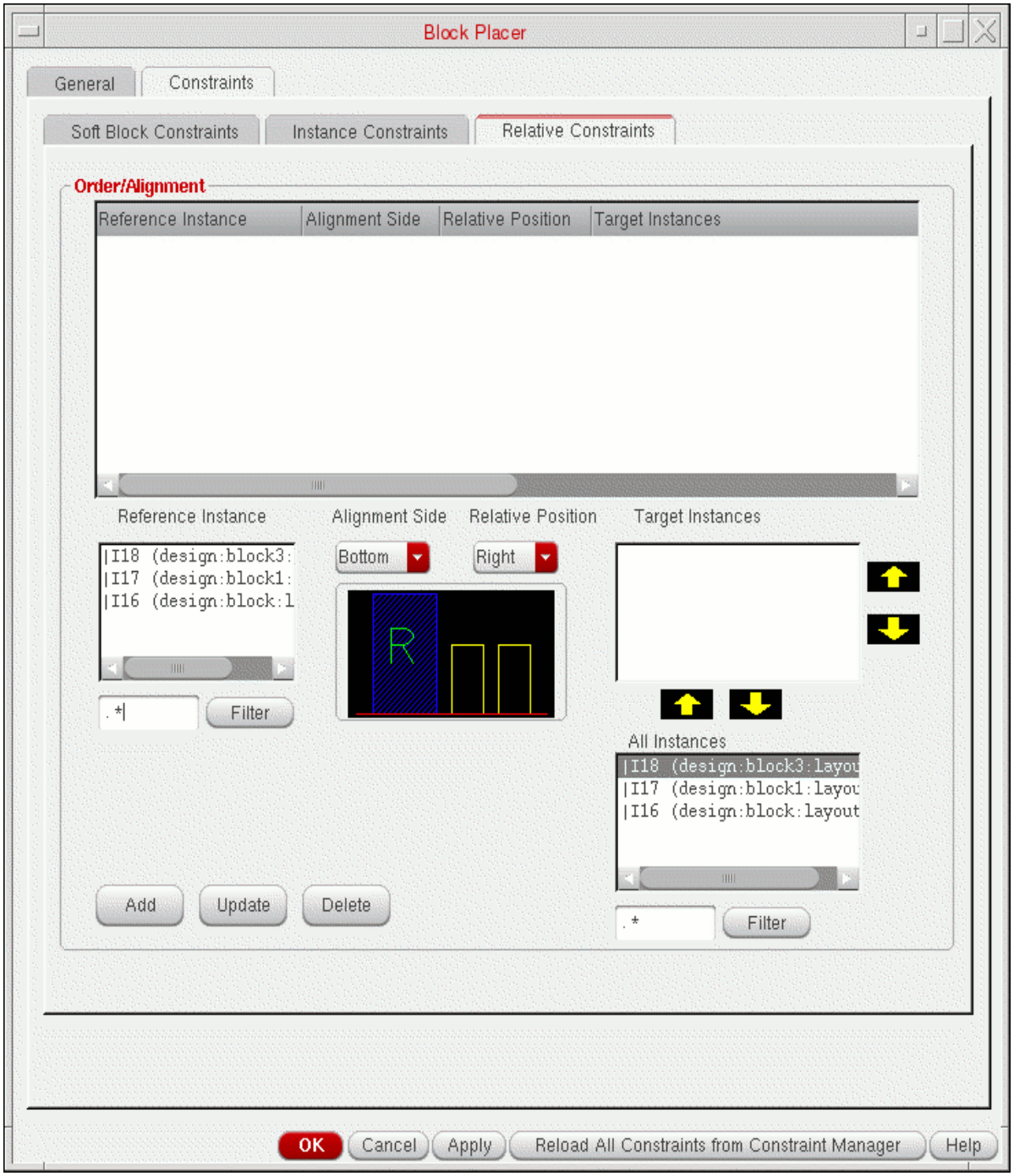
In this case, the target blocks are placed in the specified order on the left side of the reference block. This is because the relative position parameter is set to left.

However, if you do not specify any value for the relative position parameter, the target blocks are placed either on the left or right of the reference block depending on the available space .



© 2020 All Rights Reserved.

The *Relative Constraints* tab is shown in the graphic.



The various controls in this tab are as follows:

© 2020
All Rights Reserved.

- *Order/Alignment* — This table is initially populated with all the existing *Alignment* constraints present in the Constraint Manager.
- *Reference Instance* — Lists all the instances in the cellview. You can select only one instance from this field to add and update a constraint. If you select a row in the constraints table, the corresponding reference instance value is highlighted in this field.
- *Alignment Side* — Lists the possible edges of an instance to be used for alignment. The possible values are “Left”, “Right”, “Top” and “Bottom”. If you select a row in the constraints table, the corresponding alignment side is highlighted in this field.
- *Relative Position* — Lists the possible relative positions of the target instance with respect to the reference instance. If you do not want to specify a relative position, you can specify the values as None.

  If you select a row in the table, the corresponding relative position value is highlighted in the *Relative Position* field. However, if you select multiple rows, the *Update* field is grayed out.

  The following table lists the various alignment sides and the corresponding possible relative positions.

Alignment Side	Possible Relative Position
Left	Top, Bottom, None
Right	Top, Bottom, None
Top	Left, Right, None
Bottom	Left, Right, None

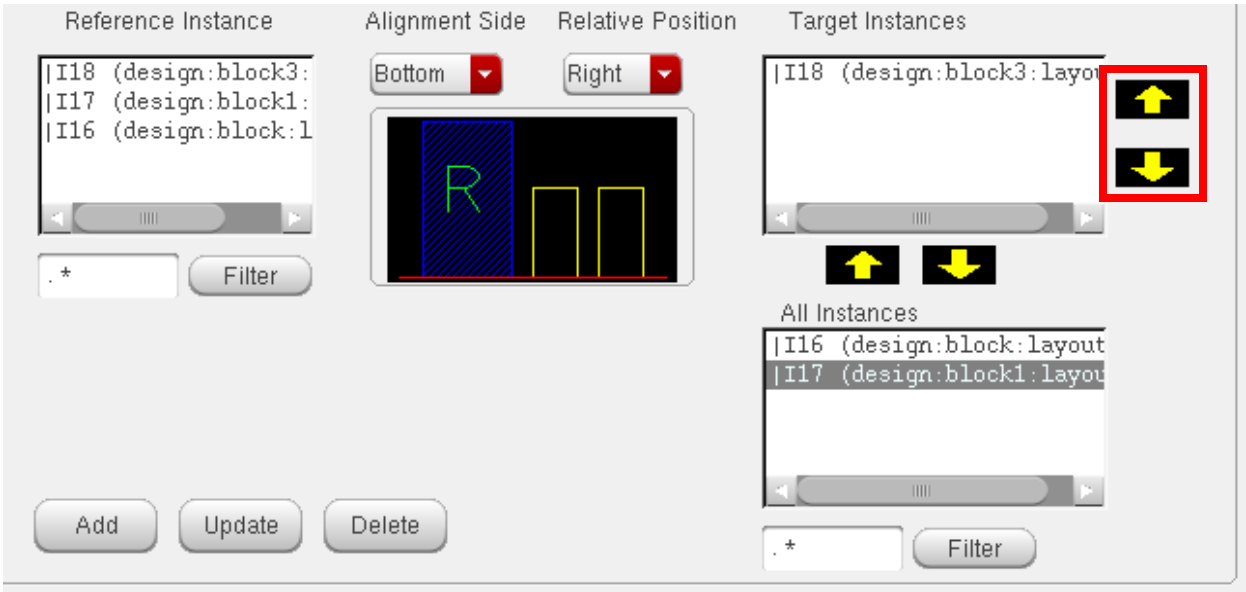
  For example, if you specify the alignment side as Left, then the target instances can be placed either on top or bottom of the reference instance. You can specify top or bottom by using the *Relative Position* field.

  The graphic below the *Alignment Side* and *Relative Position* fields shows how the target and the reference instances will be placed based on the specified settings.

- *Target Instances* — Lists all the blocks that will be aligned with the reference block and the order in which they will be aligned. You can specify the order in which the blocks

© 2020 All Rights Reserved.

should be aligned by using the arrow buttons on the right of this field as shown in the graphic.



Clicking the up arrow button places the blocks closer to the referenced instance. Similarly, clicking the down arrow button places the blocks away from the referenced instance.

You can also remove the instances from the constraint by moving them to the *All Instances* box. You can also add target instances by selecting an instance from the *All Instances* box and using the up arrow button placed above the *All Instances* box.

If you select a row in the table, the corresponding target instances are highlighted in the *Target Instances* field. However, if you select multiple rows, *Update* field is grayed out.

- All Instances — Lists all the instances in the design except the fixed and locked instances. You can select multiple instances in this field and move them to the *Target Instances* field by using the up arrow button.

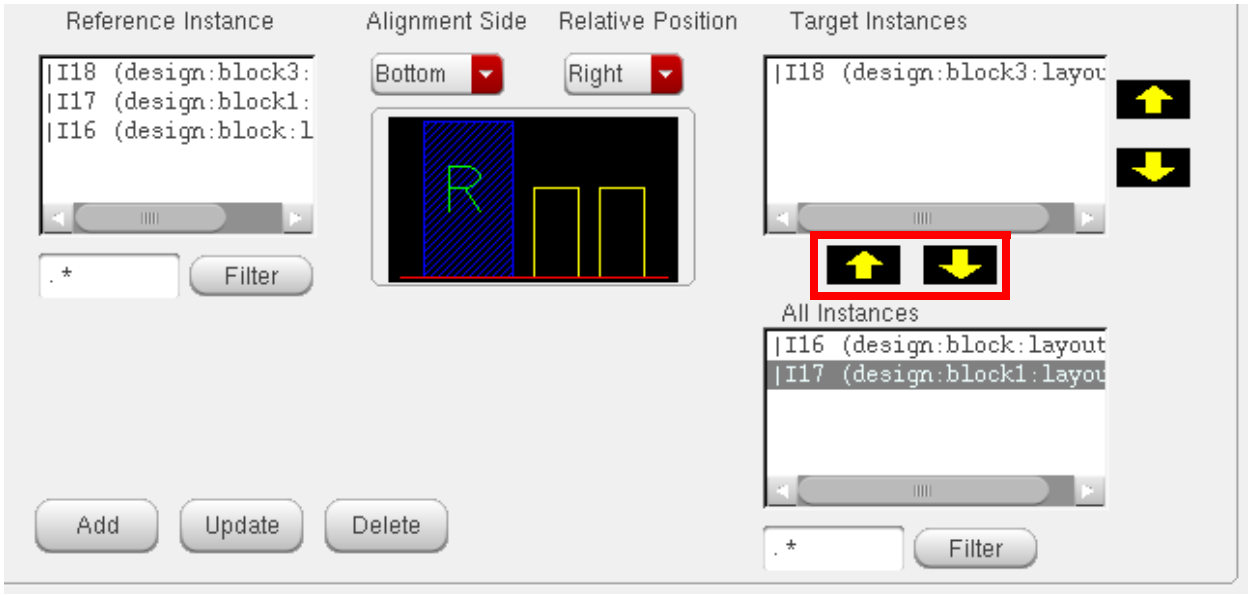
You can add, modify, and delete constraints using this form.

To add a constraint, perform the following steps:

1. Select the reference instance from the *Reference Instance* field.
2. Specify the side in the *Alignment Side* drop down list.
3. Specify the position in the *Relative Position* drop down list.

© 2020 All Rights Reserved.

4. Specify the target instance by moving an instance from the *All Instances* list box to the *Target Instance* list box as shown in the graphic.



5. Adjust the order of Target Instances using the arrow keys next to the field.
6. Click *Add*.

   The constraint is added.

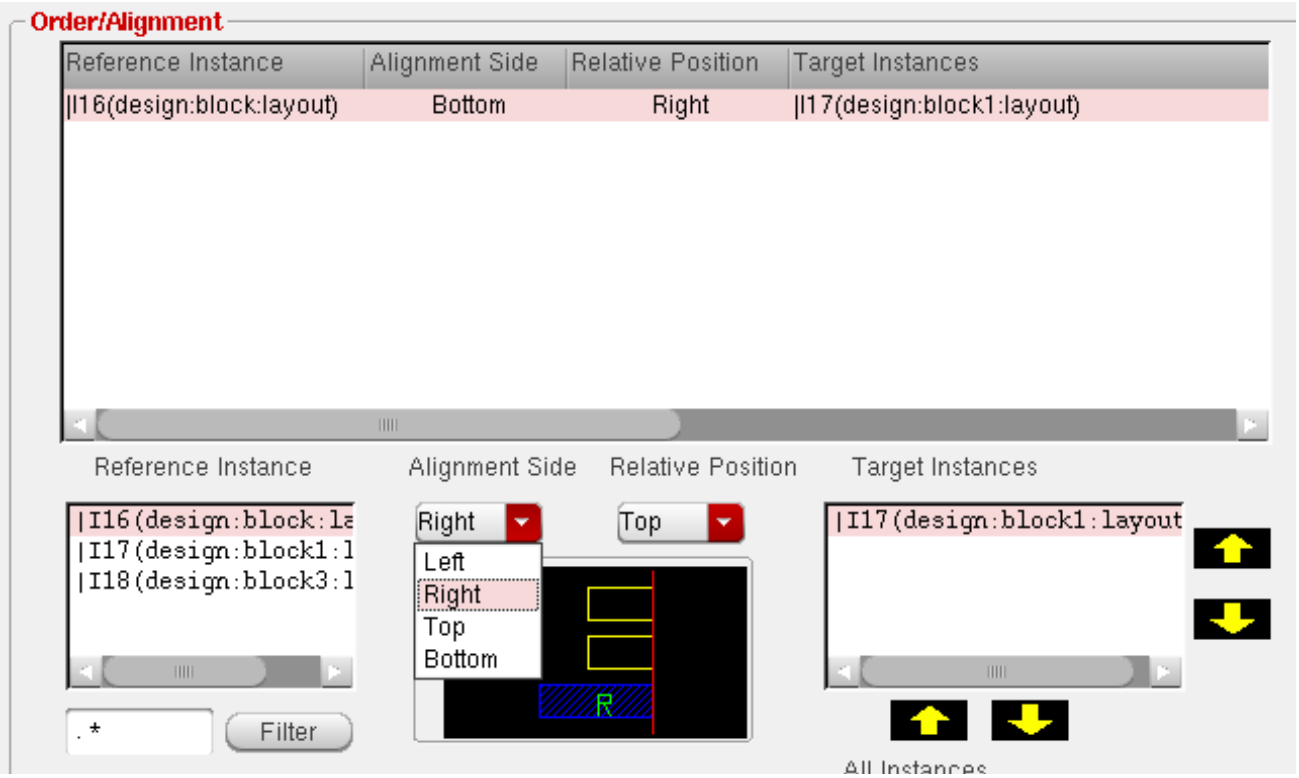
To update a constraint, perform the following steps:

1. Select the constraint in the constraints table.

   The corresponding reference instance, side, position, and target instances are displayed in the *Reference Instance*, *Alignment Side*, *Relative Position*, and *Target Instance* fields.

© 2020
All Rights Reserved.

2. Modify the values in the *Alignment Side*, *Relative Position*, and *Target Instance* list box as shown in the graphic.



3. Click *Update*.

The constraint is updated. While updating a constraint, if you specify a value that violates the values in the Constraint Manager, a warning message is displayed in the CIW and the previous values are restored.

Note: If multiple constraints in the table are selected, the Update field is grayed out.

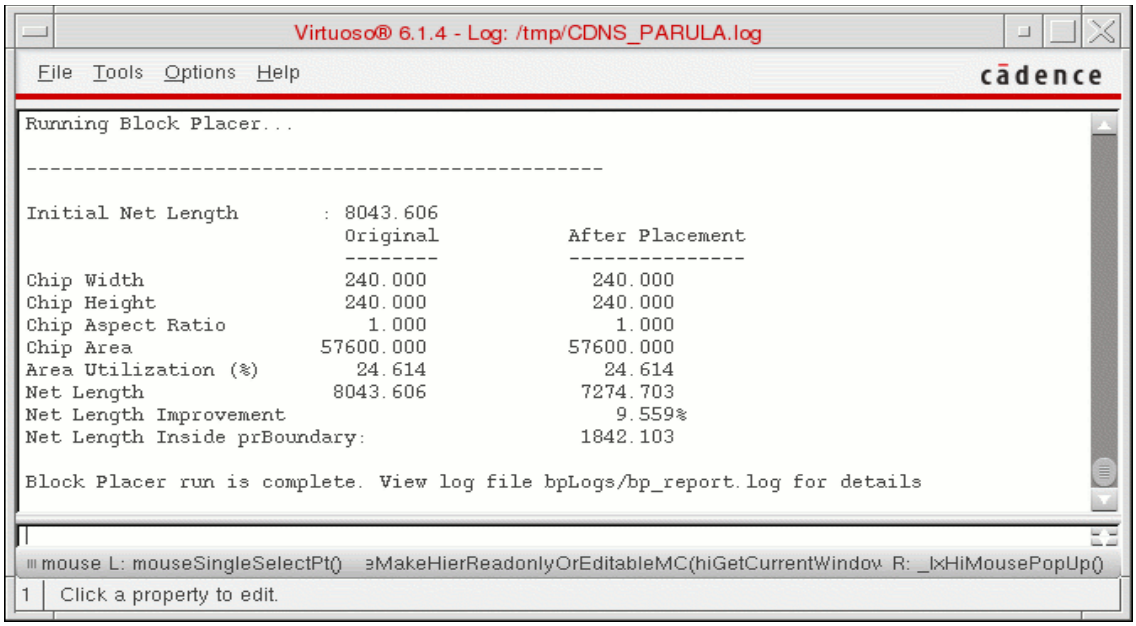
To delete a constraint, perform the following steps:

1. Select the constraint(s) in the constraints table
2. Click *Delete*.

The constraint is deleted.

© 2020 All Rights Reserved.

After block placer is run on the entire design, click *OK* in the *Block Placer* form. A summary report of the placement of all hard and soft blocks in the design is displayed in the CIW as shown in the following figure.



Note: After block placer is run, the blocks in the design automatically fits in the workspace.

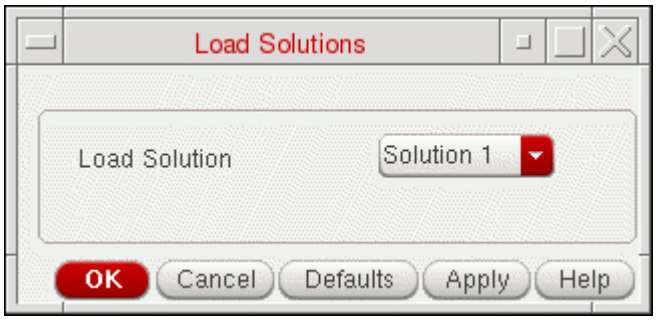
Load Solutions

By default, block placer loads the best solution among the solutions it generates. You can also load the other solutions that block placer generates. To load a solution, click *Floorplan -> Block Place -> Load Solution* command. The *Load Solutions* form is displayed as shown in the following figure.

© 2020 All Rights Reserved.

Important

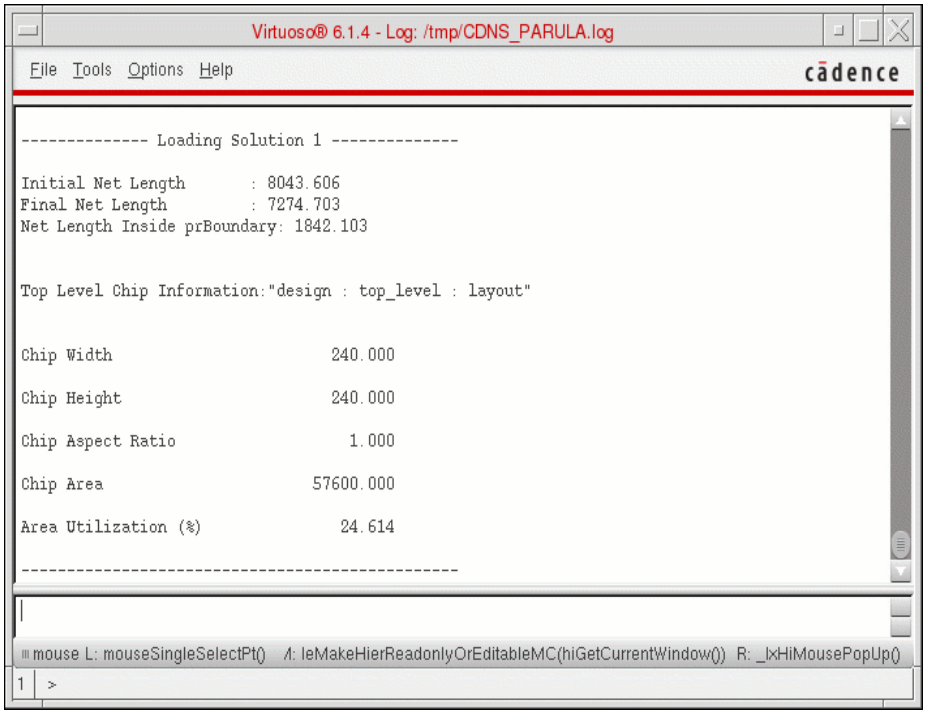
(ICADVM20.1 Only) In Layout EXL, choose *Plan – Block Planning – Load Solutions*.



The load solution command reads the number of solutions present for a design in the directory specified by BPDIR variable (or *bpLogs* directory within the testcase directory) and lists the number of solutions in the *Load Solution Number* field in the form. You can select a solution number in the field and load it. You can undo the changes made by the new solution by using the *Undo* command. When you undo the changes, block placer restores the previous state.

© 2020 All Rights Reserved.

Once you have selected the solution number, click *OK* to load it. A summary report of loading the solution is displayed in CIW as shown in the following figure.



Note: After the solution is loaded, the design automatically fits in the workspace.

Report Placement Statistics

Use the *Floorplan – Block Place – Report Placement Statistics* option to view the placement statistics report at any time after block placement.

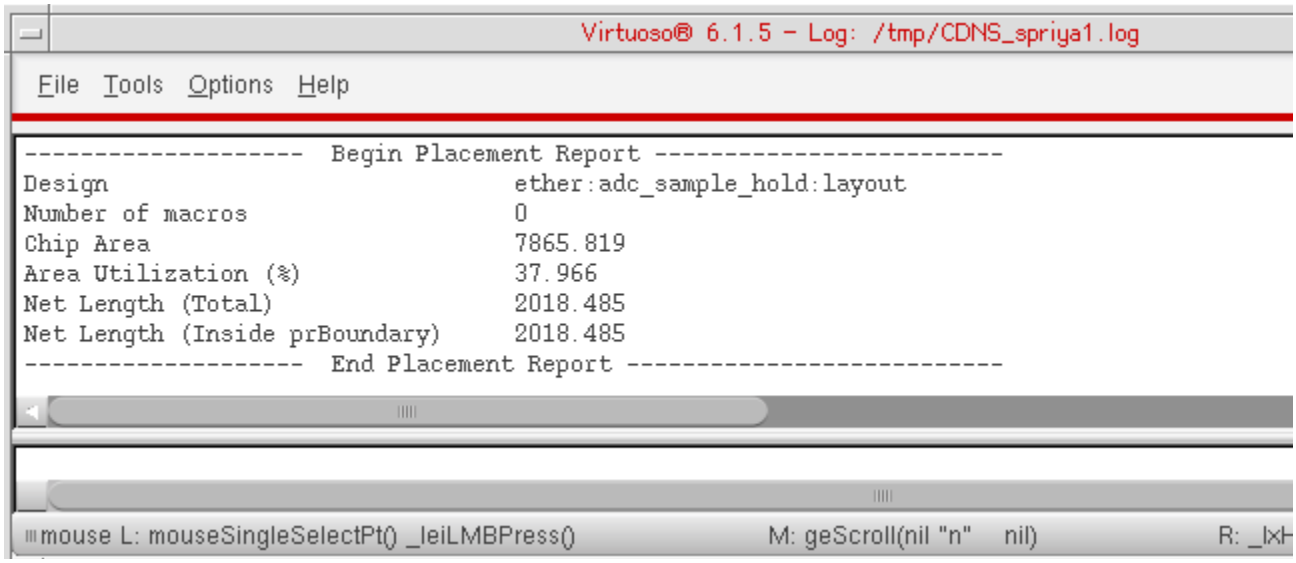
Important

In Layout EXL, choose *Plan – Block Planning – Report Placement Statistics*.

Typically, after running the block placer, you may want to make minor manual adjustments or automated adjustments such as using the Soft Block Update commands to remove overlaps or pulling blocks inside boundary. After making the adjustments, use the *Floorplan > Block*

© 2020 All Rights Reserved.

Place > Report Placement Statistics option to display the placement report in the CIW. You can use the placement report to review the placement statistics and analyze the results.



© 2020 All Rights Reserved.

6

Updating Soft Blocks

You use the *Floorplan – Soft Block Update* commands to adjust soft blocks in the design.

You can use these commands to remove soft block overlaps, snap soft blocks to the routing grid or manufacturing grid, pull soft blocks to the core area, and to push the top-level implementation of power structures to the block level.

The *Floorplan – Soft Block Update* menu provides the following sub-options:

- Push Pre-Routes: Pushes the top-level implementation of power structures to the block level.
- Adjust Blocks: Adjust soft blocks by abutting and pushing the blocks.
- Remove Block Overlap: Remove overlaps in blocks.
- Snap Soft Blocks to Grid: Pulls soft blocks to manufacturing or placement grid.
- Pull Soft Blocks inside PR Boundary: Pulls soft blocks inside the PR boundary.
- Edit Soft Blocks: Lets you modify the attributes of soft blocks in a design.

Push Pre-Routes

During floorplanning, you often need to push the top-level implementation of power structures to the block level. Floorplanner provides the *Push Pre-Routes* command that enables you to do this.

You can use this command to push the following objects to the block level:

- Path
- Path segment
- Rectangle
- Custom and standard vias

© 2020 All Rights Reserved.

- Custom and Standard rows
- Blockages

Important

The *Push Pre-Routes* command can be used for pushing only orthogonal paths, routes, and path segments to the block level.

If the *Power - Ground Net(s)* option is selected in the *Push Into Blocks* form, then the supported objects can be pushed only if the nets corresponding to these objects have the ‘ground’ or ‘power’ signal types. However, if the *Regular (non-Power/Ground) Net(s)* option is selected, then the supported objects can be pushed only if the nets corresponding to these objects have the signal types, such as ‘tieHi’, ‘tieLow’, ‘clock’, and ‘signal’. In addition, the supported shape objects should be on the layers for which the layer function in the technology file is defined as ‘metal’. The *Push Pre-Routes* command pushes the top-level power structures down to one level only.

Depending on whether the soft block instance is fed by the top-level net to which the objects are attached to, the *Push Pre-Routes* command pushes the top-level power structures to the block level as-is or as routing blockages. If the soft block instance is fed by the same top-level net, the net geometry (associated vias and supported shape objects) are pushed as they are. In this case, square pins are created at the intersection of the net geometry and the PRboundary. These pins are created inside the PRboundary.

If the soft block is not fed by the same top-level net, the power structures are pushed as routing blockages.

Power geometries that are close (within the maximum of the minSpacing value) to the PRboundary, but are placed outside the PRboundary, are pushed as routing blockages. Vias that have connectivity with routes are pushed as three routing blockages.

Note: Stand-alone vias that do not have connectivity with any route or path or pathSeg are not pushed into blocks.

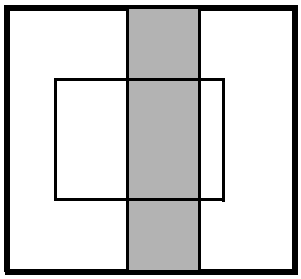
© 2020 All Rights Reserved.

The following table summarizes the behavior.

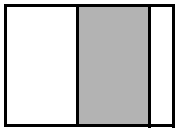
Soft block is fed by the top-level net of any signal type	The net geometry completely overlaps the soft block instance boundary	■ Overlapping portion of the net geometry is removed from the top level as shown in Figure 1. ■ The net geometry is pushed as-is to the block and two square pins are created at the intersection of the net geometry and the PRboundary.
	The net geometry partially overlaps the soft block instance boundary	■ The net geometries are retained at the top level as shown in Figure 2. ■ The net geometries are pushed as they are to the block and two pins are created at the intersection of the net geometry and the PRboundary.
	The net geometry is close (within the maximum of the minSpacing value) to the PRboundary, but is placed outside the PRboundary	■ The net geometries are retained at the top-level. ■ The net geometries are pushed as routing blockages. ■ Each via is pushed as three routing blockages.
Soft block is not fed by the top-level net of any signal type	The net geometry completely or partially overlaps the soft block instance boundary	■ The net geometries are retained at the top level as shown in Figure 3. ■ The net geometries are pushed as routing blockages. In case of a via, three routing blockages are created.
	The net geometry is close (within the maximum of the minSpacing value) to the PRboundary, but is placed outside the PRboundary	■ The net geometries are retained at the top-level. ■ The net geometries are pushed as routing blockage. ■ Each via is pushed as three routing blockages.

© 2020 All Rights Reserved.

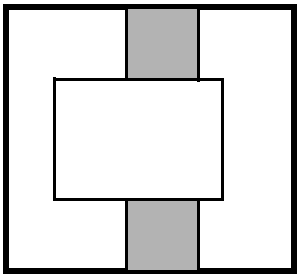
Figure 1



Top level before the push

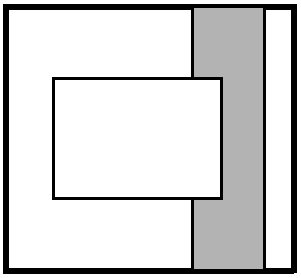


Block after the push

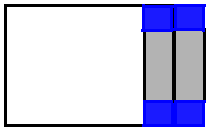


Top level after the push

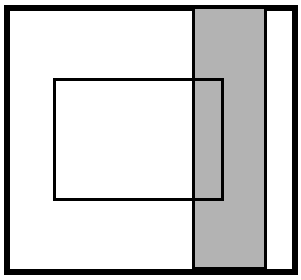
Figure 2



Top level before the push

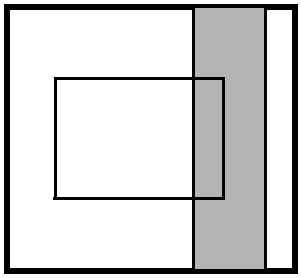


Block after the push



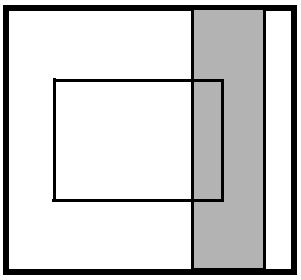
Top level after the push

Figure 3



Top level before the push

Block after the push



Top level after the push

© 2020
All Rights Reserved.

To push the top-level power structures to the block level:

1. Select the soft block to which the overlapping power geometries are to be pushed.
2. Choose *Floorplan – Soft Block Update – Push Pre-Routes*.

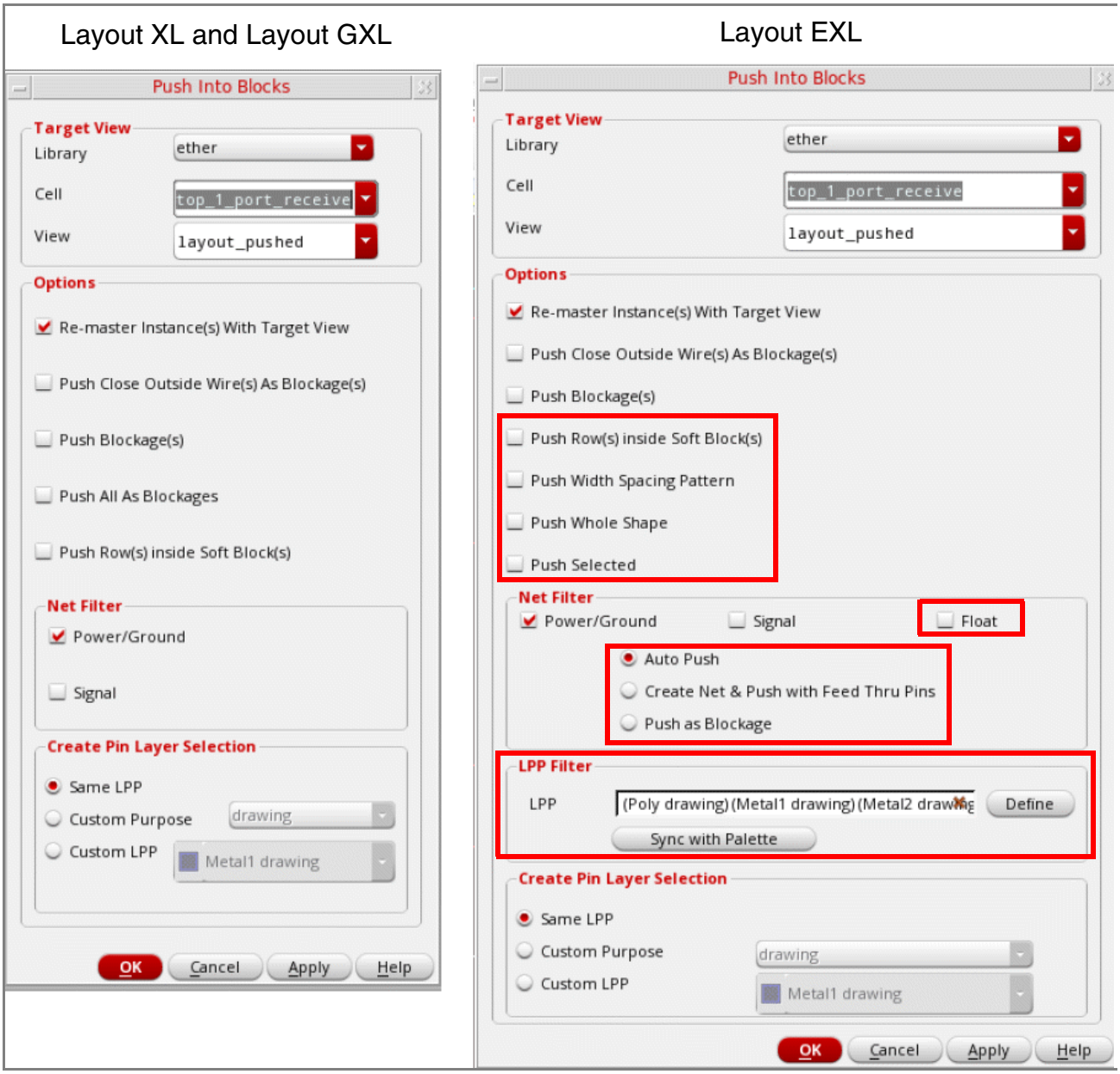
Important

In Layout EXL, choose *Plan – Block Planning – Push Pre-Routes*.

If soft blocks were not selected before invoking the form, select the required soft block instances. The Push Into Blocks form is displayed.

© 2020 All Rights Reserved.

When launched from Layout EXL, the Push Into Blocks form includes additional options, as shown below:



3. Specify the target cellview in the *Target View* section. If the form is invoked after selecting a soft block, the library, cell, and view names are populated by default. You can edit these values based on your requirements.

4. Specify the following settings in the *Options* section:

© 2020
All Rights Reserved.

- *Re-master Instance(s) With Target View*: Re-masters the selected soft block using the target design. This option is useful when the target design and the existing master design are different and you want to re-master the selected soft block using the target design.
- *Push Close Outside Wire(s) As Blockage(s)*: Pushes the pre-route objects that do not overlap the soft block, but are in close proximity into the selected soft block instance. The pre-route objects that are located at distances lesser than the `minSpacing` value from the block are also pushed as blockages.
- *Push Blockage(s):* Pushes blockages inside the soft block instance.
- (IC6.1.8 Only) *Push All As Blockages:* Pushes the routes that have connectivity as blockages and not as routes with pins created on them.
- *Push Row(s) inside Soft Block(s):* Pushes the standard and custom rows inside the soft block. Overlapping rows and the rows that are completely inside the boundary are not impacted. To avoid any possible row region abutment, the size of the constituent instances is shrunk by one unit. Also, only full height rows, not partial height rows, are pushed inside soft blocks. The rows outside the boundary are not pushed, as shown in Figure 4.

  Note: In case of standard rows, the overlapping part of the row with integer number of sites will be pushed. In case of custom rows, rail definitions and components of the row will also be pushed inside the boundary along with the row, as shown in Figure 5.

  Note: Follow pins are created at the intersection of rows and the power stripes. Therefore, pushing rows is important for follow pin creation. After pushing power structures, follow pins cannot be created because rows remain at the top level. However, after pushing rows, the follow pins can be appropriately created.
- (ICADVM20.1 Only) *Push Width Spacing Pattern*: Pushes top-level WSP patterns into sub-blocks such that the top-level WSP pattern and the soft block WSP pattern are aligned. Before running the command, ensure that the PR boundaries of the selected soft blocks are aligned with the period of the top-level WSP grid.
- (ICADVM20.1 Only) *Push Whole Shape*: Specifies that whole shapes that overlap the selected soft block must not be clipped when they are pushed inside the blocks.
- (ICADVM20.1 Only) *Push Selected*: Pushes only the selected nets and shapes inside the soft block. You can use this option to push objects in the incremental mode. For example, power nets can be pushed first followed by signal nets. The target design will have both power and signal routes. If this option is not used, the target design will have only signal routes.

© 2020 All Rights Reserved.

5. In the *Net Filter* group box, select the net shapes to be pushed into the selected soft block:
   - *Power/Ground Net(s)*: Selects the net shapes that are connected to power and ground pins.
   - *Signal*: Selects the net shapes that are connected to signal pins.
   - (ICADVM20.1 Only) *Float*: Selects all floating net shapes, which are the net shapes without any connections.
6. (ICADVM20.1) Select the required push mode for shapes:
   - *Auto Push*: Automatically pushes all shapes into the selected soft blocks. Floating shapes are pushed in as floating shapes, connected shapes are pushed in as connected shapes, and unconnected shapes are pushed in as blockages.
   - *Create Net & Push with Feed Thru Pins*: Creates missing nets before pushing all shapes into the selected soft blocks. Here, floating shapes are pushed in as floating shapes and connected shapes are pushed in as connected shapes. However, unconnected shapes are pushed in as feed-thru terminals.
   - *Push as Blockage*: Pushes all shapes as blockages.
7. (ICADVM20.1 Only) In the *LPP Filter* section, specify the layers that can be pushed into soft blocks.
   a. Click *Define* to display the LPP Selection dialog box. By default, all LPPs are selected.
   b. De-select the LPPs in which shapes are not to be pushed.
   c. Close the LPP Selection dialog box.
8. (ICADVM20.1 Only) Click *Sync with Palette* to filter LPPs based on their selectability settings in the Palette assistant. Only the LPPs that are selectable in the Palette assistant are selected.
9. In the *Create Pin Layer Selection* section, select one of the following options to specify the layers on which pins are to be created.
   - *Same LPP*: Specifies that pins are to be created on the same layers and purposes as the source pin shapes.
   - *Custom Purpose*: Creates pins on the same layers as the source pin shapes and allows you to specify the purpose.
   - *Custom LPP*: Lets you specify the layer and purpose on which pins are to be created.

© 2020 All Rights Reserved.

10. Click *OK*.

The top-level power structures are pushed to the block level.

© 2020 All Rights Reserved.

Figure 4

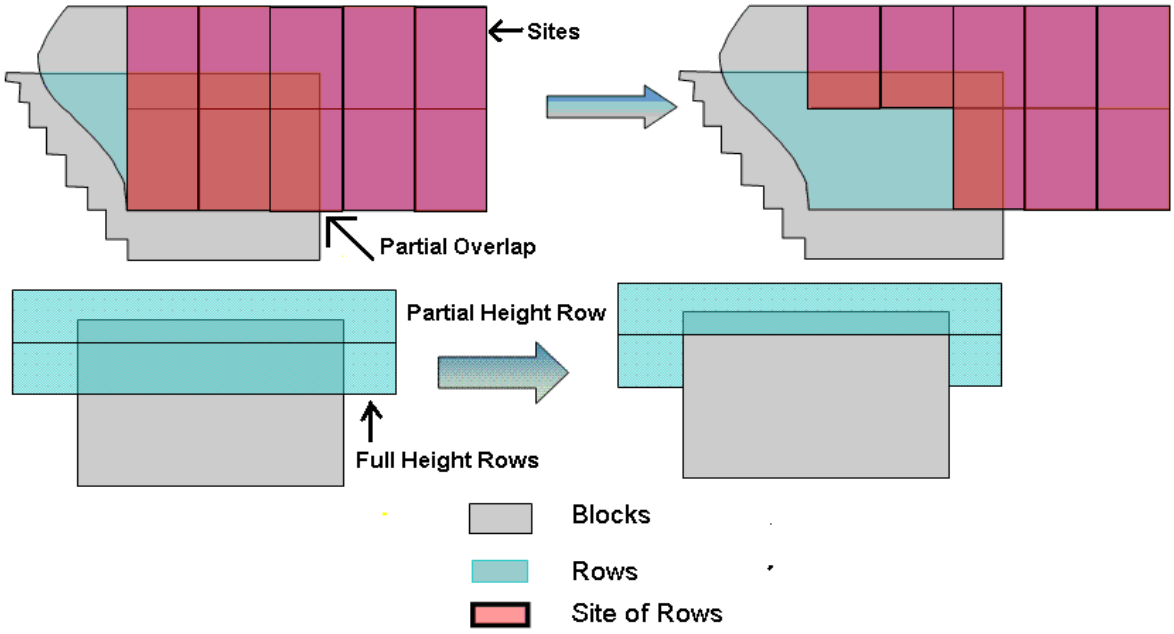
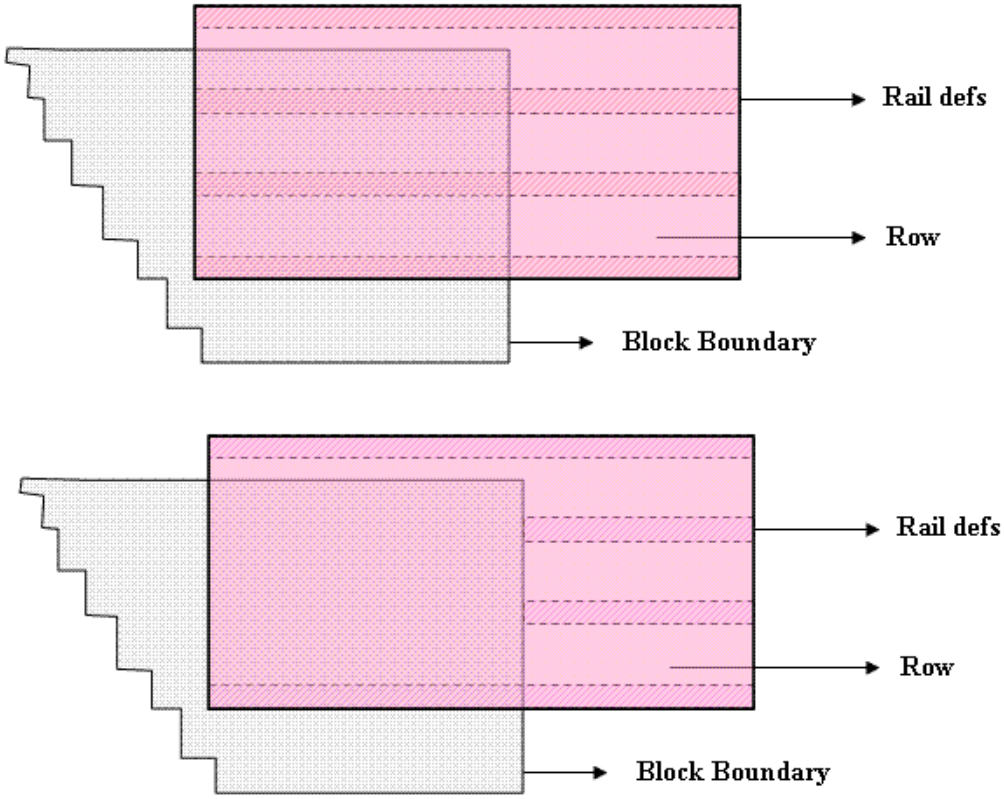


Figure 5

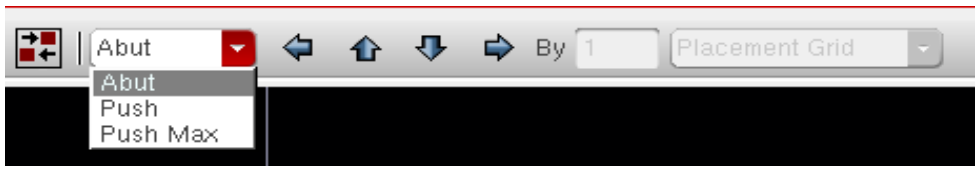


© 2020
All Rights Reserved.

Adjust Blocks

The *Adjust Blocks* command enables you to adjust the existing floorplan by abutting and pushing the soft blocks in the design. These commands work only in the pre-selection mode.

When you invoke the Adjust Blocks command by clicking *Soft Block Update* -> *Adjust Blocks*, the Floorplan toolbar expands to display the Adjust Blocks sub-toolbar. You can invoke the Adjust Blocks sub-toolbar by clicking the Adjust Blocks icon as well. The following graphic shows the sub-toolbar.

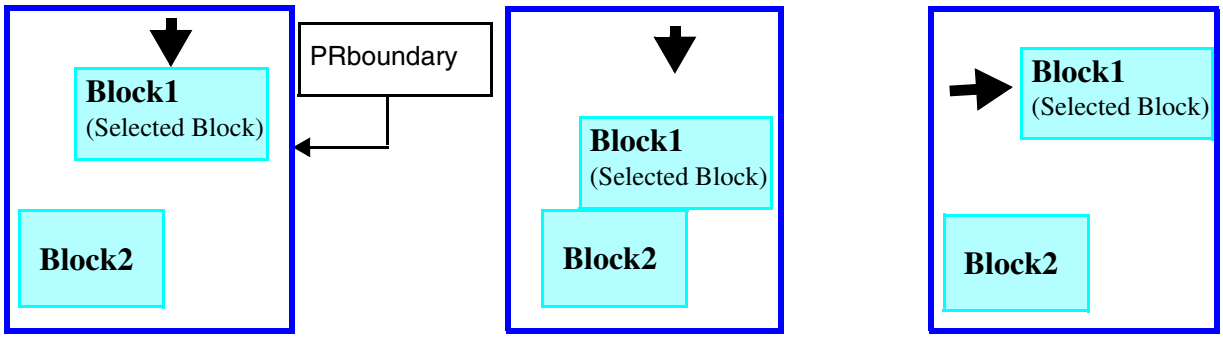


The *Adjust Blocks* command supports three sub-commands, such as *Abut*, *Push*, and *Push Max*, and each sub-command supports four actions, such as up, down, left, and right. These commands enable you to operate on the selected block when you click the arrow icons. For example, to abut a soft block to the bottom of the design PRboundary, select the soft block and click the down arrow icon. The soft block abuts to the bottom of the design PRboundary.

Note: The PRboundary can either be rectangle or rectilinear.

Abut

Use the *Abut* option to move the selected block in the specified direction until it abuts another block or the PRboundary. When you abut a soft block, the soft block does not overlap another block or move off the design PRboundary.



Original position of Block1 before the abut operation.

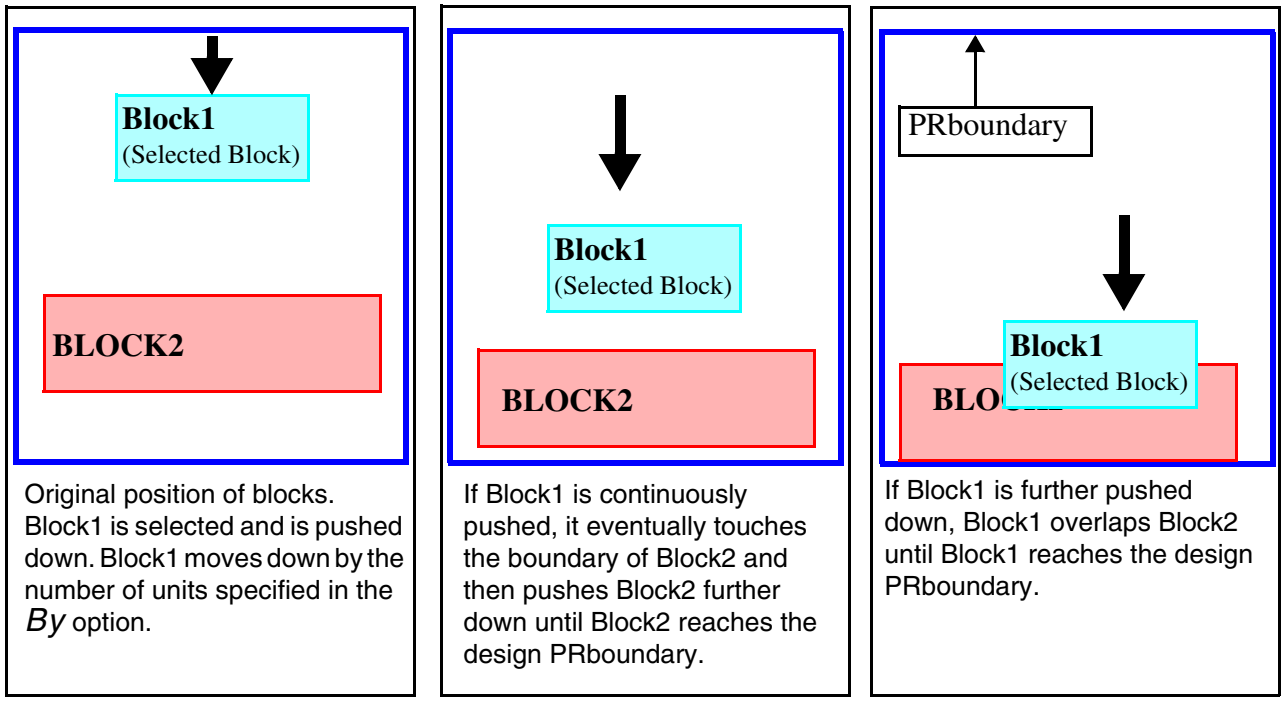
When you abut the block down, it snaps to Block2.

When you abut the block to the right, it snaps to the design PRboundary.

© 2020
All Rights Reserved.

Push

Use the *Push* option to move the selected block in the specified direction by the number of units you specify in the *By* option. The *Push* option does not check for overlapping blocks but does not move off the design PRboundary. Each time you click an arrow icon, the selected block moves in the specified direction until it reaches the design PRboundary. If it abuts another block, it pushes the block as well until the adjacent block reaches the PRboundary. If you continue to further push the block in the same direction, the selected block overlaps the adjacent block until the selected block reaches the design PRboundary, as shown in the graphic.



Original position of blocks. Block1 is selected and is pushed down. Block1 moves down by the number of units specified in the *By* option.

If Block1 is continuously pushed, it eventually touches the boundary of Block2 and then pushes Block2 further down until Block2 reaches the design PRboundary.

If Block1 is further pushed down, Block1 overlaps Block2 until Block1 reaches the design PRboundary.

Push Max

Use the *Push Max* option to push the selected block in the specified direction until it abuts the PRboundary. This option works the same way as the *Push* option, except that it pushes the soft block to the PRboundary in one go. If another block lies in the path of the selected block, the *Push Max* option pushes the block to the PRboundary. The *Push Max* option ensures that the soft block does not overlap another block or move off the core area.

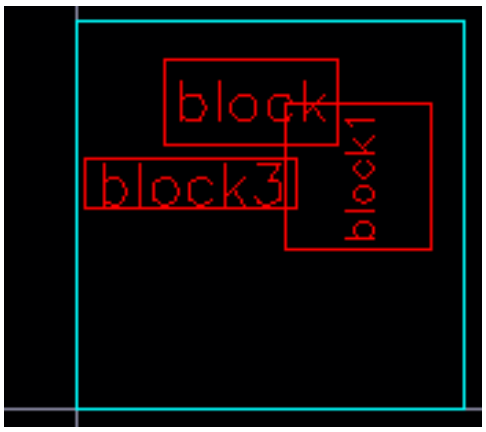
© 2020 All Rights Reserved.

Remove Block Overlap

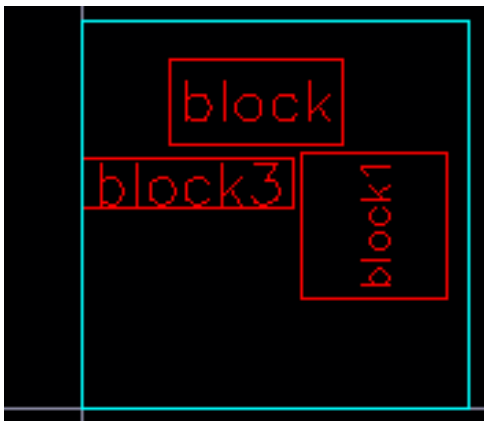
Use the *Remove Block Overlaps* command to place overlapping hard blocks and soft blocks adjacent to each other. This command works on pre-selection as well as post-selection modes. To remove overlaps, select the overlapping soft blocks and then click the *Floorplan – Soft Block Update – Remove Block Overlaps* command.

Note: In Layout EXL, choose *Plan – Block Planning – Remove Block Overlaps*.

The overlaps are removed and the blocks are placed adjacent to each other as shown in the graphic.



The three blocks overlap each other.



When you select the three blocks and click the *Remove Block Overlaps*, the overlaps are removed.

Snap Soft Blocks to Grid

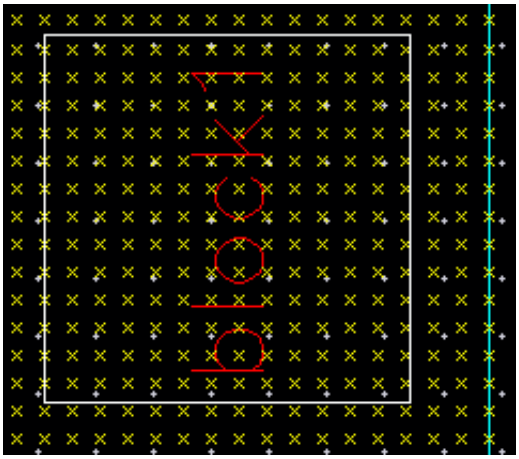
Use the *Snap Soft Blocks to Grid* command to snap the soft blocks to placement grid or the manufacturing grid. This command works on pre-selection as well as post-selection modes.

If the soft block is a digital soft block, it is snapped to the placement grid. Else if the soft block is a custom soft block, it is snapped to the manufacturing grid. In the following graphic, Block1 is a digital block and when the *Snap Soft Blocks to Grid* command is used, Block1 is

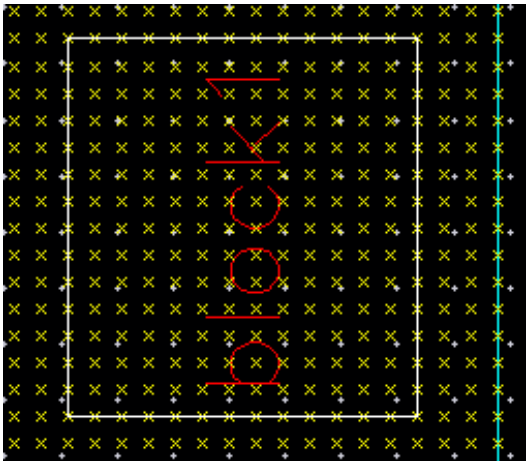
© 2020 All Rights Reserved.

snapped to the placement grid (represented by the x in the graphic). To know more about soft blocks, refer to the Editing Soft Blocks section in the Virtuoso Layout Suite XL User Guide.

Block Type	Edge Snapping
CUSTOM	Manufacturing grid
DIGITAL	Placement grid



Position of Block1 before snapping.



After snapping, Block1 is snapped to the placement grid.

Pull Soft Blocks Inside PR Boundary

Use the *Pull SoftBlocks inside PR Boundary* command to pull the soft blocks overlapping the PRboundary to the core area. If IO rows exist inside the PRboundary, the soft blocks are pulled inside the IO rows, which is the core area.

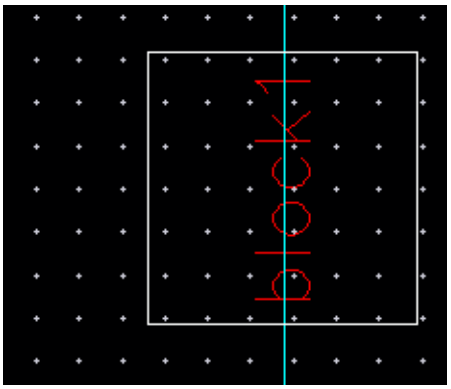
To pull a soft block in the core, select the soft blocks and then click *Floorplan – Soft Block Update – Pull SoftBlocks inside PR Boundary*.

Important

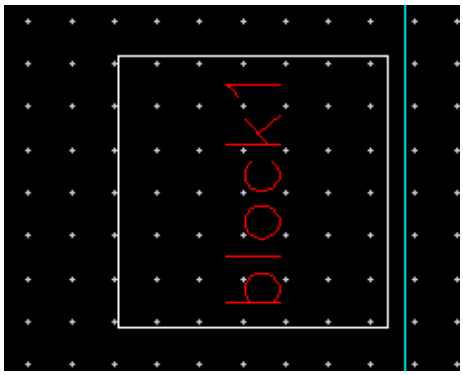
In Layout EXL, choose *Plan – Block Planning – Pull SoftBlocks inside PR Boundary*.

© 2020 All Rights Reserved.

The following graphic shows the position of Block1 before and after the block is pulled.



Block1 overlaps the PRboundary



Block1 is pulled in the core area

Edit Soft Blocks

Use the *Edit Soft Block* command to modify the attributes of individual soft blocks taking into account the placement and connectivity requirements in your design.

You can use this command to modify the following soft block attributes.

- Type of soft block that will be created
- Height, width, area, or shape of the place and route boundary
- Length, width, layer, and signal type of soft pins
- Routing and placement of obstructions inside a block

To know more about editing soft blocks, refer to the Editing Soft Blocks section in the Virtuoso Layout Suite XL User Guide.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

7

Creating Pin Group Guides

This chapter contains the following sections:

- Introduction to Pin Group Guides
- Creating a Pin Group Guide
- Comparing Pin Group Guides
- Placing Pin Group Guides as per Schematic or Symbol

Introduction to Pin Group Guides

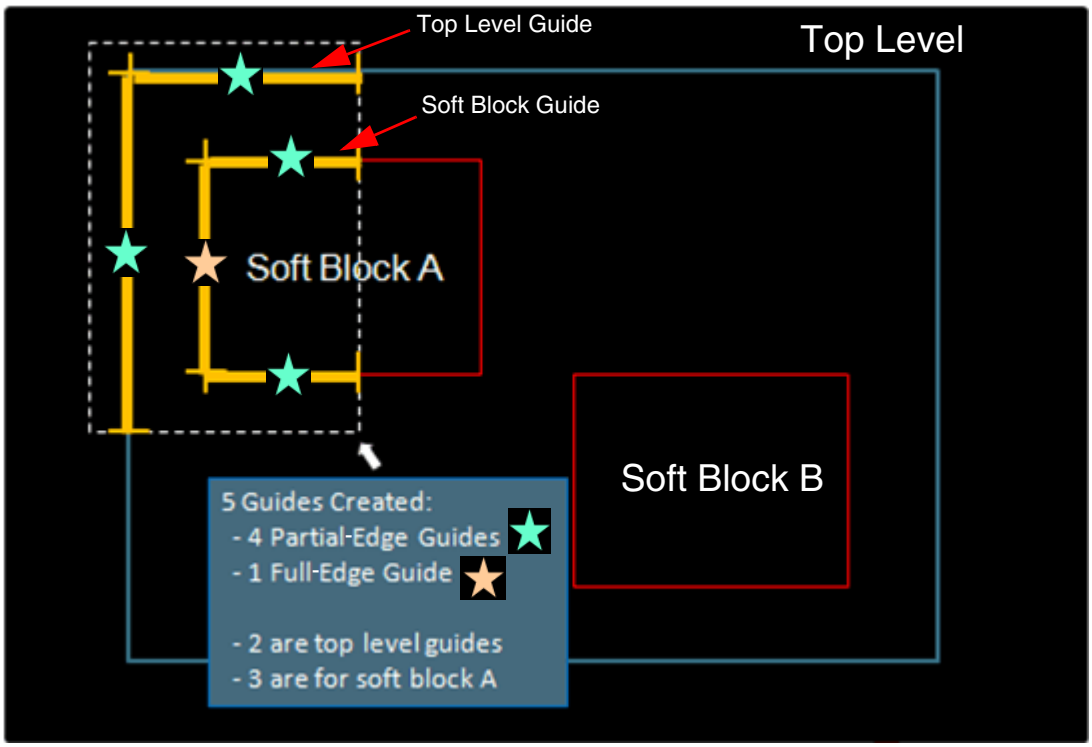
Virtuoso® Floorplanner lets you create pin groups and assign pin constraints to these groups. This feature is useful for designs with a large number of pins, such as mixed-signal designs, where it would be difficult and time-consuming to create pin constraints on individual pins. You can create pin groups, assign them to full or partial edge(s), and apply process constraints, such as layer range and spacings, to the entire pin group through a dedicated pin group guide constraint.

A pin group guide constraint:

- comprises one or more pins and may have one or more pin group guides.
- is a placement constraint for the pin group, and is defined on partial or complete edges of blocks.

© 2020 All Rights Reserved.

Pins are placed on the pin group guides. In the following example, one full-edge pin group guide and four partial-edge pin group guides are created.

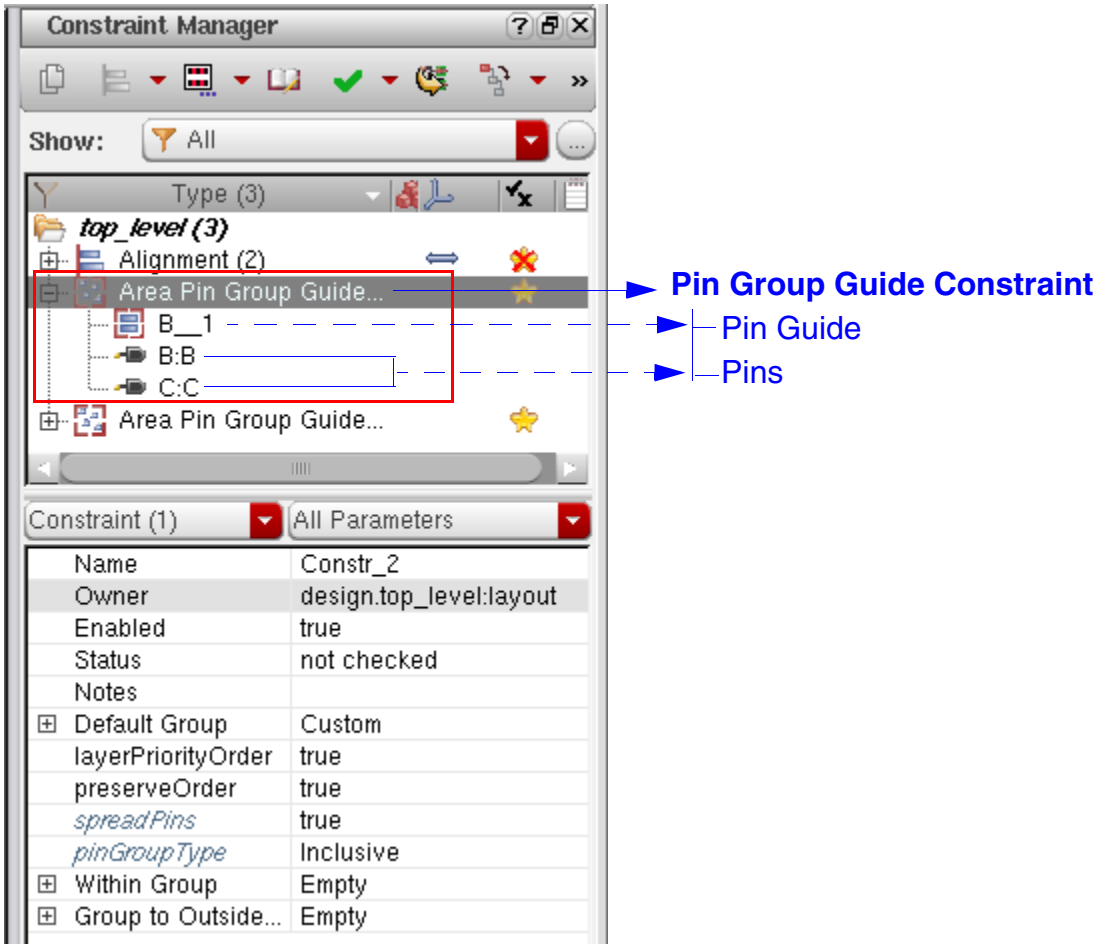


Note: For mixed-signal designs, the pin group guide constraint data is inter-operable with Innovus™ Digital Implementation System (Innovus).

Use the options in the Pin Group Guide form to create and edit pin group guide constraints. These constraints are created in the Virtuoso constraint system. Therefore, you can use the Virtuoso Constraint Manager assistant to verify and further customize these constraints. In the following snapshot, the `Constr_2` pin group constraint is selected in the Constraint

© 2020 All Rights Reserved.

Manager assistant and its attributes are displayed in the Property Editor. The `Constr_2` pin group guide constraint comprises pins `B:B` and `C:C`, and a pin guide `B__1`.



The pin group guide constraint selected in the Constraint Manager assistant is highlighted in the layout design.

After creating the pin group guide constraints, you need to run the Pin Optimizer to achieve the final pin placement. The Pin Planner and Pin Optimizer honor the pin group guide constraints. The Pin Optimizer places the pins accordingly, while maintaining design correctness.

© 2020
All Rights Reserved.

Creating a Pin Group Guide

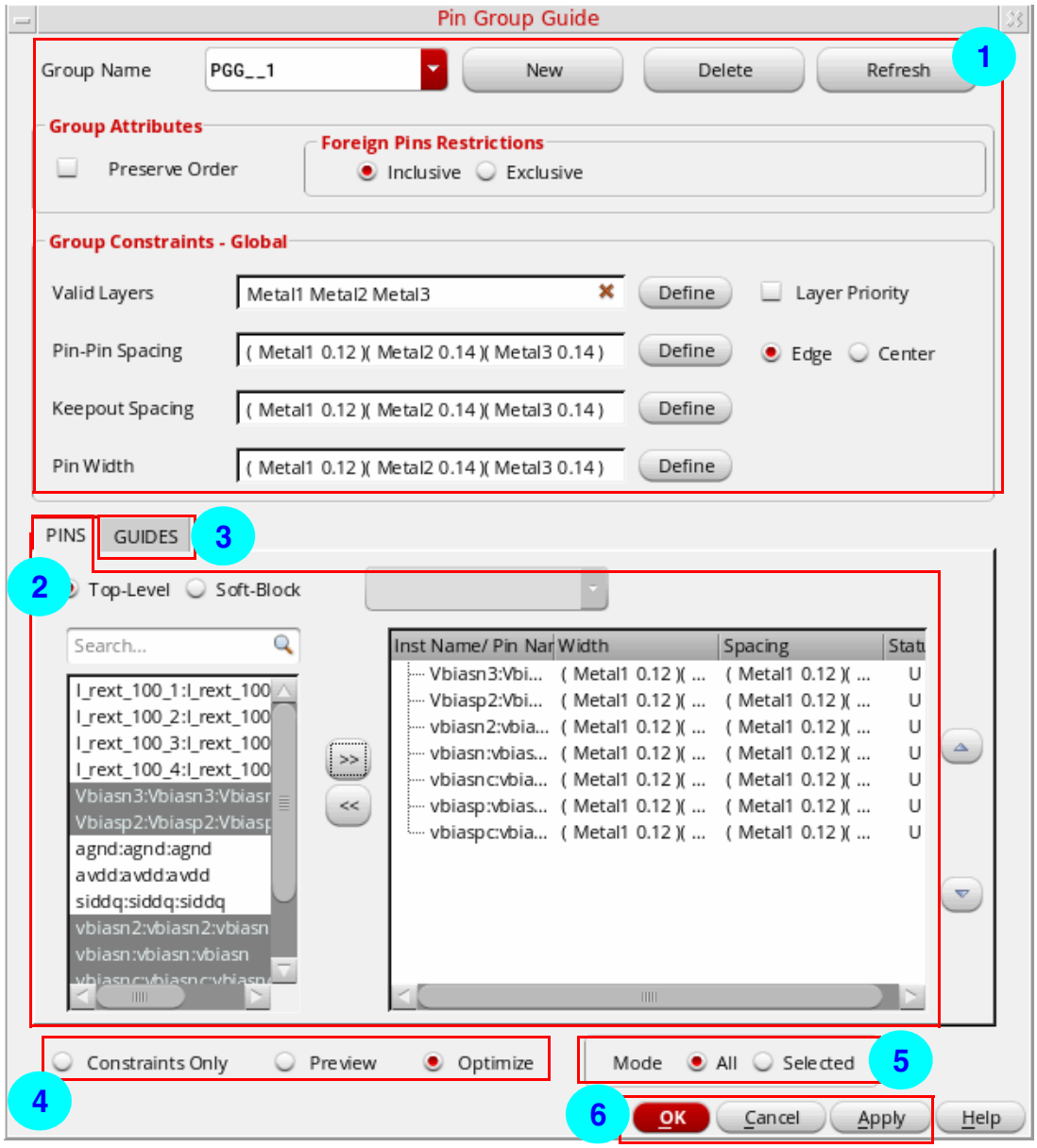
Virtuoso Floorplanner provides options to create pin groups and guides interactively. When you select pins on the layout, then the pins are selected and highlighted in the form, and vice versa.

Video

For quick reference of this feature, refer to the Pin Group and Guide (PGG) Support in VFP demonstration.

© 2020 All Rights Reserved.

Choose *Floorplan – Pin Group Guides – Create* to display the Pin Group Guide form.



Important

In Layout EXL, choose *Plan – Pin Planning – Create/Edit Pin Group Guide*.

The steps for creating a pin group guide are:

1. Specify the pin group tabulates. Use the following options:

   - <u>Group Name</u>

© 2020 All Rights Reserved.

- Group Attributes
- Group Constraints - Global

2. Select the pins to be included in the pin group guide. Use the options on the PINS tab.
3. Draw the pin group guide on the layout canvas. Use the options on the GUIDES tab.
4. Choose the required option:
   - **Constraint Only:** Updates only the pin group guide constraint.
   - **Preview:** Provides a visualization of the placement of the pins on the pin group guides.
   - **Optimize (default):** Updates the pin group guide constraint and places pins on the guide. The pins are optimized to obtain the shortest possible net length.
5. Choose a *Mode*. This option is only applicable to the Pin Optimizer.

   The *Mode* is set to *All* or *Selected* depending on whether all or some of the pins are selected.
6. Click *OK* or *Apply* to create the pin group guide.

At any point before creating the pin group guides, you can click *Refresh* to discard the changes made to the form. *Refresh* loads the default values from the database, technology file, and constraint. Whenever the database, technology file or constraints are modified, then ensure that you click *Refresh* to update form values accordingly.

Group Name

To create a new group, either type the new group name in the *Group Name* field or click *New* to generate a unique group name. You can edit the automatically-generated group name.

To edit the attributes of an existing group, select the group name from the *Group Name* drop-down list.

To delete an existing pin group, choose the group name from the *Group Name* drop-down list and click *Delete*.

Group Attributes

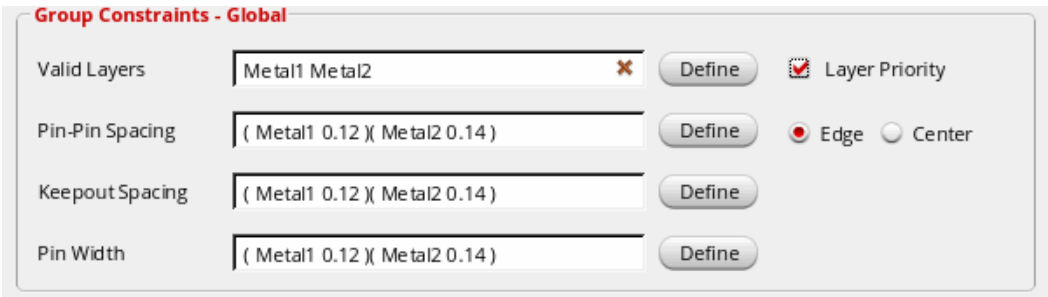
If you select the *Preserve Order* check box in the *Group Attributes* section, then the Pin Optimizer honors the order of pins in the pin group constraint and optimizes the pins in that sequence.

© 2020 All Rights Reserved.

The *Foreign Pins Restrictions* section in the *Group Attributes* group box includes the following options:

- **Inclusive**: Supports the presence of foreign pins within the pin group guide area. Therefore pin group guides with different pins can overlap in the layout design. The *Inclusive* option should be selected for overlapping pin group guides.
- **Exclusive**: Does not support the presence of foreign pins within the pin group guide area. In this case, pin group guides with either one or both pin group guides set to *Exclusive* cannot overlap in the layout design. Suitable error messages are displayed in the CIW.

Group Constraints - Global



The *Group Constraints - Global* section provides the following options to define global constraints for all pins in the pin group:

- **Valid Layers:** Specifies valid layers for pin placement. Click *Define* to display the Layer selection pop-up box, which lists all the layers that are available in the current design.

Add or remove the layers by selecting or clearing the corresponding check boxes.

© 2020
All Rights Reserved.

Click ✖ to delete all layers from the *Valid Layers* field.

If the *Layer Priority* check box is selected, then pins are optimized based on their layer priority. Layers are prioritized in the order in which they are listed in the *Valid Layers* field. You can alter the sequence by deleting the layers from the *Valid Layers* field and typing the layer names in the required sequence, or by selecting layer names in the required sequence from the *Layer selection* list.

Tip

Either set the matchLabelLayerWithPin environment manually or from the *Floorplanning* tab of the Floorplan Global Options form (Layout GXL Options form in IC6.1.8) to place pin labels on the same layers as their corresponding pins. Therefore, if a pin is re-layered during placement, then the pin label is also moved to the same layer.

Related SKILL functions:

- `dbSetPinGroupGuideValidRoutingLayers`: Sets the value of the validRoutingLayers constraint on the given area pin group guide.
- `dbGetPinGroupGuideValidRoutingLayers`: Returns the value of the validRoutingLayers constraint set on the specified area pin group guide.
- `dbSetPinGroupGuideLayerPriorityOrder`: Sets the value of the layerPriorityOrder constraint on the given area pin group guide.
- `dbGetPinGroupGuideLayerPriorityOrder`: Returns the value of the layerPriorityOrder constraint set on the specified area pin group guide.
- `dbPinGroupGuideHasLayerPriorityOrder`: Returns a boolean value indicating that the layerPriorityOrder constraint is set on the specified area pin group guide.
- `dbUnsetPinGroupGuideValidRoutingLayers`: Deletes the validRoutingLayers constraint on the given area pin group guide.
- `dbUnsetPinGroupGuideLayerPriorityOrder`: Deletes layerPriorityOrder constraint on the given area pin group guide.

- **Pin-Pin Spacing:** Specifies the spacing between adjacent pins on each layer. The *Pin-Pin Spacing* field lists layer names and the corresponding spacing values. Instead of typing the values, click *Define* to display the Pin-Pin Spacing Constraints pop-up box. All valid layers are automatically populated in the *Layer Name* column. Specify the spacing values for these layers and close the window.

  Choose either *Edge* or *Center*, depending on how the spacing needs to be applied.

© 2020 All Rights Reserved.

Important

Pin spacing values cannot be:

- Smaller than the `minSpacing` or the `tableSpacing` values that are specified in the technology file.
- Greater than the longest edge of the PR boundary.

Related SKILL functions:

- `dbSetPinMinPinSpacing`: Sets the value of the minSpacing constraint on the given pin.
- `dbPinHasMinPinSpacing`: Returns a boolean value indicating whether the minSpacing constraint is set on the specified pin.
- `dbGetPinMinPinSpacing`: Returns the value of the minSpacing constraint set on the specified pin.
- `dbUnsetPinMinPinSpacing`: Deletes the minSpacing constraint on the given pin.
- `dbSetPinGroupGuideMinPinSpacing`: Sets the value of the minSpacing constraint on the given pin group guide.
- `dbPinGroupGuideHasMinPinSpacing`: Returns a boolean value indicating whether the minSpacing constraint is set on the specified pin group guide.
- `dbGetPinGroupGuideMinPinSpacing`: Returns the value of the minSpacing constraint set on the specified pin group guide.
- `dbUnsetPinGroupGuideMinPinSpacing`: Deletes the minSpacing constraint on the given pin group guide.

- **Keepout Spacing:** Defines the minimum edge-to-edge spacing between pins and other components. In other words, the K*eepout Spacing* value defines the area around the pin within which no object can be placed.

  Similar to pin spacing values, you can either directly type the layer name and keepout spacing values, or open the Keepout Spacing Constraints pop-up box and specify the spacing value for each valid layer.

Important

Keepout spacing values cannot be:

© 2020 All Rights Reserved.

- Smaller than the `minSpacing` or the `tableSpacing` values that are specified in the technology file.
- Greater than the longest edge of the PR boundary.

Related SKILL functions:

- `dbSetPinGroupGuideKeepoutSpacing`: Sets the value of the keepoutSpacing constraint on the given pin group guide.
- `dbPinGroupGuideHasKeepoutSpacing`: Returns a boolean value indicating that the keepoutSpacing constraint is set on the specified pin group guide.
- `dbGetPinGroupGuideKeepoutSpacing`: Returns the value of the keepoutSpacing constraint set on the specified pin group guide.
- `dbUnsetPinGroupGuideKeepoutSpacing`: Deletes the keepoutSpacing constraint on the given pin group guide.

- **Pin Width:** Specifies the width of pins for each layer. Type the layer name and pin width values in the *Pin Width* field, or click *Define* to open the Width Constraints pop-up box, where you can specify the pin width value for all valid layers.

Important

Pin width values cannot be:

- Smaller than the `minWidth` value that is specified in the technology file.
- Greater than the longest edge of the PR boundary.

Related SKILL functions:

- `dbSetPinMinPinWidth`: Sets the value of the minWidth constraint on the given pin.
- `dbPinHasMinPinWidth`: Returns a boolean value indicating whether the minWidth constraint has been applied to the specified pin.
- `dbGetPinMinPinWidth`: Returns the value of the minWidth constraint set on the specified pin.
- `dbUnsetPinMinPinWidth`: Deletes the minWidth constraint on the given pin.
- `dbSetPinGroupGuideMinPinWidth`: Sets the value of the minWidth constraint on the given pin group guide.
- `dbPinGroupGuideHasMinPinWidth`: Returns a boolean value indicating whether the minWidth constraint is set on the specified pin group guide.

© 2020 All Rights Reserved.

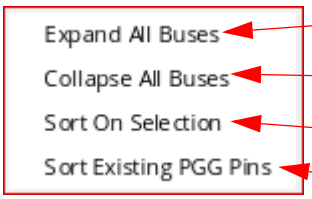
- `dbGetPinGroupGuideMinPinWidth`: Returns the value of the minWidth constraint set on the specified pin group guide.
- `dbUnsetPinGroupGuideMinPinWidth`: Deletes the minWidth constraint on the given pin group guide.

PINS tab

Use the PINS tab to select pin members for the pin group guide constraint.

The PINS tab comprises a pin browser on the left side and a pin group table on the right side. The pin browser enlists all the shape pins found in the current scope. It does not list pins that are vias. The entries are expressed as `termName:pinName:figName` because the pin figure is the physical entity that you are placing. If a pin has more than two pin figures attached to it, but the pin figures have the same name, they are represented by a single entry in the pin list. Pins that you select in the layout design are automatically selected in the pin browser. Similarly, pins selected in the pin browser are automatically highlighted in the layout design.

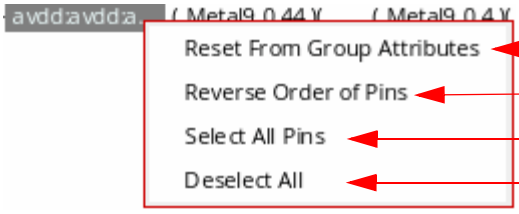
Select the required pins in the pin browser and right-click to display the shortcut menu, which provides the following options:



- Expand All Buses — Expands all buses included in the selected pins
- Collapse All Buses — Collapses all buses
- Sort On Selection — Sorts the selected pins
- Sort Existing PGG Pins — Places pins in the following order:
  1. Pin members of pin group guide constraints.
  2. Fixed and locked pins.
  3. Pin members of an alignment constraint.
  4. Unconstrained pins.

Use to move pins from the pin browser to the pin group table.

After moving pins to the pin group table, right-click to display the shortcut menu in the pin group table, which provides the following options:



- Reset From Group Attributes — Resets the pin list from the associated group attributes
- Reverse Order of Pins — Reverses the order of the selected pins
- Select All Pins — Selects all pins in the pin guide table
- Deselect All — Deselects all pins in the pin guide table

© 2020 All Rights Reserved.

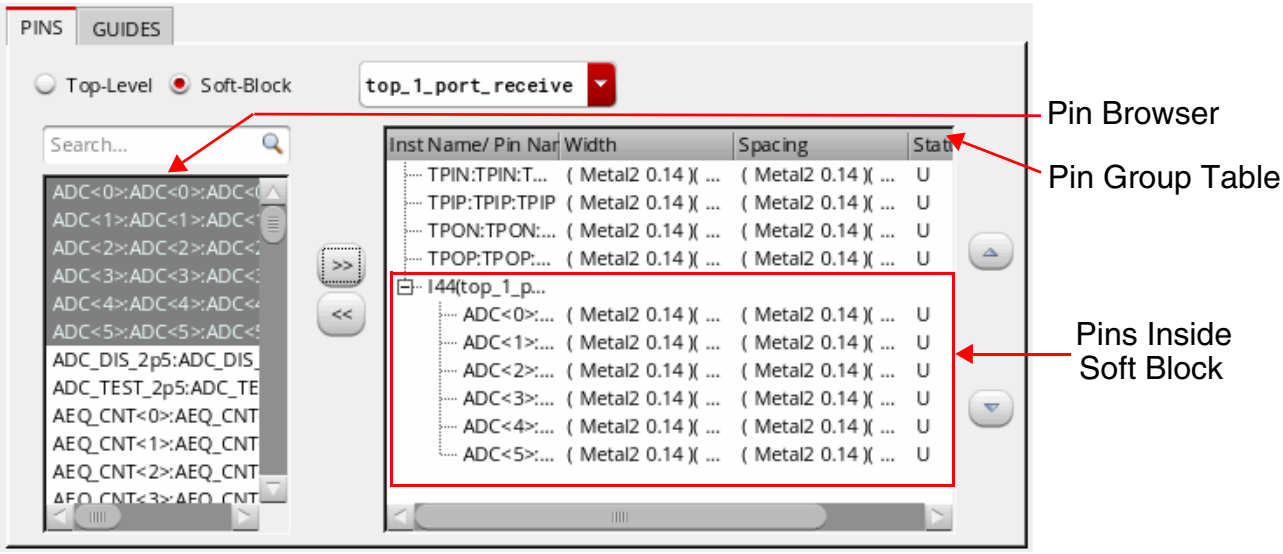
You can add top-level pins, soft block pins, or a mix of both to the pin group table. To display a list of soft block pins, choose *Soft Block*. Select the required soft block from the drop-down list. Pins in the selected soft block are displayed in the pin browser. Also the focus in the layout design changes to the selected soft block.

Note: Pins can be added only once in a pin group guide.

Use the add (>>) or remove buttons to add or remove pins from the pin group table. The pin group table also provides the *Pin Name*, *Width*, *Spacing*, and *Status* values in corresponding columns. *Status* indicates the placement status of the pins. Valid statuses are:

- *P* - Placed
- *U* - Unplaced
- *L* - Locked
- *F* - fixed

Note: You can directly edit the pin width, spacing, and status values in the pin group table. Double-click the required cell and select the required values from the drop-down list.

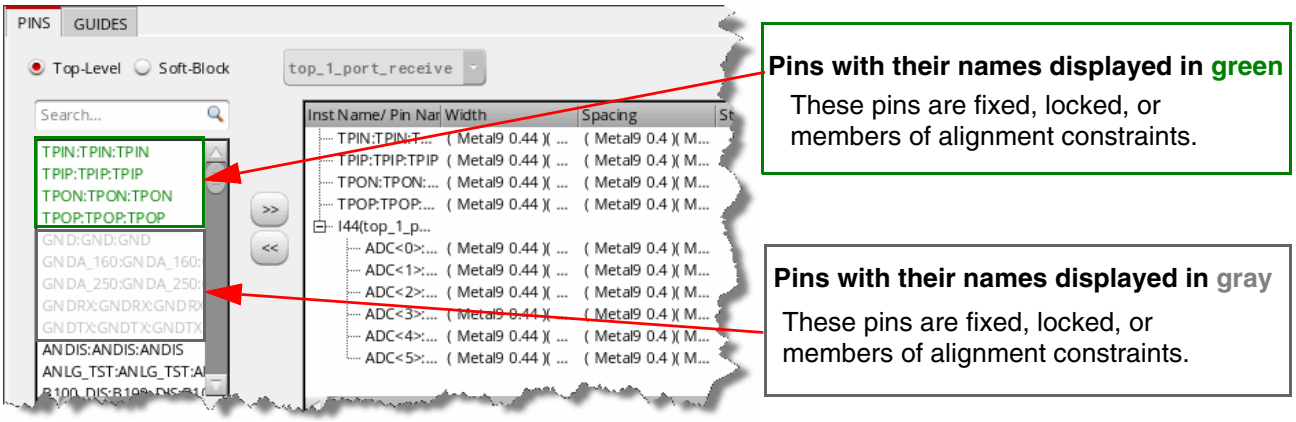


Cross-selection of pins is implemented between the layout design and the Pin Group Guide form. Therefore, the pins selected on the PINS tab are highlighted in the layout design. Similarly, the pins selected in the layout design are selected in the pin browser.

Related SKILL function: `dbAddPinToPinGroupGuide`: Adds a pin to the specified pin group guide.

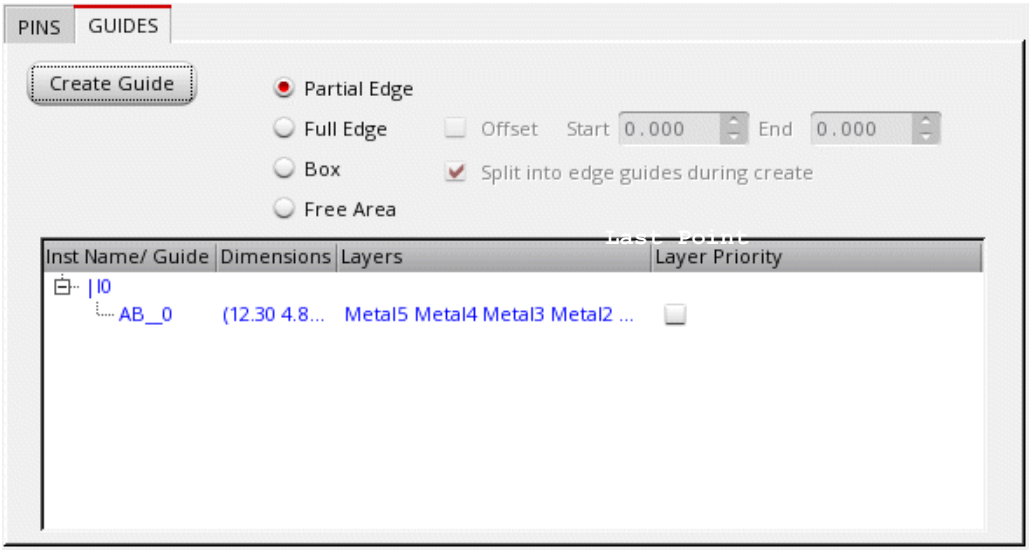
© 2020 All Rights Reserved.

You cannot add the following types of pins to the pin group table:



GUIDES tab

Use the options on the GUIDES tab to interactively create guides.



Important

If you try to create a pin group guide constraint with no pins, then an error message is displayed in the CIW, and the guide is not created.

You can create the following types of pin group guides:

- **Partial Edge:** Use this option to create pin group guides that cover partial edges.

  1. Choose *Partial Edge.*

© 2020
All Rights Reserved.

2. Click *Create Guide*.

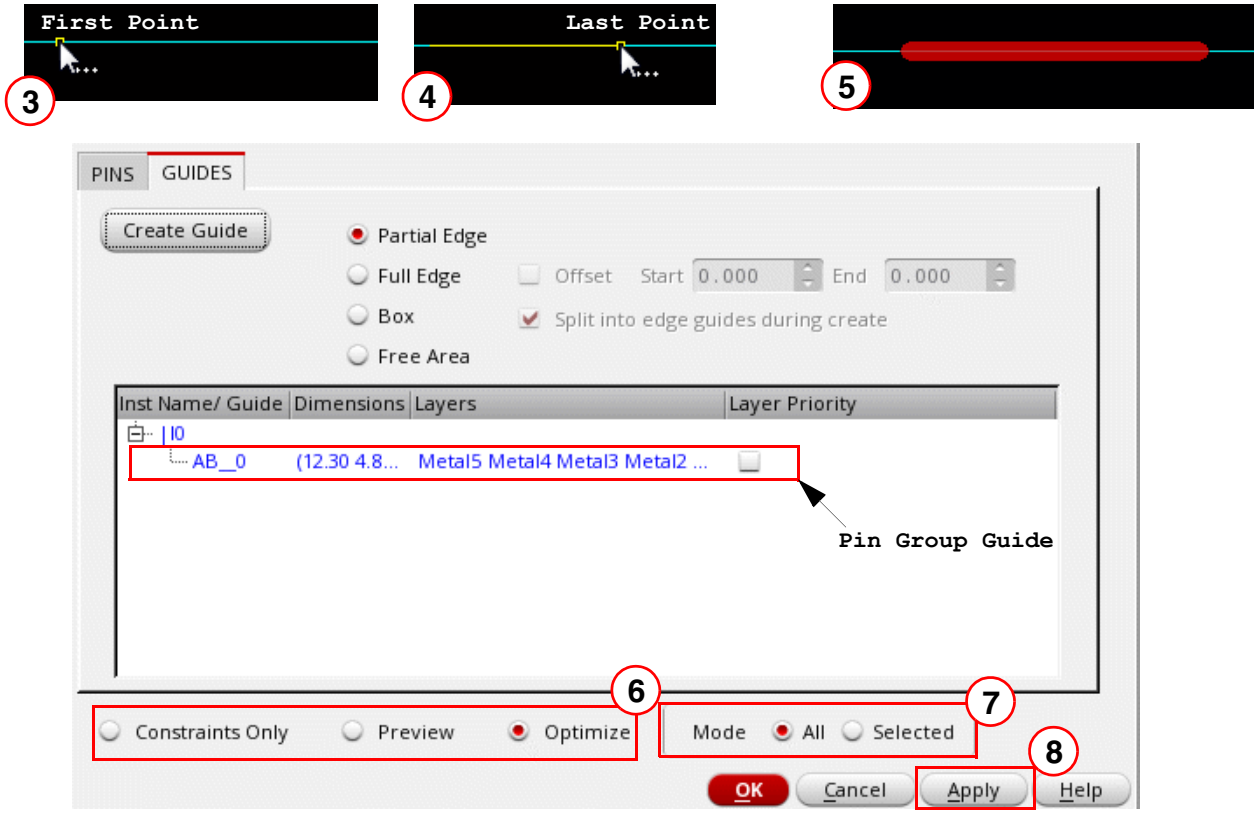
3. In the layout design, click the required edge to indicate the first point of the pin group guide. See Edges for Creating Pin Group Guides for more information.

4. Click to indicate the last point of the pin group guide. If you are creating multiple pin group guides, continue to click the edges to create the additional pin group guides.

Important

A pin group guide is not created if the length of the guide is too small for placement of the selected pins, that is if the guide is smaller than the sum of the `minWidth`, keepout, and spacing values for `metal1`. A suitable warning message is displayed in the CIW.

5. Press `Esc` when you are done.



6. Choose one of the options *Constraints Only*, *Preview*, or *Optimize* (default).

7. Choose a *Mode*.

8. Click *Apply* to submit the updates you made and view the pin group guide in the layout design.

© 2020
All Rights Reserved.

Area boundaries are created and the specified pins are placed within the pin group guide that you created.

Note: Width of the pin group guide equals the unit width area boundary.

If you enter a top-level or soft block corner as the first point of a pin group guide, then, by default, you are prompted to create a vertical guide (along the y-axis). To create a horizontal guide, press `F3` to display the Create Guide popup box. Choose *Horizontal* from the *Snap Mode* drop-down list and click *Hide* to close the Create Guide popup window. Proceed to create a horizontal pin group guide. You can use this method to create orthogonal pin group guides.

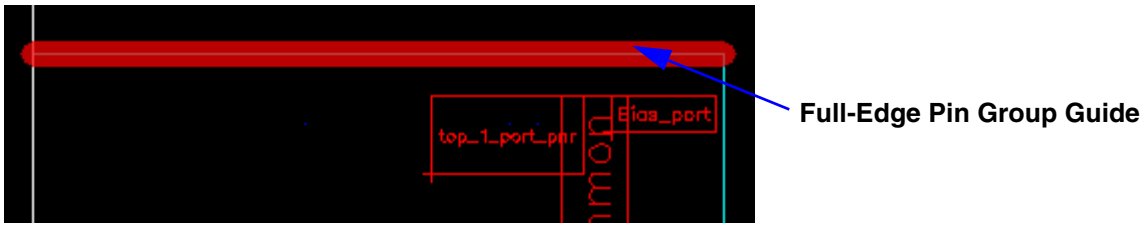
Edges for Creating Pin Group Guides

Choose one of the following edges for creating pin group guides:

- Partial Edge
- Full Edge (See Creating a Full Edge Pin Group Guide)
- Box (See Creating a Box Pin Group Guide)
- Free Area (Creating a Free Area Pin Group Guide)

Creating a Full Edge Pin Group Guide

Use this option to create pin group guides that cover the edges completely. Unlike partial-edge guides, full-edge guides can be specified with a single click on a top-level or soft block edge in the layout design. The entire edge on which you clicked is selected.

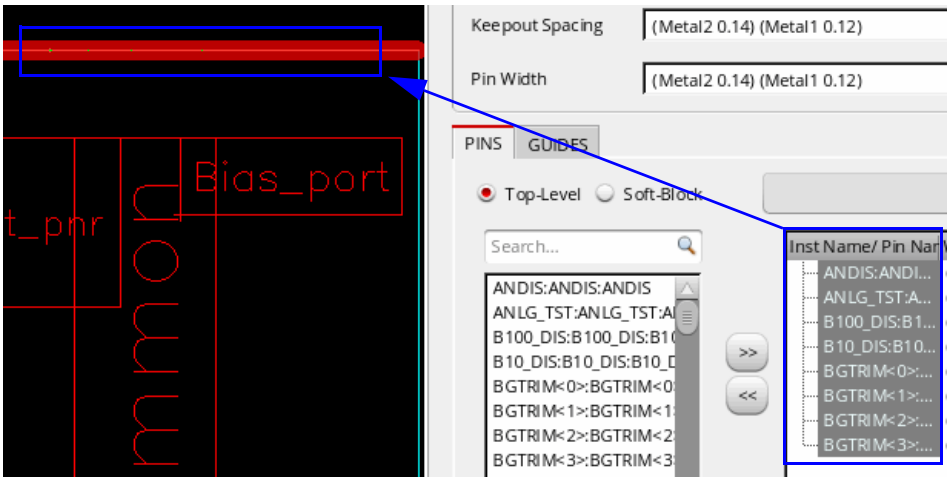


Continue to click the required edges to create multiple full-edge pin group guides.

While selecting the edges, you can go back to the Pin Group Guide form and select *Partial Edge* to start creating partial-edge guides along with the full-edge guides that you created. The Enter function automatically switches from the *Partial Edge* to *Full Edge* mode. Press Esc when you are done. Click *Apply* to submit the updates you made and to view the pin

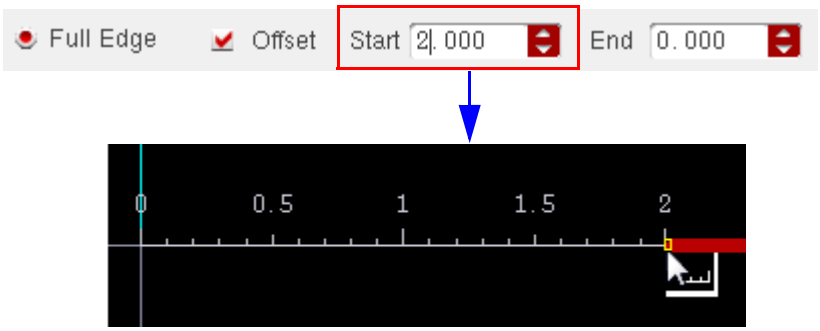
© 2020 All Rights Reserved.

group guides in the layout design. Pins are automatically placed along the pin group guide that you created.



Selected Pins are Highlighted in the Layout Window

Use the *Offset* option to specify the *Start* and *End* offset values of the pin group guide on the corresponding edges.

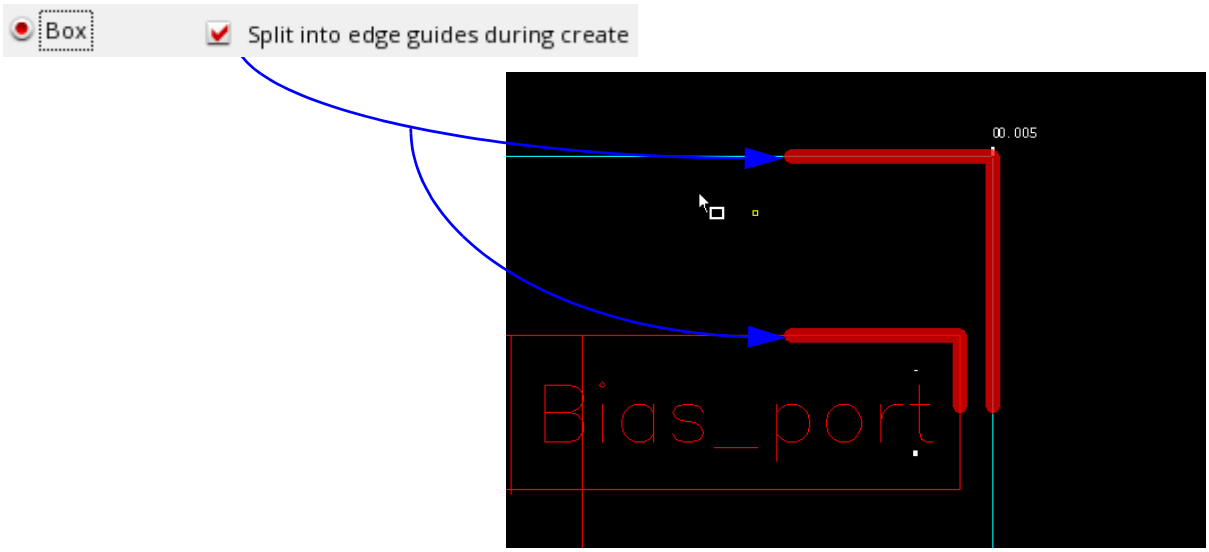


Creating a Box Pin Group Guide

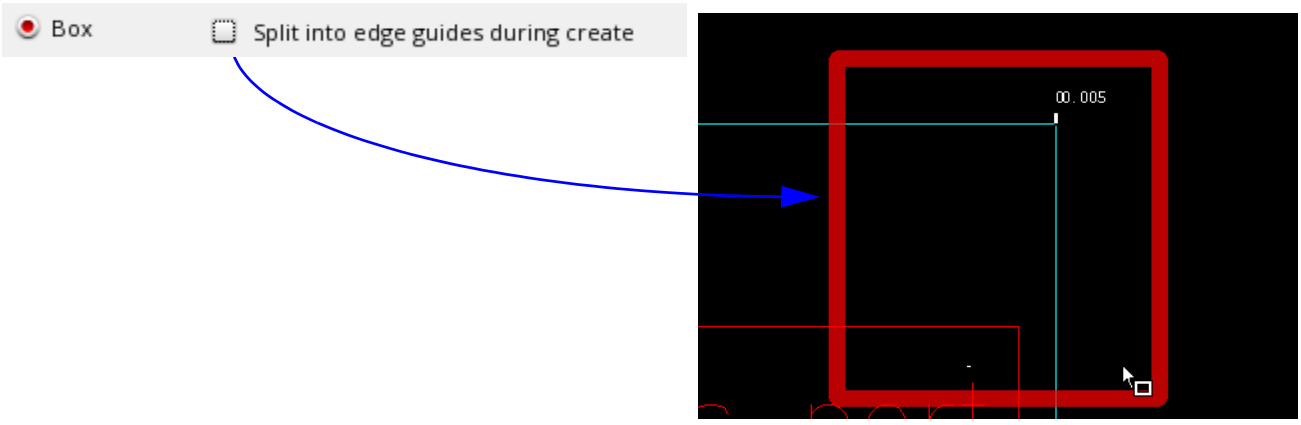
Use this option to draw one or more areas in the layout window. Pin group guides are created on the PR boundary edges that intersect the selected boxes. Press Esc when you are done. If the *Split into edge guides during create* check box is selected, top-level and soft block

© 2020 All Rights Reserved.

edges that intersect the box are created as the partial- or full-edges for creating pin group guides.



If the *Split into edge guides during create* check box is deselected, then a bounding box is created for the selected area. Pin edges within the bounding box are selected for placing pins.



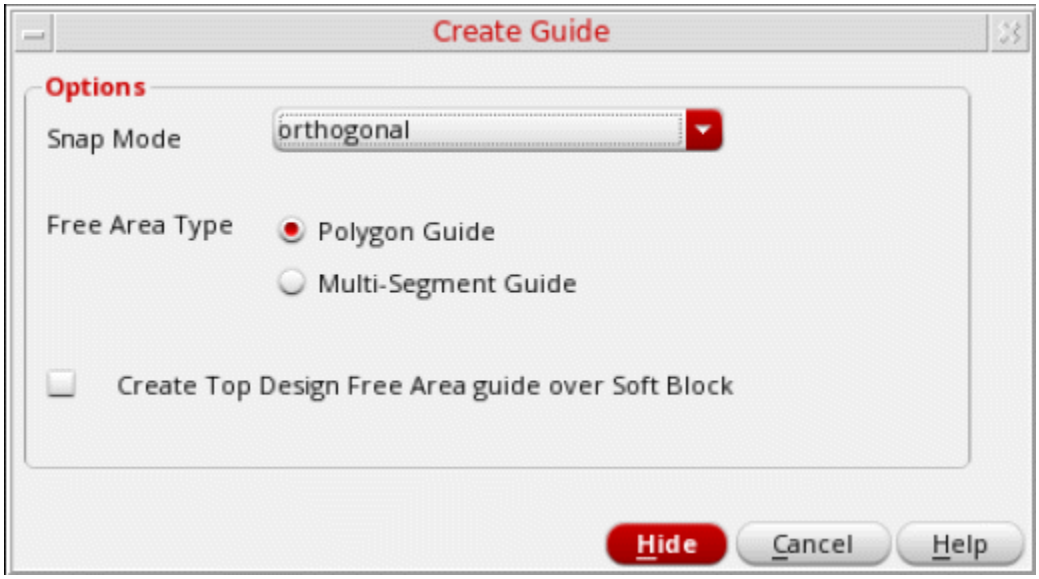
Note: Whenever the PR boundary of a digital block is modified though stretch or move operations, the area pin group guides on these blocks are automatically snapped to the PR boundary.

Creating a Free Area Pin Group Guide

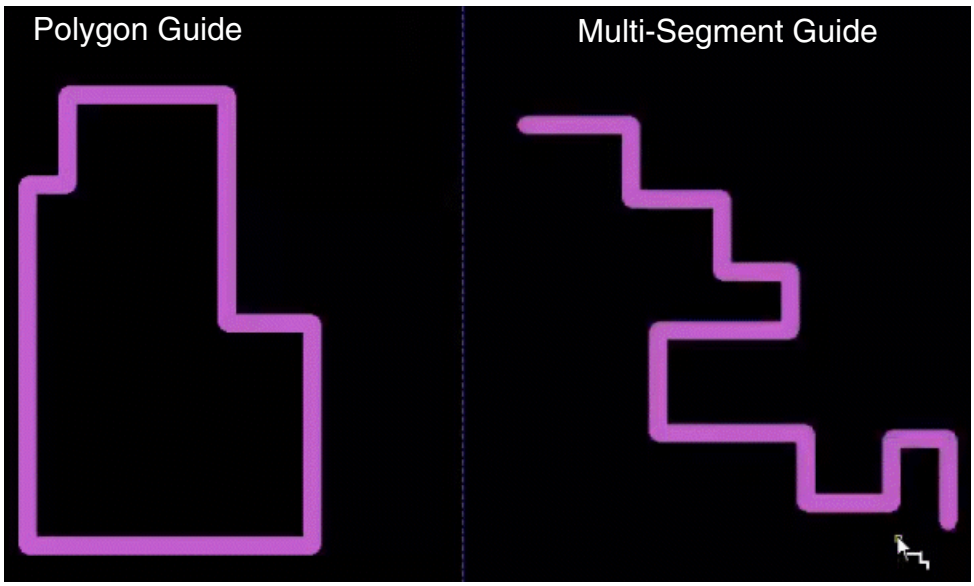
Use this option to place pins in the free area within the PR boundary of the top level or instance level (level-1) from the top level. To create a free area pin group guide:

© 2020 All Rights Reserved.

1. Select *Free Area* on the *Guides* tab.
2. Click *Create*. By default, you can create polygonal guides.
3. Click `F3` to open the Create Guide form.



4. Select a *Free Area Type*, which indicates the shape of the guide that you want to create. The following image shows the differences between polygonal and multi-segment guides.



Note: The top-level (level-0) and level-1 pin group guides are highlighted in different colors on the layout canvas so that they are visually distinguishable.

© 2020
All Rights Reserved.

5. Select *Create Top Design Free Area guide over Soft Block* to create top-level (level-0) guides over soft blocks. By default, if soft block pins are added on the *Pins* tab, then a level-1 guide is created.
6. Click *Hide*.
7. Draw the required shape for the free area pin group guides.

Important

Free area pin group guides must be completely enclosed within the PR boundary of the block.

8. Either press `Enter` or double-click to indicate the end point.

   For polygonal guides, the polygon is auto-completed. Polygons should contain at least four vertices.

   For multi-segment guides, double-click or enter indicates the end point.

Important

Free area pin group guides must not overlap (partially or completely) with any boundary. If there is an overlap, the pin group guide is not created and a warning message is displayed in the CIW.

A free area pin group guide as per your specifications is created.

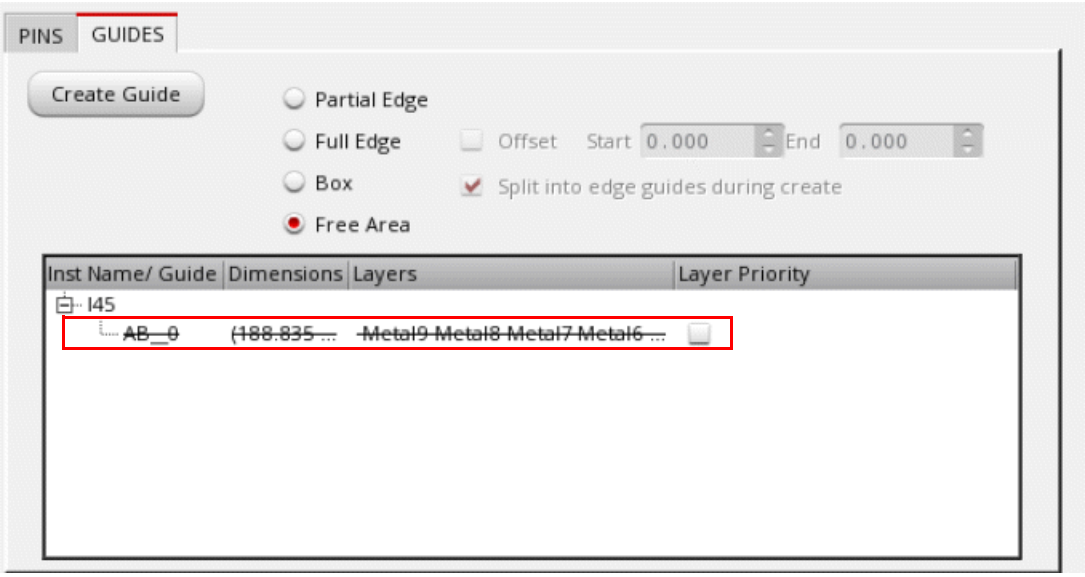
Deleting Pin Group Guides

To delete a guide:

1. Select the instance name from the list on the *GUIDES* tab.
2. Right-click to display the shortcut menu.

© 2020 All Rights Reserved.

3. Select *Delete Guide*. The selected guide is marked for deletion as displayed in the figure below:



4. Click *Apply* to remove the guide from the *Guides* list.

For more information about the area pin group guide constraint, see Area Pin Group Guide section in the *Virtuoso Unified Custom Constraints User Guide*.

For more information about the edge pin group guide constraint, see Edge Pin Group Guide section in the Virtuoso Unified Custom Constraints User Guide.

Related SKILL functions:

- Area Pin Group Guides:
  - `dbCreateAreaPinGroupGuide`: Creates an area pin group guide with the specified name in the given cellview.
  - `dbGetAreaBoundaryPinGroupGuide`: Returns the pin group guide associated with the specified area boundary.
  - `dbGetAreaPinGroupGuideAreas`: Returns the list of area boundaries associated with the specified area pin group guide.
  - `dbAddAreaToPinGroupGuide`: Adds an area boundary to the specified area pin group guide.
  - `dbGetAreaBoundaryValidRoutingLayers`: Returns the value of the validRoutingLayers constraint on the given area boundary.

© 2020 All Rights Reserved.

- dbAreaBoundaryHasLayerPriorityOrder: Returns a boolean value indicating that the layerPriorityOrder constraints set on the specified area boundary.
- dbAreaBoundaryHasValidRoutingLayers: Returns a boolean value indicating that the validRoutingLayers constraint is set on the specified area boundary.
- dbGetAreaBoundaryLayerPriorityOrder: Returns the value of the layerPriorityOrder constraint that is set on the specified area boundary.
- dbSetAreaBoundaryLayerPriorityOrder: Sets the value of the layerPriorityOrder constraint on the given area boundary.
- dbSetAreaBoundaryValidRoutingLayers: Sets the value of the validRoutingLayers constraint on the given area boundary.
- dbUnsetAreaBoundaryLayerPriorityOrder: Deletes the layerPriorityOrder constraint on the given area boundary.
- dbUnsetAreaBoundaryValidRoutingLayers: Deletes the validRoutingLayers constraint on the given area boundary.
- dbSubAreaFromPinGroupGuide: Removes an area boundary from the specified area pin group guide.

- Edge (Partial and Full) Pin Group Guide:
  - dbCreateEdgePinGroupGuide: Creates an edge pin group guide with the specified name in the given cellview.
  - dbGetPinGroupGuideEdgeBoundary: Returns the PR boundary associated with the specified edge pin group guide.
  - dbGetPinGroupGuideEdgeName: Returns the edge name of the specified edge pin group guide.
  - dbGetPinGroupGuideEdgeStartOffset: Returns the start offset of the edge for the specified edge pin group guide.
  - dbGetPinGroupGuideEdgeStopOffset: Returns the stop offset of the edge for the specified edge pin group guide.
  - dbRemovePinGroupGuideEdge: Removes the PR boundary and edge name specification from the specified edge pin group guide.
  - dbSetPinGroupGuideEdge: Specifies new or updates existing PR boundary edge information for the edge pin group guide.

© 2020 All Rights Reserved.

- ❑ `dbSetPinGroupGuideEdgeStartOffset`: Sets the start offset of the edge on the edge pin group guide.
- ❑ `dbSetPinGroupGuideEdgeStopOffset`: Sets the stop offset of the edge on the edge pin group guide.

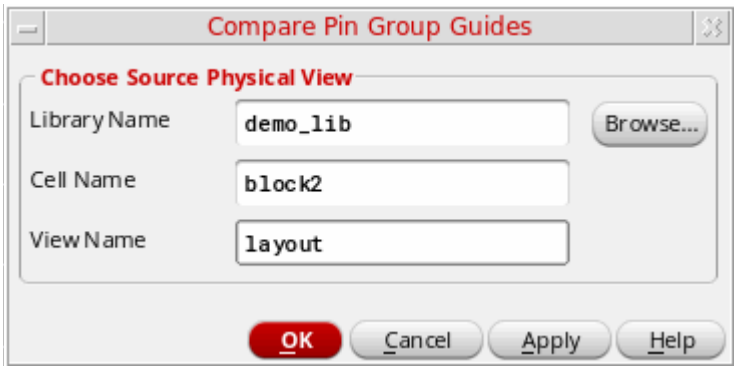
■ Other related SKILL functions:

- ❑ `dbFindPinGroupGuideByName`: Searches for the pin group guide with the specified name in the given cellview.
- ❑ `dbGetFigPinGroupGuide`: Returns the pin group guide associated with the specified figure.
- ❑ `dbGetPRBoundaryPinGroupGuides`: Returns the list of pin group guides associated with the specified PR boundary.
- ❑ `dbGetPinGroupGuides`: Returns the list of pin group guides associated with the specified cellview.
- ❑ `dbGetPinPinGroupGuides`: Returns the list of pin group guides associated with the specified pin.
- ❑ `dbAddFigToPinGroupGuide`: This function adds a via or a non-text shape to the specified pin group guide.
- ❑ `dbSubFigFromPinGroupGuide`: Removes a via or a non-text shape from the specified pin group guide.
- ❑ `dbSubPinFromPinGroupGuide`: Removes a pin from the specified pin group guide.
- ❑ `dbPinGroupGuideHasValidRoutingLayers`: Returns a boolean value indicating whether the validRoutingLayers constraint is set on the specified area pin group guide.

© 2020 All Rights Reserved.

Comparing Pin Group Guides

Select *Floorplan – Pin Group Guides – Compare* to display the Compare Pin Group Guides form.



Important

In Layout EXL, choose *Plan – Pin Planning – Compare Pin Group Guides*.

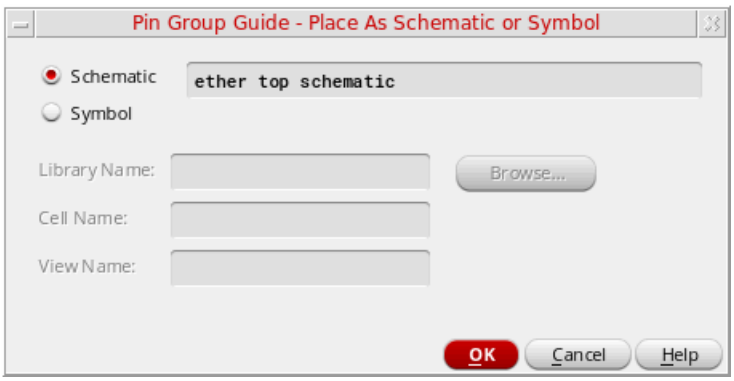
Use this form to compare the pin group guides in two different cellviews. Either type the `lib:cell:view` of the source cellView or click Browse and select the cellView.

© 2020 All Rights Reserved.

Placing Pin Group Guides as per Schematic or Symbol

Virtuoso Floorplanner lets you generate pin group guides and place pins as per their relative positions in the schematic or symbol view. Separate pin group guides are created for each side of the PR boundary, and pins are distributed as per their specified positions in the schematic or symbol view. To create a pin group guide as per the relative positions of pins in the schematic view:

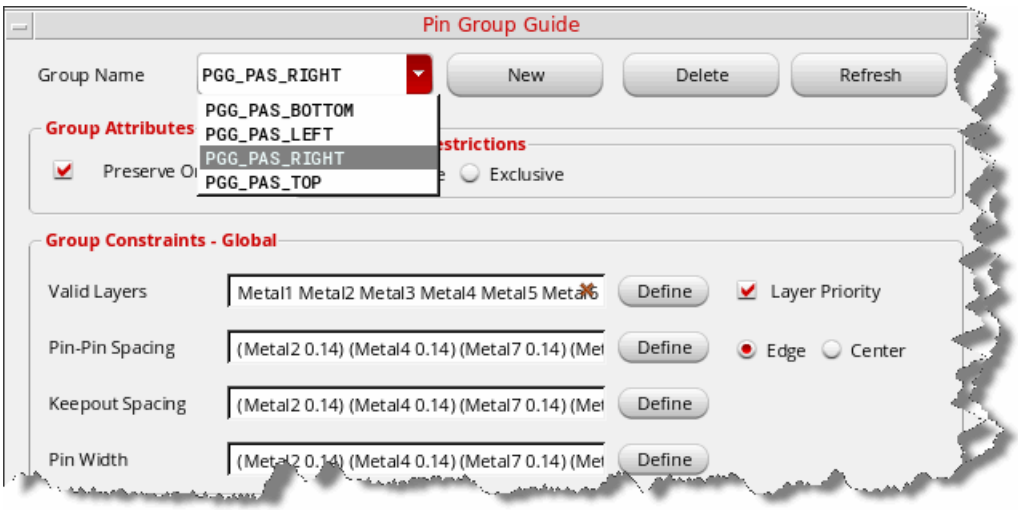
1. Choose *Floorplan – Pin Group Guide – Place As Schematic/Symbol*.



Important

In Layout EXL, choose *Plan – Pin Planning – Place Pin Group Guide as Schematic/Symbol*.

2. Choose *Schematic* or *Symbol*, as per your requirements. Default is *Schematic*.
3. Click *OK*. The Pin Group Guide form is displayed.



© 2020
All Rights Reserved.

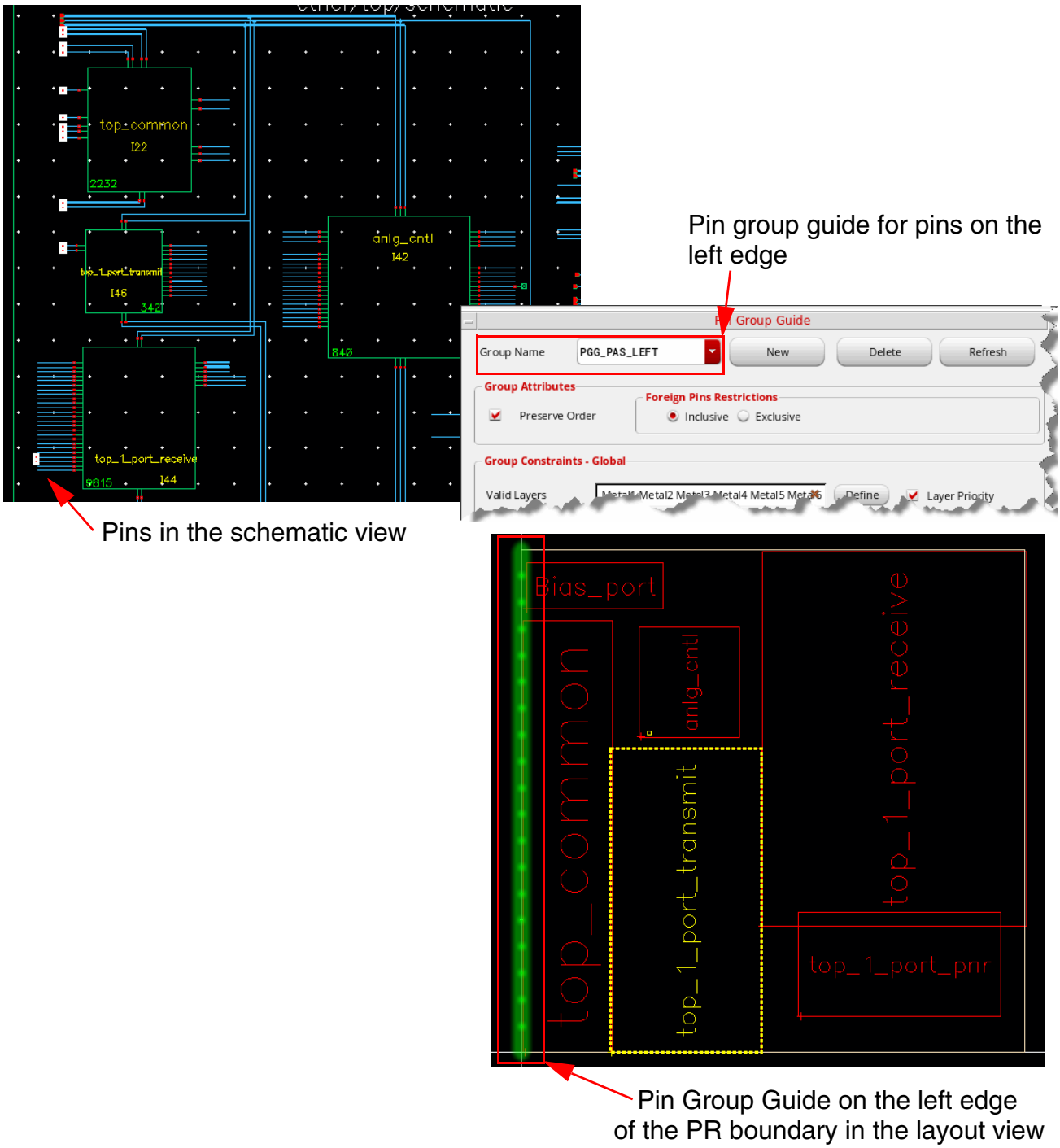
Group Name lists four different pin group guides, one for each PR boundary edge. Any existing pin group guides and alignment constraints in the design are deleted.

Note: Pin group guides are created only for the edges that have pins in the schematic view. For example, consider a schematic with pins only at the top and left edges of the PR boundary. Here, only two pin group guides will be created, one at the top edge and the other at the left edge.

4. Click *OK*.

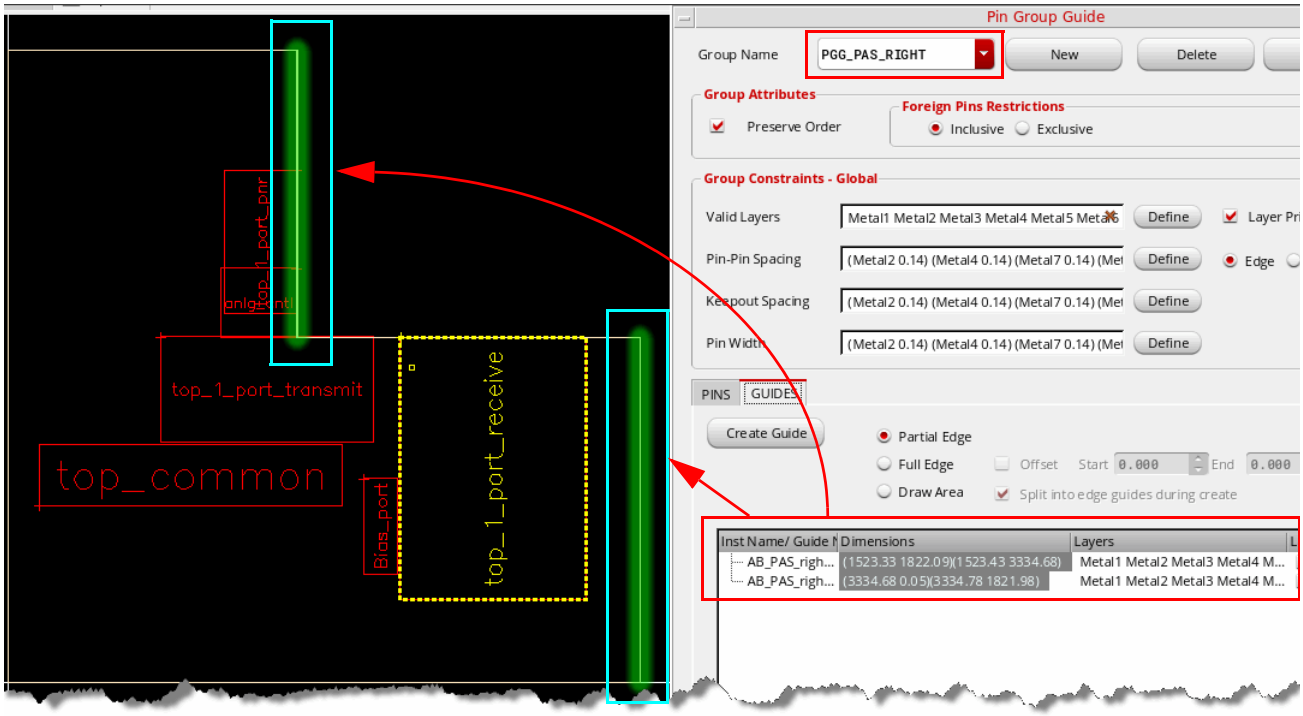
© 2020 All Rights Reserved.

Here is a snapshot of a pin group guide created on the left edge of a rectangular PR boundary:



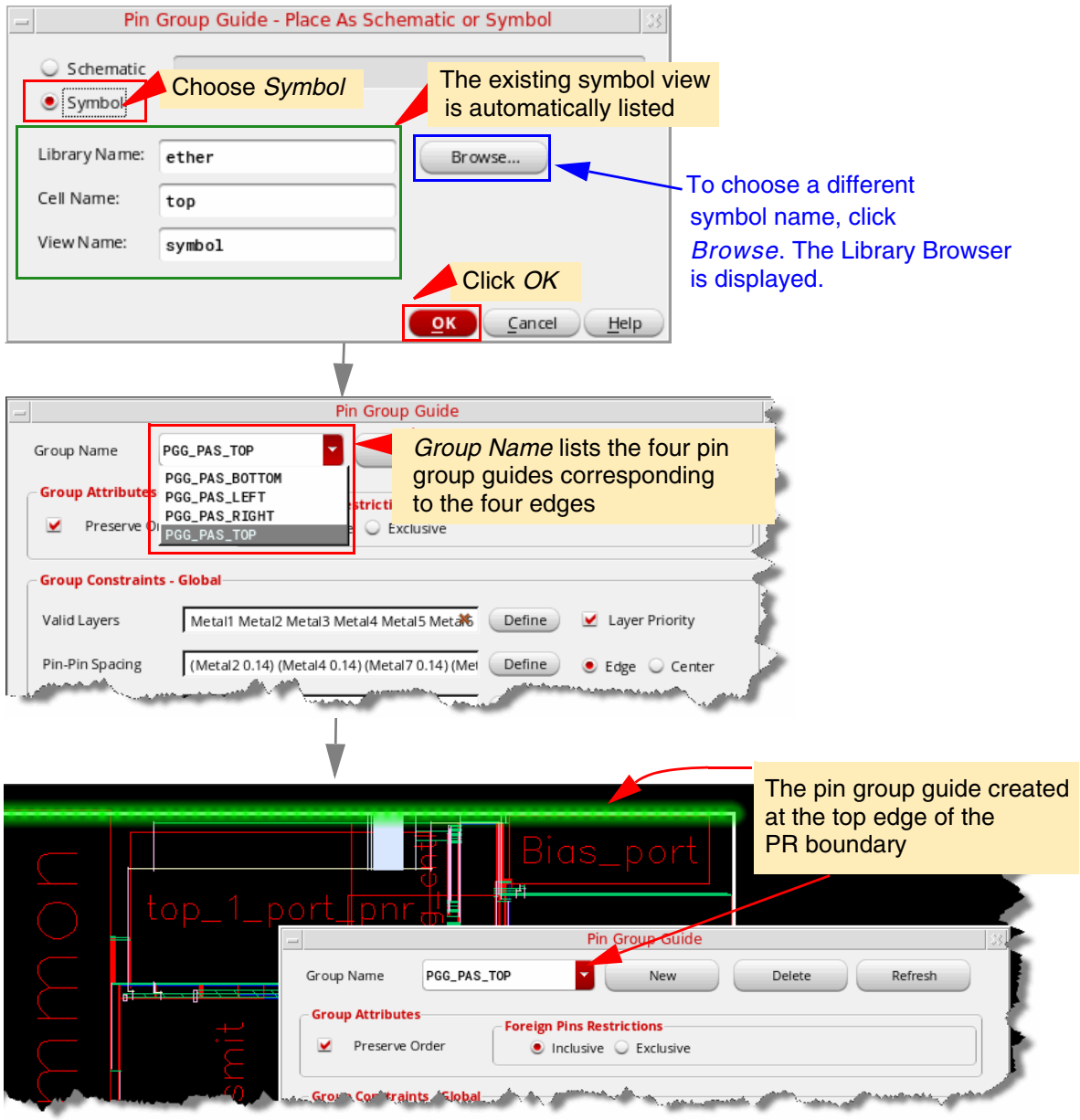
© 2020
All Rights Reserved.

In case of a rectilinear PR boundary, a pin group guide is created for each edge. In the following example, notice that two pin group guides are created that correspond to the two right edges.



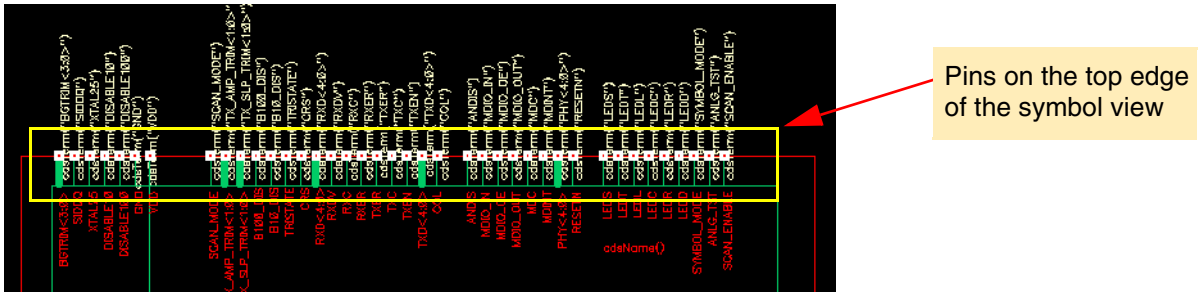
© 2020
All Rights Reserved.

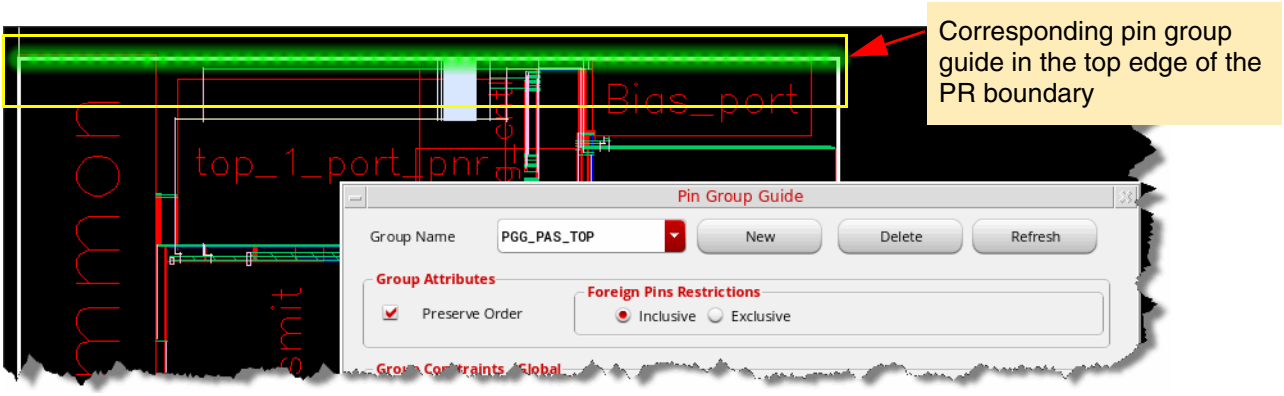
The following figure depicts the steps for creating pin group guides based on symbol views:



© 2020
All Rights Reserved.

Here is a quick view of pins at the top edge of the symbol view, and their corresponding pin group guide in the layout view:





© 2020
All Rights Reserved.

© 2020 All Rights Reserved.

8

Pin Placement

This chapter describes Virtuoso® Floorplanner's capability to place and optimize pins. This chapter discusses the following topics:

- Pin Planning
  - Using the Pin Placement Form
- Pin Optimization
  - Using the Pin Optimization Tab
  - Synchronizing Pin Optimization Options between Command Line and GUI
  - Alignment Issues During Pin Optimization
  - WSP/SP Support (ICADVM20.1 Only)
- Using a Pin Constraints File
- Running the Pin Checker (ICADVM20.1 Only)

After placing blocks, next you plan and optimize pins in the layout design. Use either *Floorplan – Pin Planner* or *Floorplan – Pin Optimization* command to open the Pin Placement form. The Pin Placement form includes the Pin Planning and Pin Optimization tabs.

Important

In Layout EXL, these commands are available under *Plan – Pin Planning*.

Use Pin Planner to:

- Assign or refine pin constraints, such as the side constraint (Left, Right, Top, Bottom), placement status (Fixed, Locked), and pin order
- Assign or refine pin attributes, such as pin layer and size

© 2020 All Rights Reserved.

When an alignment constraint is defined, a basic placer engine distributes the pins on the edge. For example, if five pins are assigned to the left edge, then the left edge is divided into five sections and the pins and evenly distributed. However, any existing shapes, blockages, or spacing rules on these edges are not considered. Pin planner only provides a visualization of the placement of the pins.

Pin Optimizer:

- Is a powerful algorithm that intelligently decides where to place the pins in design for optimal wire length, while honoring the constraints
- Determines the placement of pins based on several factors, such as the minimal net length between the pins, routability of pins, and DRC-correct placement, while honoring the blockages, all constraint sets on the pins, including the ones done through Pin Planner.

Important

The Pin Placement form supports level-1 editing even when the top design is open in the read-only mode or when the top design has no pins. You can descend into and edit the soft blocks of such designs.

© 2020 All Rights Reserved.

Pin Planning

This section explains how you can use the pin planning functionality to set constraints on and plan the placement of the top-level or level-1 pins in your design.

Tip

You can use Pin Planner functionality only if there is a place and route boundary available in the layout view. To generate a place and route boundary, use the *Connectivity – Generate – All From Source* command. For more information, see Design Boundary in the *Virtuoso Layout Suite XL User Guide*.

Using the Pin Placement Form

To use the Pin Placement form:

1. Open the Pin Placement form.

© 2020 All Rights Reserved.

The *Pin Planner* tab is displayed listing all the pins in the current scope in the pin table.

The entries are expressed as *termName:pinName:figName* because the pin figure is the physical entity that you are placing. If a pin has more than two pin figures attached to it, but the pin figures have the same name, they are represented by a single entry in the pin table.

The constraints generated by Pin Planner also use the *termName:pinName:figName* convention, thus enabling the constraints system to distinguish between pin figures in constraints from pins. This format is used only when the pin has more than one figure. If

© 2020 All Rights Reserved.

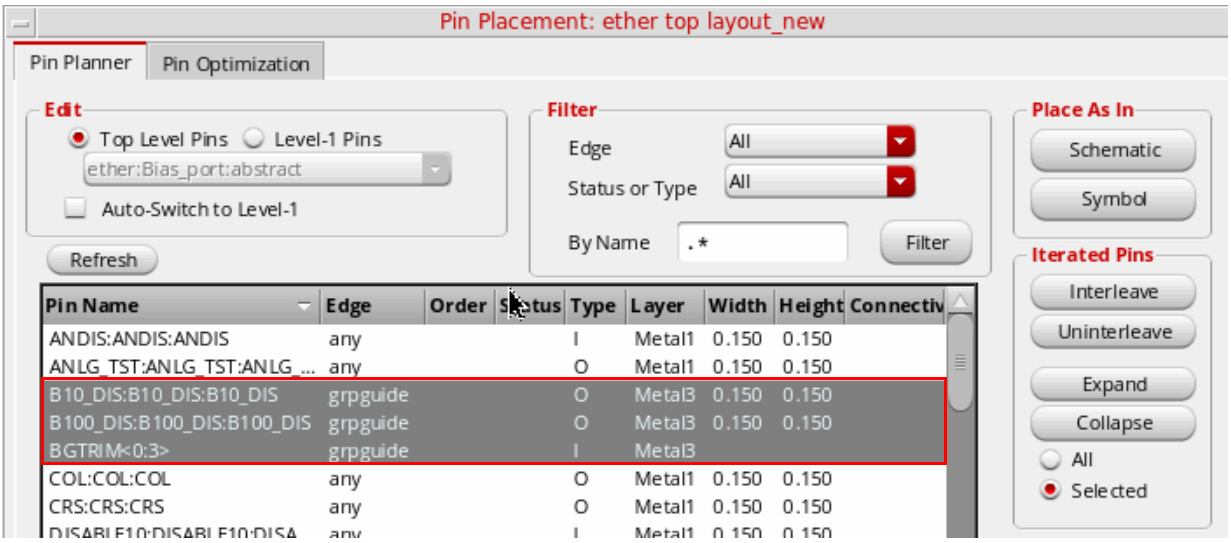
a pin has only one figure then the old format *termName:pinName* is used when creating the constraint.

You can filter and otherwise manipulate the list using the controls provided. Click the *Pin Name* column heading to view an alphanumeric listing of the pins. For example:

- `APIN` is listed before `BPIN`
- `APIN2` is listed before `APIN19`
- Bus bit `APIN<4>` is listed before bus bit `APIN<18>`

When a pin is selected in the pin table, Pin Planner recognizes the corresponding net PRO and displays the values in the *Layer*, *Width* and *Height* fields. You can edit these values manually, while ensuring that the net PRO constraint values are not violated. If the specified pin size is less than the `minWidth` defined in net PRO, then the pin size is reverted to its original value. If the specified pin *Width* or *Height* is equal to or greater than the net PRO value, then the new value is applied. For more information, see Support for Net PROs in Virtuoso Floorplanner.

Pin Planner supports the placement and distribution of member pins in pin groups in the specified guide region.



You can continue to use Pin Planner to create and display side (alignment) constraints for pins. However, if you assign an alignment constraint to a pin that is already part of a pin group guide constraint, then an error message is displayed in the CIW.

(ICADVM20.1 Only) The *Connectivity* column annotates whether pins are connected to buried pins or multiple hard block pins.

2. Use this form to do one or more of the following.

© 2020 All Rights Reserved.

- Place pins in the same relative positions as in the schematic view. Click *Schematic* in the *Place As In* section.
- Place pins in the same relative positions as in the symbol view. Click *Symbol* in the *Place As In* section. Pin Planner looks for a symbol view at the same location as the schematic view, followed by the layout view. If not found at both locations, an appropriate warning message is displayed and pins are not placed.
- Place pins in fixed positions on the boundary. See:
  - Placing a Pin on a Boundary Edge
  - Placing Ordered Pins on a Boundary Edge
- Place pins in fixed positions that are not on the boundary. See:
  - Placing a Pin in a Fixed Position not on the Boundary.
- Place pins directly on lower-level instance terminals on the same layer. See:
  - Placing a Pin on a Lower-level Instance Terminal
- Align level-1 instance pins with the top-level routes. See:
  - Aligning Level-1 Pins to Top-Level Routes
- Align soft block pins with the level-1 routes. See:
  - Aligning Level-1 Pins with Level-1 Routes
- (ICADVM20.1 Only) Place soft block or top-level pins that are connected to multiple hard block pins. See:
  - Placing Pins that Are Connected to Hard Block Pins (ICADVM20.1 Only)
- Place level-1 pins below top-level buried pins. See:
  - Placing Buried Pins (ICADVM20.1 Only)
- Place pin groups with any given pin pitch spacing. See:
  - Spacing Pins
- Create vertical and horizontal rails. See:
  - Converting a Pin into a Rail
  - Converting a Rail into a Pin
- Report pins and their assigned locations.
- Expand and collapse iterated bus pins (`Q<7:0>` = `Q<7>`, `Q<6>`, `Q<5>`, and so on.)

© 2020 All Rights Reserved.

- Handle multiple pins on the same net, `gnd.1 gnd.2 gnd.3`.

3. When you have finished setting pin constraints, use the *Re-Place Pins* pull-down to specify which pins can be moved by the placer. For more information about these options, see Pin Optimization.

4. Click *Apply* to set the new attributes for the specified pins.

   Note: Pins with placement status as “none” or “unplaced” will not be reinitialized. Instead, a warning message is issued indicating the presence of such pins.

5. Click *Defaults* to return all pins to their default positions.

Important

Using both the Pin Placement form and the Constraint Manager to constrain pins can result in conflicting and redundant constraints. Create pin constraints using *either* the Pin Placement form or the Constraint Manager and its associated SKILL functions.

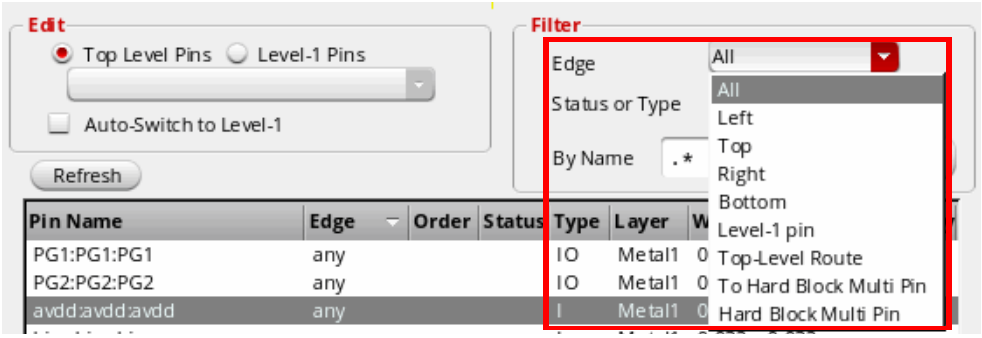
Placing a Pin on a Boundary Edge

To place a pin in a fixed position on the boundary,

1. From the layout window menu bar, choose *Floorplan – Pin Planner*.

   The Pin Placement form is displayed.

2. Choose *Unplaced* from the *Status or Type* drop-down list.



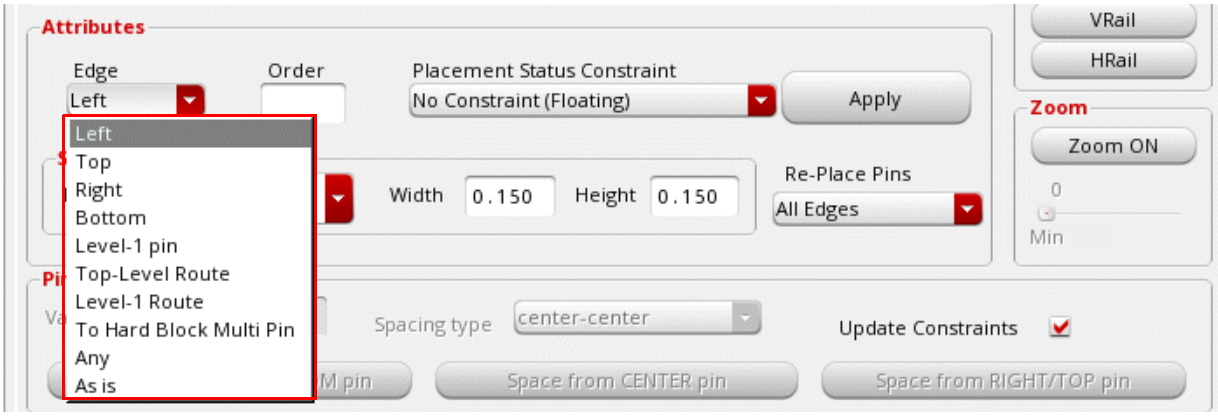
3. Select the pin you want to place from the pin table.

   You can expand iterated bus pins (for example, `A<7:0>`) using the *Expand* button in the *Iterated Pins* group box. This expands the bus into individual pins, which you can then place individually with different constraints.

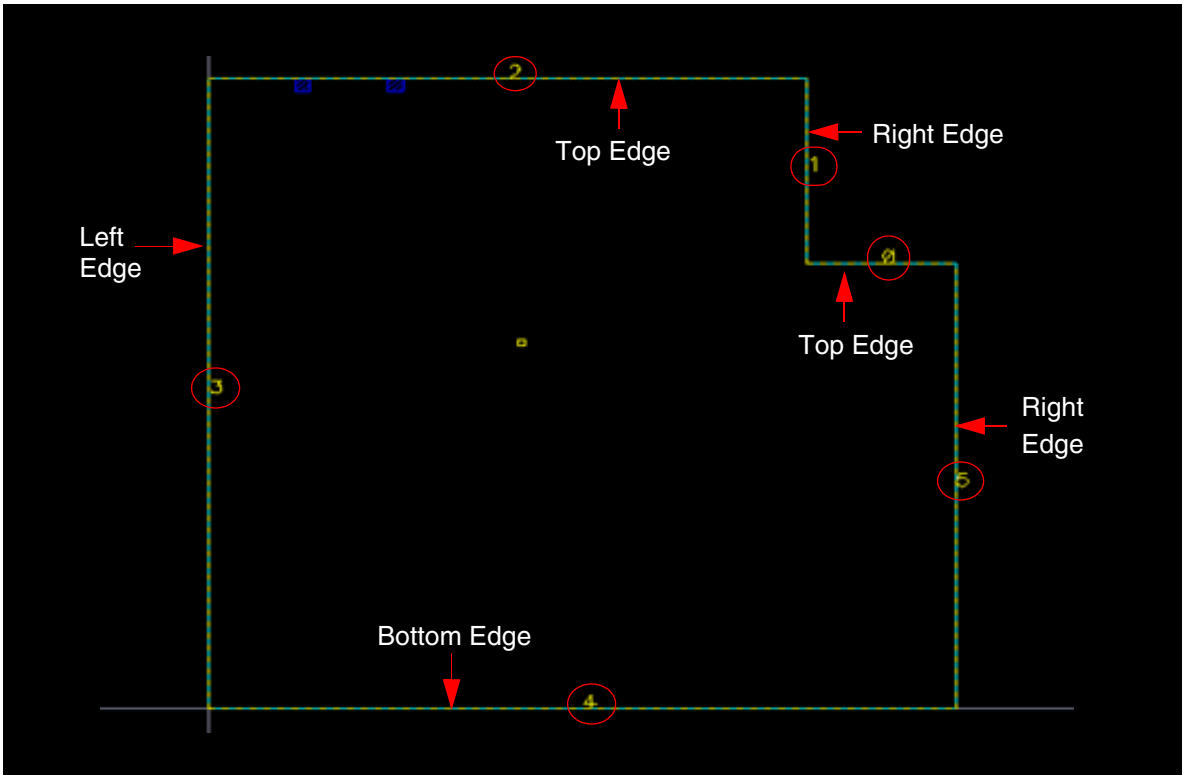
© 2020 All Rights Reserved.

Note: Use the enableNoPinsBetweenBusPins environment variable to disable placement of other pins between bus pins.

4. From the *Edge* drop-down list, choose the edge on which the selected pins are to be placed. For rectangular boundaries, the edge sides are listed as *Left*, *Top, Right*, *Bottom, Level-1 pin, Top-Level Route*, *To Hard Block Multi Pin*, *Any,* and *As is*.



For rectilinear boundaries, both the edge numbers and the edge sides are listed.



© 2020
All Rights Reserved.

Edge numbers are assigned in the increasing order, starting with zero, which is assigned to the edge at the right-top vertex, followed by the adjoining edge in the anti-clockwise direction.

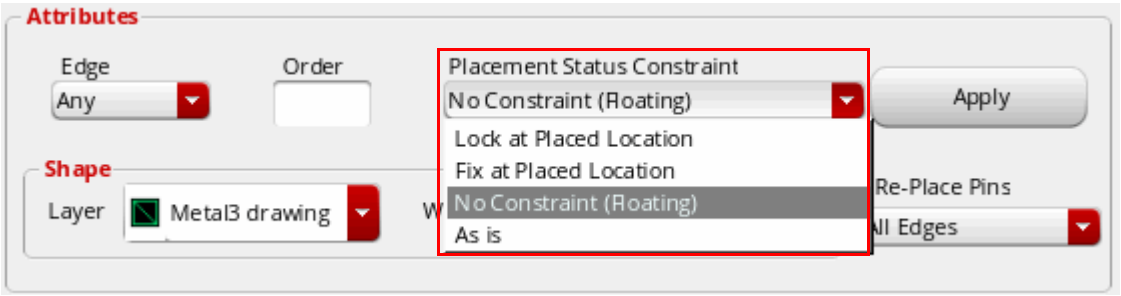
An edge side for rectilinear boundaries may correspond to one or more edges. In the above example, the "`Top`" edge side can correspond to edge numbers `0` or `2`. Therefore, Pin Planner has a choice of edges.

Note: The alignment Edge/Side constraint is disregarded and a warning message is displayed in the following situations:

- *Situation 1* - Edge number is specified for a cell with a rectangular boundary.
- *Situation 2* - The specified edge does not have enough available slots.
- *Situation 3* - The PR boundary is not a member of the alignment constraint.

For situations 1 and 2, although the alignment Edge/Side constraint is disregarded, the pitch value, if set, is honored. For situation 3, however, the pitch value is not honored.

5. Choose *Fix at Placed Location* from the *Placement Status Constraint* drop-down list.



Fix at Placed Location places the pin at a location on the boundary and then fixes the pin at that location, meaning that it cannot be moved by the automatic placement functions; however, you can move it manually, for example, using the *Edit – Move* command.

Tip

Use *Lock at Placed Location* to lock the selected pins after they are placed. A locked pin cannot be moved either by the automatic placement functions or manually by the user.

6. Click *Apply* to align the pin to the specified boundary edge in the layout.

Note: If required, use the *Re-Place Pins* pull-down to limit the pins that can be moved by the placer when you click *Apply*.

© 2020 All Rights Reserved.

The *Status* field is updated in the pin table and the label in the *Placement Status Constraint* field changes to show the coordinates of the center of the pin.

7. Repeat **step 3** through **step 6** for each pin you want to place on the boundary.
8. Click *Close* to close the Pin Placement form.

Placing Ordered Pins on a Boundary Edge

To place a group of pins in a specific order along a boundary edge:

1. From the layout window menu bar, choose *Floorplan – Pin Planner*.

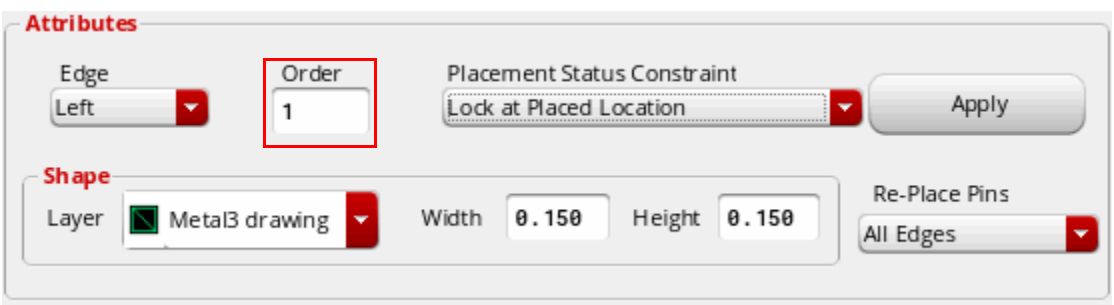
   The Pin Placement form is displayed.

2. Select one or more pins from the pin table.

   Change the order of the pins by selecting a pin in the list and clicking on of the arrows that surround the *Move pins* label. The up and left arrows move items up in the list; the down and right arrows move items down.

   Use the *Swap order* option to swap the position of two selected pins.

3. Choose the edge you want from the *Edge* drop-down list.
4. Type a number in the *Order* field.



You can type any number into the field; the selected pins are assigned an integer based on the order in which they are listed in the pin table.

When an *Order* is specified, the *Width* and wire *Type* values are ignored. Pins are placed in adjacent tracks following the specified order. Set the enableTrackSkipping environment variable to `t` to honor the pin order along with the width and wire type values. If the adjacent track does not meet the required wire type or width requirements, then tracks are skipped until a suitable track is found.

Note: By default, enableTrackSkipping is set to `nil`. Therefore, pins with order constraint are placed in continuous adjacent tracks meeting the order constraint. The

© 2020 All Rights Reserved.

wire type and width requirements are ignored.

5. Click *Apply* to apply each pin assignment.

   Note: If required, use the *Re-Place Pins* pull-down to limit the pins that can be moved by the placer when you click *Apply*.

   The pins are aligned to the specified boundary edge in the layout in the order you specified.

Placing a Pin in a Fixed Position not on the Boundary

To place a pin in a fixed location not aligned with the boundary,

1. From the layout window menu bar, choose *Edit – Move*.
2. In the layout window, select an unplaced pin and move it to the required location.
3. From the layout window menu bar, choose *Floorplan – Pin Planner*.

   The Pin Placement form is displayed.
4. Select the pin in the pin table.
5. Choose *Fix at Placed Location* from the *Placement Status Constraint* drop-down list.
6. Click *Apply*. The pin is fixed at its current location and is not aligned with any edge.

Note: You can optionally assign the pin to an edge immediately prior to step 5. If you do this, the pin is fixed at the location but the pin width and design rules are projected to the specified edge. This projection leaves open a channel on the aligned edge for the fixed pin to facilitate better routing from outside the boundary.

Placing a Pin on a Lower-level Instance Terminal

To place a pin directly onto its corresponding instance terminal at a lower level of the design hierarchy:

1. From the layout window menu bar, choose *Floorplan – Pin Planner*. The Pin Placement form is displayed.
2. Select the pin from the pin table.
3. Choose *Level-1 pin* from the *Edge* drop-down list.

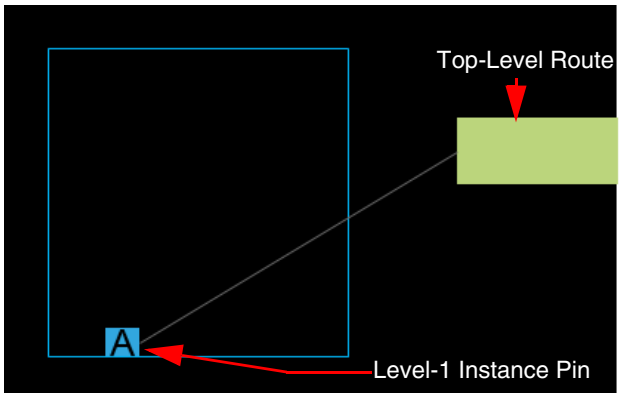
© 2020 All Rights Reserved.

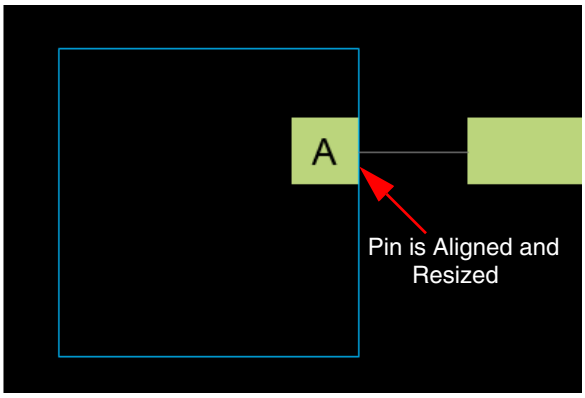
4. Choose *No Constraint (Floating)* from the *Placement Status Constraint* drop-down list.
5. Click *Apply*. The pin you selected is placed over its corresponding instance terminal in the layout window.

The *Connectivity* column in the pin table annotates whether pins are connected to buried pins or multiple hard block pins. Pins are re-layered and re-sized automatically. Pin labels are updated if the matchLabelLayerWithPin environment variable is set to `t`.

Aligning Level-1 Pins to Top-Level Routes

Pin Planner lets you align level-1 instance pins with the top-level routes to which they are connected. You can also choose to resize the pins to match the dimensions of the corresponding top-level route, as shown in the following diagram:





To align level-1 instance pins with the top-level routes:

1. Select *Level-1 Pins* in the *Edit* section of Pin Planner.
2. Select *Top-Level Route* from the *Edge* drop-down list.
3. Click *Apply*.

The pin resize behavior is controlled by the alignMatchRefSize environment variable. Set it to `t` to enable pin resizing.

Aligning Level-1 Pins with Level-1 Routes

Pin Planner lets you align level-1 pins with level-1 routes that belong to the same net. You can also use environment variables to define the following pin settings:

- Relayer the level-1 pin to the same layer as the level-1 route: alignMatchRefLayer

© 2020 All Rights Reserved.

- Resize the level-1 pin to match the width of the level-1 route: alignMatchRefSize

To align pins with the level-1 routes:

1. Set the required environment variables in the CIW.
2. Select the required pins in the *Edit* section of Pin Planner.
3. Select *Level-1 Route* from the *Edge* drop-down list.
4. Click *Apply*.

Alignment of level-1 pins with level-1 routes is supported in the following scenarios:

- In the Presence of a Guide Pin
- In the Absence of a Guide Pin

In the Presence of a Guide Pin

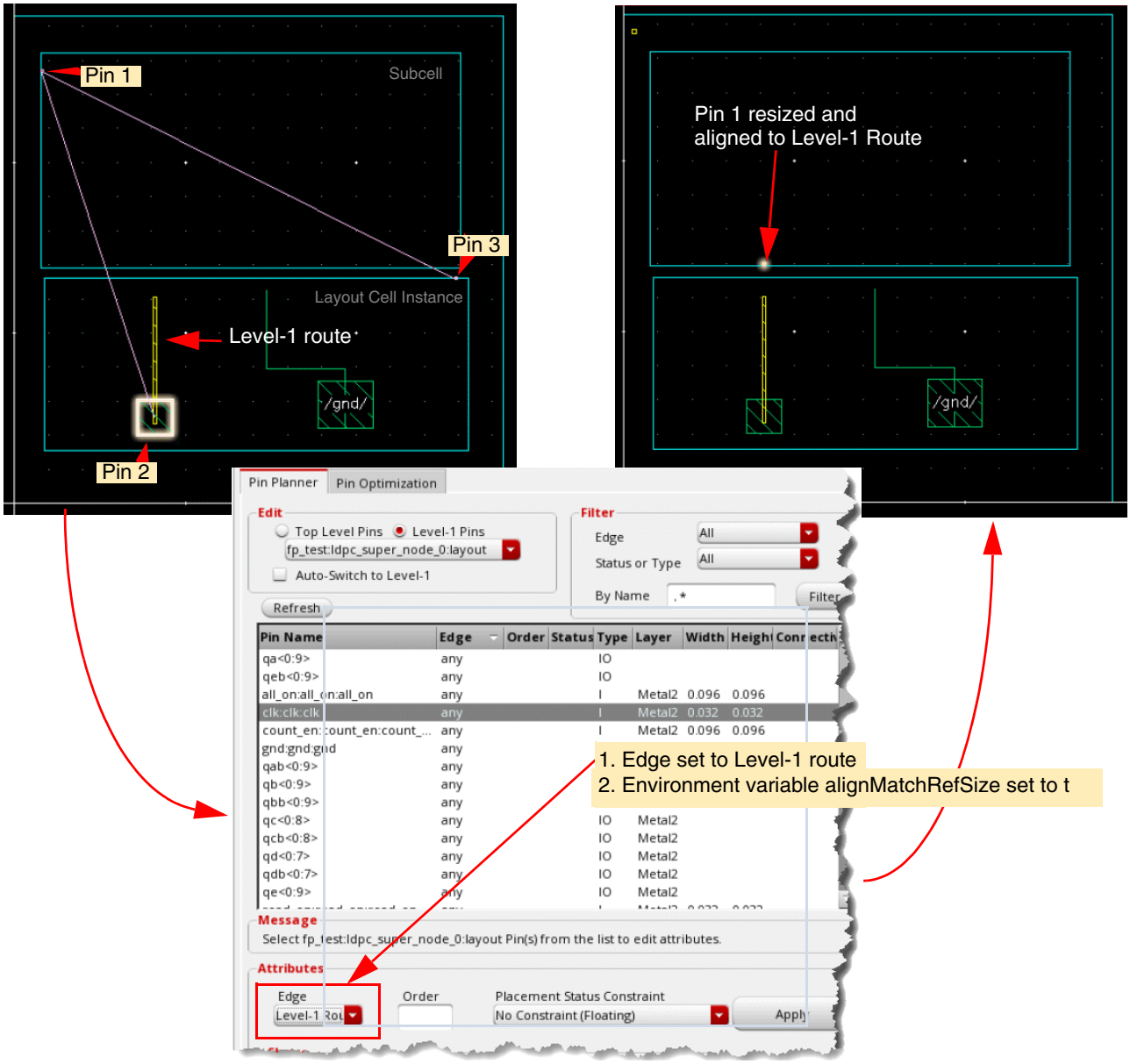
In the following example, a single net connects the following pins:

- Pin 1 - A level-1 pin `clk`, located in a subcell
- Pin 2 - A top-level pin placed on a level-1 route, within a layout cell instance
- Pin 3 - A level-1 pin, which is the guide pin, located in the layout cell

The alignMatchRefSize environment variable is set to t.

© 2020 All Rights Reserved.

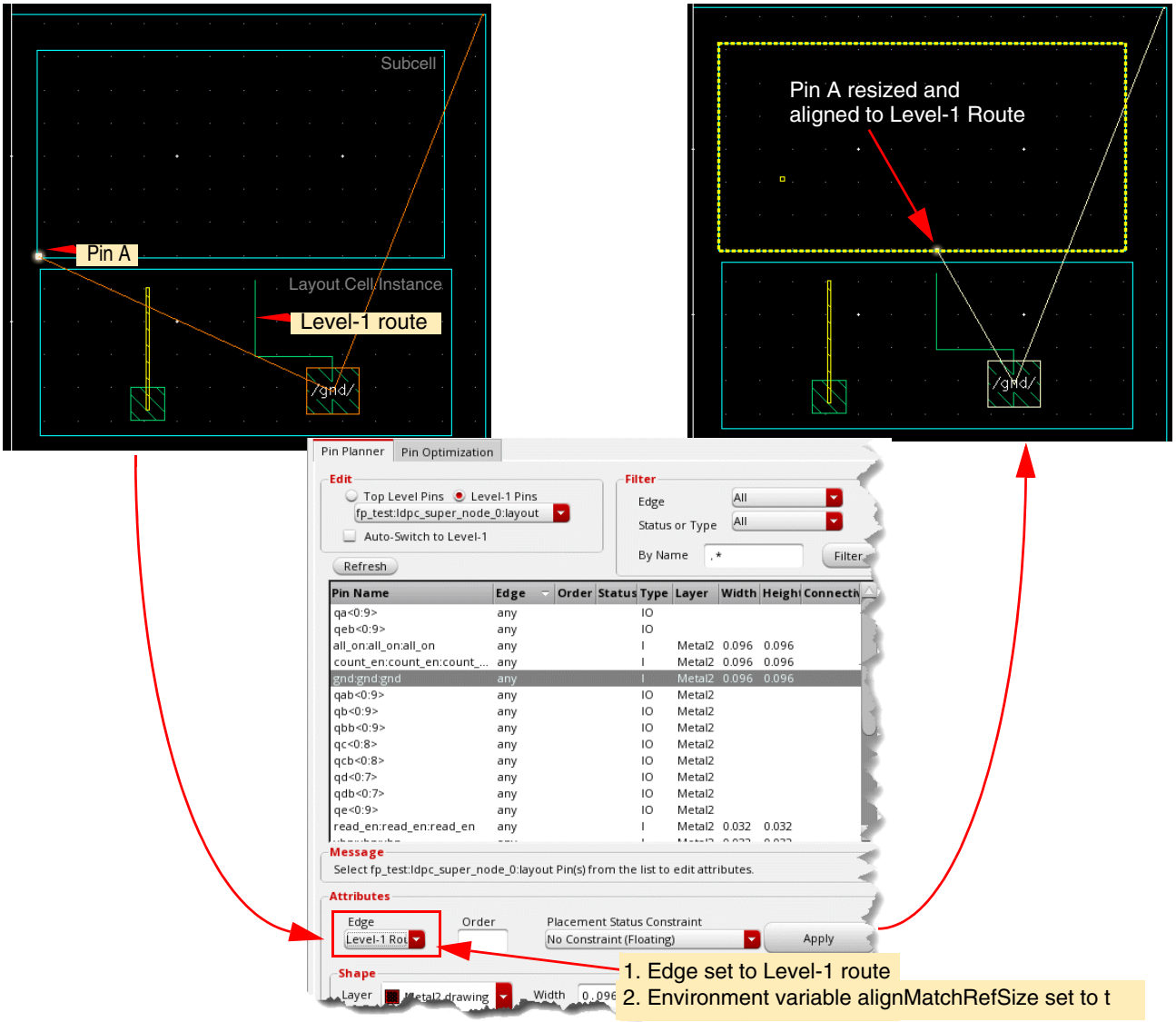
With the *Edge* set to *Level-1 Route*, Pin Planner aligns Pin 1 with the Level-1 route. The guide pin helps retain the net connectivity during pin placement and optimization.



© 2020
All Rights Reserved.

In the Absence of a Guide Pin

In the following example, there are no guide pins. Pin A, a level-1 pin, is located in a subcell. It is aligned with a Level-1 route and is resized as per the width of the level-1 route.



Placing Pins that Are Connected to Hard Block Pins (ICADVM20.1 Only)

There may be situations where one top-level or level-1 pin is connected to two or more hard block pins. To place such pins:

1. From the layout window menu bar, choose *Floorplan – Pin Planner*.

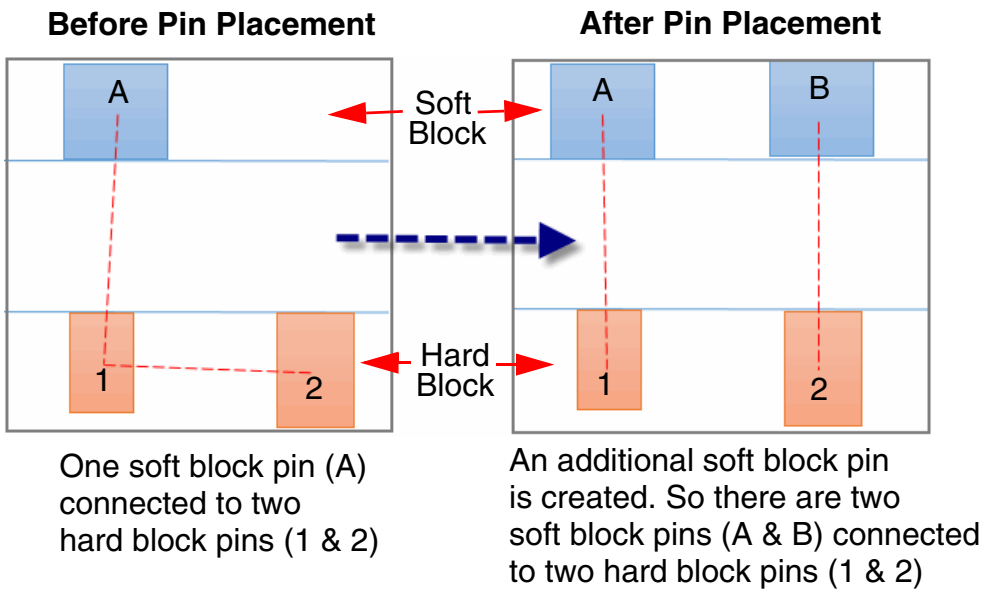
© 2020 All Rights Reserved.

The Pin Placement form is displayed.

2. Select the soft block or top-level pin from the pin table.
3. Choose *To Hard Block Multi Pin* from the *Edge* drop-down list.
4. Choose *No Constraint (Floating)* from the *Placement Status Constraint* drop-down list.
5. Click *Apply*.

The *Connectivity* column in the pin table annotates whether pins are connected to buried pins or multiple hard block pins.

Additional pins are created in the soft block (or at the top-level) to match the number of pins in the hard block. Therefore, each soft block or top-level pin is connected to a corresponding hard block pin, as shown below:



© 2020
All Rights Reserved.

A summary report is displayed in the CIW that provides detailed information about hard block pin placement:

```
================================================================================
To Hard Block Pin Report for
 Soft pin: "pad_resetb:P__80:P__80"
 Instance: "|I8 (fp_test:ldpc_digital:layout)"
================================================================================

Created and placed soft pins
----------------------------------------------
Creating and placing pin "pad_resetb:P__80:P__80_4 (|I8 (fp_test:ldpc_digital:layout))" corresponding to hard block pin
"pad_resetb:pad_resetb:pad_resetb (fp_test:ldpc_top_NODE:layout_WSP_TOP_demo)"
Creating and placing pin "pad_resetb:P__80:P__80_3 (|I8 (fp_test:ldpc_digital:layout))" corresponding to hard block pin
"pad_resetb:P__142:P__142 (fp_test:ldpc_top_NODE:layout_WSP_TOP_demo)"

Summary
----------------
Number of existing soft pins found:                              2
Number of existing soft pins which are placed                    2
Number of existing soft pins which are not placed                0

Number of created and placed soft pins:                          2

Total number of connected hard pins:                             4
Number of connected hard pins for which soft pins are placed:    4
Number of connected hard pins for which no soft pins are placed: 0
```

The summary report includes the following information:

- A list of messages generated while placing soft block pins that are connected to multiple hard block pin.
- Number of soft pins found.
- Number of soft block pins that are aligned to hard block pins.
- Number of soft block pins that could not be aligned to hard block pins.
- Number of new soft pins created.
- Number of hard pins found.
- Number of hard block pins to which soft block pins have been aligned.
- Number of hard block pins to which soft block pins could not be aligned.

Placing Buried Pins (ICADVM20.1 Only)

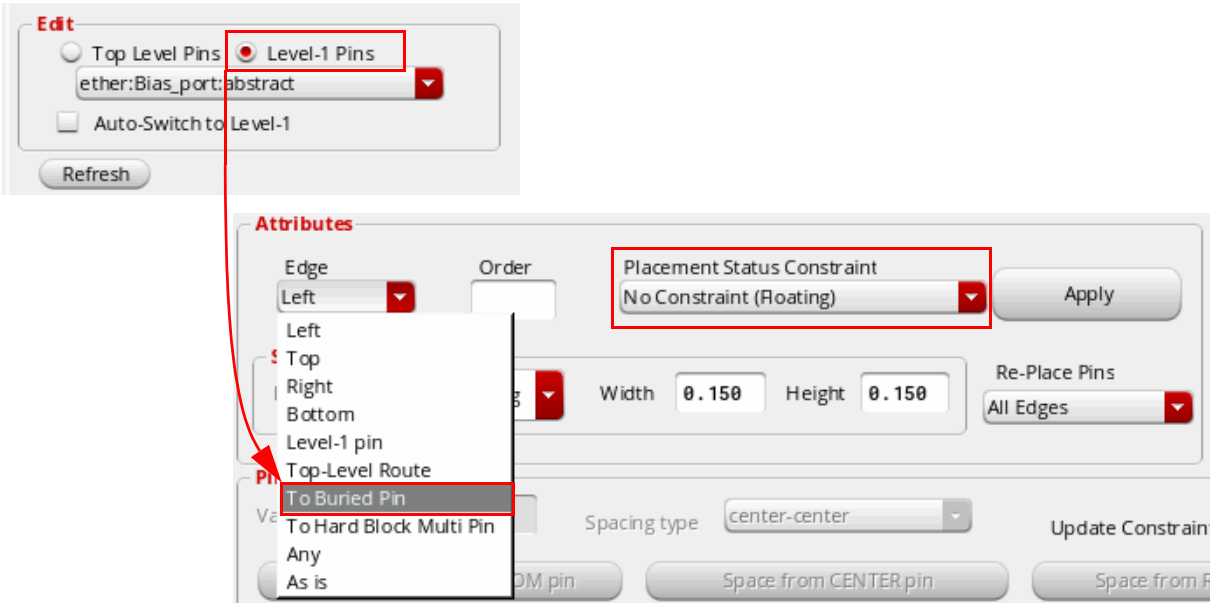
Buried pins are the top-level pins that are also bound by a level-1 PR boundary. To place a level-1 pin below a top-level buried pin:

1. From the layout window menu bar, choose *Floorplan – Pin Planner*.

© 2020 All Rights Reserved.

The Pin Placement form is displayed.

2. Choose *Level-1 Pins* in the *Edit* region.
3. Select the level-1 pin from the pin table.
4. Choose *To Buried Pin* from the *Edge* drop-down list.



5. Choose *No Constraint (Floating)* from the *Placement Status Constraint* drop-down list.
6. Click *Apply*.

The level-1 pin is moved below the top-level pin. Its size and layer are matched to the buried pin. The status of both the pins (level-1 pin and buried pin) is marked as *Fixed* in the *Status* column of the pin table.

Spacing Pins

Spacing values are enforced only between adjacent pins on an edge, even if the spacing is set between two non-adjacent pins. If you specify a spacing between a pin and an iterated pin, the software applies the spacing between the last pin of the iterated pin and the individual pin. If the specified spacing cannot be satisfied, you see a warning in the CIW and the pins are not moved.

To set the exact distance between two or more ordered pins or between the individual bits of a bus pin:

© 2020 All Rights Reserved.

1. From the layout window menu bar, choose *Floorplan – Pin Planner*.

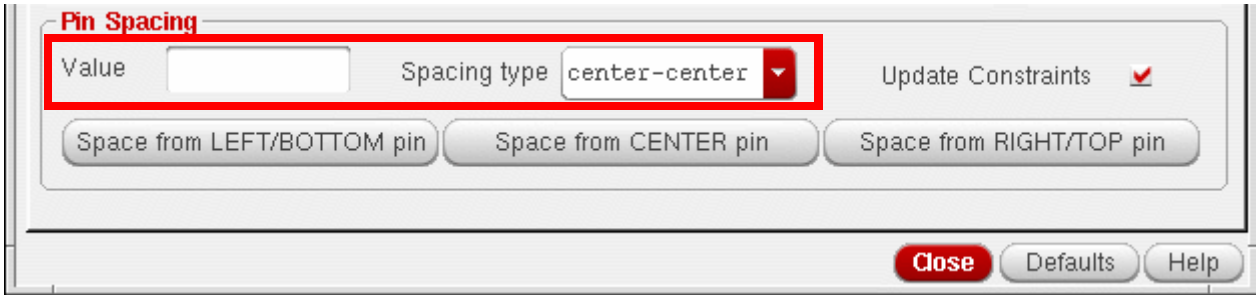
   The Pin Placement form is displayed.

2. Filter the list of pin names as required using the *Display* drop-down list.

3. Select the pins from the pin table.

4. In the *Pin Spacing* group box, type the spacing you require in the *Value* field, choose the type of spacing from the *Spacing type* pull down, and check *Update Constraints* to automatically update any constraints (for example, alignment constraints) associated with the selected pins.

   Note: If a pin to be spaced has the alignment constraint set up, the placer reads the constraint to determine the appropriate access direction for the pin.



5. Click one of the *Space* buttons, depending on which pin is to be the reference pin for the spacing operation. For information about the spacing options, see Pin Optimization.

Important

Pins are spaced and constraints updated only by clicking one of the *Space* buttons in this section and not by clicking the *Apply* button in the *Attributes* section of the Placement Planning form.

Converting a Pin into a Rail

To convert a pin into a rail,

1. From the layout window menu bar, choose *Floorplan – Pin Planner*.

   The Pin Placement form is displayed.

2. Choose *Unplaced* from the *Status or Type* drop-down list.

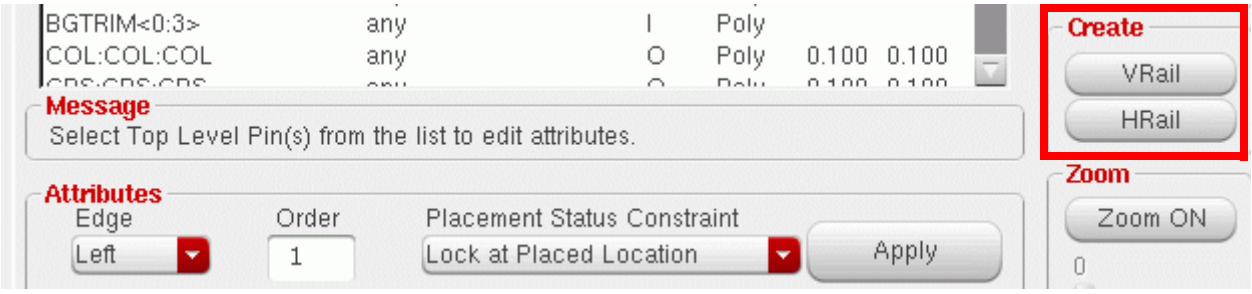
3. Select a pin from the pin table.

   Note: To select multiple pins to be converted to a rail, hold down the `Ctrl` key and select

© 2020 All Rights Reserved.

the pins.

4. Choose an edge from the *Edge* drop-down list.
5. Click *Apply*. The pin is placed on the boundary. If you have selected multiple pins, they will all be placed on the boundary.
6. Select the pin in the layout and do one of the following:
   - Click *HRail* to convert it into a horizontal rail.
   - Click *VRail* to convert it into a vertical rail.



When a pin is changed into a rail, it becomes aligned with two edges of the boundary. For a horizontal rail, the alignment is to the left and right edges. For a vertical rail, the alignment is the bottom and top edges. The rail stretches with the boundary in the Constraint-Aware Editing mode.

Important

The Pin Placement form does not distinguish layer information when the pins are placed on the boundary edges. For example, you cannot convert a `metal2` pin to a rail if that pin occupies the same location as a `metal1` pin.

7. (Optional) Use the Constraint Manager Assistant to view the alignment constraints on the rail.

Converting a Rail into a Pin

To convert a rail back into a pin:

1. From the layout window menu bar, choose *Floorplan – Pin Planner*.

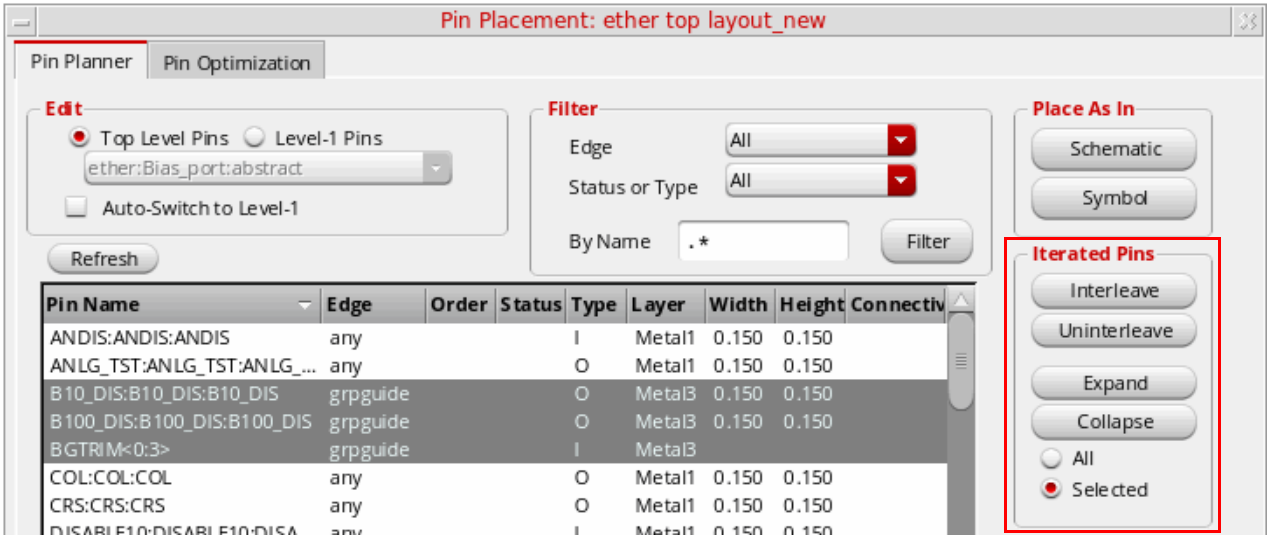
   The Pin Placement form is displayed.
2. In the layout window, select the rail you want to change.
3. In the Pin Placement form, click *unHRail* or *unVRail* as required.

© 2020 All Rights Reserved.

The rail is converted back into a pin and returned to its original size and edge, regardless of the railing direction.

Interleaving the Bits of Iterated Pins

Use the Pin Placement form's *Interleave* button to interleave the bits of the selected iterated pins and automatically apply an order constraint to constrain how the individual bits will be placed.



The selected set must contain at least one pin already assigned to an edge. Any unplaced pins are automatically placed on the same edge and their bits are interleaved. Interleave does not work if the pins in the selected set are assigned to more than one edge or to no edge at all. In this case, the command generates an error message and does nothing.

Important

You can no longer select only unplaced pins, select an *Edge* in the *Attributes* section, and then press *Interleave*. The edge must be specified before you select the pins to be interleaved.

If you select both buses and single pins, interleave processes them in two steps: first it interleaves the buses, then it gets the single pin array, splits it into two parts, and interleaves those also.

For example,

Before Interleave:

```
A<0:2> A1 B<0:2> B1 C<0:1> C1 D1 E1 F1
```


© 2020 All Rights Reserved.

After Interleave:

```
A<0> B<0> C<0> A<1> B<1> C<1> A<2> B<2> A1 D1 B1 E1 C1 F1
```

The *Uninterleave* command works with pins assigned to any number of edges. However, it does not move pins from one edge to another, but instead deals with the pins on each edge separately. *Uninterleave* does nothing with unplaced pins - they are ignored.

© 2020 All Rights Reserved.

Pin Optimization

Pin optimization is done to position pins of blocks in a manner that helps obtain the shortest possible net length at a particular level in the design. Pin optimization must be done even for blocks that are placed by the block placer because the aggregate length of nets connecting pins of the blocks may not be optimal or the shortest. Pin optimization should be done before routing the design.

The table lists the objects that you can select for pin optimization.

Object	Description
top-level and soft block pins	Selected top-level and soft-block pins are optimized.
top-level PR boundary	All top-level pins are optimized.
Soft block instances	Pins in the selected soft block instances are optimized.
Nets	Pins associated with the selected nets are optimized.

Note: When the soft block pins are optimized using Pin Optimizer, the soft block master becomes editable automatically.

Pin Optimizer tries to align all connected pins in the design, including the top-level pins connected to level-1 pins and the level-1 pins connected to other level1-pins.

While creating a layout, you should first place the top-level and level-1 pins and then optimize the placed pins. You can place pins using the *Pin Planner* tab in the Pin Placement form. For more information on how to place pins, refer to Pin Planning. Using the *Pin Planner* tab, you can also set the side constraints on the top-level and the level-1 pins.

You can run Pin Optimizer either using *Floorplan – Pin Optimization* or through the `vpaOptimizePins` command-line SKILL API.

Important

Pin optimization cannot be done if the total net length of a design is zero. Pin Optimizer, when run for such designs, issues a warning message and exits.

Selecting Pins for Optimization

You can choose to optimize all pins or only a few selected pins in the design. To select the required pins, either select the individual pins or select the nets, instances, or PR boundary

© 2020 All Rights Reserved.

that contain the required pins. In addition, you can use the `VFP_EXT_MULTIFIG_THRESHOLD SHELL` environment variable to filter out the required pins from the selected pins. Use this environment variable to choose the nets that contain the required number of pins. For example, if the `VFP_EXT_MULTIFIG_THRESHOLD SHELL` environment variable is set to `2`, then the pins of only those nets that connect two pins or less are selected for optimization. Pins connected to the remaining nets are treated as fixed pins, and are therefore not optimized. Therefore, this environment variable serves as an additional check on the number of pins selected for optimization.

The default value of the `VFP_EXT_MULTIFIG_THRESHOLD SHELL` environment variable is `50`, which is used by Pin Optimizer in the default mode. To optimize the pins of nets that have not been selected as a result of the default value 50, increase the threshold value.

Examples:

To increase the threshold to value 100:

```
setShellEnvVar("VFP_EXT_MULTIFIG_THRESHOLD" "100")
```

To decrease the threshold to value 2:

```
setShellEnvVar("VFP_EXT_MULTIFIG_THRESHOLD" "2")
```

Support for net PRO definitions

Pin Optimizer recognizes net process rule override (net PRO) definitions in the design, and performs the following tasks:

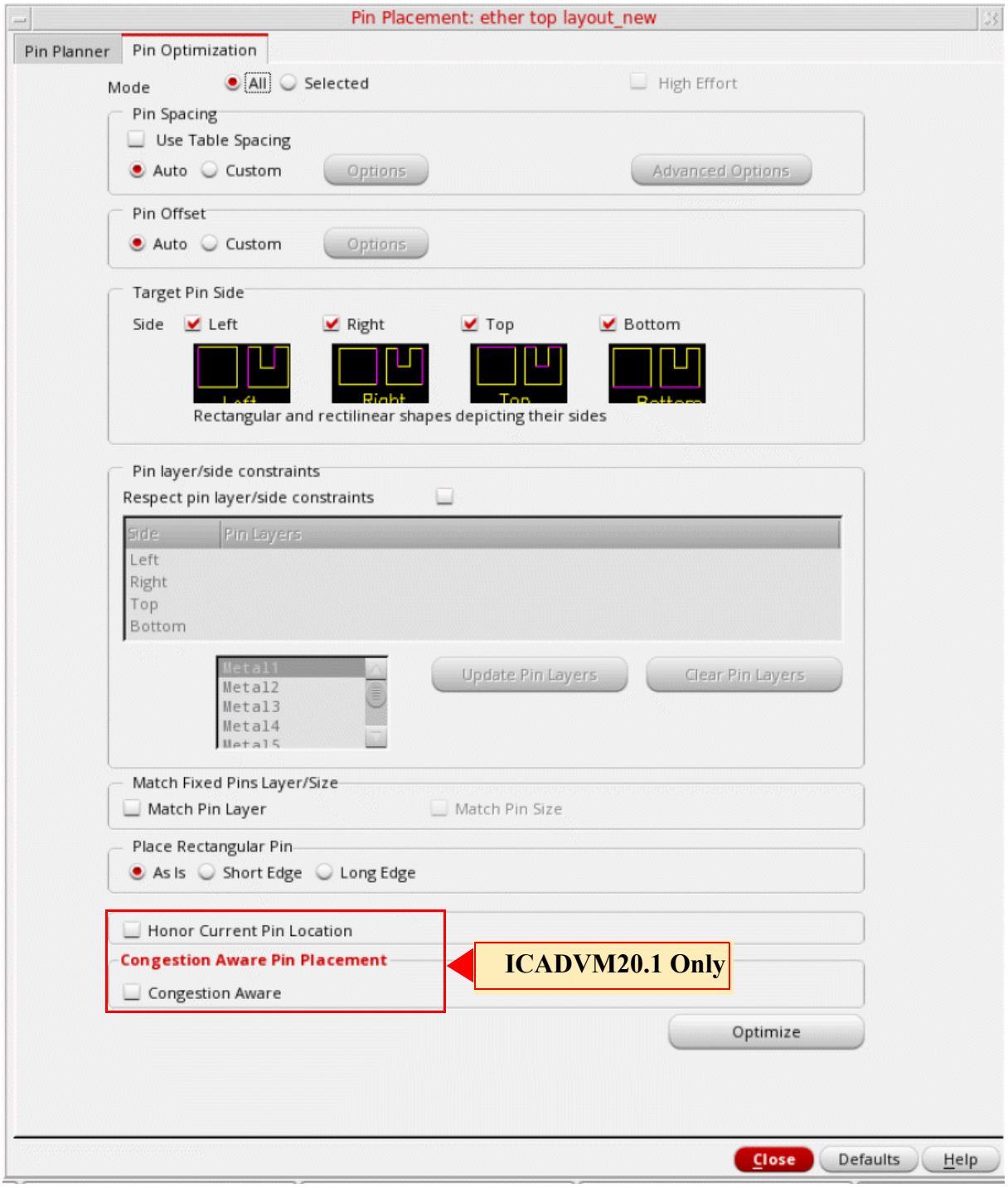
- Resizes the pins connected to the net to the minimum width specified in net PRO.
- Moves pins that are connected to the net to the valid layers specified in net PRO.
  - If there is a single valid layer, then all pins on the net are re-layered to the valid layer, and the pins are placed at feasible locations to ensure minimum net length.
  - If there is a list of valid layers, then the behavior depends on the state of the *Respect layer priority for valid routing layers constraint on net* option in the Floorplan Global Options form (Layout GXL Options form in IC6.1.8). For more information about this option, see Respecting Layer Priority for Valid Routing Layers.
- Applies the minimum spacing value mentioned in the net PRO to all pins in the selected net.

For more information, see Support for Net PROs in Virtuoso Floorplanner.

© 2020 All Rights Reserved.

Using the Pin Optimization Tab

Choose *Floorplan – Pin Optimization* to display the *Pin Optimization* tab of the Pin Placement form.



© 2020 All Rights Reserved.

The *Mode* is set to *All* or *Selected* depending on whether all or some of the pins are selected in the layout. You can change your initial selection by selecting the required *Mode* option in the form.

In the *Selected* mode, the *High Effort* option is available. With this option selected, Pin Optimizer runs extra iterations to get better results (better than normal run). Pin Optimizer checks all available slots to achieve the least possible net length. However, these improvements are limited to the pins placed on parallel edges.

The options on the *Pin Optimization* tab are grouped as follows:

- *Pin Spacing*
- *Pin Offset*
- *Bus Pins*
- *Target Pin Side*
- Pin Layer/Side Constraints
- Match Fixed Pins Layer/Size
- Place Rectangular Pin

Pin Spacing

While placing the pins in a design, Pin Optimizer should maintain a minimum spacing between the pins. The *Pin Spacing* section provides various options to specify pin spacing values either from the technology file or by letting you enter the values manually.

Caution

The specified spacing value may be ignored when there are connected pins on the same net, but in different blocks. If the PR boundaries of the blocks are abutted, then the pins would also be abutted, without considering any potential DRC spacing violations.

Select the *Use Table Spacing* check box to scale up the spacing values as per the table spacing rules defined for different widths of the pins. If this option is turned off, then use the minimum spacing values that are defined for the layers in the technology file.

Environment Variable: tableSpacingRule

Pin Optimization is layer aware. Therefore, you can use or define separate pin spacing values for each layer.

© 2020 All Rights Reserved.

Video

For quick reference of this feature, refer to the Advanced Pin Optimization Support in Virtuoso Floor Planner (VFP) demonstration.

Select one of the following pin spacing modes to define pin spacing values for the various layers in the current design.

Auto

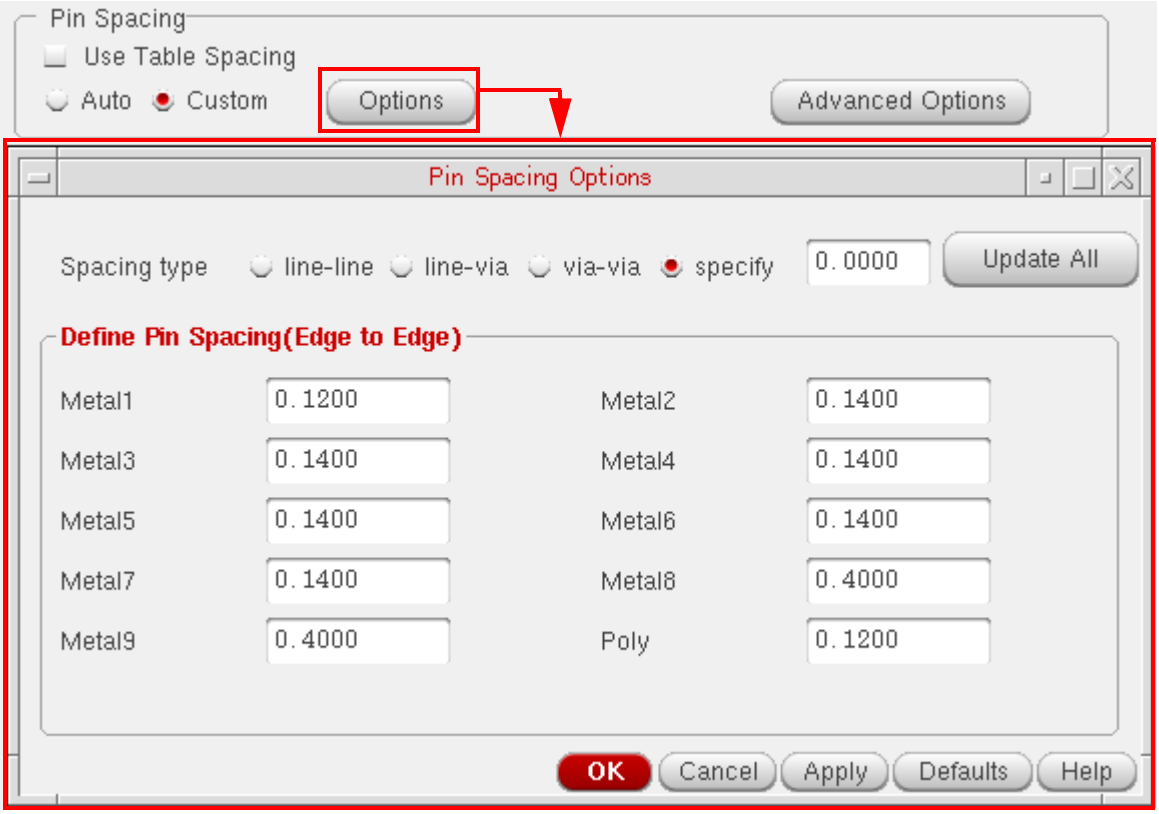
When the *Auto* option, which is the default option, is selected, the minimum spacing values are based on the line-line spacing mode. If the *Use Table Spacing* check box is selected, then the spacing values are defined as per the table spacing rules for different pin widths.

Custom

The *Custom* option lets you select a pin spacing type for specifying the pin spacing values for each layer. Click the *Options* button to display the Pin Spacing Options form.

© 2020 All Rights Reserved.

Environment Variable: spacingMode

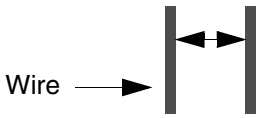


Note: If the *Use Table Spacing* option is selected, then the spacing values are defined as per the table spacing rules for different pin widths.

You can choose one of the following *Spacing Type* options:

- line-line

  This option uses the minimum spacing between two wires on a layer for spacing pins. Pin Optimizer reads appropriate spacing rules from the technology file to identify the spacing value. Use this option if you are using a single layer. In multi-layer designs, line-line spacing can cause congestion when vias are inserted, because of the extra jogs that you add to wires to allow for vias.

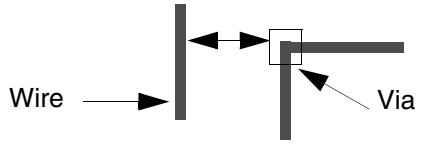
  

  The options in the *Define Pin Spacing (Edge to Edge)* section are not editable in this mode. However, the layer spacing values that are retrieved from the technology file are displayed in the individual fields.

© 2020
All Rights Reserved.

- line-via

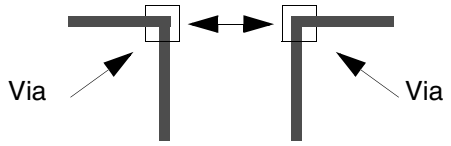
  This option uses the minimum spacing between a wire and a via on a layer for spacing pins. In addition to the minimum spacing rules, Pin Optimizer reads valid via definitions and minimum extension rules from the technology file to calculate the spacing value.



  The options in the *Define Pin Spacing (Edge to Edge)* section are not editable in this mode. However, the layer spacing values that are retrieved from the technology file are displayed in the individual fields.

- via-via

  This option uses the minimum spacing between two vias on a layer for spacing pins. Similar to the line-via option, Pin Optimizer reads valid via definitions, minimum spacing rules, and minimum extension rules from the technology file to calculate the spacing value.



  The options in the *Define Pin Spacing (Edge to Edge)* section are not editable in this mode. However, the layer spacing values that are retrieved from the technology file are displayed in the individual fields.

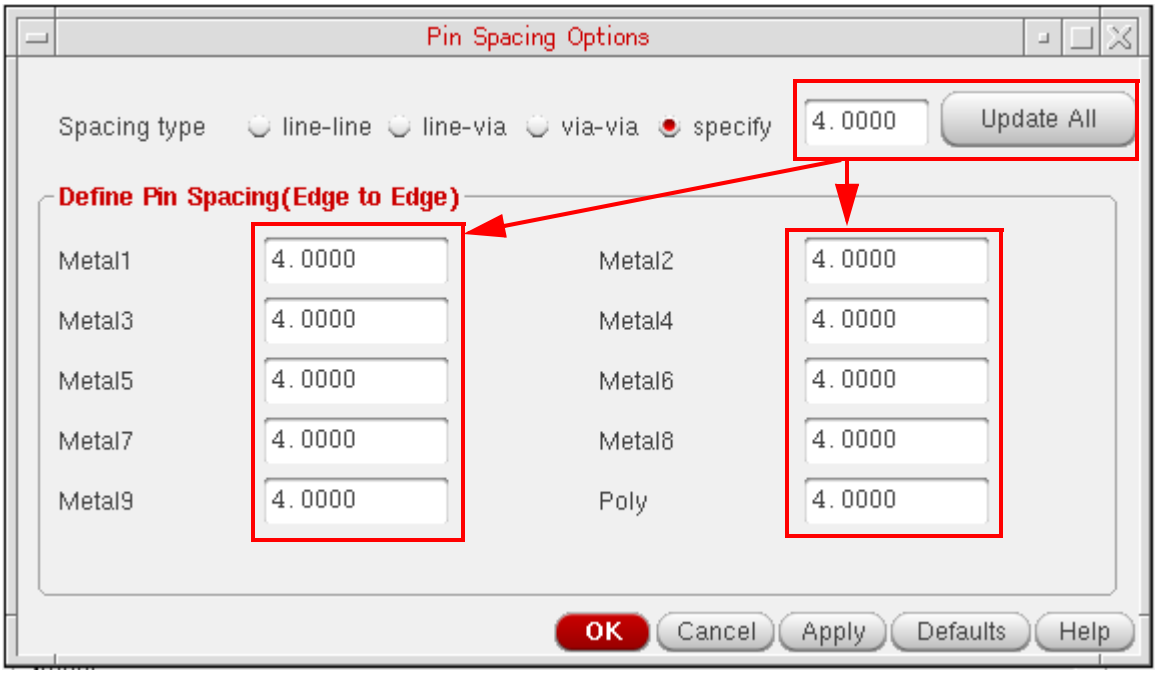
- specify

  This option enables you to enter pin spacing values for each layer in the current design manually. By default, the pin spacing values that are retrieved from the technology file are displayed in the individual fields. You can specify the same or different spacing values for each layer.

  Environment Variables: spacingType, spacingValue

© 2020 All Rights Reserved.

To specify the same pin spacing value for all layers, enter the value in the text box next to the *specify* option and click *Update All*.



To specify different pin spacing values for each layer, type the values in the corresponding text boxes. Pin spacing values cannot be smaller than the `minSpacing` or the `tableSpacing` value specified in the technology file.

Environment Variable: perLayerSpacingList

Clicking *Defaults* changes the form values to the default values, which correspond to the original factory settings.

Important

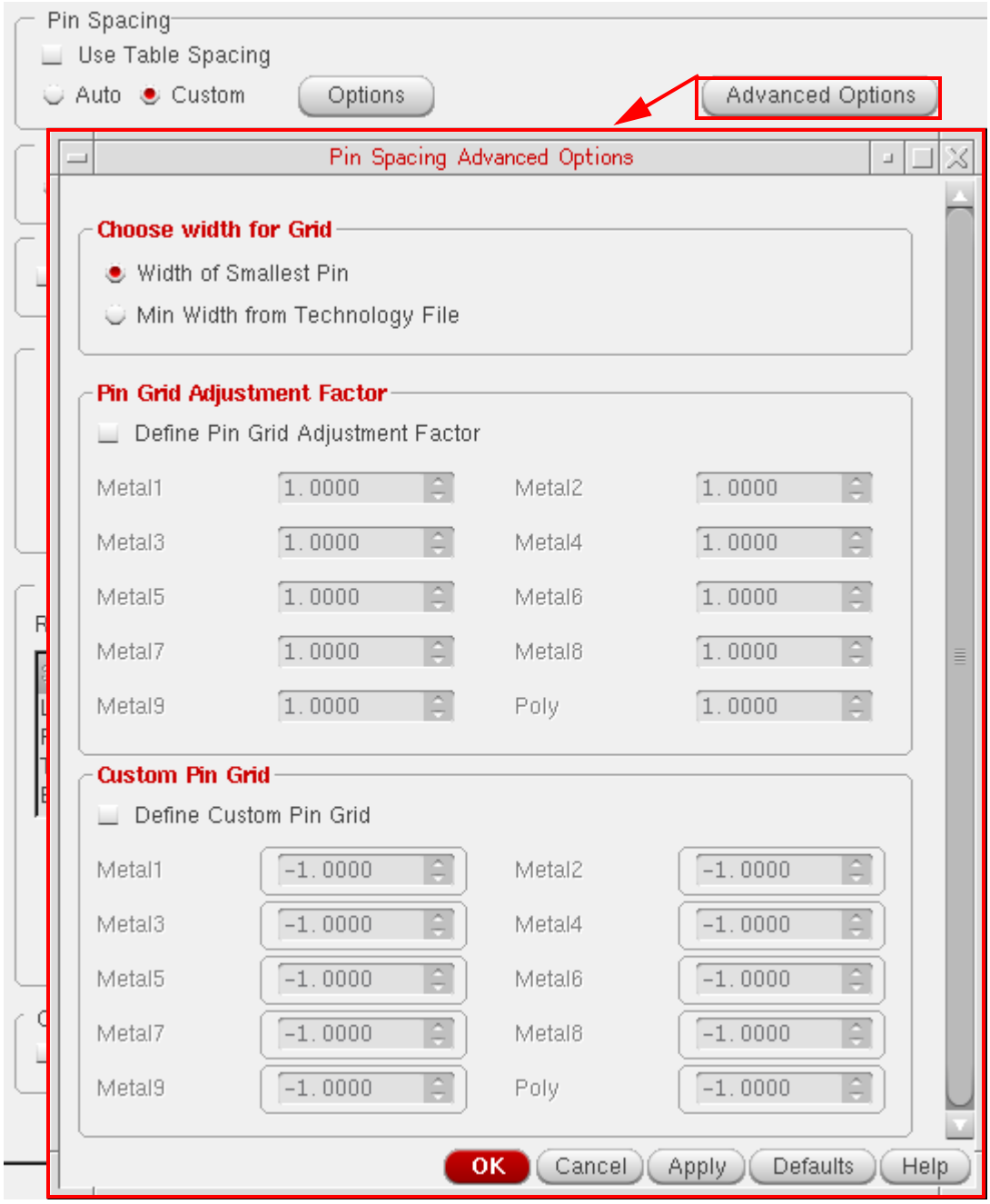
Values specified in the *Define Pin Spacing (Edge to Edge)* are the minimum values. These values are scaled up as per the table spacing rules if the *Use Table Spacing* check box is selected.

Pin Optimizer checks for pin overlaps and the validity of the spacing value on each layer.

© 2020
All Rights Reserved.

Advanced Options

Click the *Advanced Options* button to display the Pin Spacing Advanced Options form.



Use options in the Pin Spacing Advanced Options form to define pin grids. A pin grid defines the points at which pins need to be placed. Each layer has a separate pin grid. Typically, the pin grid value is the sum of the layer spacing value and the pin width value.

© 2020 All Rights Reserved.

The Pin Spacing Advanced Options form provides the following options to define the pin grid:

- **Choose width for Grid:** Use this option only when the pins on a layer have different pin widths. In this section, specify whether the width of the grid should correspond to the *Width of the Smallest Pin* in the design, or to the value that is picked from the technology file (default). The *Choose width for Grid* option is available for both, the *Auto* and *Custom* pin spacing modes.

  Note: For layers that do not have any pins, the layer `minWidth` value is considered the minimum pin width for creating the pin grid.

  Environment Variable: widthForPinGrid

- **Pin Grid Adjustment Factor:** In this section, specify the value by which the pin grid needs to be scaled for each layer. For example, if you specify the Pin Grid Adjustment Factor as 0.5 for a layer, then the width of the grid is reduced to half its current value. The *Pin Grid Adjustment Factor* option is available for both, the *Auto* and *Custom* pin spacing modes.

  Environment Variables: perLayerAdjustmentFactorList, pinGridAdjustmentFactor

- **Custom Pin Grid:** In this section, specify absolute values for the width of the pin grid for each layer. Pin spacing values, as specified in the Pin Spacing Options form, are applied. The *Custom Pin Grid* option is available only for the *Custom* pin spacing mode.

  Environment Variable: customPinGrid, perLayerCustomPinGridList

Clicking *Defaults* changes the form values to the default values, which correspond to the original factory settings.

Pin Offset

Pin offset defines the first valid point for placing a pin on a grid. The pin offset value specifies the distance between the adjacent side and the centre point of the first pin. Pin offset values are layer aware, which means that each layer can have a different pin offset value.

Use the *Control Pin Offset* section of the Pin Placement form to define pin offset values for all layers in the current design.

Select one of the following modes:

- **Auto:** This is the default mode. During pin optimization, the first pin is placed on a grid such that it is aligned with the corner pins of the adjacent side and there is no spacing violation. Pin Optimizer also ensures that there is no wastage of grid space.

© 2020 All Rights Reserved.

In this mode, Pin Optimizer respects both, the pin spacing value and the pin pitch from the instance corners.

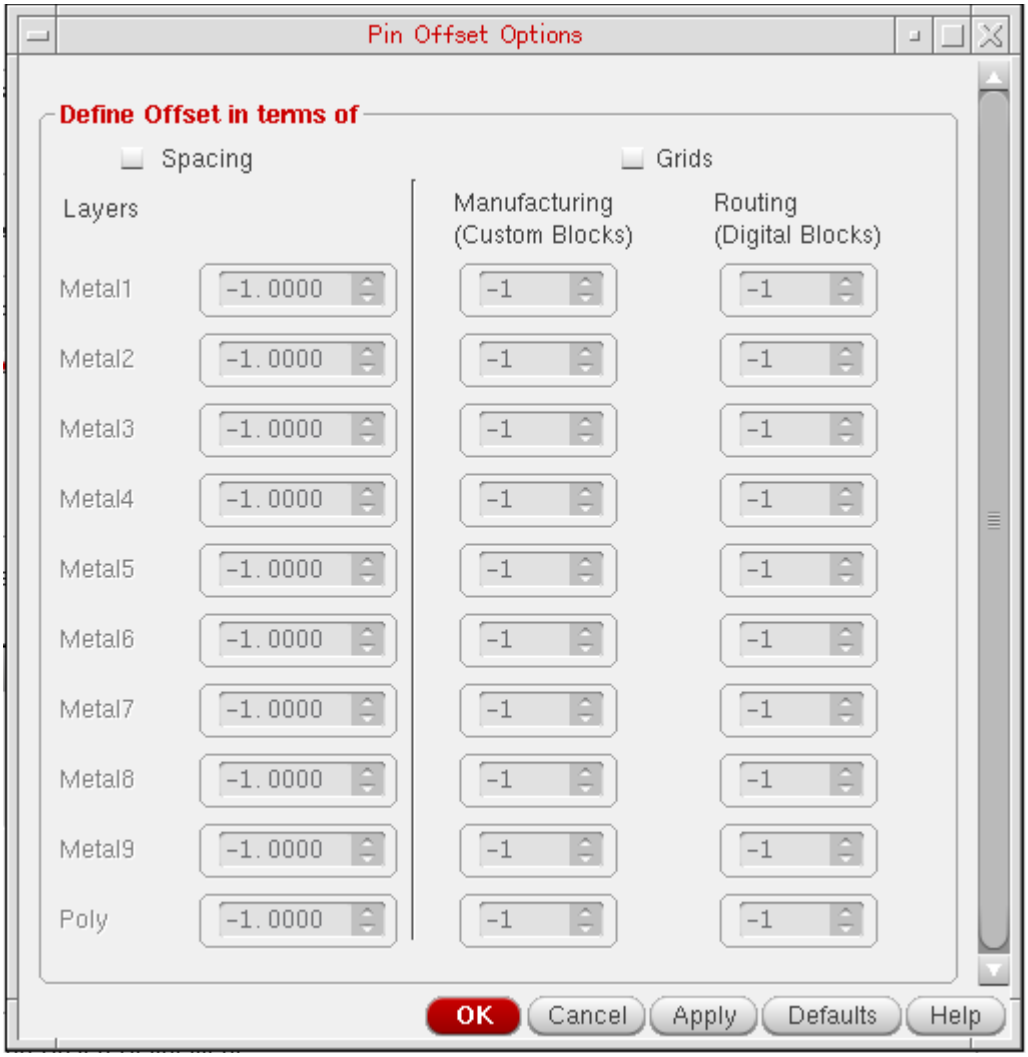
Note: The *Options* button is not available in this mode.

- **Custom:** This mode provides options to control the pin offset values for all layers in a design. Custom Offset number is the value from the boundary edge to the first pin slot to which the pin needs to be centered. In this mode, Pin Optimizer respects the pin pitch values from instance corners, independent of offset mode. However, Pin Optimizer does not respect the pin spacing from instance corners, but ensures that pins do not lie outside the instance boundaries.

  Note: Adding a Pin Grid Adjustment factor for custom pins may be error-prone as the distance from the boundary edge to pin edge may not be the value used to adjust the offset.

© 2020 All Rights Reserved.

When you select the *Custom* check box, the *Options* button is made available. Click the *Options* button to display the Pin Offset Options form.

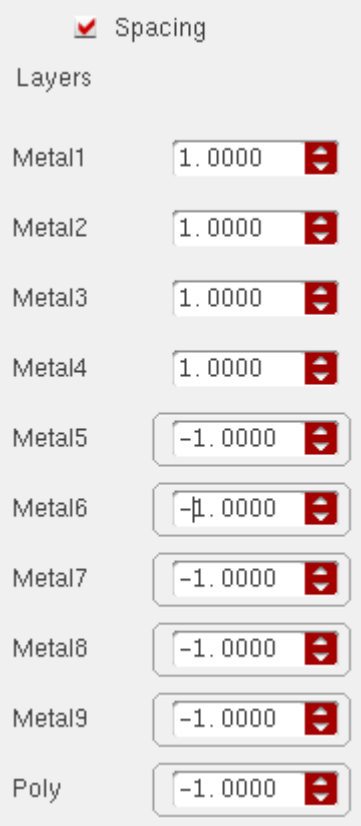


Environment Variable: offSetMode

Use the Pin Offset Options form to define pin offset values in terms of one or both of the following parameters:

© 2020
All Rights Reserved.

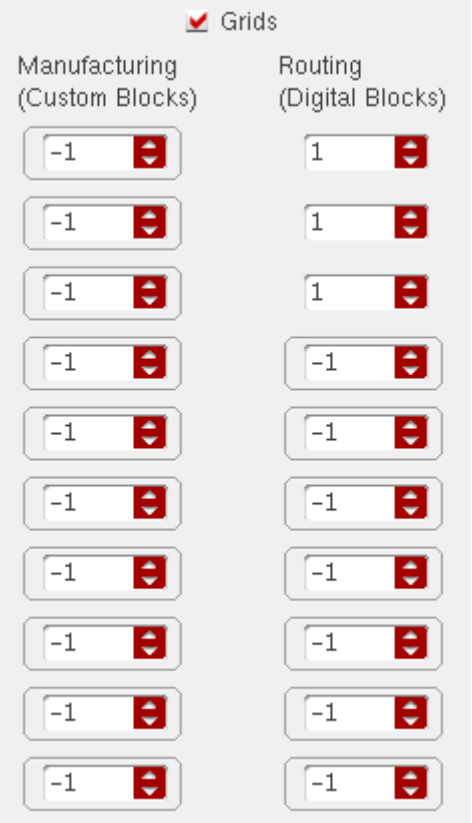
- **Spacing:** Specifies the absolute space between the cell boundary and the first pin.



Environment Variables: offSetBySpacing, perLayerOffsetBySpacingList

© 2020 All Rights Reserved.

- **Grids:** Specifies the number of grids to skip before placing the first pin. Manufacturing grids apply for custom pins, and routing grids apply for digital pins.



Choose one of the above options and specify the pin offset values for individual layers.

If the pin offset values are specified in terms of both, spacing and grids, then the grids values override the spacing values. If both, *Spacing* and *Grids* check boxes are turned off, then the *Auto* option is applied.

Environment Variables: offSetByGrids, perLayerOffsetByMfgGridList, perLayerOffsetByRoutingGridList

> **Note:** If a negative pin offset value is specified, a warning for that metal and instance is displayed and the Auto offset value is used during pin optimization.
>
> If the pin offset value is specified as zero, a warning for that metal and instance is displayed and the manufacturing grid is used as the offset value during pin optimization.

Clicking *Defaults* changes the form values to the default values, which correspond to the original factory settings.

© 2020 All Rights Reserved.

Bus Pins

Pin Optimizer places all the pins of a bus that have flexible pins on both the sides. Flexible pins are yet to be placed and can be moved during pin optimization.

For a bus that has fixed reference pins on one side and flexible pins on the other side, Pin Optimizer places only the flexible pins. The fixed reference pins are pre-placed and cannot be moved.

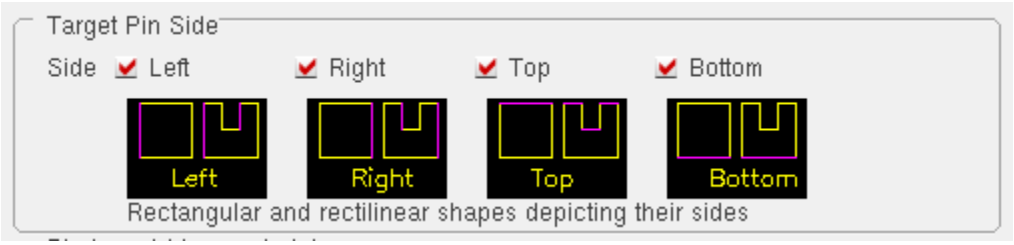
By default, Pin Optimizer does not group the flexible pins of a bus that have fixed reference pins on the other side. The nets of such buses are optimized individually. Therefore, the placement of the flexible pins is independent of the order of the fixed reference pins of the bus. This methodology gives maximal alignment, even if the fixed pins are unordered. To avoid formation of criss-cross connections, nets with fixed reference pins on one side are first optimized, followed by the other nets in the design. This is because any other location for flexible pins connected to fixed pins will result in misalignments and criss-cross flight lines.

Video

For quick reference of this feature, refer to the Using the Pin Optimizer feature demonstration.

Target Pin Side

The Target pin side is the side of the boundary on which Pin Optimizer should try to place the pins. The options in the *Target Pin Side* group box are used to specify the side of the block or the top-level PR boundary on which pins should be placed. A check box is available for each side on which pins can be placed. The image below each check box shows the definition of that side for rectangular and rectilinear blocks. For rectilinear blocks, the *Target Pin Side* setting specifies the access direction for pins.



Note: The *Target Pin Side* settings are considered after considering the constraints specified in the Constraint Manager.

© 2020 All Rights Reserved.

Pin Optimizer treats the edge of any soft block or top-level block as a range. Pin Optimizer places pins at different points of the range to find the best location so that minimum net length is achieved.

Pin Layer/Side Constraints

The Pin layer/side constraints are used to switch the pin layer depending on the side of the block on which the pin is placed by Pin Optimizer.

To set the pin layer/side constraints, you select the *Respect pin layer/side constraints* check box. You will notice that the two list boxes below the check box are available. By default, these two list boxes are disabled.

Important

(ICADVM20.1 Only) At advanced nodes, default values are automatically applied to the tables in the *Pin layer/side constraints* section. The default pin layer/side constraint values are based on the `snapGridVertical` and `snapGridHorizontal` constraints defined in the technology file.

- WSSPDefs with `snapGridVertical` constraint have tracks intersecting with the left and right edges.

- WSSPDefs with `snapGridHorizontal` constraint have track intersecting with the top and bottom edges.

The first list box has two columns i.e *Side* and *Pin Layers*. *Side* specifies four different sides as “*Left*” “*Right*” “*Top*” and “*Bottom*”. *Pin Layers* specifies one or more pin layers for each

© 2020 All Rights Reserved.

side. You can define the pin layers for a side by selecting a pin layer from the list box before the *Update Pin Layers* button. If no side is specified, then the existing pin layer remains unchanged.

To add/update a pin layer for the side, select a pin layer from the list box before the *Update Pin Layers* button and then click the *Update Pin Layers* button. The selected pin layer is displayed in the *Pin Layers* column in the first list box. If you want to clear the pin layers of the selected side, click the *Clear Pin Layers* button.

During pin optimization, when a pin layer is switched, the pin label associated with the pin is also switched to the same layer on which the pin is changed. However, if the pin layer is not switched, the layer of the pin label will remain unchanged.

Note: The pin label should be associated with the pin.

Related environment variables: enablePinLayerSideConstraints, leftSidePinLayersConstraints, rightSidePinLayersConstraints, topSidePinLayersConstraints, bottomSidePinLayersConstraints.

Match Fixed Pins Layer/Size

To change the layer or purpose of the 'to be placed' or movable pins to match the layer of purpose of the connected fixed pins, select the *Match Pin Layer* check box. For example, if a movable pin is on `(Metal2 drawing)` layer purpose and the connected fixed pin is on `(Metal1 drawing)` layer purpose, then selecting this option will change the movable pin layer to `(Metal1 drawing)`.

To change the size of the movable pins to that of the connected fixed pins, select the *Match Pin Size* check box. For example if the target pin size is 0.6 x 0.6 and the reference pin size is 1.0 x 1.0, then selecting this option will change the target pin size to 0.6 x 0.6.

Note: *Match Pin Size* is available only when you select *Match Pin Layer*.

The options in the *Match Fixed Pins Layer/Size* section are applicable only to nets with fixed (non-movable) pins connected to movable pins. A pin is considered fixed if:

- The pin is not selected for pin optimization. This is applicable when *Mode* is set to *Selected* and the pins on one side of a net are selected. The selected pins are movable, while pins on the other side of the net are fixed.
- The placement status of the pin is set to fixed or locked.

When *Mode* is set to *All*, pins on both side of nets are selected for optimization. Therefore, there are no reference (fixed) pins, and so the *Match Fixed Pins Layer/Size* settings are

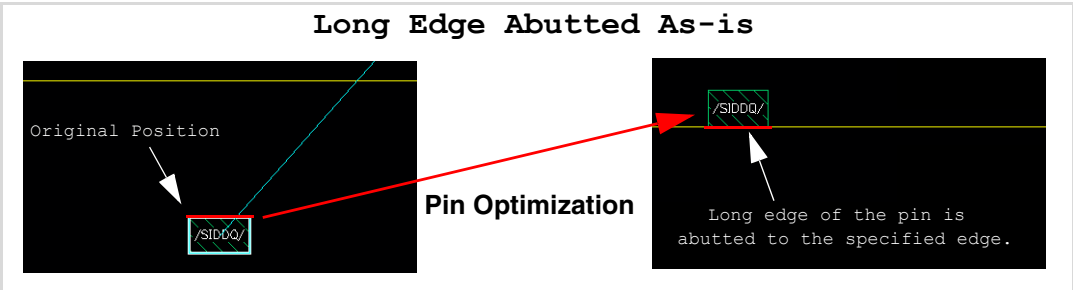
© 2020 All Rights Reserved.

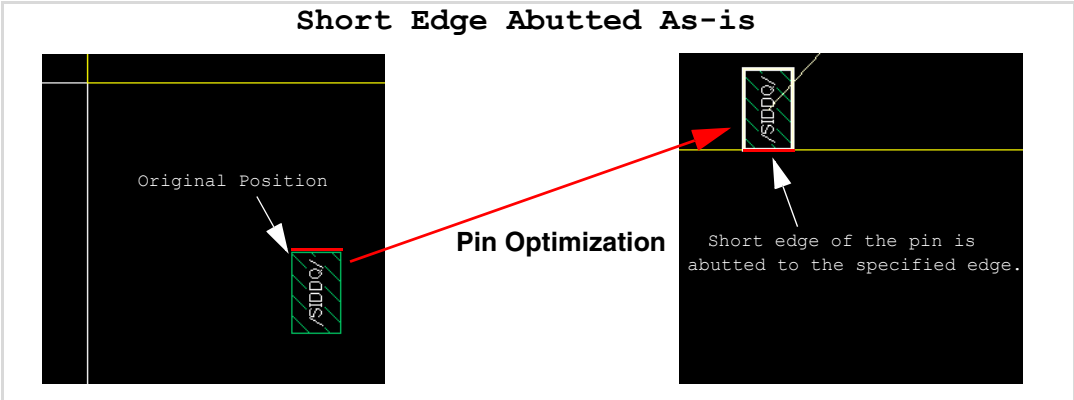
not applicable. To honor these settings, you must manually change the placement status of the required pins to fixed or locked.

Place Rectangular Pin

Pin Optimizer supports rectangular as well as square pins. For rectangular pins, you can use the options in the *Place Rectangular Pin* group box to select the pin edge that needs to be abutted to the PR boundary. Choose one of the following edge options for placing rectangular pins:

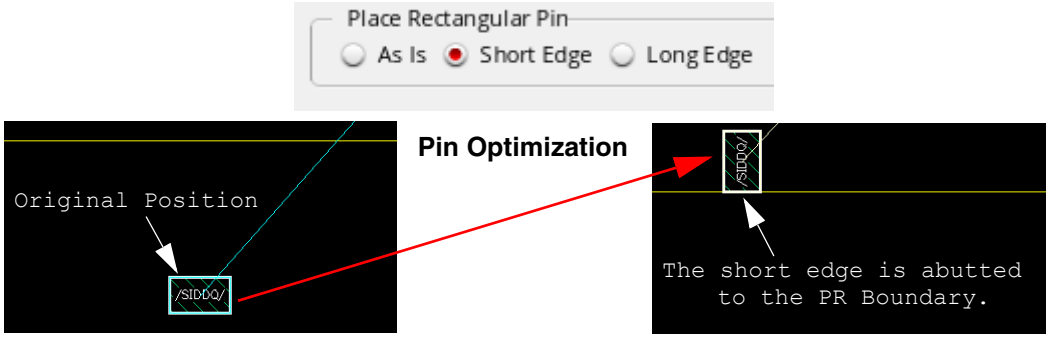
- **As Is:** During pin optimization, the rectangular pin is abutted as-is to the PR boundary, irrespective of whether the abutted edge is short or long. Depending on the original position of the pin and the PR boundary side, either the long edge or the short edge of the pin is abutted to the PR boundary, as shown below:



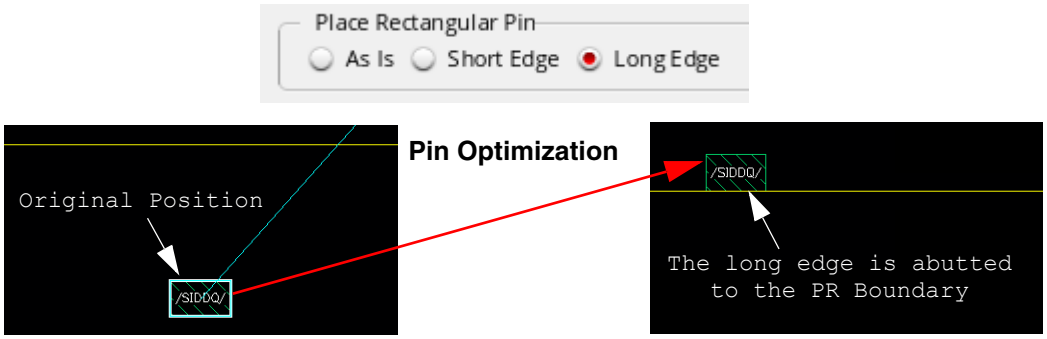


© 2020
All Rights Reserved.

- **Short Edge:** During pin optimization, the short edge of the rectangular pin is abutted to the PR Boundary, as shown below:



- **Long Edge:** During pin optimization, the long edge of the rectangular pin is abutted to the PR boundary, as shown below:



Environment variable: rectangularPinEdgeOnPRB

(ICADVM20.1 Only) Snapping Pins to WSP Tracks

Select *Honor Current Pin Location* on the *Pin Optimization* tab to snap pins (selected or all) to their nearest WSP tracks by moving them by a minimum distance from their current position. The following constraints set on the pins are honored:

- Wire Type
- Width
- Side/Edge
- Order
- `minSpacing` violations in blockages and pre-placed pins

Choose *Select Honor Current Pin Location* in the following situations:

© 2020 All Rights Reserved.

- When optimizing unconnected pins leads to issues such as bunching of pins. Unconnected pins are pins with either no connectivity, or they are connected to instances that are located or placed outside the PR boundary. To avoid such bunching issues, choose *Select Honor Current Pin Location*. The pins are only snapped to their nearest WSP tracks, while honoring any applicable pin constraints.
- When pins are placed after performing design placement using placement tools such as Pin Planner, Virtuoso Custom Digital Placer, and Virtuoso Analog Placer. The pins are snapped to their nearest tracks, while honoring any applicable pin constraints.

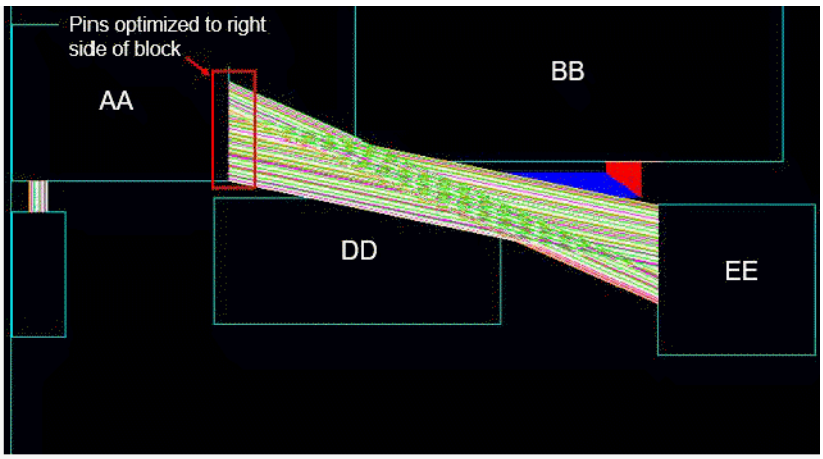
Note: When *Select Honor Current Pin Location* is selected, Pin Optimizer does not try to achieve the minimum net length.

Environment variable: honorCurrentPinLocation

Congestion-Aware Pin Placement (ICADVM20.1 EXL Only)

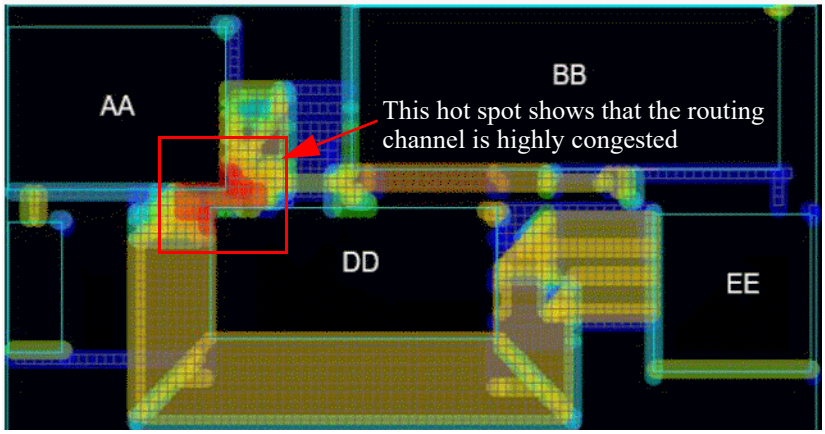
By default, Pin Optimizer places pins considering DRC, minimum wire length, design constraints, and process rules, but does not consider routing information such as routing channel paths and congestion data. In certain designs, this might cause hot spots after pin optimization.

The following image shows the result of standard wire length-based pin optimization on a design.



© 2020 All Rights Reserved.

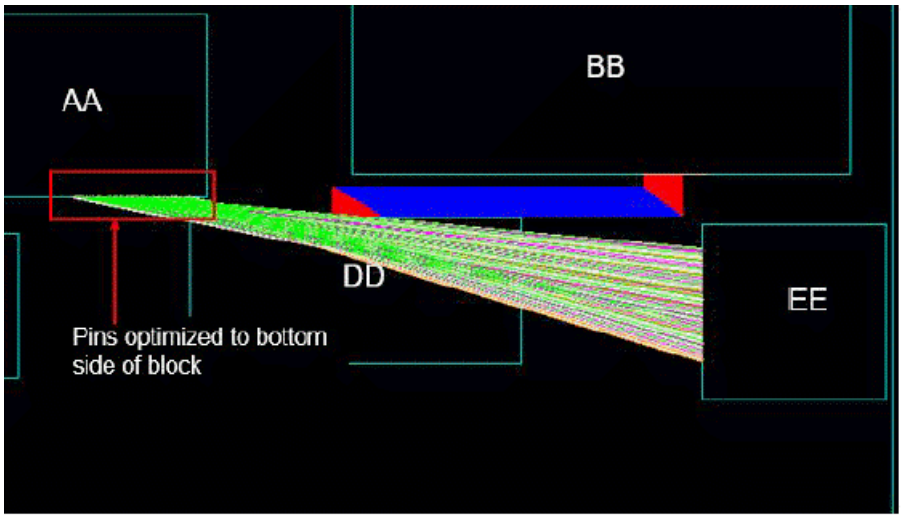
After running global routing and congestion analysis on the design, the routing channel appears highly congested.



When pin placement becomes congestion aware, the routability of such designs improves.

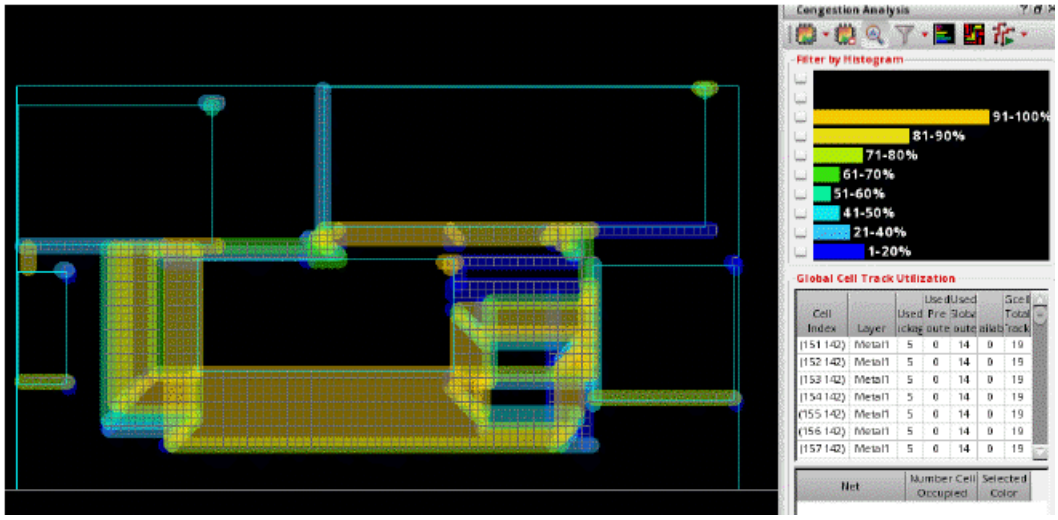
Select *Congestion Aware* in the *Congestion Aware Pin Placement* section of Pin Optimizer to consider the channel congestion while optimizing pins.

The following example shows how congestion-aware pin optimizer helps with pin placement. Pin Optimizer internally uses the results from Global Router to find congestion-aware placement positions for pins and places the pins accordingly.



© 2020
All Rights Reserved.

The following image shows the result of congestion analysis after running congestion-aware pin optimizer.



For more information about running congestion analysis using Global Router, see Running Congestion Analysis.

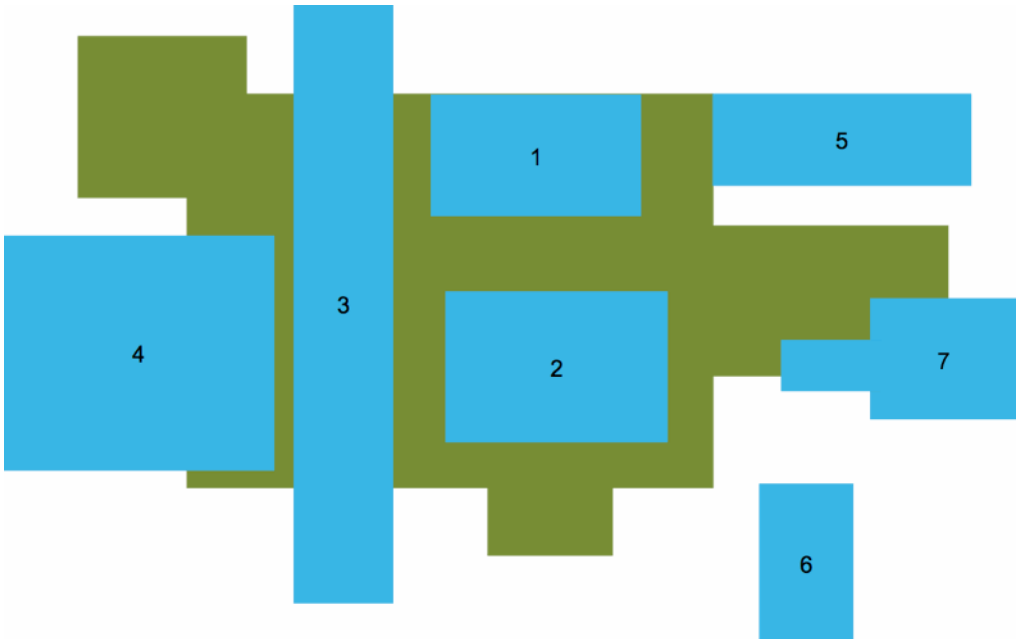
Running Pin Optimizer in the congestion-aware mode may result in an increase in total net length, which is expected because target pins might be placed further away. This is done to increase the routability of the design without increasing the design size or modifying the block locations.

```
*************************************************************
*                                                           *
*    End Congestion Map Data Clearing                       *
*                                                           *
*************************************************************
 Net Length Before Congestion Aware Optimization = 80741

 Net Length After Congestion Aware Optimization = 85101

-----------------------------------------------------
```


© 2020 All Rights Reserved.

Optimizing Pins that are Partially Outside the PR Boundary

Pin Optimizer supports optimization of pins in instances that are partially outside the top-level PR boundary. Set the optimizeInstancesPartiallyOutsidePRB environment variable to `t` to enable optimization of such pins. Consider the following placement of instances:



The following table explains the behavior when `optimizeInstancesPartiallyOutsidePRB` is set to `nil` (default) and `t`, respectively:

optimizeInstancesPartiallyOutsidePRB	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7
nil	Optimized	Optimized	Not Optimized	Not Optimized	Not Optimized	Not Optimized	Not Optimized
t	Optimized	Optimized	Not Optimized	Optimized	Optimized	Not Optimized	Optimized

Blocks 4, 5, and 7 are partially outside the PR boundary. Therefore pins in these blocks are optimized when `optimizeInstancesPartiallyOutsidePRB` is set to `t`.

However all vertices of Block 3 are outside the PR boundary, while block 6 is completely outside the PR boundary. So pins in these blocks are not optimized even when `optimizeInstancesPartiallyOutsidePRB` is set to `t`.

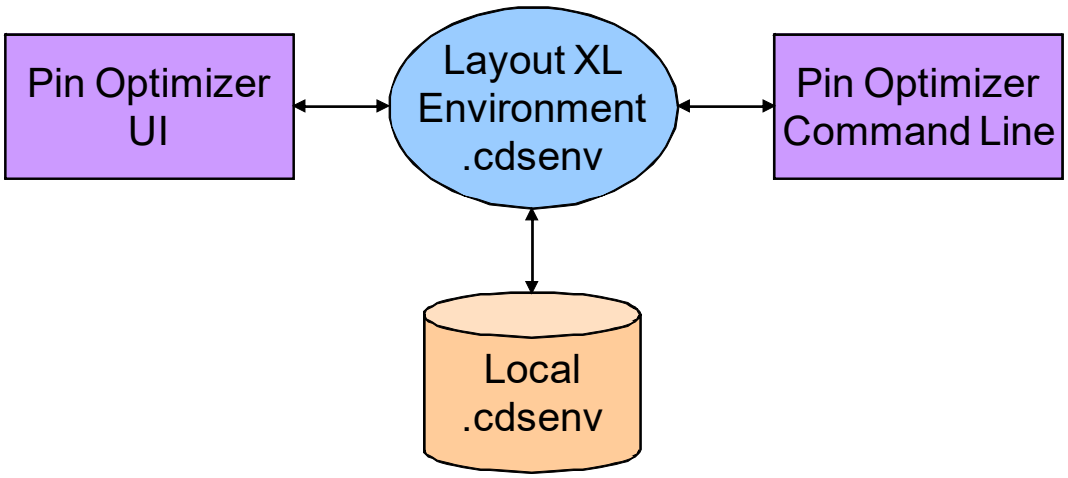
Note: To consider an instance to be partially inside or outside the PR boundary, at least one of its corner (vertex) should lie inside the PR boundary.

© 2020 All Rights Reserved.

Synchronizing Pin Optimization Options between Command Line and GUI

Use the `vpaOptimizePins` command-line SKILL API to run pin optimization without opening the *Pin Optimization* GUI.

Each pin optimization option in the GUI and its corresponding pin optimization option in the SKILL API is associated with a Layout XL Floorplan environment variable. This set of environment variables keeps the pin optimization options synchronized between the GUI and SKILL API, as shown in the figure below. For more information, see Layout XL Floorplan Environment Variables.



When you run the `vpaOptimizePins` SKILL API, you specify values for various pin optimization options. If you do not specify the value for a pin optimization option when you run the SKILL API, the value of the option is read from the corresponding Layout XL Floorplan environment variable. The Layout XL Floorplan environment variables are updated with the existing pin optimization option values once the command has successfully completed. Now, when the GUI is opened, the pin optimization options are updated with these values from the Layout XL Floorplan environment variables.

Note: If the Pin Optimization GUI is already open when the SKILL API is run, the options in the GUI are not updated. You need to close the GUI and reopen it to view the updated options.

When you specify the pin optimization options in the GUI and click *Optimize*, the Layout XL Floorplan environment variables are updated with the values that exist in the GUI. The SKILL API read the default values of the pin optimization options from these updated Layout XL Floorplan environment variables.

© 2020 All Rights Reserved.

Tip

The updated Layout XL Floorplan environment variables can be saved for future in a local `.cdsenv` file on the disk. For more information, refer to Customizing Your Design Environment in the *Virtuoso Design Environment User Guide*.

The following example explains how the values of the pin optimization options in the SKILL API are synchronized with the options on the *Pin Optimization* tab.

```
vpaOptimizePins(?layoutLCV '("ether" "top" "before_pin_opt") ?spacingType
"specify" ?spacingValue 0.1)
```

The values that are assigned to the pin optimization options when the command above is run are listed below.

Argument	Value	Notes
`spacingType`	specify	provided by user
`spacingValue`	0.1	provided by user
selectedOnly	nil	Read from the Layout XL Floorplan environment variable
targetPinSide	left, right, top, bottom	Read from the Layout XL Floorplan environment variable

Values returned are:

- `t` - Pin optimization was successfully run. The Layout XL Floorplan environment variables were updated as per the specified values.
- No value - Pin optimization failed on some of the blocks (partially) or on all blocks (completely). Related error messages are displayed in the CIW.

If you now open the Pin Placement form, you will notice that the values displayed on the *Pin Optimization* tab are the same as those assigned to the pin optimization options while

© 2020 All Rights Reserved.

running the `vpaOptimizePins` SKILL API. The following figure shows the pin optimization options, in the GUI, with the updated values.

© 2020
All Rights Reserved.

Using cphID In Conjunction With vpaOptimizePins

A special optional argument, `cphId`, is available for the `vpaOptimizePins` API, which identifies a physical configuration cellview created using the `cphCreatePhyConfig` API. This optional argument can be used along with the `vpaOptimizePins` API only when the soft blocks in the design are created through command-line-based CPH-GPH flow. The `cphId` argument acts as a quick prototype to check whether the generated soft blocks can be correctly used by the pin optimizer engine. In a way, this argument impacts the behavior of Pin Optimizer and the returned value. There is no corresponding GUI-equivalent for the `cphId` option in the Pin Placement form. If the `cphId` is not specified, then Pin Optimizer retains its original behavior and there is no impact of the CPH-GPH flow on Pin Optimizer behavior.

On executing the API, one of the following values is returned:

- `t` - Pin optimization was successfully run.
- `nil` - Pin optimization failed on all blocks
- A list of blocks - Pin optimization failed on some of the blocks. The list of blocks indicates whether pin optimization was performed for each soft block.

```
(
(sbId  lib:cell:view  status)
(sbId lib:cell:view   status)
)
```

  - `sbId` – ID of the soft-block that was created using `cphSbDefineSoftBlock` API
  - `lib:cell:view` – Physical LCV of the soft block
  - `status` – Status of the soft block after running Pin Optimizer. Values returned are:
    - `t` – Soft block is successfully optimized.
    - `nil` – Soft block could not be optimized.

Constraints Supported by Pin Optimizer

Pin Optimizer supports the following constraints.

- Side Constraint
- Order Constraint
- Priority Constraint

© 2020 All Rights Reserved.

- Pitch Constraint
- Status Constraint

Side Constraint

In the *Target Pin Side* group box on the *Pin Optimization* tab, you can specify the side of the block or the top-level PR boundary on which the pins should be placed. For example, if you want to place all the pins in the design or the selected pins on the left side of the soft block, you select the *Left* check box and deselect the *Right*, *Top*, and *Bottom* check boxes. Pin Optimizer tries to place all or the selected pins on the left edge.

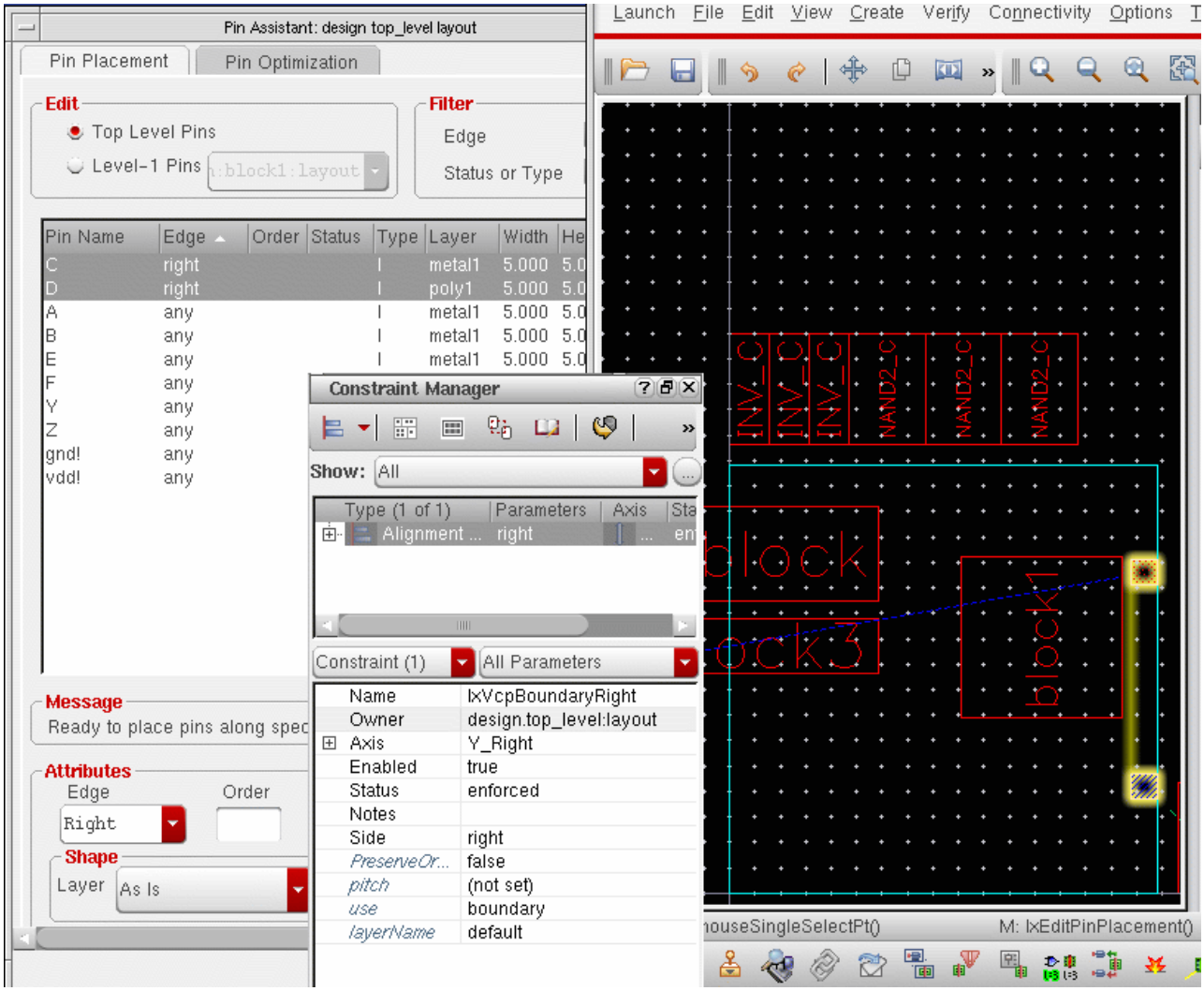
However, if you want to place a selective few pins on another edge, you can specify the side on which pins will be placed by using Pin Planner. For example, if you want only five pins in the design to be placed on the right edge of the block and the remaining pins on the left edge of the block, you can selectively specify the side for the five pins by using Pin Planner.

To specify the side for a pin using Pin Planner, perform the following steps:

1. Select the pins in the *Pin Placement* form.
2. Specify the side for the selected pin in the *Edge* drop-down box.
3. Click *Apply*.

© 2020 All Rights Reserved.

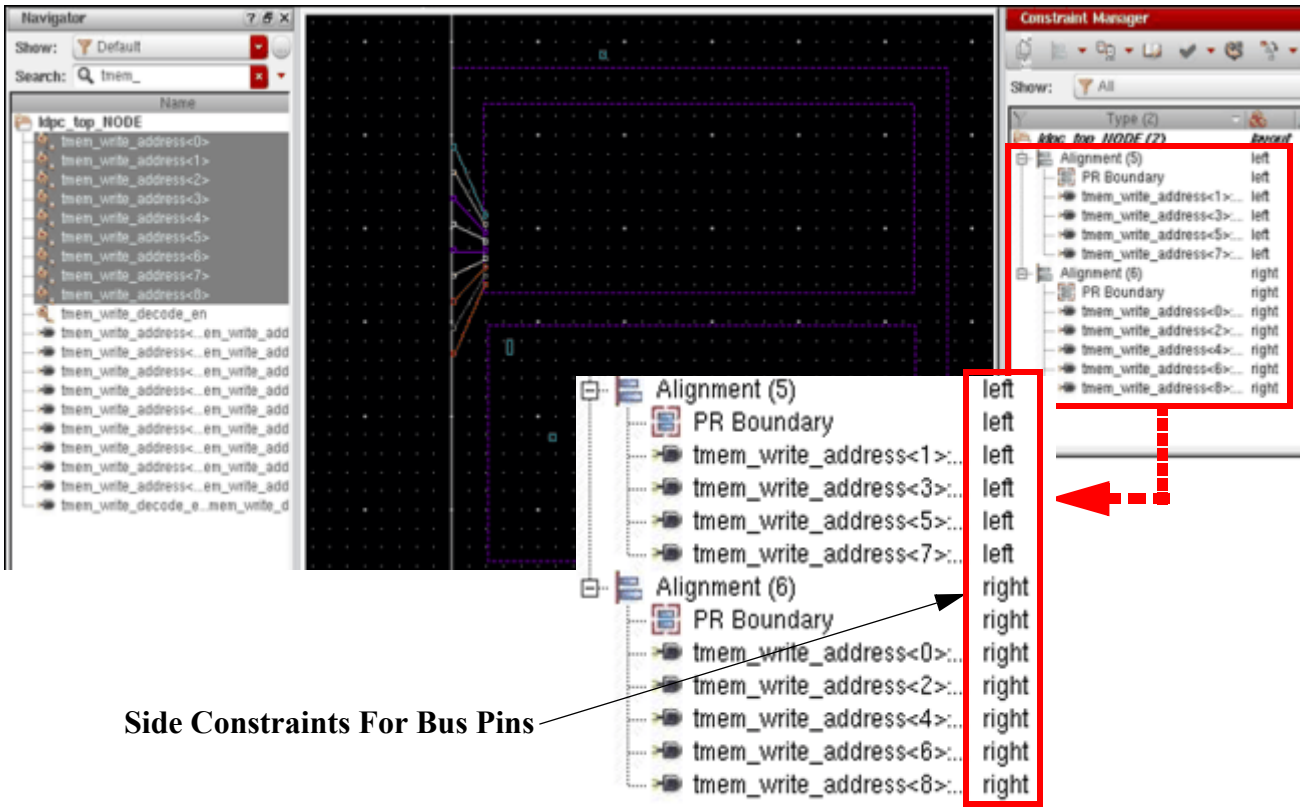
The pins are placed on the side specified in the *Edge* drop-down box as shown in the graphic.



When you specify the side for selective pins in the design, Pin Planner internally creates an alignment constraint for the selected pins and updates the Constraint Manager accordingly. Now if you optimize the pins in the design, Pin Optimizer respects the alignment constraint for the selected pins. For rectilinear blocks and top level PR boundary, the target side refers to the access direction of pins.

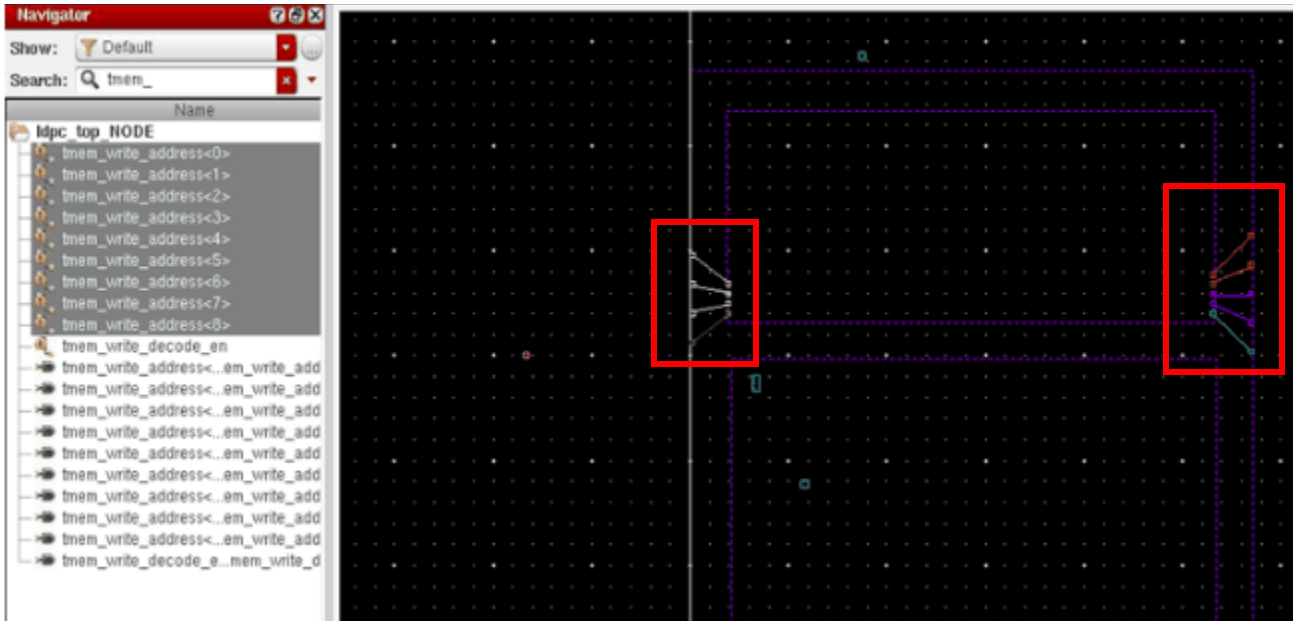
© 2020 All Rights Reserved.

When side constraints are defined for bus pins, during pin optimization, the bus pins are split based on connected pin side constraints. Consider bus pins in the following design. Some of the pins are assigned to the right side, and some are assigned to the left side.



© 2020
All Rights Reserved.

During pin optimization, the bus pins are split depending on the specified target pin side to achieve minimum net length.



Order Constraint

Pin Optimizer enables you to specify the order in which Pin Optimizer places the pins during optimization. The priority of respecting the Order Constraints is as follows:

- **Alignment Constraint**

  You can specify the order of the optimized pins by using Pin Planner. To specify the order of three pins using Pin Planner, perform the following steps:

  1. Select the pins in the *Pin Placement* form.

  2. Specify the order for the selected pin in the *Order* field.

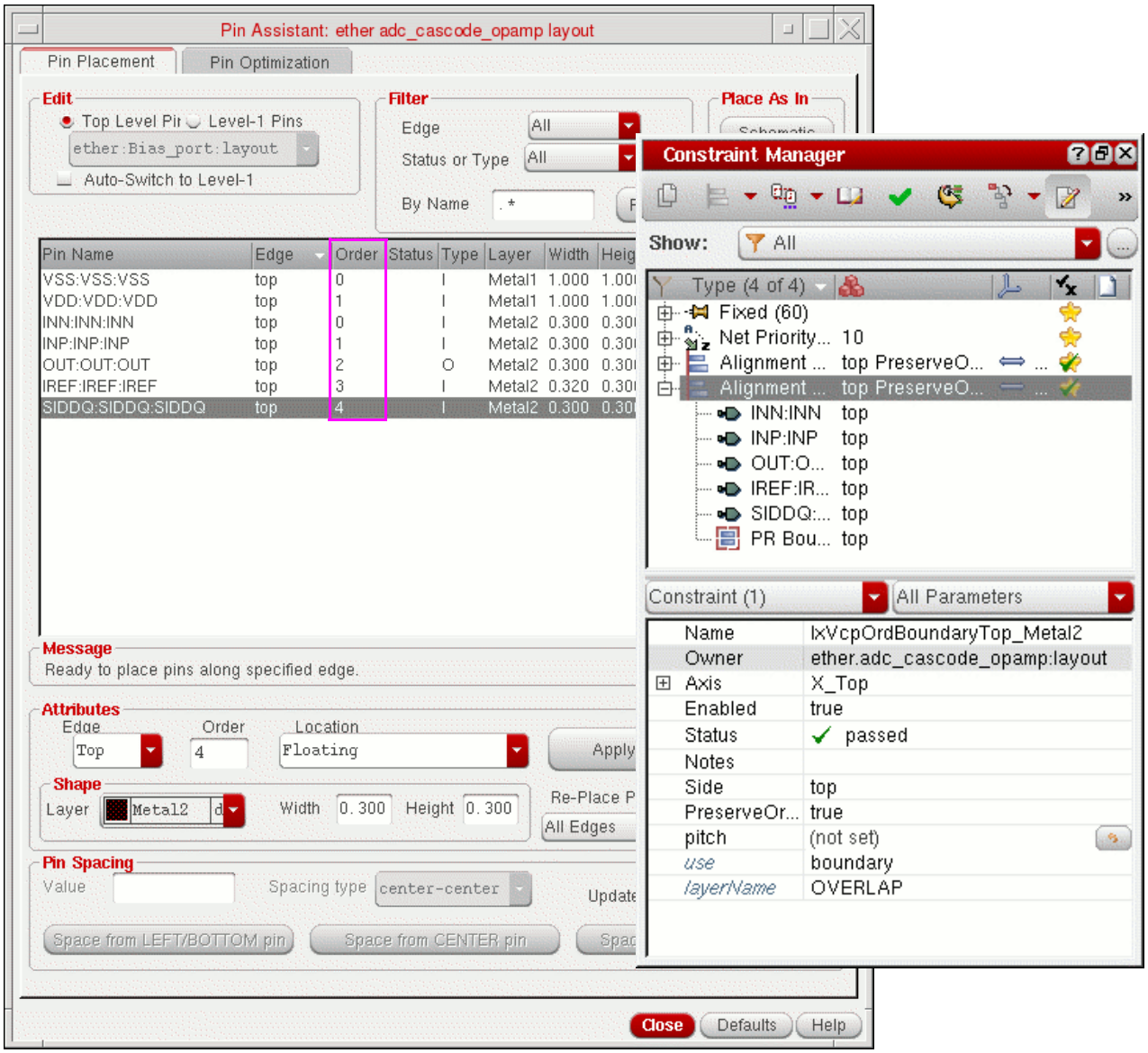
  3. Click *Apply*.

  Pin Optimizer respects the order of the optimized pins in the Alignment constraint. This is done by specifying the value of the *PreserveOrder* attribute of alignment constraint to TRUE as shown in the following figure. By default, the *PreserveOrder* attribute is set to false.

  Also, when the pitch value is set in the alignment constraint with *PreserveOrder* set to TRUE, Pin Optimizer will consider it as the spacing required between the pins and will try to place those pins with this pin spacing (i.e the pitch value).

© 2020 All Rights Reserved.

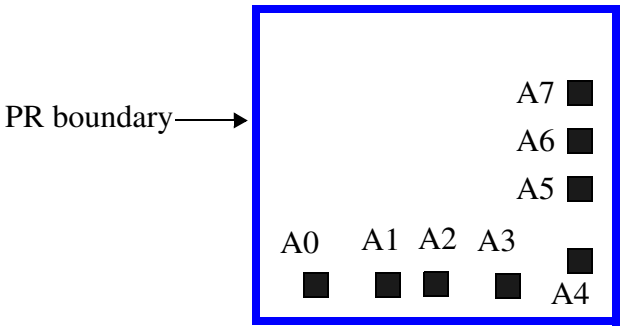
Note: If the *PreserveOrder* attribute is set to false, then the value specified for the *pitch* attribute will be ignored.



© 2020 All Rights Reserved.

- **Bus Constraint**

  If you want the A0, A1, A2, A3, A4, A5, A6, and A7 pins to be arranged in the same order, define the Bus constraint using Constraint Manager. Pin Optimizer respects the Bus constraint and places the pins as shown in the graphic.



  Note: When selecting bus pins for optimization, ensure that you select all pins on both sides of the bus. If only a few pins are selected, then the bus constraint is not recognized. The pins are treated as scalar pins and placed accordingly.

  To define the Bus constraint in Constraint Manager, perform the following steps:

  a. Select the nets in the Navigator window.

  b. Click *Bus* in the constraints list of Constraint Manager.

  The Bus constraint is defined for the pins and Pin Optimizer respects the constraint during optimization.

- **Bus Pins in Layout**

  By default, Pin Optimizer respects the order of bus pins in the layout during optimization. Bus nets get transferred to the layout from the schematic.

Priority Constraint

Pin Optimizer enables you to specify the priority in which it optimizes the pins. For example, if you want P0 to be optimized before P1, you can specify a higher priority for P0 as compared to P1. Pin Optimizer uses the associated priority of the pins during optimization and pins with higher priority are optimized first. You can specify the priority of pins in the -128 to 128 range, where -128 is the lowest priority.

To define the priority of a pin, perform the following steps:

1. Select the net in the Navigator window.

© 2020 All Rights Reserved.

2. Click *Net Priority* in the constraint list of Constraint Manager.
   The *Create Priority* window is displayed.
3. Specify the priority in the *Priority* textbox.
4. Click *OK*.
   The priority of the pin is set.

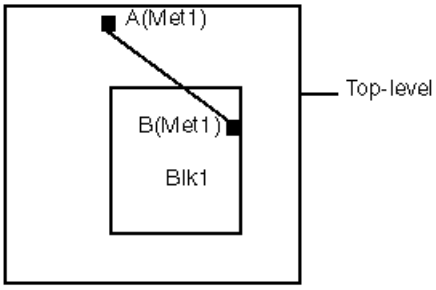
Now if you optimize the pins, Pin Optimizer optimizes the pins based on the associated priority.

Support for Rectangular and Square Pins and Blockages

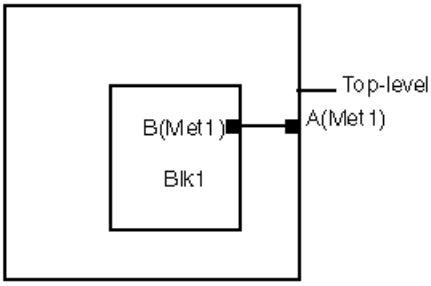
In addition to the supported constraints described above, Pin Optimizer supports the following:

- Pin optimization supports rectangular as well as square pins. If a design consists of pins that are polygonal or rectilinear in shape, they are placed using their bounding box shape.
- Pin Optimizer supports routing blockages on top-level and level-1. For example, if you have pins on the Metal1 layer and a routing blockage is defined on the same layer, then while placing the pins, Pin Optimizer places the pins away from the blockage with the shortest possible net length. Consider the placement of pin in the figures given below.

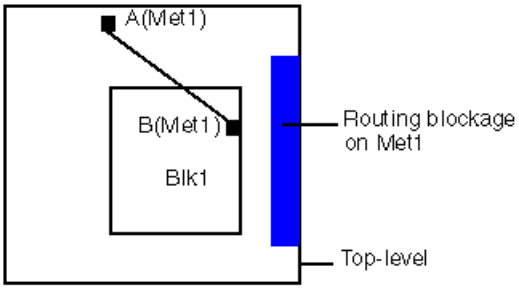
© 2020 All Rights Reserved.



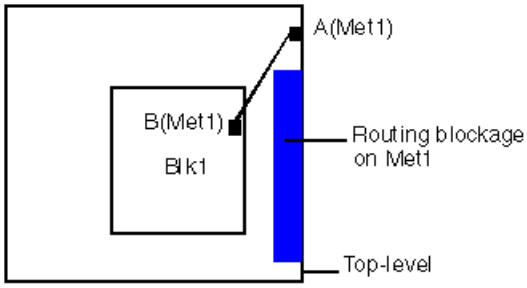
Reinitialized Pins Before Pin Optimization
When no Routing Blockage Exists



Placement of Pin After Pin Optimization
When no Routing Blockage Exists



Reinitialized Pins Before Pin Optimization
When Routing Blockage Exists

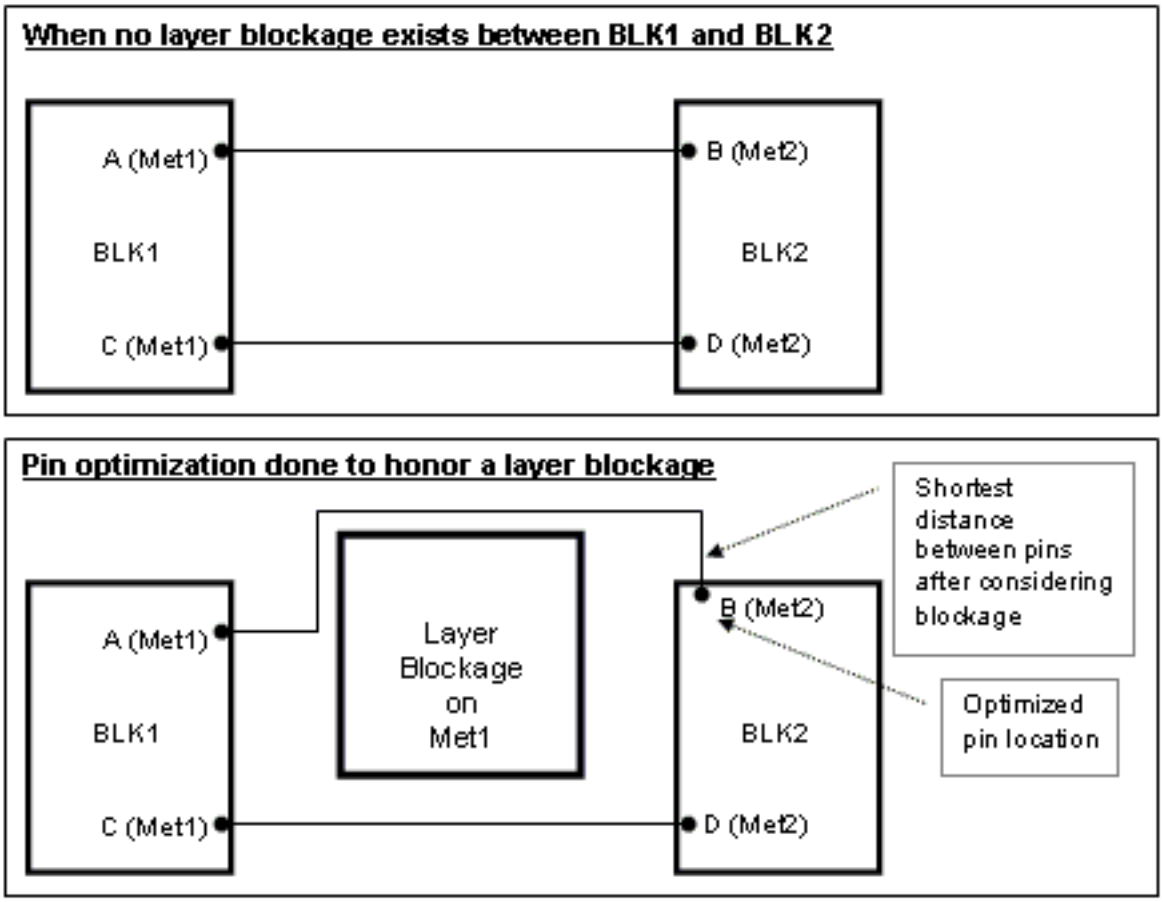


Placement of Pin After Pin Optimization
When Routing Blockage Exists

- Pin Optimizer also supports layer blockages of the design. If there is a layer blockage between two blocks, Pin Optimizer ensures the following while moving the pins:
  - Connects the pins with the shortest possible nets
  - Circumvents the layer in between the blockages

© 2020 All Rights Reserved.

If a layer blockage is on a particular layer and the pins are on another layer, then the blockage will not be considered. Consider the example in the diagrams below:



Pin Optimizer does not move pins with “firm” and “locked” status. If the design has previously placed pins whose status is not firm, Pin Optimizer re-places them and snaps all pins to the placement or manufacturing grid depending on the block type.

© 2020 All Rights Reserved.

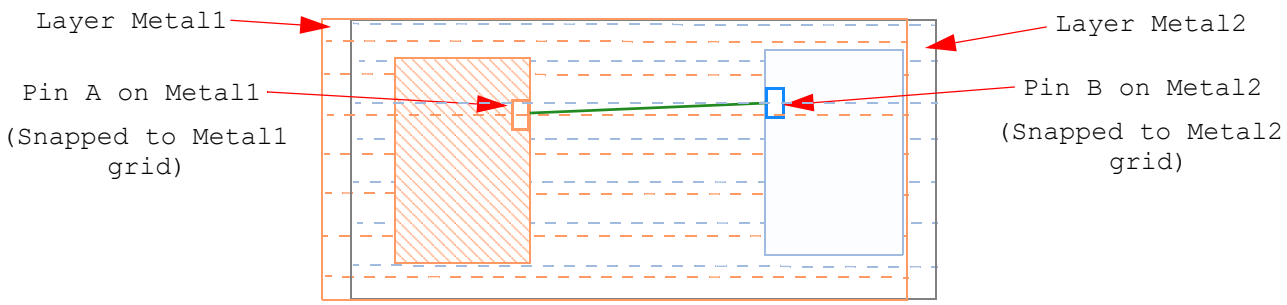
Alignment Issues During Pin Optimization

There are certain possible scenarios where running Pin Optimizer may lead to certain alignment issues among the connecting pins.

Scenarios:

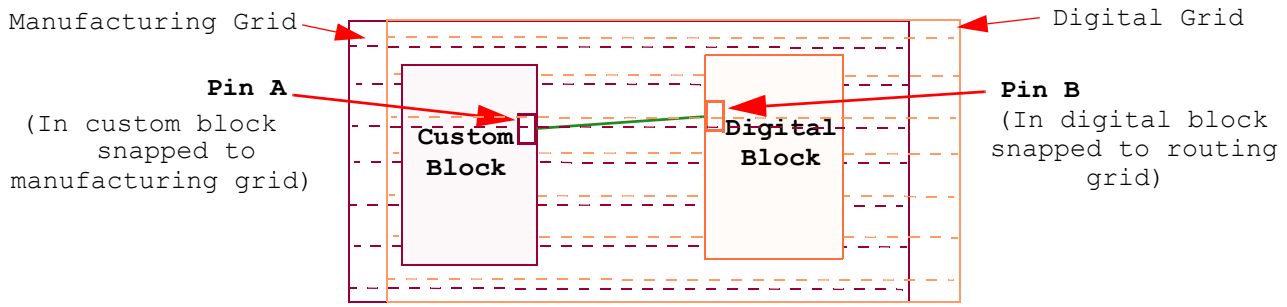
- **When the connected pins are located on different layers:** Each layer may have different pin grids on which the pins are placed. This may result in alignment issues.

  In the following example, Pin A is on metal1 and Pin B is on metal2. Metal1 and metal2 have different pin grids. Pins A and B are placed and snapped to their corresponding layer grids. This has resulted in misalignment between the two pins.



- **When there are multi-pin nets:** Optimizing all pins in these nets may cause alignment issues.

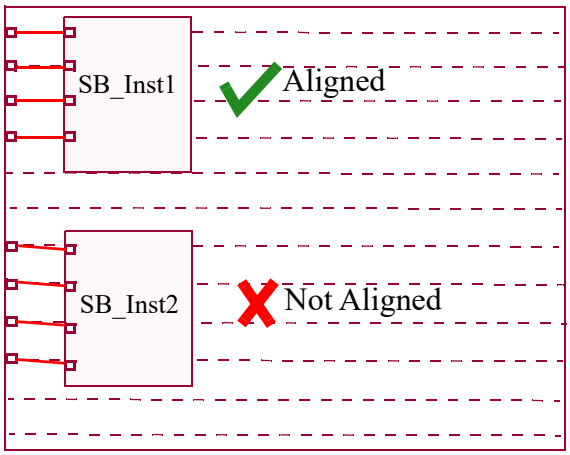
- **When the connected pins belong to different types of blocks:** Pins in custom blocks are snapped to the manufacturing grid, those in digital blocks are snapped to the digital or routing grid, and the ones assigned to block type `nil` are snapped to the user-defined grid. Therefore, there may be misalignments when such pins are connected. The following example depicts misalignment between connected pins that are located in different types of blocks—custom block and digital block.



- **When there are multiple instances of a block:** Pins in one instance will be aligned correctly. However, there may be alignment issues with the other instances. In the

© 2020
All Rights Reserved.

following example, there are two instances of a soft block. Pins in these instances are connected to top-level pins. On running Pin Optimizer, pins in one of the instances are aligned correctly, whereas there is a misalignment with the other instance.



© 2020
All Rights Reserved.

Viewing the Slot Density Report

In case the Pin Optimizer fails, use the detailed slot density report to view information about occupied and unoccupied slots for each soft block in the design. To view the slot density report, you need to set the `PIN_OPT_PIN_DENSITY_INFO` shell environment to any positive integer number, for example:

```
setShellEnvVar("PIN_OPT_PIN_DENSITY_INFO" "1")
```

```
Virtuoso® ICADV12.2-64b - Log: /servers/cic_pd/saching/6.1.7.performance/tools.lnx86/dfII/test/vfp/pinOpt/pgg_width_on_pins/saching_1.
File  Tools  Options  Help
The pin optimizer ran successfully..

Checking pin overlaps ...
End pin overlap checking.
Checking pin spacing ...
End pin spacing checking.
*WARNING* (DB-270000): dbCreateLayerBlockage: Points for blockage contains coincident or collinear vertices.
Pin assignment starts execution.
*WARNING* (VPA-21455): The bus pins 'c1<0:24>' of cellview 'PinTest3:Cell1:layout' cannot be placed on the cell boundary due to insufficient spa
there is sufficient space available at the cell boundary before you run pin optimization again.
*WARNING* (VPA-21464): Cannot place bus pins 'c1<0:24>' because none of the edges of cellview 'PinTest3:Cell1:layout' have sufficent contiguous
these pins together.

Detailed Pin/Slot Density Information :  Edge : Occupied Slots (LayerName Pin/Slot Density)
                                               Required Slots (LayerName Pin/Slot Density)
    Design : PinTest3:Top:layout
         Edge BOTTOM :Occupied Slots By Assigned Pins/Shapes => (M1 0)
                      Required Slots by UnAssigned Pins      => (M1 0)
         Edge RIGHT :Occupied Slots By Assigned Pins/Shapes => (M1 0.293)
                        Required Slots by UnAssigned Pins      => (M1 0.333)
         Edge TOP :Occupied Slots By Assigned Pins/Shapes => (M1 0)
                  Required Slots by UnAssigned Pins      => (M1 0)
         Edge LEFT :Occupied Slots By Assigned Pins/Shapes => (M1 0.293)
                        Required Slots by UnAssigned Pins      => (M1 0.333)
    Design : PinTest3:Cell2:layout
         Edge BOTTOM :Occupied Slots By Assigned Pins/Shapes => (M1 0)
                      Required Slots by UnAssigned Pins      => (M1 0)
         Edge RIGHT :Occupied Slots By Assigned Pins/Shapes => (M1 0.415)
                        Required Slots by UnAssigned Pins      => (M1 0.472)
         Edge TOP :Occupied Slots By Assigned Pins/Shapes => (M1 0)
                  Required Slots by UnAssigned Pins      => (M1 0)
         Edge LEFT :Occupied Slots By Assigned Pins/Shapes => (M1 0.415)
                        Required Slots by UnAssigned Pins      => (M1 0.472)
    Design : PinTest3:Cell1:layout
         Edge BOTTOM :Occupied Slots By Assigned Pins/Shapes => (M1 0)
                      Required Slots by UnAssigned Pins      => (M1 0)
         Edge RIGHT :Occupied Slots By Assigned Pins/Shapes => (M1 0.415)
                        Required Slots by UnAssigned Pins      => (M1 0.472)
         Edge TOP :Occupied Slots By Assigned Pins/Shapes => (M1 0)
                  Required Slots by UnAssigned Pins      => (M1 0)
         Edge LEFT :Occupied Slots By Assigned Pins/Shapes => (M1 0.415)
                        Required Slots by UnAssigned Pins      => (M1 0.472)

*Error* VPA-30454: Failed to optimize the pins. Check the previous messages for more information.
```


© 2020 All Rights Reserved.

WSP/SP Support (ICADVM20.1 Only)

Pin Optimizer has been enhanced to support width-aware and constraint (CST)-aware optimization of pins.

Pin Optimizer recognizes the Width Spacing Snap Pattern grid defined using the `widthSpacingSnapPatternDef` rule in the technology file. The `widthSpacingSnapPatternDef` definitions are available in the technology database as well as in the design. Pin Optimizer recognizes both global and local `widthSpacingSnapPatternDef` definitions.

For more information about the `widthSpacingSnapPatternDef` rule and related technology file information, see the Width Spacing Pattern Layer Rules section in *Virtuoso Technology Data ASCII Files Reference*.

The pin optimization and snapping behavior for connected pins is different from that of unconnected pins. In both cases, the pin width is determined and compared with the WSP/SP track width. The pin width is determined by comparing the width of the pin with the `minWidth` constraint based on the CST lookup. The larger value is considered.

Example:

Pin width=0.4 (**considered**)

CST=0.3

For information about CST lookup, see the Process Rules for Objects section in V*irtuoso Unified Custom Constraints User Guide*.

Pin Optimizer honors the pin layer/side constraints while placing the pins on the WSP/SP tracks. The pin layer/side constraints are honored only when WSP/SP tracks are available on the specified side. For example, consider a technology in which `Metal2` WSP tracks are available in the horizontal direction. The pin layer/side constraint on this layer is honored only when the pins are assigned to the *Left* or *Right* side. The constraint is not honored when pins are assigned to the *Top* or *Bottom* side because there are no WSP/SP tracks on this side.

For more information about the pin layer/side constraint, see Pin Layer/Side Constraints.

Connected Pins

Connected pins are snapped to tracks with widths greater than or equal to the pin width, while ensuring the minimum net length. While snapping, pins are resized to square-shaped pins with dimensions that match the WSP/SP track width. Pins are centered along the WSP/SP tracks on the snapping layer that is identified in the WSP/SP definition.

© 2020 All Rights Reserved.

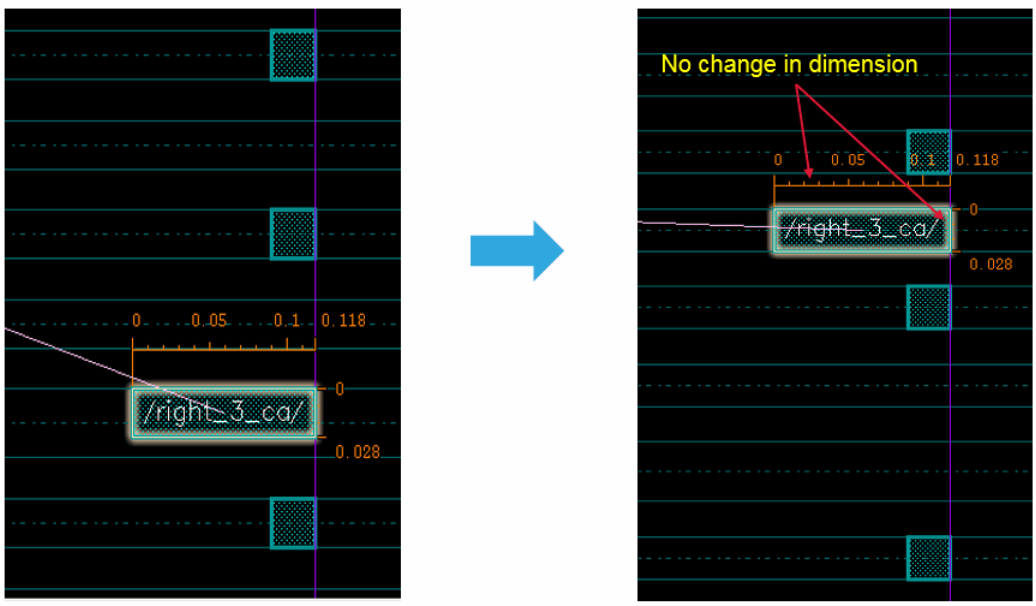
Caution

The original pin area may not be maintained during pin snapping.

Set the retainLongPin environment variable to `t` to retain the dimension of long, rectangular pins while snapping to WSP/SP tracks. The default value is `nil`, in which case pins are resized to square-shaped pins to match the WSP/SP track width.

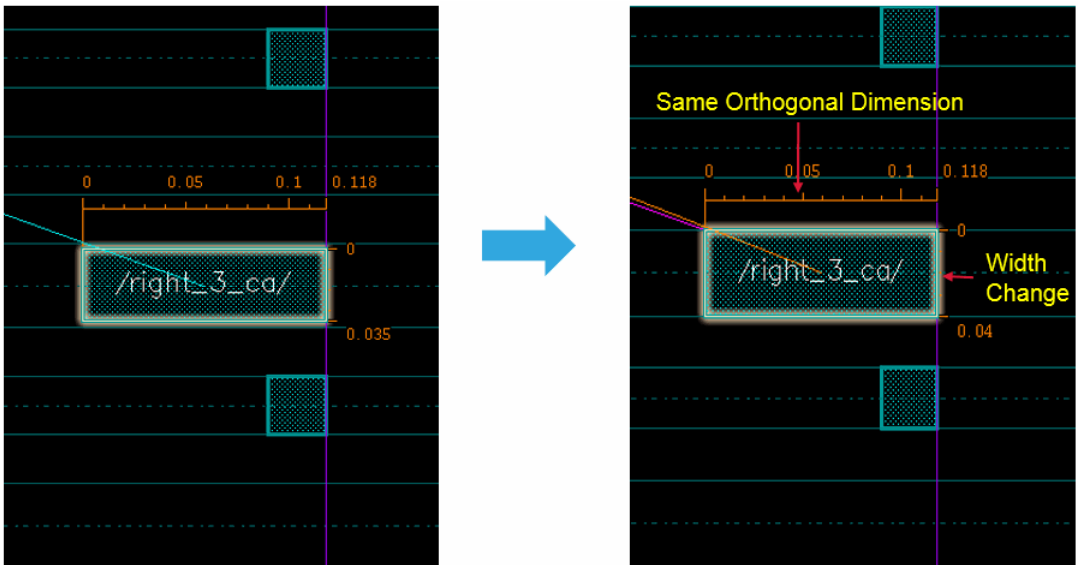
The following examples depict the snapping behavior in various scenarios when `retainLongPin` is set to `t`.

Example 1: A long pin is snapped to a track of the same width. There is no change in the pin dimensions.

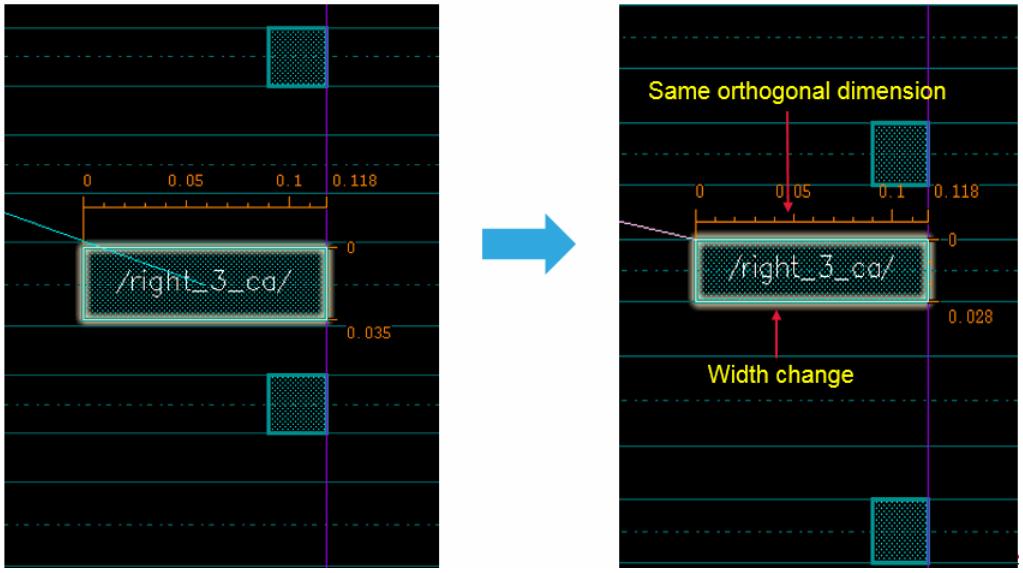


© 2020
All Rights Reserved.

Example 2: A long pin is snapped to a wider track. The pin width is adjusted as per the track width, but the orthogonal pin dimension remains the same.

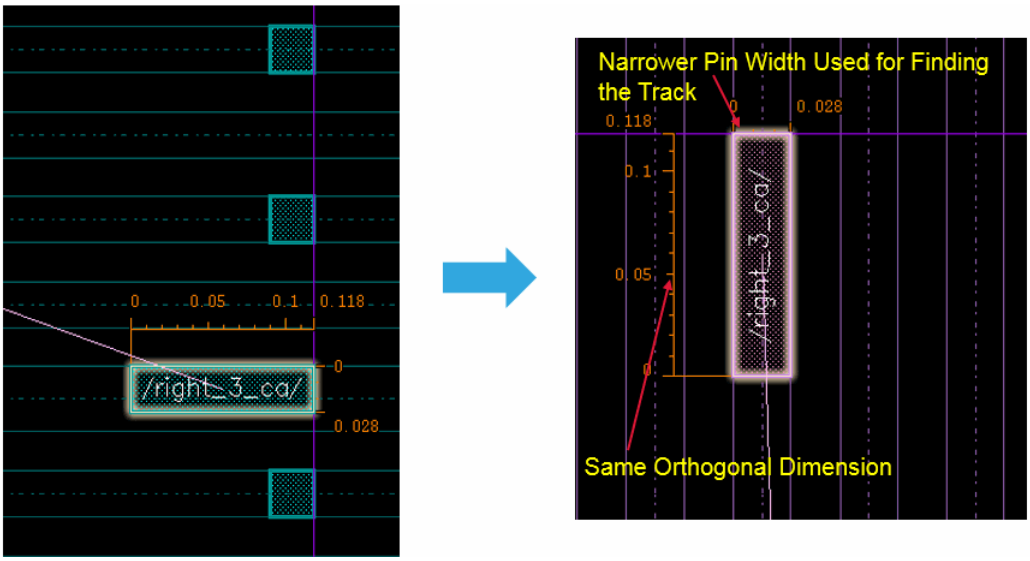


Example 3: A long pin is snapped to a narrower track. The pin width is adjusted as per the track width, but the orthogonal pin dimension remains the same.

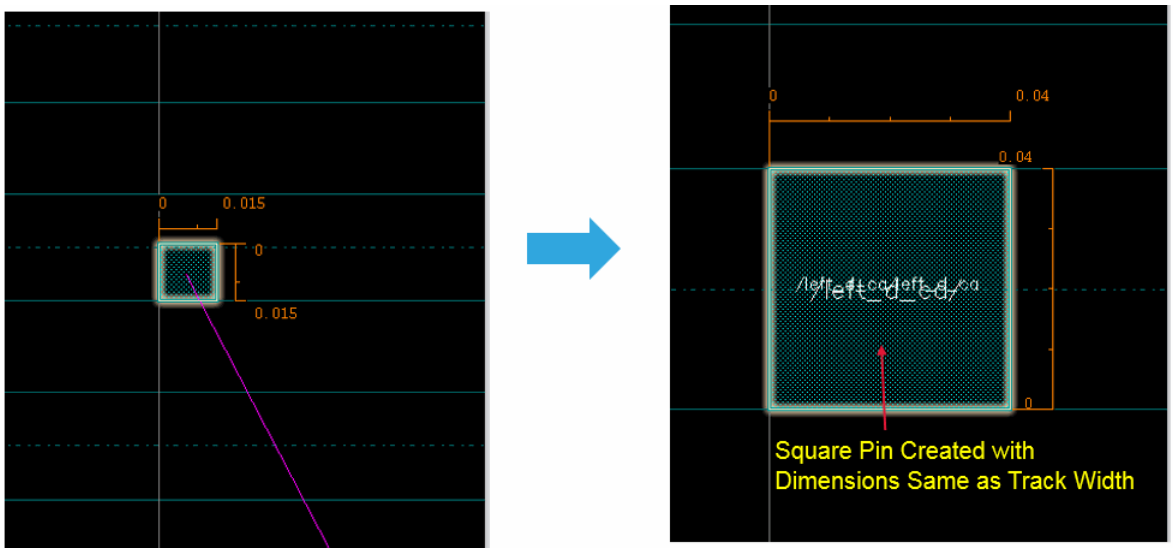


© 2020
All Rights Reserved.

Example 4: A long pin is snapped to an orthogonal edge, typically due to a layer change. The shorter pin dimension is used to find a suitable width-aware track. The shorter pin edge is snapped to the track, and so the orthogonal dimension is maintained.



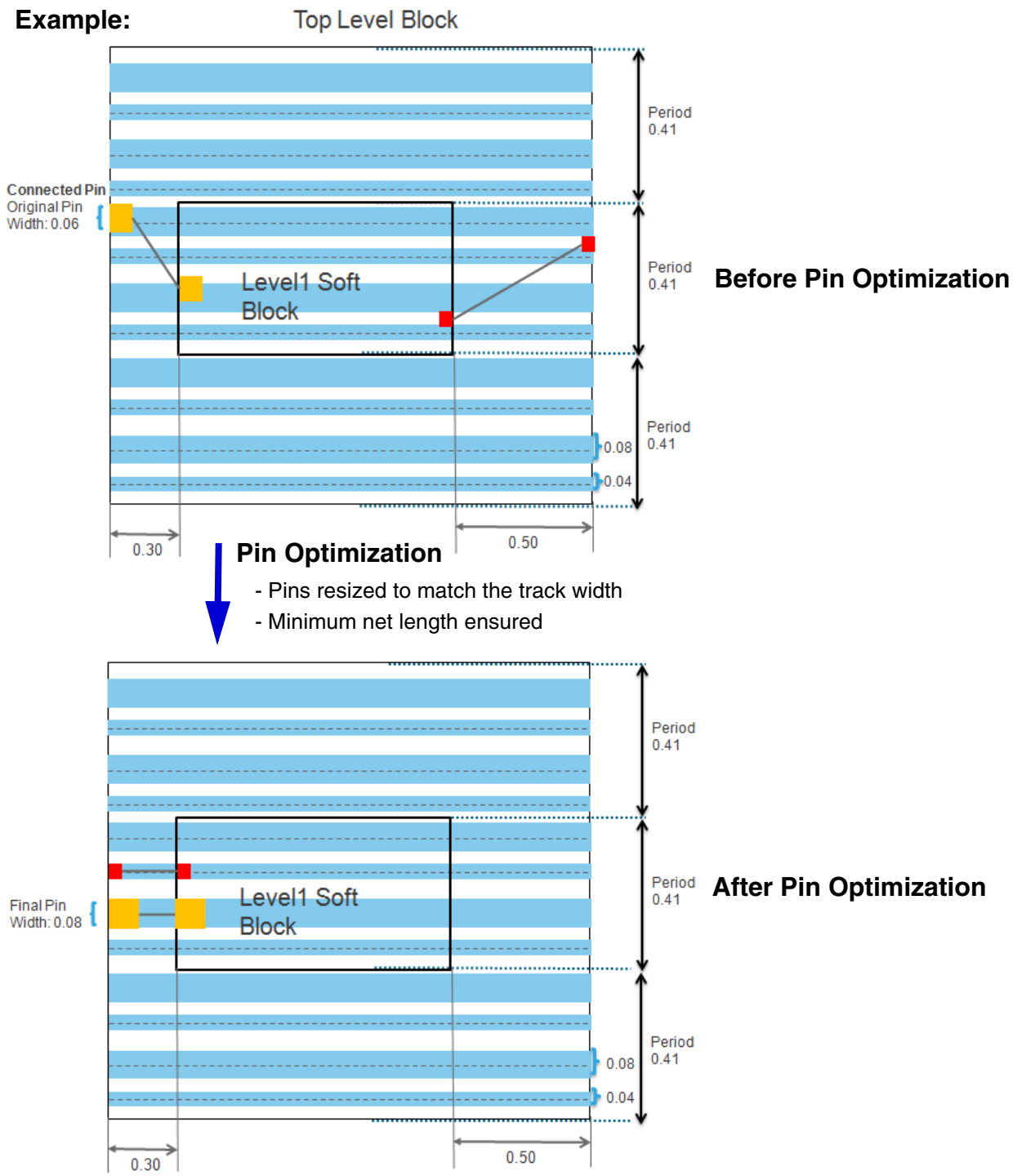
Example 5: A square pin is snapped to a wider width track. A square pin is created that matches the width of the snapped track.



© 2020
All Rights Reserved.

Note: The Alignment (Side/Edge), Order, and Fixed constraints continue to have the highest priority. For example, if `pin_1` has the side `Left` edge constraint, then only the tracks on the left side are considered.

Example:



© 2020
All Rights Reserved.

In the above figure, for the yellow pins, only those tracks with width >=0.6 are considered for placement; whereas for the red pins, all the tracks are considered.

During pin optimization, the yellow pin is snapped to a track such that the minimum net length is achieved.

The red pin has multiple tracks available. Such pins are snapped to ensure minimal net length, and not minimal pin resize. Further, the pins are placed on the left to ensure minimum net length. While snapping, the pins are resized to square-shaped pins with dimensions that match the WSP/SP track width.

If no matching WSP/SP tracks are found, then the pins are snapped to tracks such that the minimum net length is achieved.

Note: If WSP/SP tracks are not available, then pins are snapped to the manufacturing or routing grids, depending on the block type.

To match a pin's layer and size to the connected fixed pin's layer and size, use the options in the *Match Fixed Pins Layer/Size* section on the *Pin Optimization* tab of the Pin Placement form. For more information about this option, see Match Fixed Pins Layer/Size.

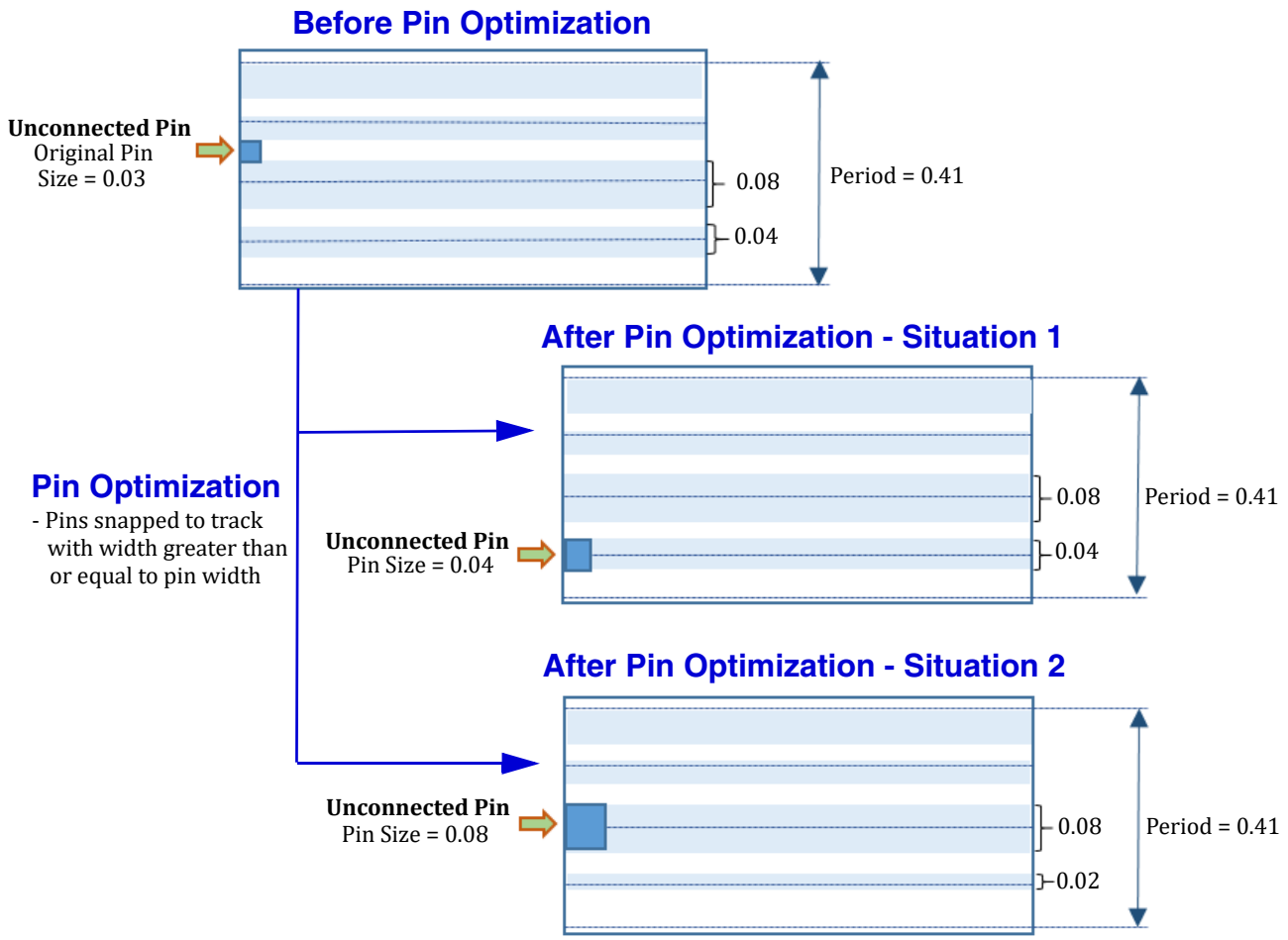
Unconnected Pins

Unconnected pins are snapped to WSP/SP tracks with widths equal to or greater than the pin width. While snapping, the pins are resized to square-shaped pins with dimensions that match the WSP/SP track width. The following figures depict the snapping behavior of a pin with an original size 0.03.

In **Situation 1**, there are two available tracks of widths 0.08 and 0.04. So, the pin is snapped to the WSP/SP track with width 0.04, and resized accordingly.

© 2020 All Rights Reserved.

In **Situation 2**, there are two available tracks of widths 0.08 and 0.02. This time, the pin is snapped to the WSP/SP track with width 0.08, and resized accordingly.



Blockage Support

Pin Optimizer supports routing blockages. Therefore, during pin optimization, a pin is not placed on a WSP track if the placement would lead to an overlap between the pin and blockages on the pin's layer. For more information, see Support for Rectangular and Square Pins and Blockages.

Pin Snapping for Overlapping Regions

In certain designs, there could be an overlap between two WSP regions that have the same snapping layer. The following types of overlaps are possible:

© 2020 All Rights Reserved.

- Partial, between two regions
- Exact, between two regions of the same size
- Complete enclosure of a smaller region within another larger region

Pin Optimizer follows certain rules that define the pin snapping behavior for such overlapping regions. For more information about these rules, see Pin Snapping for Overlapping Regions.

Support for the `allowedWireTypes` Constraint

Pin Optimizer has been enhanced to support the `allowedWireTypes` constraint. When this constraint is set on a net, pins on that net are placed on tracks with allowed wire types. Pin Optimizer matches the `allowedWireTypes` constraint specified on the nets with the track `wiretype` value. If multiple matches are found, then the tack with the minimum net length is chosen for placement.

For more information on the `allowedWireTypes` constraint, see Assigning a Wire Type to a Net (ICADVM20.1 Only).

Note: Pin Optimizer achieves the minimum net length while honoring the constraints and the priority order mentioned below:

a. Matching wire type

b. Matching width

For information about CST lookup, see the Assigning a Wire Type to a Net (ICADVM20.1 Only) > Default Constraint Types section in Virtuoso Unified Custom Constraints User Guide.

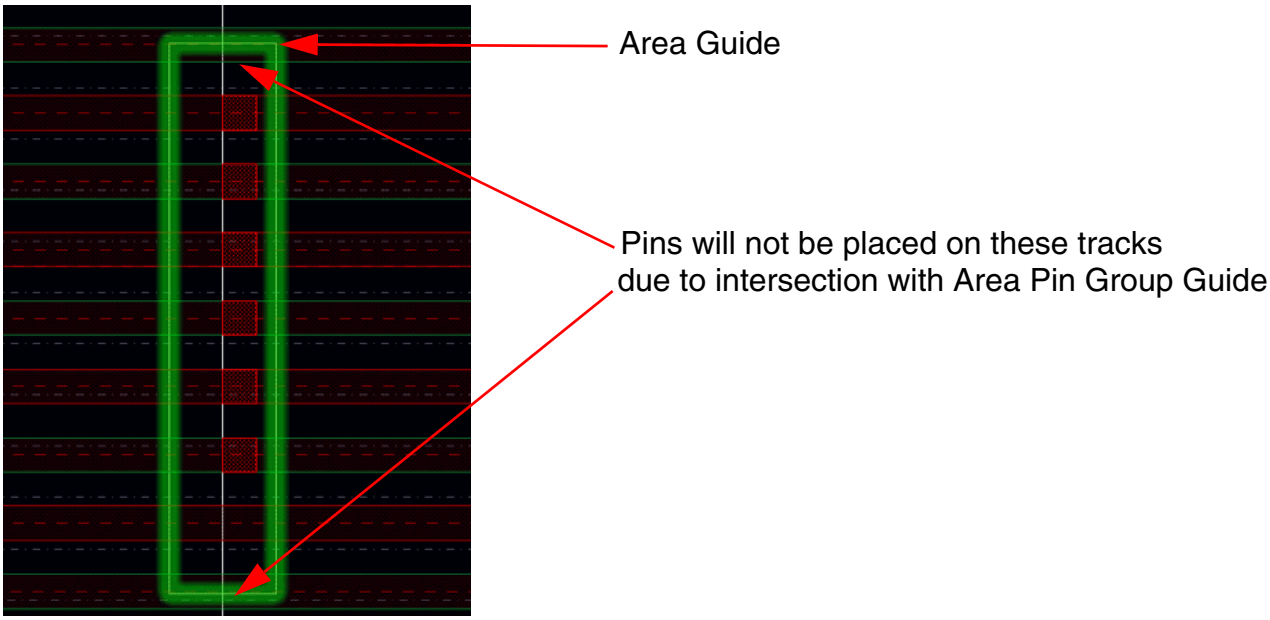
Caution

As WSP is a space-corrected grid, a pitch value does not make sense. Therefore, it is not supported.

© 2020 All Rights Reserved.

Support for Pin Group Guides

If a design with WSP patterns has a pin group guide defined on its PR boundary, then the pins in the pin group guide are snapped to nearest WSP/SP track within the guide. However, if a pin group guide only intercepts a WSP/SP track, then the pins cannot be placed on the intercepting track, as shown below:

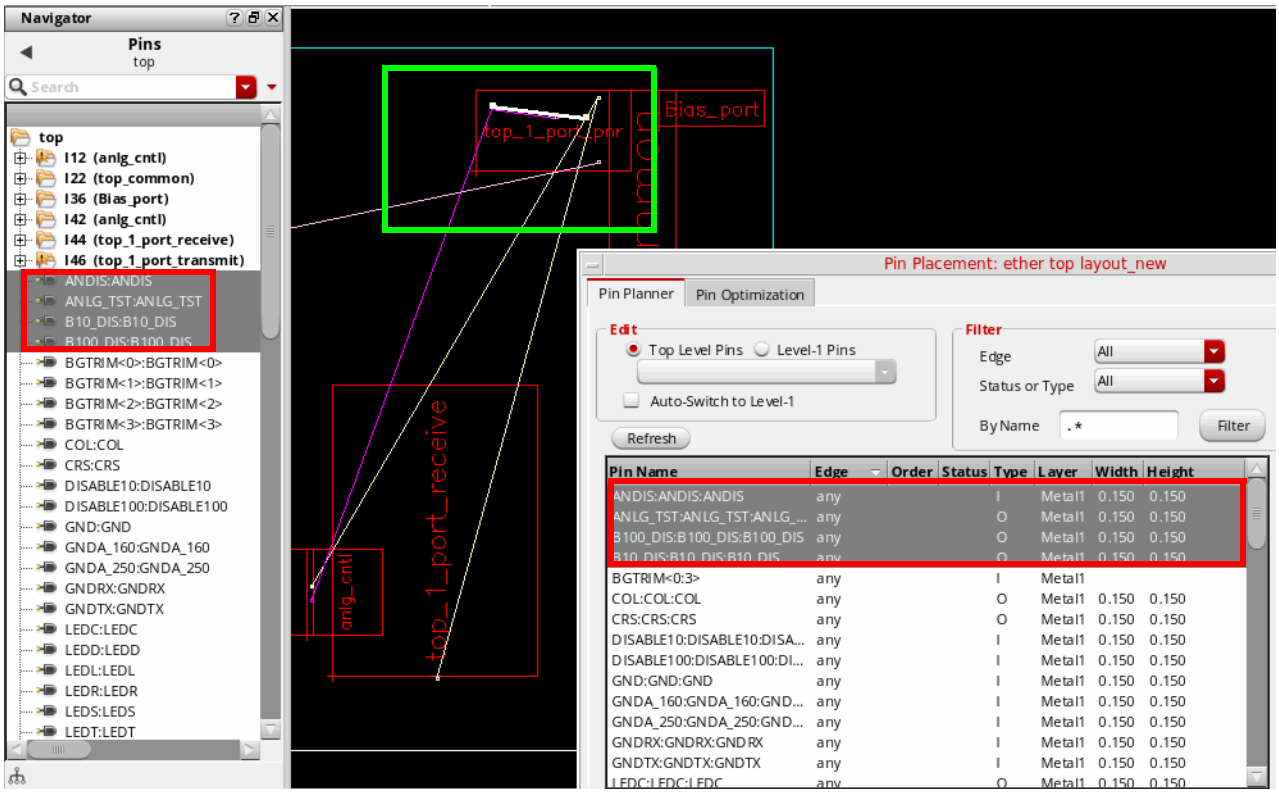


For more information about pin group guides, see Creating Pin Group Guides.

© 2020
All Rights Reserved.

Cross-Selection of Pins

Any pin that you select in the pin table, as shown below, also gets selected in the navigator assistant and the layout canvas, or vice versa.



Important

The pin selection use-model has been enhanced such that any pins selected in the pin table are now **highlighted and selected** in the layout canvas. This enables directly selecting the pins from the pin table and editing them in the layout canvas.

If you select a top-level pin in the layout canvas and the *Pin Planner* tab has the **Top Level Pins** edit mode selected, the pin table displays the selected pin as highlighted. However, if the edit mode selected in the *Pin Planner* tab is **Level-1 Pins**, the form automatically switches to the **Top Level Pins** mode to show the cross-selection. The automatic mode switching works as long as the selected layout pins belong to the same level—top or level-1. If you select a mix of top-level and level-1 pins in the layout canvas, the *Pin Planner* tab continues to be in the current selected mode and displays only those pins as highlighted in the pin table that belong to the current edit mode.

© 2020 All Rights Reserved.

The table below summarizes how the *Pin Planner* tab handles the cross-selection depending on the layout pins selected. Let us assume here that the default edit mode selected is **Top Level Pins:**

Selected Pin	Pin Planner Tab Behavior
`Only top pins`	Shows the selected top pins in the pin table.
`Only soft block instance pins`	Automatically switches to the **Level-1 Pins** mode and shows the selected instance pins in the pin table.
`Top pins + Soft block instance pins`	No automatic mode switching. The **Top Level Pins** mode shows the selected top pins and the **Level-1 Pins** mode shows the selected instance pins.
`Only soft block instance`	Automatically switches to the **Level-1 Pins** mode but shows no selection in the pin table as the instance “pin” is not selected.
`Soft block instance + Soft block instance pins`	Automatically switches to the **Level-1 Pins** mode and shows the selected instance pins in the pin table.
`Top pins + Soft block instance`	No automatic mode switching. The **Top Level Pins** mode shows the selected top pins.
`Top pins + Soft block instance + Soft block instance pins`	No automatic mode switching. The **Top Level Pins** mode shows the selected top pins and the **Level-1 Pins** mode shows the selected instance pins.
`Top pins + Soft block instance + Soft block instance pins + Pins of another soft block`	No automatic mode switching. Each level shows the selected pins at that level.

Note:

- To add to your selection set in the pin table, incrementally select the pins using the `Ctrl` or `Shift` key, as appropriate.

© 2020 All Rights Reserved.

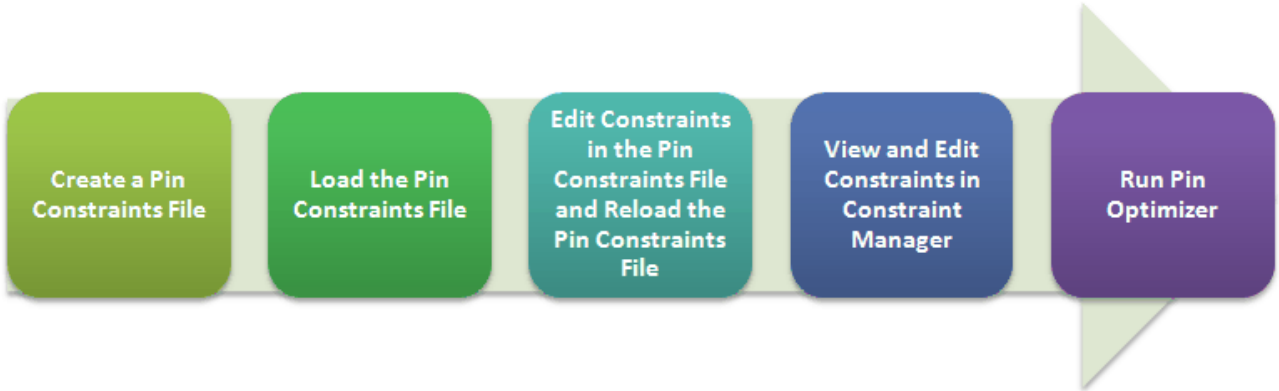
- To nullify your existing selection:
  - In the pin table, click any pin. This nullifies the previous selection and shows only the clicked pin as selected.
  - In the layout canvas, click anywhere. This nullifies the previous selection and shows no pins as selected. However, if you click a pin in the layout canvas, the previous selection is nullified and the clicked pin appears as selected.

© 2020 All Rights Reserved.

Using a Pin Constraints File

Typically, designers use Pin Optimizer, Pin Planner, or Constraint Manager to set pin constraints. These methods are useful while setting pin constraints for smaller designs with few pins. However, for large, complicated designs with hundreds of pins, these methods may require a lot of time and effort. For such designs, you can create a pin constraints file that includes all the pin constraint definitions. This file can be loaded using a SKILL function. Once loaded, constraints are created, which can be viewed and edited in the Constraint Manager. During pin optimization, these constraints are honored.

The following image depicts the use of a pin constraints file for defining constraints:



Step 1: Creating a Pin Constraints File

A pin constraints file is a SKILL file with the following constraints entered in a SKILL list:

- Edge Constraint
- Pitch Constraint
- Status Constraint

The pin constraints file also includes:

- A Sanity Checker to validate all parameters before running Pin Optimizer.
- An API that automatically loads constraints from the pin constraints file while the file is being read.

You can create a pin constraints file by either entering the constraints manually in a text editor or using a script to generate the file.

© 2020 All Rights Reserved.

Tip

A sample pin constraints file, `vfpLoadPinCons.il`, is available in the `dfII/samples/vfp/` directory for your reference. Use the file as a reference for creating a pin constraints file.

For more information about the pin constraints file, see Pin Constraints File.

To know more about the constraints supported by Pin Optimizer, see Constraints Supported by Pin Optimizer.

Step 2: Loading the Pin Constraints File

Use the `load(file_name)` command to load the pin constraints file. The constraints in the file are read and loaded in the Constraint Manager.

Tip

Utility functions to load pin constraint file with and without sanity checkers are available in the sample file. To use these functions, un-comment them as per your requirement.

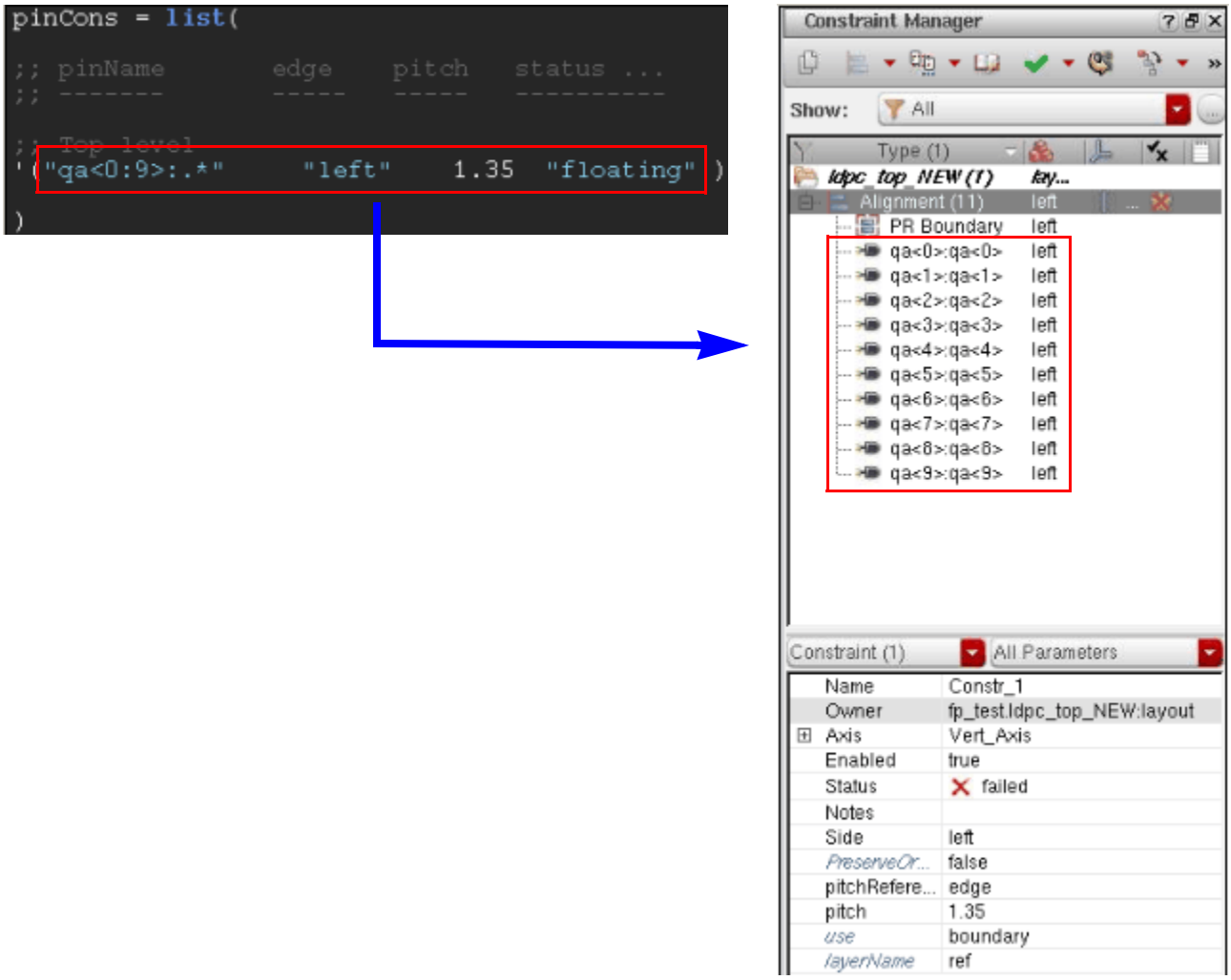
Step 3: Edit Constraints in the Pin Constraints File and Reload the File

While loading the pin constraints file, parsing errors, if any, are displayed in the CIW. Make the required edits to the pin constraints file and reload the updated file. Ensure that there are no parsing errors.

© 2020 All Rights Reserved.

Step 4: Viewing and Editing Constraints in the Constraint Manager

Open the Constraint Manager to view the constraints. Notice that the constraints in the pin constraints file are updated in the Constraint Manager. Make edits, if required, before running Pin Optimizer.

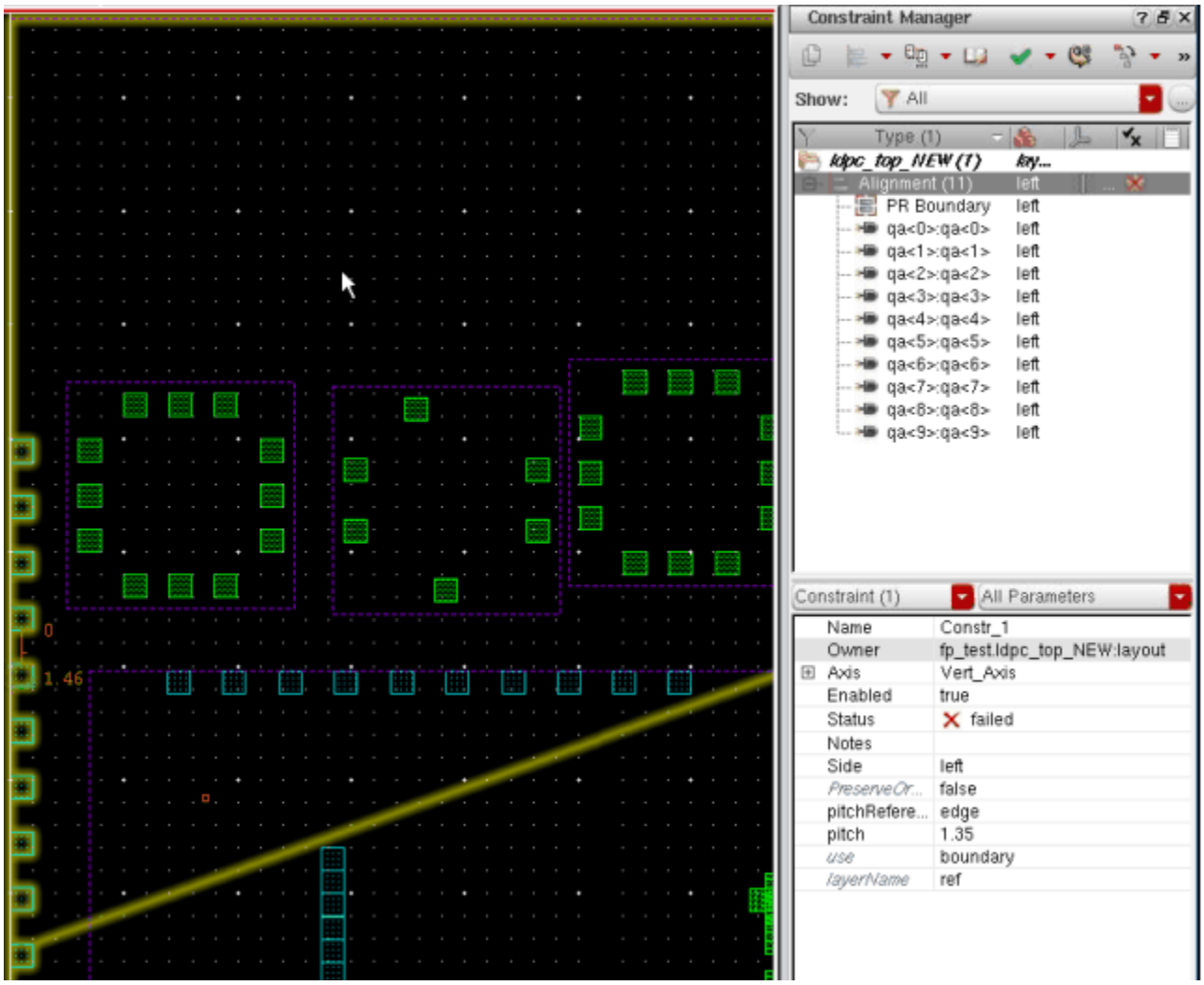


Step 5: Running Pin Optimizer

To run Pin Optimizer, choose *Floorplan – Pin Optimization*. For more information about Pin Optimization, see **Using the Pin Optimization Tab**.

© 2020 All Rights Reserved.

Notice that Pin Optimizer has honored the constraints during pin optimization.



© 2020
All Rights Reserved.

Running the Pin Checker (ICADVM20.1 Only)

Pin Checker lets you perform the following checks on pins and report the results:

- Check whether the pin layer directions match the assigned edge directions.
- Check whether soft blocks pins are:
  - Aligned.
  - On the same layer.
  - Of the same size.
- Check for exact pin-to-pin abutment of connected pins.
- Report nets with multiple pin shapes.
- Report pin overlaps, if any.

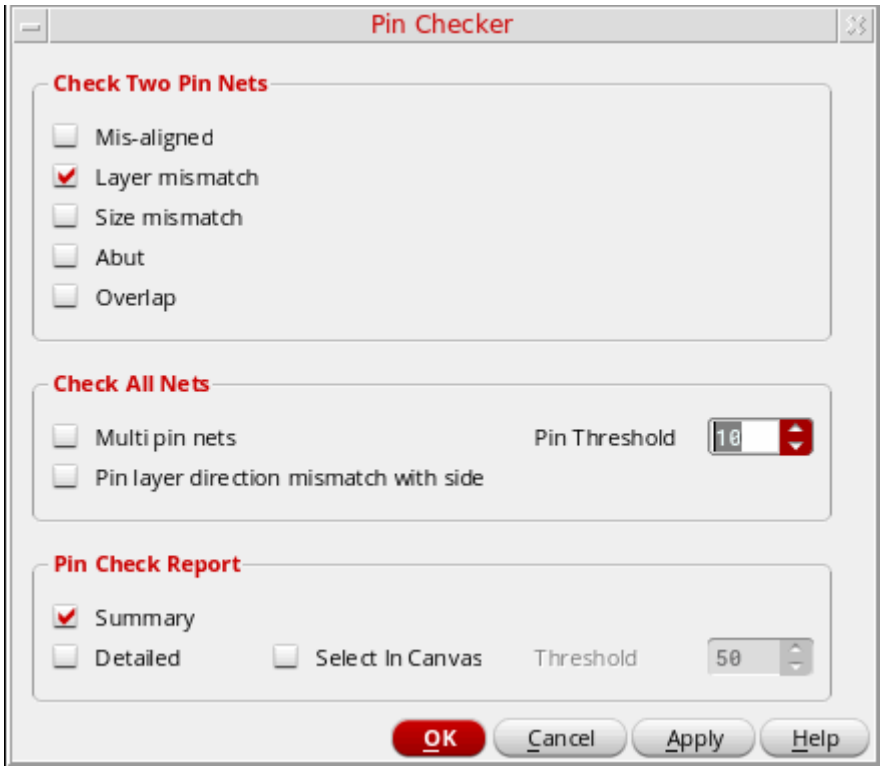
Important

It is recommended that you use the Pin Checker to validate Pin Optimization results.

© 2020 All Rights Reserved.

Using the Pin Checker

Select *Floorplan – Pin Checker* to display the Pin Checker form.



The form is divided into three sections that let you perform the following tasks:

- Check Two Pin Nets
- Check All Nets
- Customize the Pin Check Report

Check Two Pin Nets

The options in this section are applicable only to the nets that are connected to two pins. Violations are reported in the Pin Check Report.

- **Mis-aligned:** Checks whether the connected pins are correctly aligned.
- **Layer mismatch:** Checks whether the connected pins are on the same layer.
- **Size mismatch:** Checks whether the dimensions of the connected pins are the same.
- **Abut:** Checks whether the connected pins are abutted to each other.

© 2020 All Rights Reserved.

- **Overlap:** Checks whether the connected pins overlap each other.

Check All Nets

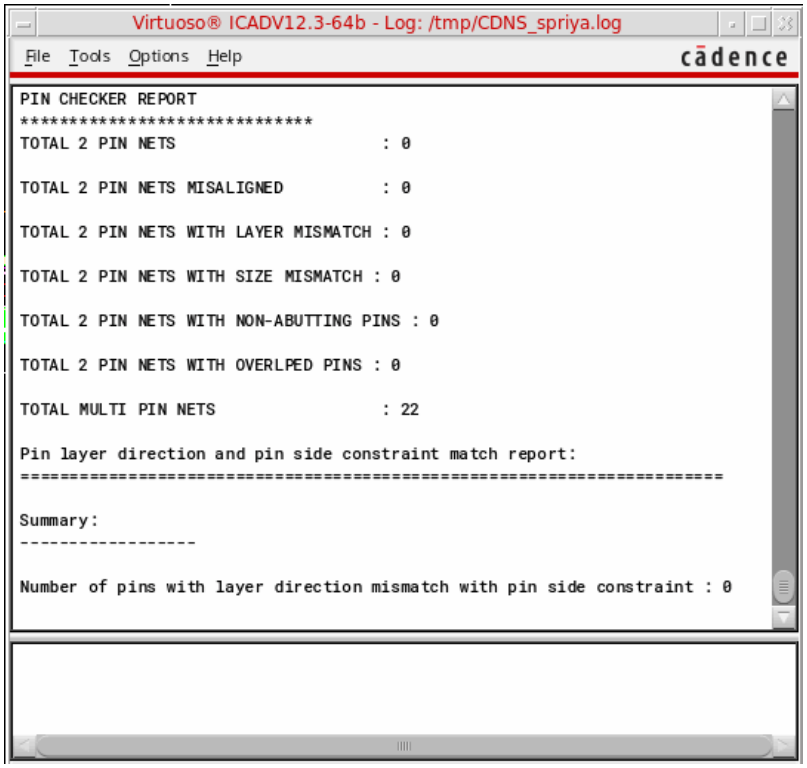
The options in this section are applicable to all nets. Violations are reported in the Pin Check Report.

- **Multi pin nets:** Identifies nets that have multiple pins and lists their names in the Pin Check Report. Specify a *Pin Threshold* to restrict the check to nets that have the specified number of pins.
- **Pin layer direction mismatch with side:** This check, applicable only to the pins that have side layer constraints, identifies the pins for which pin layers do not have tracks in the specified pin side.

Customize the Pin Check Report

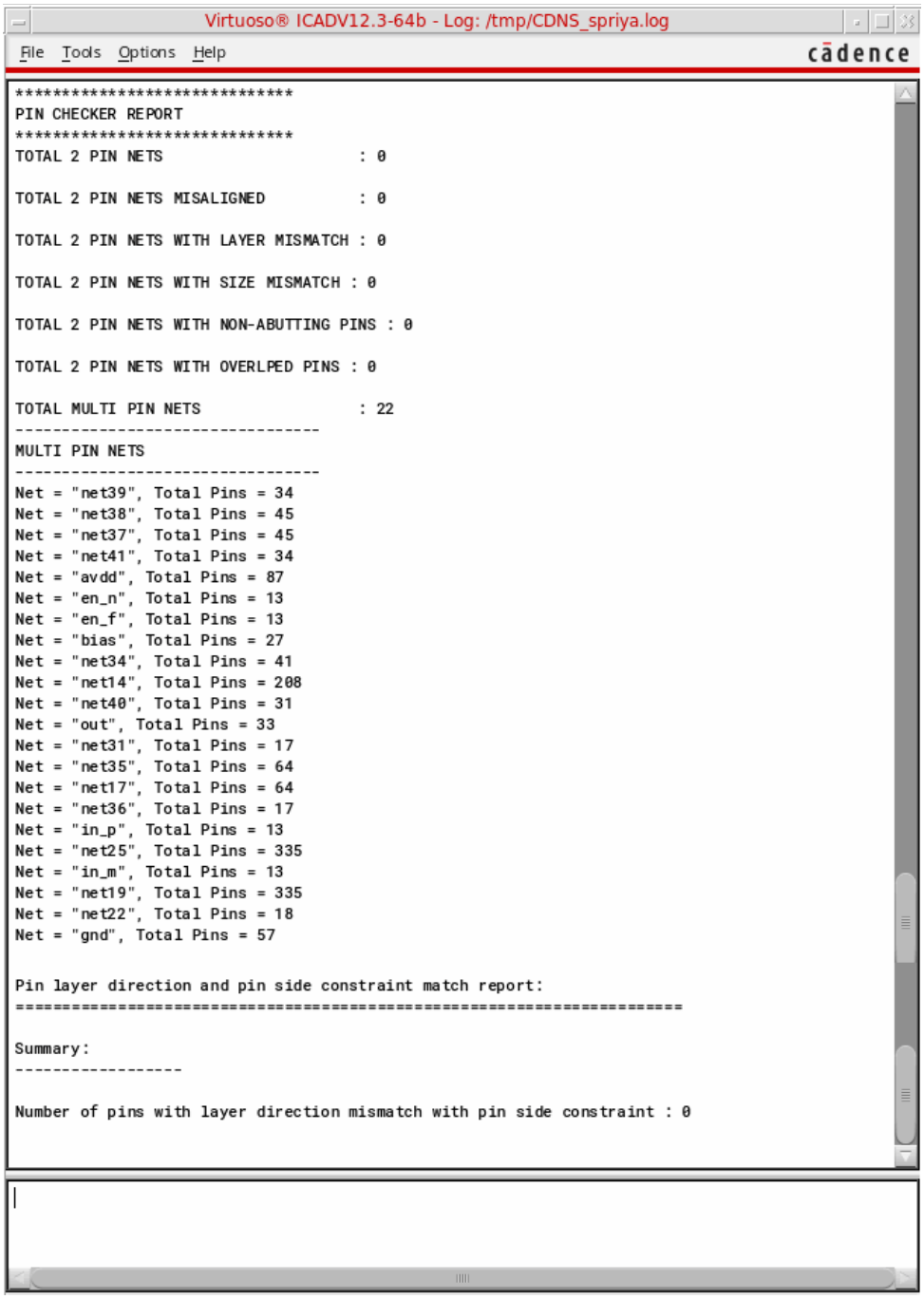
After running Pin Checker, the Pin Check Report is generated in the CIW. This report provides information about the violations. The following options in the Pin Check Report section let you customize the information displayed in the report:

- **Summary:** Generates a summary report about the checks, as shown below:



© 2020 All Rights Reserved.

- **Detailed:** Generates a detailed report in the CIW with information about nets as per the settings. The violations are also listed in the *Misc* tab of Annotation Browser. The following image shows a detailed report:



- **Selected In Canvas:** Restricts the information displayed in the Pin Checker Report only to those nets that are selected in the design canvas.

© 2020 All Rights Reserved.

- **Threshold:** Specifies the maximum number of nets to be considered for each check.

© 2020 All Rights Reserved.

9

Using the Pin Tool

Virtuoso Floorplanner includes the Pin Tool, which is a unified interface for all pin-related tasks, such as creating pins, resizing pins, planning and optimizing pins, editing pin attributes, and setting their location constraints.

Before launching the Pin Tool, ensure that the blocks in the design are placed. You can then use the Pin Tool to define the pin location constraints, alignments, and pin group guides, and set the pin attributes.

This chapter contains the following topics:

- Understanding the Pin Tool Interface
  - Pin Browser
  - Label Browser
  - Net Shapes Browser
- Creating Pins
  - Creating Pins
  - Creating Pins from Labels
  - Creating Pins from Net Shapes
  - Creating Pins Automatically
- Promoting Pins
- Resizing Pins
- Making Pins Electrically Aware
- Merging Pins
- Planning and Optimizing Pins

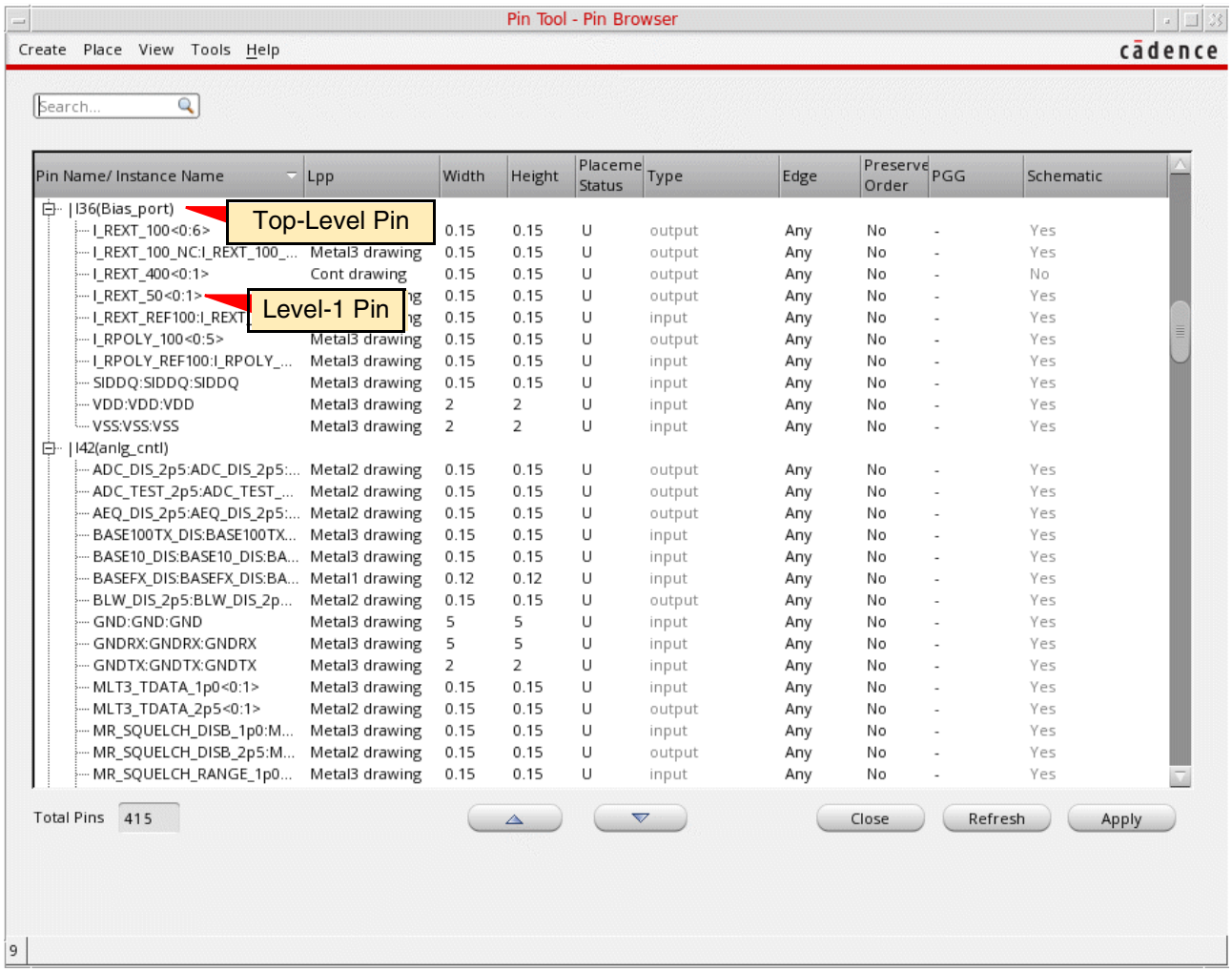
© 2020 All Rights Reserved.

Understanding the Pin Tool Interface

Choose *Floorplan – Pin Tool* or click the Pin Tool button in the Floorplanning toolbar to launch the Pin Tool.

Important

In Layout EXL, choose *Plan – Pin Planning – Pin Tool*.



The Pin Browser appears, which lists all the top-level and level-1 pins that are available in the current layout design.

© 2020 All Rights Reserved.

Menu Bar

The Pin Tool menu bar includes the following menus and commands:

Menu	Command	Description
Create	*Creating Pins*	Creates pins in the layout canvas.
	Creating Pins from Labels	Creates pins from existing labels in the cellview.
	Creating Pins from Net Shapes	Creates pins from existing net shapes in the cellview.
	Promote Pins	Promotes pins from a lower level in a design hierarchy to a higher level.
Place	*Pin Planner*	Lets you assign pin constraints and attributes.
	Pin Optimizer	Positions pins of blocks in a manner that helps obtain the shortest possible net length at a particular level in the design.
	Resize Pins	Lets you register and run pin resize estimator SKILL hook functions on all or selected pins in the cellview.
View	*Pins*	Opens the Pin Browser.
	Labels	Opens the Label Browser.
	Net Shapes	Opens the Net Shapes Browser.
Help		Opens Help documentation.

© 2020 All Rights Reserved.

The Pin Tool Browser Windows

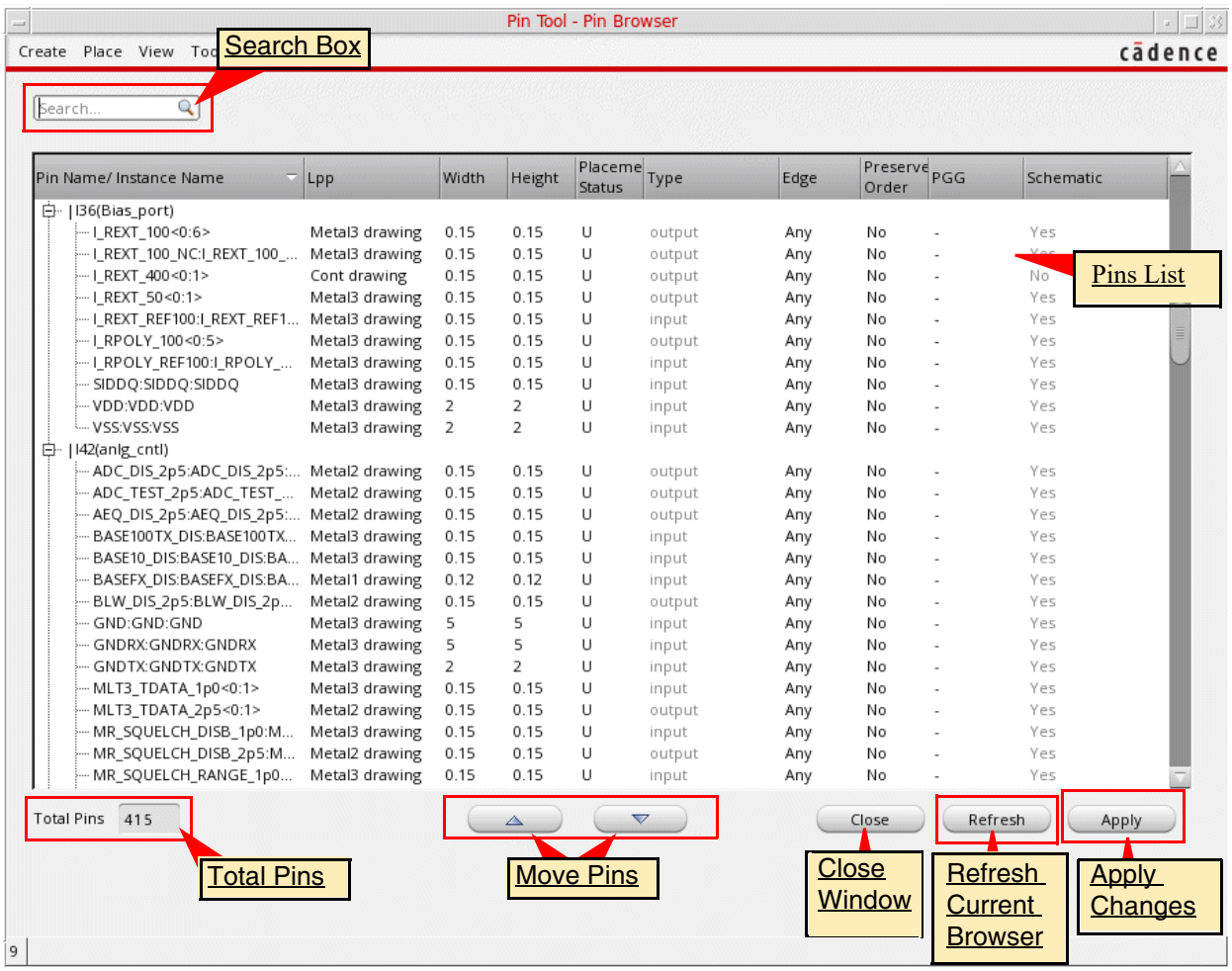
The Pin Tool lets you invoke the following Pin Tool browser windows:

- Pin Browser (default): Lists all pins that are available in the design.
- Label Browser: Lists all labels that are available in the design.
- Net Shapes Browser: Lists all net shapes that are available in the design.

Use the commands in the *View* menu to switch between these browser windows.

Pin Browser

The Pin Browser lists all the top-level and level-1 pins that are available in the current layout design. The interface includes a pins list and a few additional options to customize the display of pins. Use the *Search* box to filter the pins in the pins list.



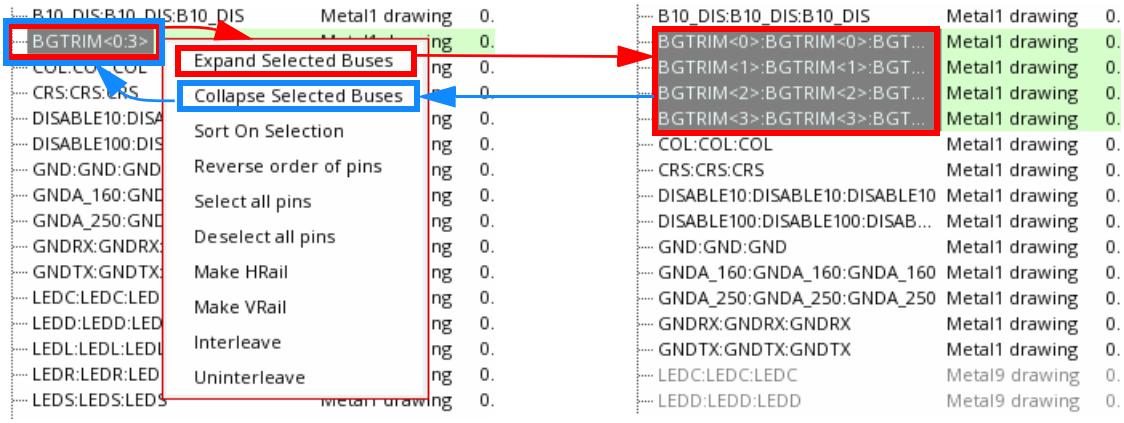
© 2020 All Rights Reserved.

Pins List

The pins list includes the following information about top-level and level-1 pins:

- **Pin Name/Instance Name:** Specifies the pin or instance name. The pin entries are expressed in the format `termName:pinName:figName`.

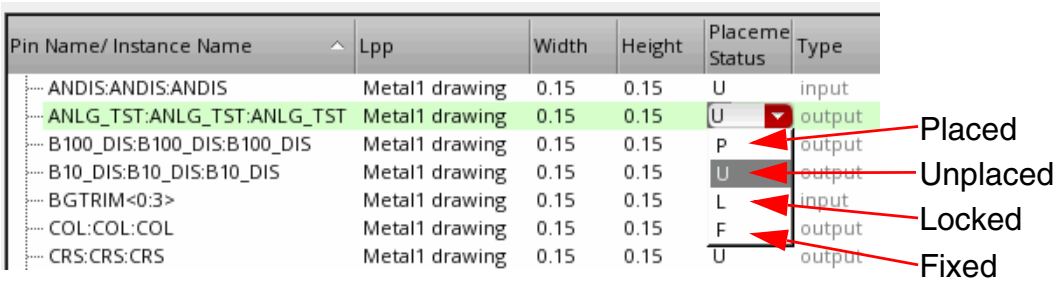
  The pins list also includes names of iterated bus pins. By default, bus pins are in the collapsed state. To expand buses, select the required bus pins, right-click, and select *Expand Selected Buses* from the shortcut menu. To collapse buses, choose *Collapse Selected Buses* from the shortcut menu.



- **LPP:** Specifies the layer-purpose pair on which the pin is drawn. You can edit the value directly in the Pin Browser. Click *Apply* to move pins to the new LPP in the database, and therefore in layout canvas.

- **Width** and **Height:** Specify the width and height of the pin. You can edit these values directly in the Pin Browser. Press `Enter` to accept the new value. Click *Apply* to update the revised pin dimension in the layout canvas.

- **Placement Status:** Represents the placement status of the pins. You can reset the value directly in the Pin Browser. Valid values are:

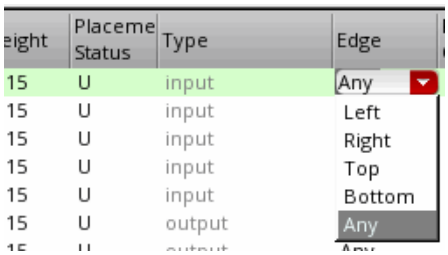


- **Type:** Specifies the pin type: *input*, *output*, or *input/output*. It is a read-only column.

© 2020 All Rights Reserved.

- **Edge:** Indicates the edge Alignment constraints set on the pins. The constraint is honored by various Virtuoso engines, such as the pin optimizer, pin planner, and constraint-aware editing engines.

  You can change the *Edge* directly in the Pin Browser. Valid values are:



- **Preserve Order:** Indicates whether the pin order specified in the Alignment constraint must be honored during pin placement. If selected, pins on the same edge are placed in the same order as they are listed in the Pin Browser.

- **PGG:** Specifies the name of the pin group guide that controls the placement of the pin. Pins that are associated with pin group guides cannot be edited directly in the Pin Browser, and are therefore displayed in **gray**.

  Click the *PGG* name in the Pin Browser to open the Pin Group Guide form, make the required changes, apply the changes, and close the form. Click *Refresh* in the Pin Tool to update the values as per the changes made in the Pin Group Guide form.

- **Schematic:** Specifies whether the top-level pins exist in the corresponding schematic cellview. This information is displayed only if the schematic view is open. The value is not editable.

Additional Options

- Use the *Search* box to filter the pins in the Pin Browser. For example, if you type a string `P` in the *Search* box, only the pins with their names starting with letter `P` are listed. As you type additional strings, the search results are further refined.

- View the total number of pins below the pins list. It is the sum of the top-level and level-1 pins in the list.

- Use the arrow buttons to change the order of the pins by moving them up or down in the list.

- Click *Refresh* when pin attributes have been changed outside the Pin Tool to replicate the changes in the Pin Browser.

- Click *Apply* to propagate the changes you made in the Pin Browser to the layout canvas.

© 2020 All Rights Reserved.

- Click *Close* to close the Pin Tool.

Using the Pin Browser

You can perform the following tasks in the Pin Browser:

- Select Pins
- Edit Pin Attributes
- Sort Pins
- Convert Pins to Rails
- Interleave Bus Pins

Select Pins

The Pin Browser lets you select pins in the pins list and edit their attributes. When you select a pin in the Pin Browser, the corresponding pin is highlighted in the layout canvas. Similarly, pins selected in the layout canvas are highlighted in the Pins Browser. To select all pins in the Pin Browser, right-click anywhere inside the Pin Browser and click *Select all pins* from the shortcut menu. After making the required edits to the pin attributes, choose *Deselect all pins* to clear selection.

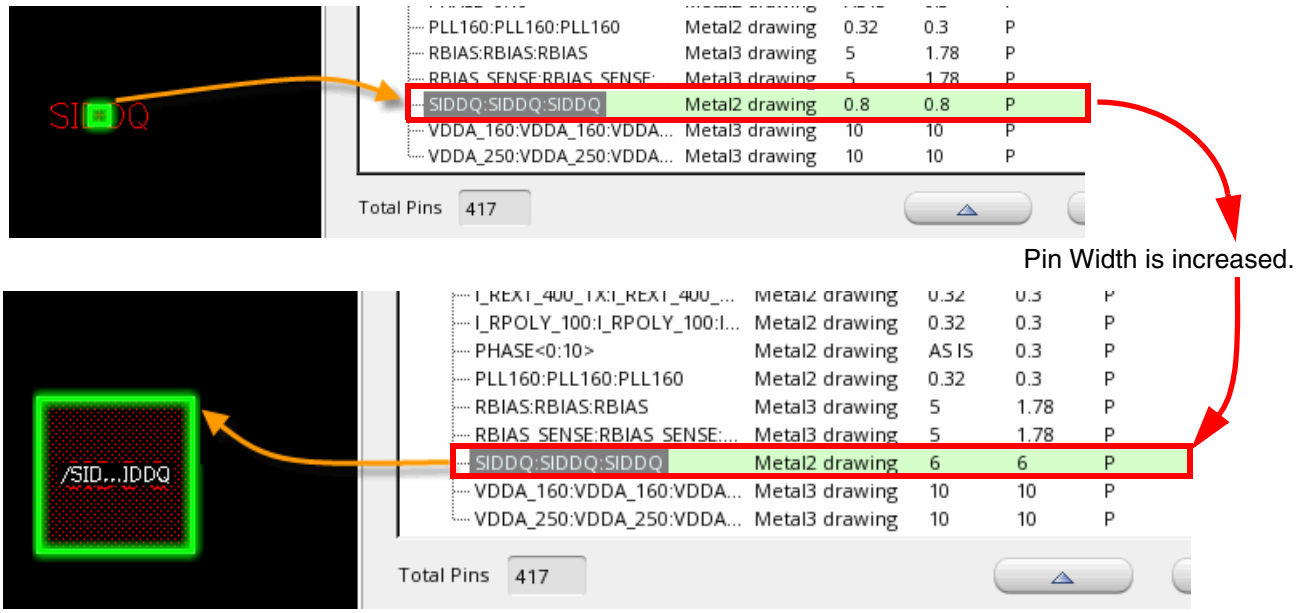
Edit Pin Attributes

The Pin Browser supports in-situ editing of the following pin attributes for top-level and level-1 pins that are not associated with any pin group guide:

- Pin Width
- Pin Height
- Placement Status
- Pin Edge
- Preserve Order setting

© 2020 All Rights Reserved.

In the example below, the width of a pin is increased. The change is applied to the layout canvas dynamically, as shown below:



The edited pin attribute is displayed in blue in the Pin Browser. You must click *Apply* to commit the change, after which the formatting of the pin attribute is reset to default.

Important

To edit the attributes of all selected pins, keep the `Shift` key pressed and make the required change to a pin entry, for example, change the *Width* of one of the selected pins, and click *Apply*. The change is propagated to all selected pins.

Sort Pins

By default, pins in the Pin Browser are sorted based on their *Pin Name/Instance Name* value. To sort by any other column, click the relevant column header; click again to reverse the sort order.

To sort a selected set of pins, right-click and choose *Sort on selection* from the shortcut menu. Choose *Reverse order of pins* to reverse the sort order.

Convert Pins to Rails

The Pin Browser lets you convert pins to either horizontal or vertical rails. The selected pin is stretched along the assigned edge to create a rail. Therefore pins assigned to the top or

© 2020 All Rights Reserved.

bottom edges are converted into VRails and pins assigned to the left or right edges are converted to HRails.

To create a rail:

1. Select the required pins in the Pin Browser
2. Right-click to display the shortcut menu.
3. Choose either *Make HRail* or *Make VRail*.

   Make HRail elongates the selected pins horizontally, from the left edge of the boundary to the right edge.

   Make VRail elongates the selected pins vertically, from the top edge of the boundary to the bottom edge.

These options are also available on the *Pin Planner* tab of the Pin Placement form. For more information, see Pin Planning.

Interleave Bus Pins

You can interleave the bits of the selected bus pins in the Pin Browser and automatically apply an order constraint to constrain how the individual bits will be placed.

Example:

Before Interleave:

```
A<0:2> A1 B<0:2> B1 C<0:1> C1 D1 E1 F1
```

After Interleave:

```
A<0> B<0> C<0> A<1> B<1> C<1> A<2> B<2> A1 D1 B1 E1 C1 F1
```

The selected bus must contain at least one pin that is already assigned to an edge. Unplaced pins are automatically placed on the same edge and their bits are interleaved. Interleave does not work if the pins in the selected set are assigned to more than one edge or to no edge at all.

To interleave bus pins:

1. Select the required pins in the Pin Browser. Ensure that the selected pins are assigned to valid edges.
2. Right-click and choose *Interleave*. An order constraint is applied automatically to specify how the individual pins will be placed.

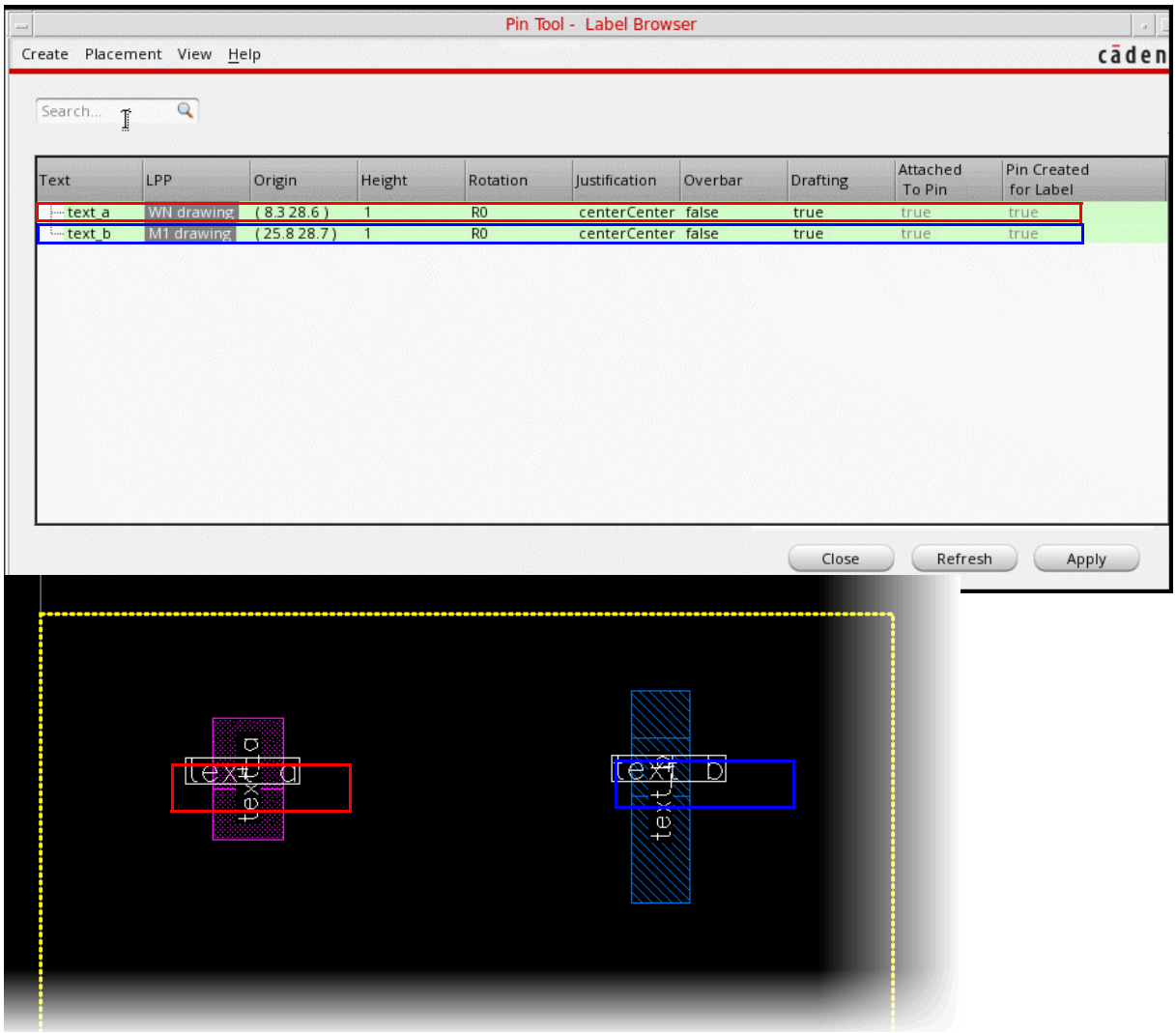
© 2020 All Rights Reserved.

The *Uninterleave* command works with pins assigned to any number of edges. However, it does not move pins from one edge to another, but instead deals with the pins on each edge separately. Uninterleave does nothing with unplaced pins; they are ignored.

The *Interleave* and *Uninterleave* options are also available on the *Pin Planner* tab of the Pin Placement form. For more information, see Pin Planning.

Label Browser

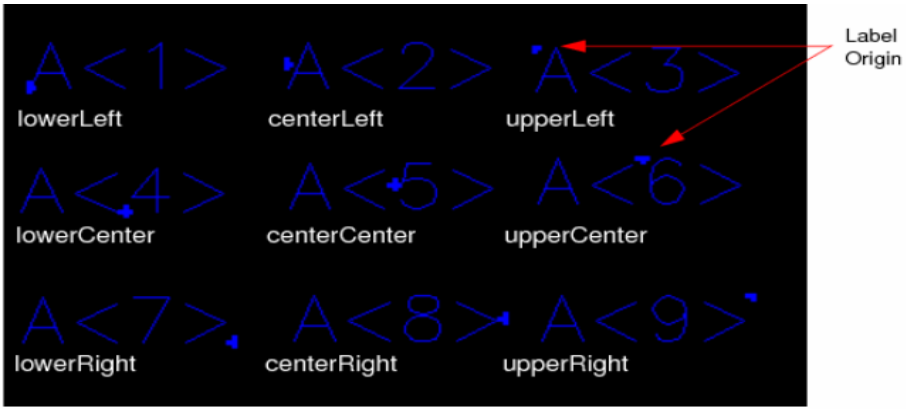
Choose *View – Labels* to display the Label Browser. This browser displays information about all labels that are available in the current cellview. The Pin Tool supports cross-selection of labels between the Label Browser and the layout canvas.



The Label Browser includes the following information about labels:

© 2020
All Rights Reserved.

- **Text:** Specifies the label name.
- **LPP:** Specifies the layer and purpose on which the label is drawn.
- **Origin:** Specifies the coordinates of the lower left corner of the label. The *Origin* and the *Justification* values together determine the label position. You can edit the value directly in the Label Browser. Click *Apply* to implement the change in the layout canvas.
- **Height:** Specifies the height of the label. You can edit the value directly in the Label Browser.
- **Rotation:** Specifies the orientation of the label. You can edit the value directly in the Label Browser.
- **Justification:** Specifies the position of the label origin with respect to the label position. The following figure shows the various justification values that are available:



You can edit the value directly in the Label Browser.

- **Overbar:** Indicates how the underscore characters in the label text strings must be displayed. When set to `false`, underscores are displayed as underscores; when set to `true`, underscores are displayed as overbars on the text string as shown in the following examples:

Property Name	Appears in Design Window As
`_abcde`	$\overline{\text{abcde}}$
`ab_cde`	$\text{ab}\overline{\text{cde}}$
`_abc_de`	$\overline{\text{abc}}\text{de}$
`ab_cd_ef_gh_ij`	$\text{ab}\overline{\text{cd}}\text{ef}\overline{\text{gh}}\text{ij}$

You can edit the value directly in the Label Browser.

© 2020 All Rights Reserved.

- **Drafting:** When set to `true`, prevents the label from being rotated more than 90 degrees. You can edit the value directly in the Label Browser.
- **Attached to Pin:** Denotes whether the label is attached to any pin in the cellview.
- **Pin Created for Label:** Specifies whether pins have been created for the labels.

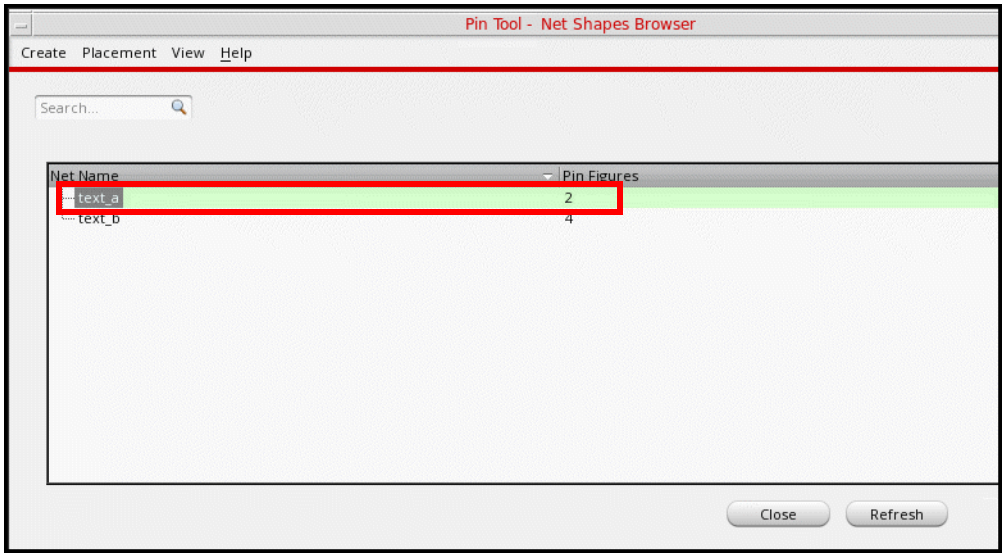
Additional Options

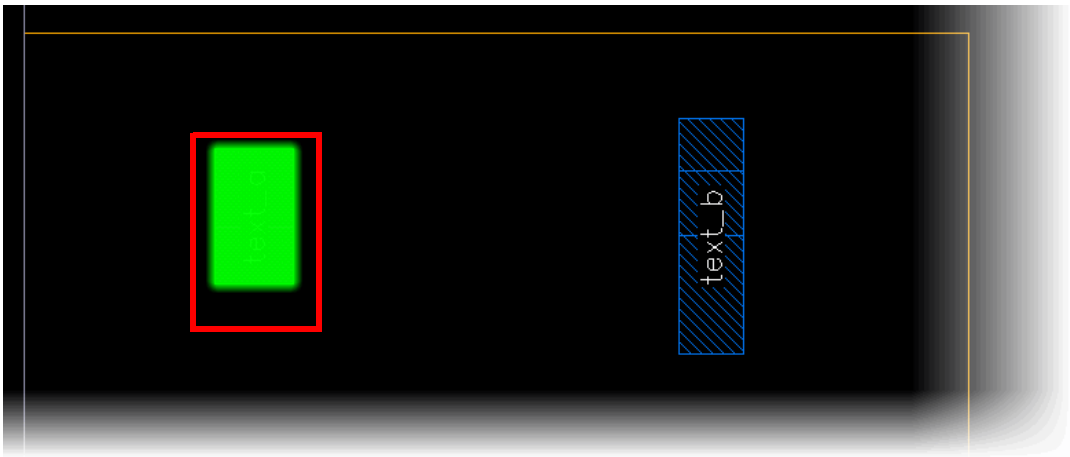
- Click *Refresh* to replicate changes in the Label Browser when label attributes have been changed outside the Pin Tool.
- Click *Apply* to replicate the changes you made to the Label Browser to the layout canvas.

© 2020 All Rights Reserved.

Net Shapes Browser

Choose *View – Net Shapes* to display the Net Shapes Browser. This browser displays information about the nets in the current cellview. The Pin Tool supports cross-selection of nets between the Net Shapes Browser and the layout canvas.





The Net Shapes Browser includes:

- **Net Name:** Lists the net names.
- **Pin Figures:** Specifies the number of pin figures that are connected to the net.

These values cannot be edited in the Net Shapes Browser. Click *Refresh* when net attributes have been changed outside the Pin Tool to propagate the changes in the Net Shapes Browser.

© 2020
All Rights Reserved.

Creating Pins

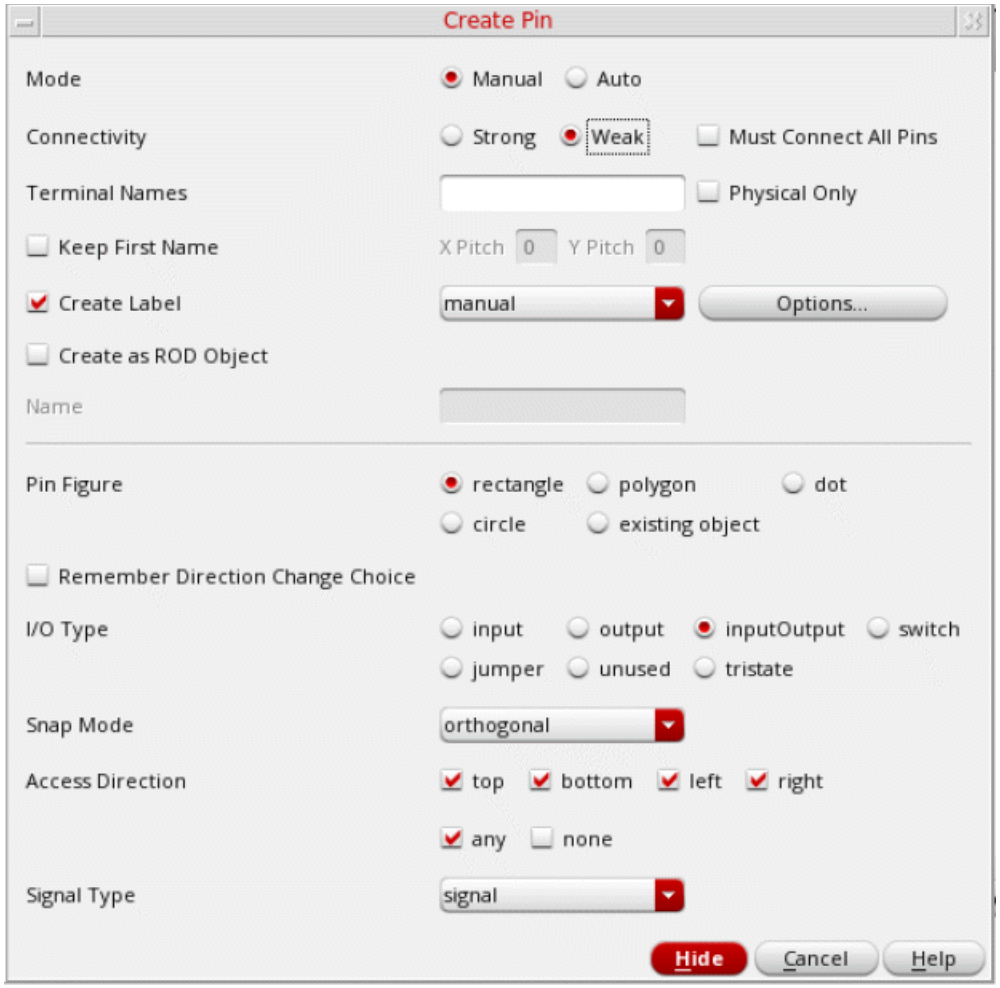
The Pin Tool lets you create new pins and customize their attributes. The *Create* menu lists the following commands to create pins from various sources:

- Creating Pins
- Creating Pins from Labels
- Creating Pins from Net Shapes
- Creating Pins Automatically

© 2020 All Rights Reserved.

Creating Pins

Choose *Create – Create Pin* to display the Create Pin form. Use this method to create pins from scratch when there are no labels or nets that can be used as source for creating pins.



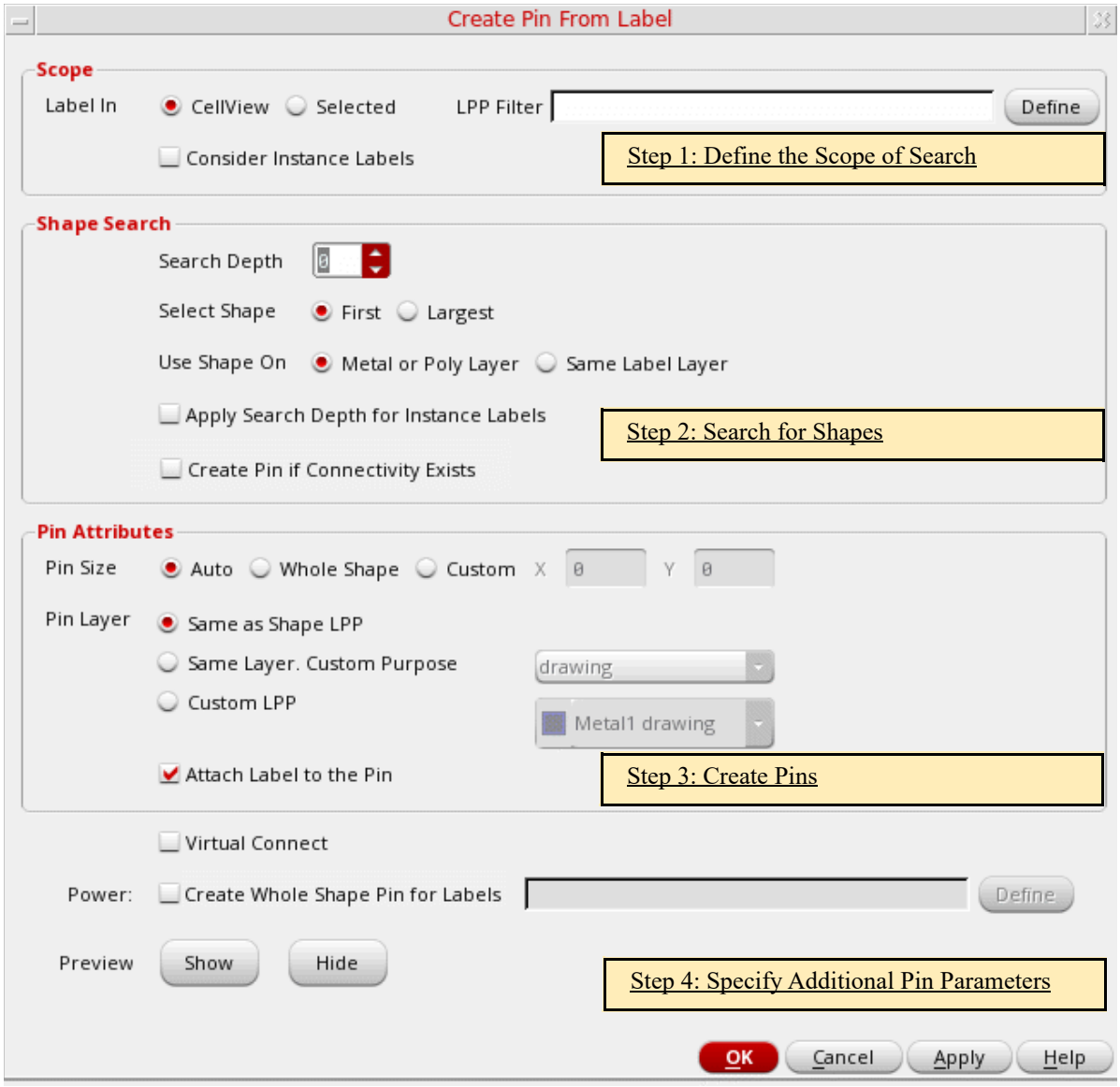
For more information, see Creating Pins.

Creating Pins from Labels

The Create Pin From Label form lets you create pins from labels and text displays in the layout cellview. The Pin Tool lets you search down the hierarchy for labels, but creates pins only at the top level. The tool searches for shapes and creates pins for labels with their origins overlapping these shapes. You can specify the pin dimensions and the layer on which the pins must be created. The generated pins are centered on the origin of the label or text display from which they are derived, and have terminal names matching the labels or text displays.

© 2020 All Rights Reserved.

To create pins from labels, choose *Create – Create Pins From Labels* to display the Create Pin From Label form. The form is divided into the following sections:



Step 1: Define the Scope of Search

1. Select either *Cellview* or *Selected* in the *Scope* section to specify the target labels.
   - *Cellview* - All top-level labels in the current cellview are considered as target labels.
   - *Selected* - The selected labels in the current cellview are considered as target labels.

© 2020 All Rights Reserved.

2. Specify the required *LPP Filter* to restrict the search only to specific layer-purpose pairs. All other LPPs are ignored. Example: `(metal2 drawing)`
3. You can choose to *Consider Instance Labels* in the search results.
   - If *Scope* is set to *Cellview*, all top-level and level-1 labels in the current cellview are considered as target labels.
   - If *Scope* is set to *Selected*, all top-level and level-1 labels in the selected instances are considered as target labels.

Step 2: Search for Shapes

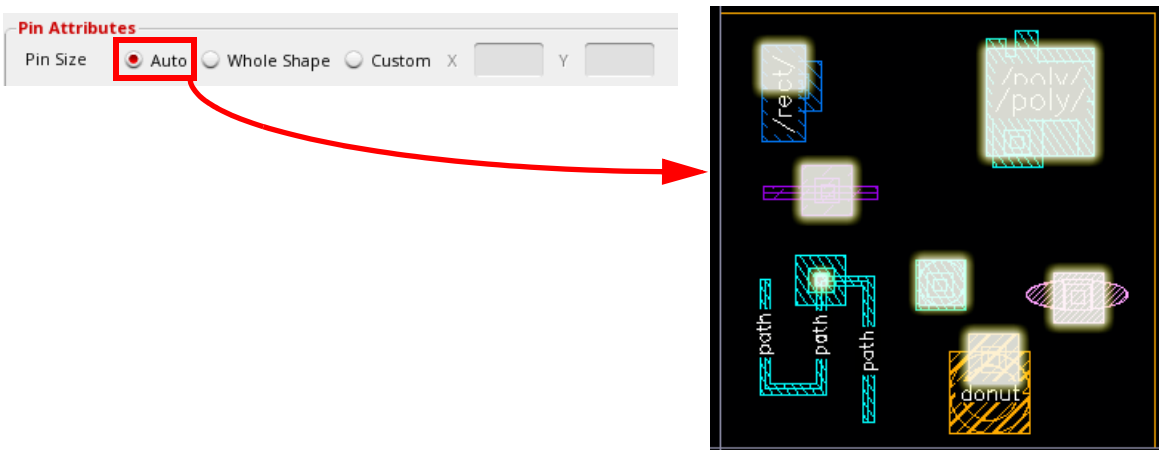
4. Specify the *Search Depth*, which indicates the level up to which shapes must be searched. For example, if *Search Depth* is set to 4, then levels 0 through 4 are searched for the required shapes. The default value is 0, which refers to the top-level (level 0).
5. Specify the shape that must be used for pin creation - *First* or *Largest*. *First* indicates that the first shape that matches the specified criteria must be used. Therefore, the search is stopped as soon as a suitable shape is found. When set to *Largest*, all matching shapes are retrieved; then the shape with the largest area is identified and used.
6. Specify the layer property based on which the search results must be filtered out - *Metal or Poly Layer* or *Same Label Layer*.
7. Select *Apply Search Depth for Instance Labels* to use the specified *Search Depth* for instance labels.
8. Select *Create Pin if Connectivity Exists* to create pins only for the labels that are associated with existing nets, terms, or pins in the current cellview. All other labels are ignored.

Step 3: Create Pins

8. Specify the *Pin Size* by choosing one of the following options:

© 2020 All Rights Reserved.

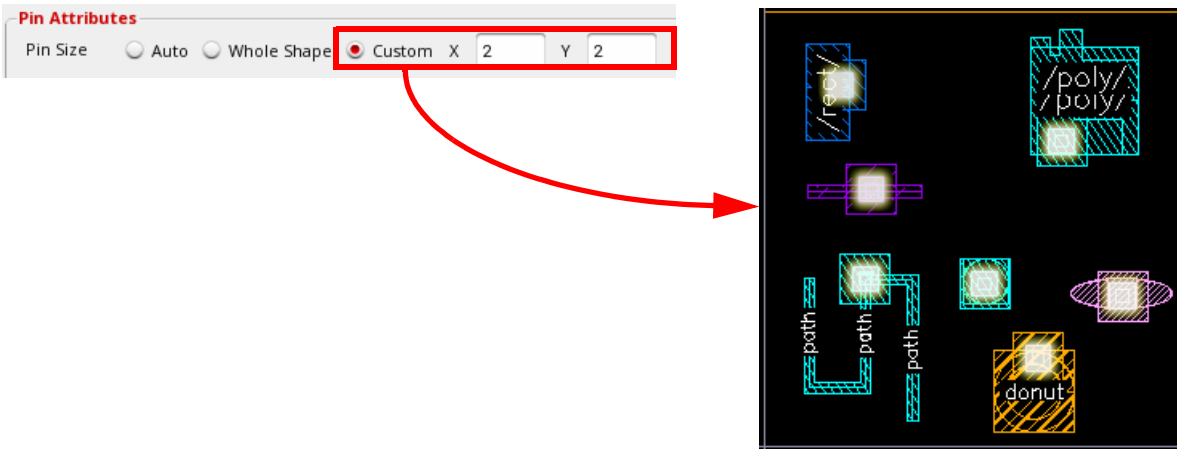
- **Auto:** Creates square pins inside the shape. The smaller dimension (width, height) is chosen as the pin side of the square pins. Pins are created along the edge of the PR boundary, so that they are accessible for defining connectivity.



- **Whole Shape:** Creates pins of the same shape as the labels.

© 2020
All Rights Reserved.

- **Custom (X and Y values):** Lets you specify the size of the pins by specifying the width and height values. The same size is applied to all output pins.



9. Specify the *Pin Layer* by choosing one of the following options:
   - **Same as Shape LPP:** Creates pins on the same layer as the overlapping shape.
   - **Same Layer Custom Purpose:** Uses the same layer as the overlapping shape, but lets you specify the purpose.
   - **Custom LPP:** Lets you specify both the layer and purpose on which pins must be created.
10. Select *Attach Label to the Pin* to attach the labels to their corresponding pins.

Step 4: Specify Additional Pin Parameters

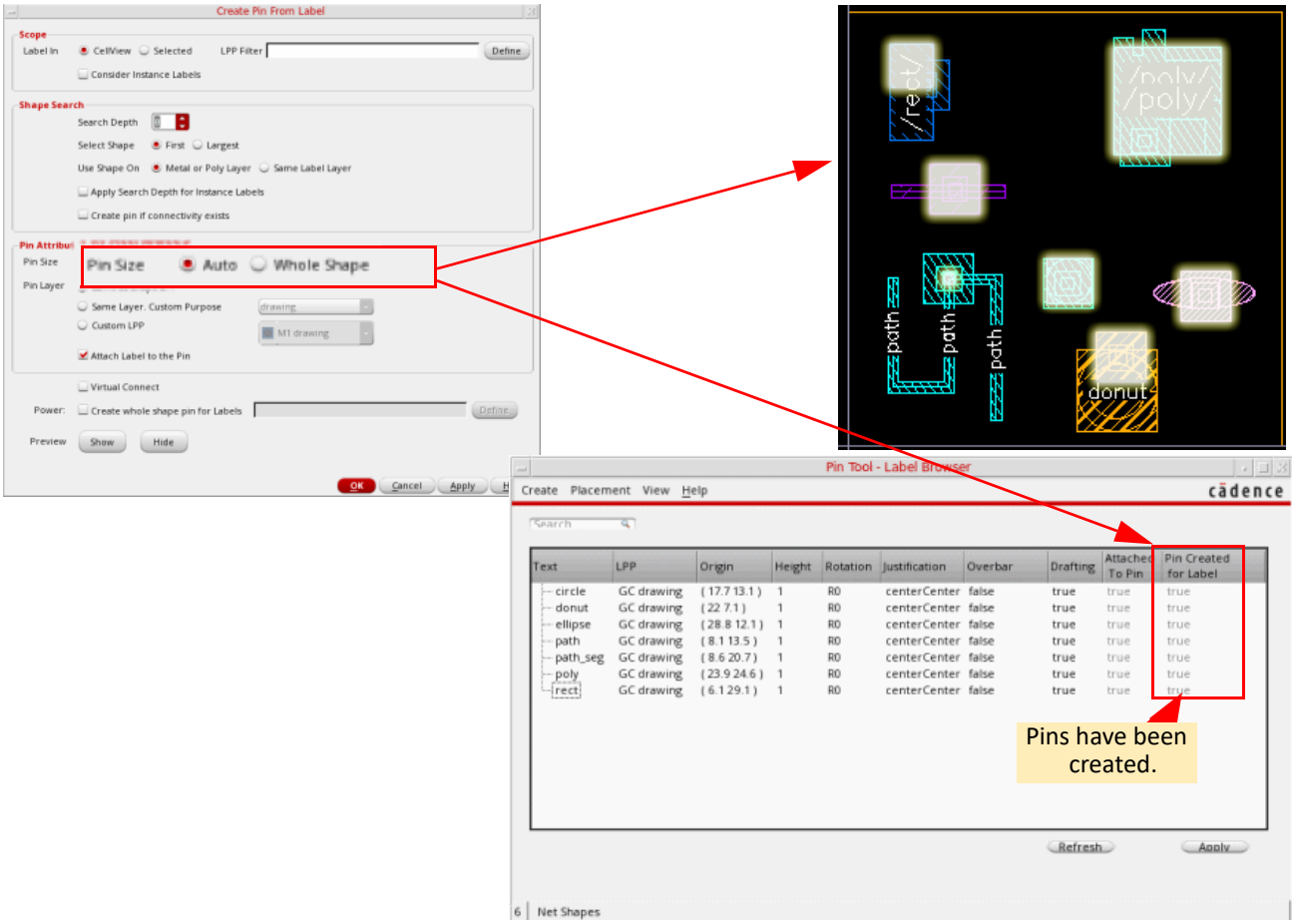
11. Virtual Connect

12. Select *Create Whole Shape Pin for Labels* in the *Power* section to create power pins of the same shape as their corresponding labels. Click *Define* to display a list of labels. Select the labels for which power pins must be created.
13. Click *Show* in the *Preview* section to see a preview of the pins that would be created.
14. Click *Hide* to close the preview.
15. Click *OK* to create the pins.

© 2020
All Rights Reserved.

Consider the following design that contains labels of different shapes. These labels are listed in the Label Browser.

The following images show the result of creating pins from the above labels:



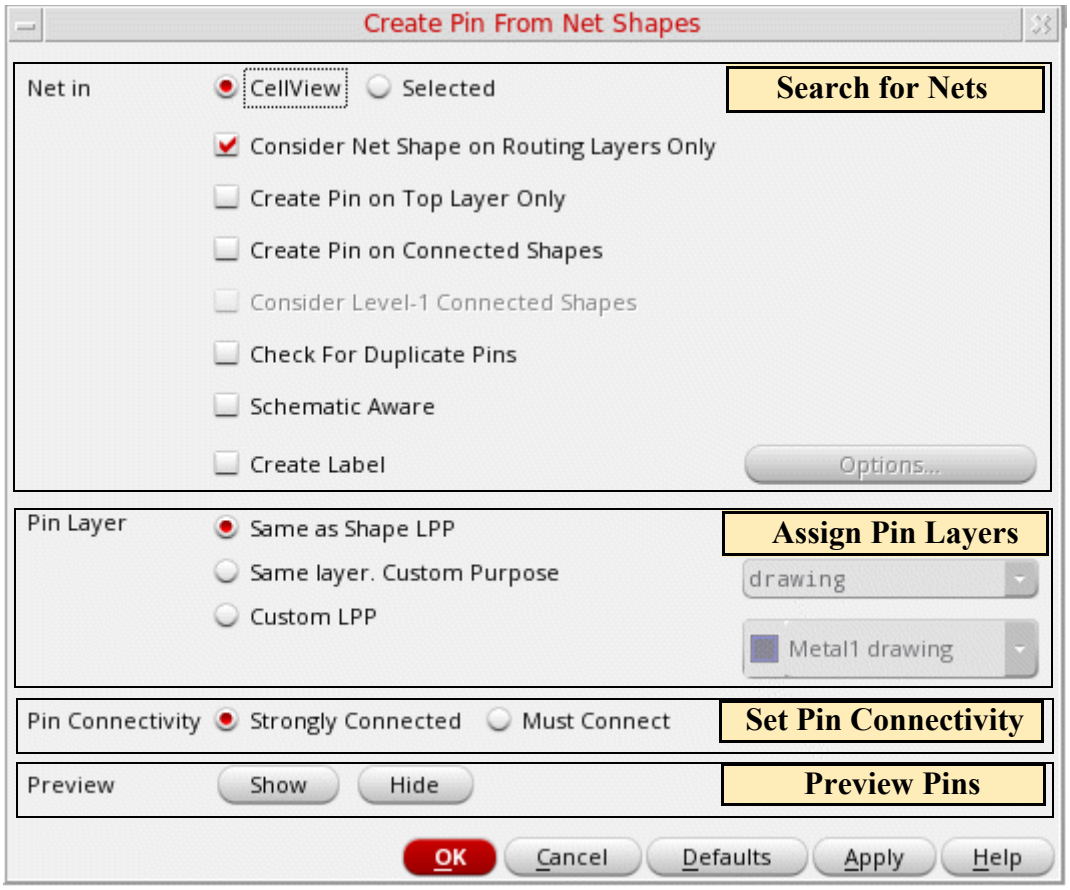
© 2020
All Rights Reserved.

Creating Pins from Net Shapes

Use the Create Pin From Net Shapes form to create pins from existing net shapes in the layout cellview. Only top-level pins can be created using this method. The Pin Tool searches for the target net shapes in the current cellview, creates new pins on top of these target net shapes, and assigns connectivity to these new pins.

Note: The Pin Tool does not modify the connectivity of existing nets or pins in the design.

Select *Create – Create Pin From Net Shapes* to display the Create Pin From Net Shapes form. The form is divided into the following sections:



To create pins from net shapes:

1. In the *Net in* section, select either *CellView* or *Selected* to specify whether net shapes must be searched in all or selected nets in the current cellview.
2. Select *Consider Net Shape on Routing Layers Only* to restrict the search to the net shapes that are located on routing layers.

© 2020 All Rights Reserved.

3. Select *Create Pin on Top Layer Only* to create pins only on the top layer shape. This option is useful when multiple shapes of the same net have different layers, and you want to create pins only on the top layer shapes. For example, if there are two net shapes `A` on metal layers `M2` and `M4`, then pins are created only on the highest metal layer `M4`.
4. Select *Create Pin on Connected Shapes* to restrict the search to those net shapes that are connected to top-level nets.
5. Select *Consider Level-1 Connected Shapes* to include net shapes that are connected to level-1 nets in the search. This option is available only when the design has level-1 shapes.
6. Select *Check for Duplicate Pins* to check for existing pins with the same names, and to not create duplicates.
7. Select *Schematic Aware* to ensure that pins are created only for those nets that have corresponding terminals in the schematic view. Pins are not created for internal nets.
8. Select *Create Label* to create labels over pins. Click *Options* to display the Set Pin Label Text Style form. Use the options in this form to define pin label settings.
9. In the *Pin Layer* section, select one of the following options to specify the LPP on which the pins must be created:
   - *Same as Shape LPP*: Creates pins on the layers on which the net shapes are located.
   - *Same Layer, Custom Purpose*: Uses the same layer as the net shape, but lets you specify the purpose.
   - *Custom LPP*: Lets you specify both the layer and purpose on which pins must be created.
10. In the *Pin Connectivity* section, set the routing preferences to one of the following:
    - *Strongly Connected* (default): Specifies that the router must connect the pins internally, within the device.

      Note: Consider pins `Term:Pin:Fig` and `Term:Pin:Fig1`. Here, only the pin fig name is different. The term name and pin name are the same. Such pins are called strongly connected pins.
    - *Must Connect*: Applies the LEF58_MUSTJOINALLPORTS property to all auto-created pin terminals, which implies that the router must connect to all pins of the selected terminal.
11. In the *Preview* section, click *Show* to see a preview of the pins. This is a visual depiction of how the pins would be created; actual pins are not created yet.

© 2020 All Rights Reserved.

12. Click *Hide* to close the preview.

13. Click *OK* to create the pins.

© 2020 All Rights Reserved.

Creating Pins Automatically

The Auto-Create Pins form provides options to search for net shapes in a design and create boundary and buried pins automatically on these net shapes.

© 2020
All Rights Reserved.

Note: By default, pins are created as groups. Set autoPinCreateFigGroup to `t` to create figGoups instead.

To create pins automatically:

1. Select *Create – Auto-Create Pins* to display the Auto-Create Pins form.
2. Set search options for net shapes as follows:
   - Specify the location in which the net shapes must be searched.
     - *CellView*: Searches for net shapes in all nets in the current cellview.
     - *Selected*: Searches for net shapes only in the selected nets in the current cellview. You can select the required nets in the Navigator assistant, *Net Shapes Browser* in the Pin Tool, or directly in the design canvas.
   - Select *Consider Net Shape on Routing Layers Only* to restrict the search for net shapes to the routing layers.
   - Select *Consider Net Shape on Selected Purpose* and select a purpose.
3. Set generic pin generation options as follows:
   - *Schematic Aware*: Specifies whether the tool must search for corresponding shapes in the schematic view before creating pins. For pins that do not have a schematic counterpart, a warning message is displayed and pins are not created.

     Note: Pins are not created for the top-level or level-1 shapes that are outside the PR boundary. A warning message with a list of pin names is displayed in the CIW.

     Note: A maximum of a 100 pin names can be listed in the warning message. If the list exceeds this limit, the pin names are excluded and only the warning message is displayed in the CIW.
   - *Delete Pins*: Deletes the existing pins in the layout canvas before generating new ones to avoid creating pins over existing pins. You can refine this by selecting one of the following settings:
     - *Auto Created* (recommended): Deletes only those pins and associated metal shapes that were generated automatically by using the Auto-Create Pins form.
     - *All*: Deletes all pins in the current cellview.

     Environment Variables: deletePinCheckbox, deletePinSelection
   - *Check for Duplicate Pins*: Checks for the presence of any similar pins in the design. If this option is selected and a similar pin is found, then a duplicate pin is not created.

© 2020 All Rights Reserved.

4. Set either one or both of the following *Signal Pin Generation* options:

- *Boundary Shape Pin*: Considers shapes that are abutted to or are straddling the PR boundary. Therefore, pins are created for the shapes that are touching or overlapping the PR boundary. For more information about straddling shapes, see Straddling Shapes.

  The following settings specify the number of signal pins to be generated when multiple boundary shapes belong to the same net:

  - *Single*: Creates a single boundary pin on the highest metal layer. For example, if there are five boundary shapes on metal layers 0 to 4, then a single pin is created on metal 4, which is the highest metal layer.

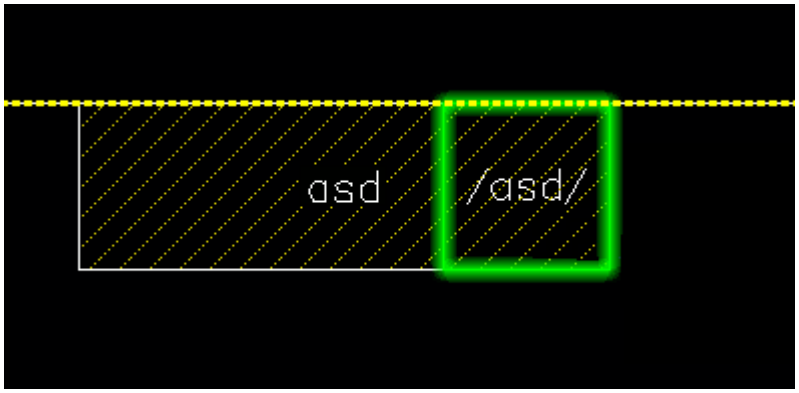
    When the autoPinPathSpineAware environment variable is set to `t`, a highest layer pin island, which consists of overlapping boundary pins belonging to the same layer and net, is promoted to the highest metal layer. A corresponding boundary pin is created in the highest metal layer. This feature supports creation of only shape pins, and not square pins.

  - *Multiple*: Provides the following layer selection options:

    - *Top Layer*: Creates multiple boundary pins on the highest metal layer.

    - *User Layer*: Lets you specify the layers on which boundary pins are to be created. By default, all layers are selected. Therefore, the same number of boundary pins as the number of boundary shapes are created in the same layers as the shapes.

  The following image shows a signal boundary pin generated by the Auto-Create Pins tool.



  When *Boundary Shape Pin* is selected, the *Boundary Signal Pin Size* options in the *Pin Attributes* section are enabled. These options control the pin shapes.

© 2020 All Rights Reserved.

- *Buried Whole Shape Pin*: Considers shapes that are completely inside the PR boundary.

  Note: Shapes that are completely outside the PR boundary are ignored.

  The following settings specify the number of signal pins to be generated when multiple buried shapes belong to the same net:

  - *Single*: Creates a single buried pin on the highest metal layer.

    When the autoPinPathSpineAware environment variable is set to `t`, a highest layer pin island, which consists of overlapping buried pins belonging to the same layer and net, is promoted to the highest metal layer. A corresponding buried pin is created in the highest metal layer. This feature supports creation of only shape pins, and not square pins.

  - *Multiple*: Provides the following layer selection options:

    - *Top Layer*: Creates multiple buried pins on the highest metal layer.

    - *User Layer*: Lets you specify the layers on which buried pins are to be created. By default, all layers are selected. Therefore, the same number of buried pins as the number of buried shapes are created in the same layers as the shapes.

  When both *Boundary Shape Pin* and *Buried Whole Shape Pin* are selected, boundary pins take precedence. For example, when both settings are set to *Single*, a single boundary pin is created on the highest metal layer.

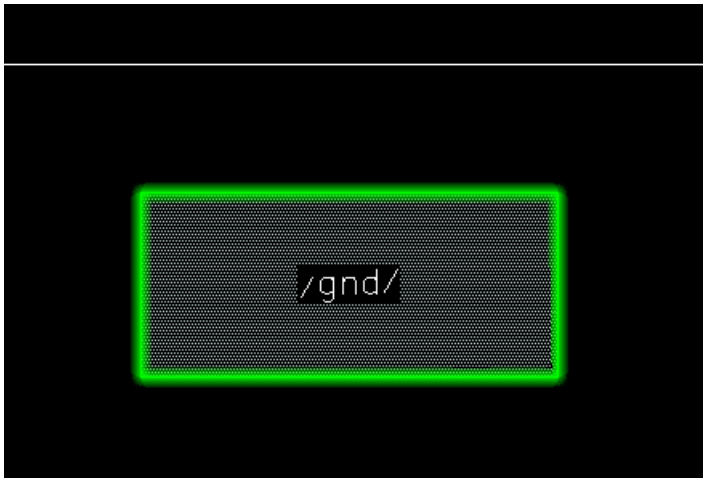
5. Set the required options in the *Power Pin Generation* section. Similar to signal shapes, you can choose to create pins for boundary shapes and buried shapes.

Power pins only support the multiple mode. Click *Define* to select the layers in which the pins must be generated. The selected layers are listed in the *Layer Selection* box.

Similar to signal pins, with both options selected, boundary pins take precedence.

© 2020 All Rights Reserved.

The following image shows a power buried (whole shape) pin generated by the Auto-Create Pins tool.



6. Set the following pin attributes:

Pin Attributes	Options	Description
Boundary Signal Pin Size (available when *Boundary Shape Pin* is selected)	*Auto*	Creates square pins. Rectangular and square path, pathseg, and polygonal shapes are recognized for pin creation. All other shapes are ignored, and a warning message is printed in the CIW.
	Whole Shape	Creates pins of the same shape as the corresponding net shapes. For example, square, rectangle, L-shaped, and path pins are supported.
Pin Layer	*Same as Shape LPP*	Creates pins on the same LPPs as the original net shapes.
	Same Layer, Custom Purpose	Lets you choose a different purpose for the same layer. Pins are created in the specified LPP.
	Custom LPP	Lets you choose a different LPP on which pins must be created.

© 2020 All Rights Reserved.

Pin Connectivity	*Strongly Connected*	Specifies that the router must connect the pins internally, within the device.
	Must join	Specifies that the router must connect the pins externally, at a higher level.
	Must Connect	Applies the LEF58_MUSTJOINALLPORTS property to all auto-created pin terminals, which implies that the router must connect to all pins of the selected terminal. Environment Variable: pinConnectivityMode
Create Text Label for Pins		Creates text labels for the pins that are generated.
(ICADVM20.1 Only) *Enable Coloring*		Uses the multiple patterning color engine to assign colors to the new pins based on their positions relative to the WSP tracks. This option is available only when the `coloringEngineEnabled` environment variable is set to `t`.
Create Metal Shapes Under Pin		Creates a metal shape under each auto-created level-1 (soft block) pin. The metal shapes are assigned to the same layers as the parent pin and to the `drawing` purpose. The metal shapes have the same dimensions as their parent pin shapes. The reason to create these metal shapes is to enable the Cadence® Assura™ RCX (Resistance, Capacitance, and Inductance Extraction) tool to detect any violations in the design.
	Path	Creates a pathSeg metal shape.
	Rectangle	Creates a rectangular metal shape.

7. (Optional) Click *Show* to see a preview of how the settings would be applied to the design. Click *Hide* to close the preview.
8. Click *Apply* to generate pins in the layout canvas. Click *OK* to create pins and close the form.

© 2020 All Rights Reserved.

Pins are automatically generated in the layout canvas. Their placement status is set to `Placed`.

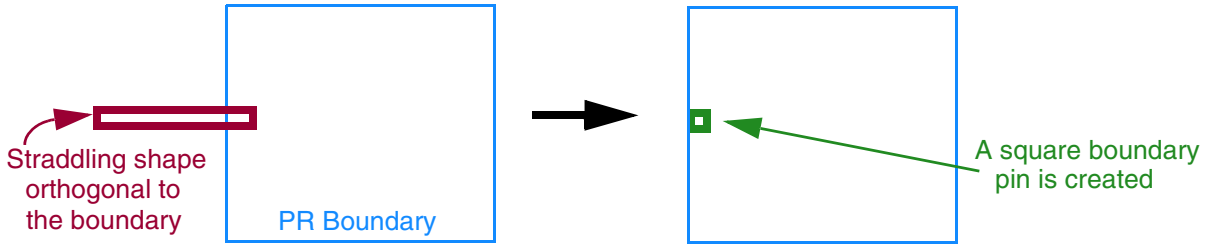
Straddling Shapes

Straddling shapes are the shapes that overlap the PR boundary. They are neither entirely inside nor outside the PR boundary. The Auto-Create Pins command recognizes such shapes and creates corresponding boundary pins.

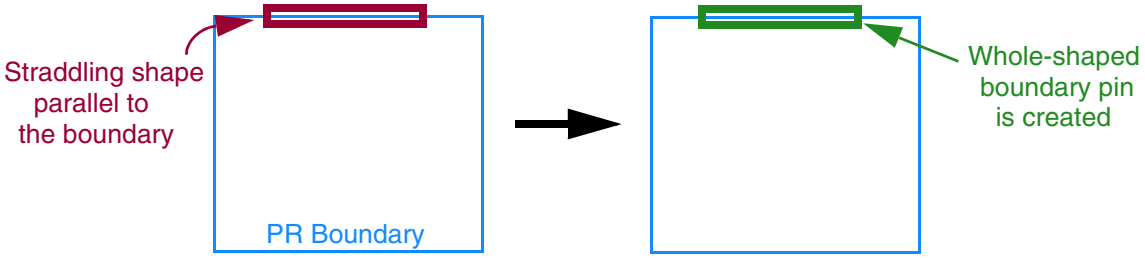
In case of power and ground straddling shapes, whole-shape boundary pins are created.

In case of signal shapes, the *Boundary Signal Pin Size* setting and the direction of the layer that contains the straddling shape determine the shape of the auto-created pin.

- When *Boundary Signal Pin Size* is set to *Auto* and the layer of the straddling shape layer is orthogonal to the PR boundary, a square-shaped boundary pin is created.

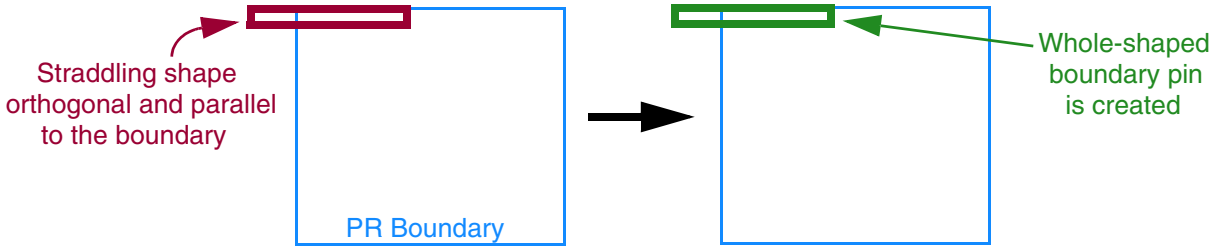


- When *Boundary Signal Pin Size* is set to *Auto* and the layer of the straddling shape layer is parallel to the PR boundary, a whole-shape boundary pin (of the same dimensions as the source shape) is created.

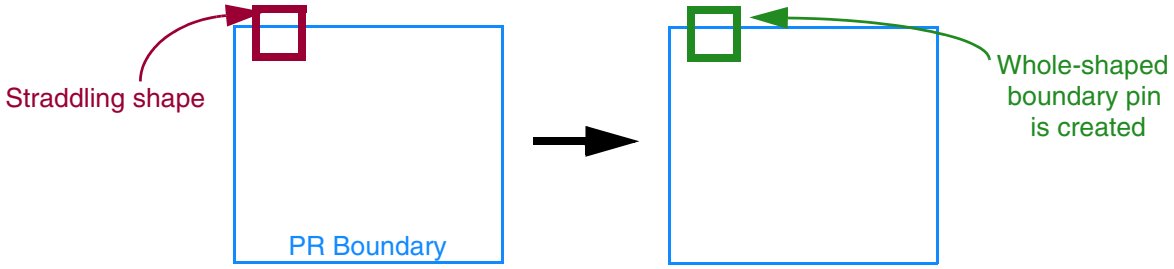


© 2020 All Rights Reserved.

- When *Boundary Signal Pin Size* is set to *Auto* and the layer of the straddling shape layer is both orthogonal and parallel to the PR boundary, a whole shape boundary pin is created.



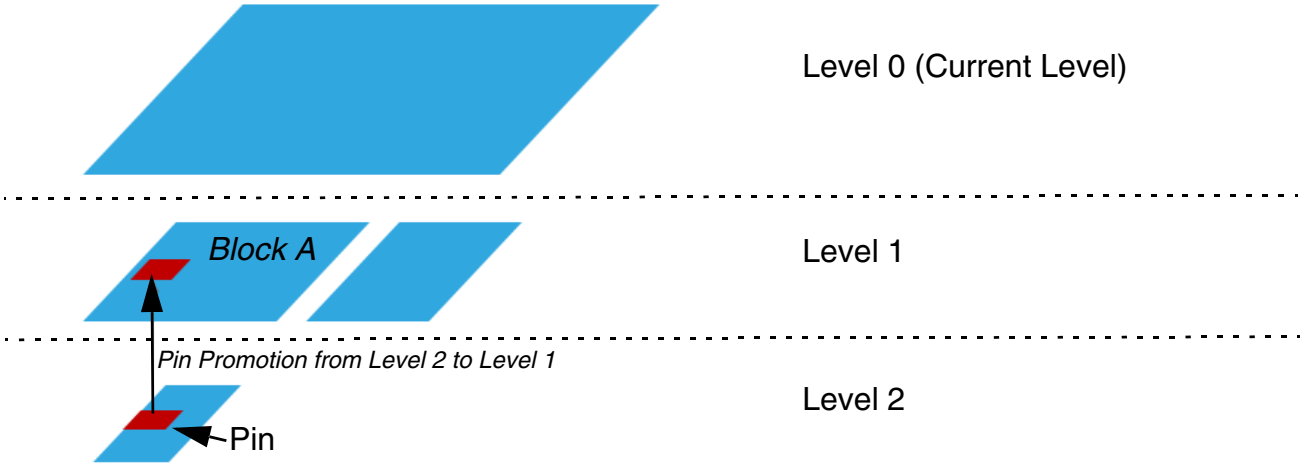
- When *Boundary Signal Pin Size* is set to *Whole Shape,* a whole shape boundary pin is created. This is irrespective of whether the straddling shape layer is orthogonal or parallel to the PR boundary.



© 2020
All Rights Reserved.

Promoting Pins

Use the Promote Pins form to promote pins from a lower level in a design hierarchy to a higher level, which can be the current level or lower than it. In the following figure, Level 0 is the current level. With Block A in the editable mode, a pin is promoted from Level 2 to Level 1.

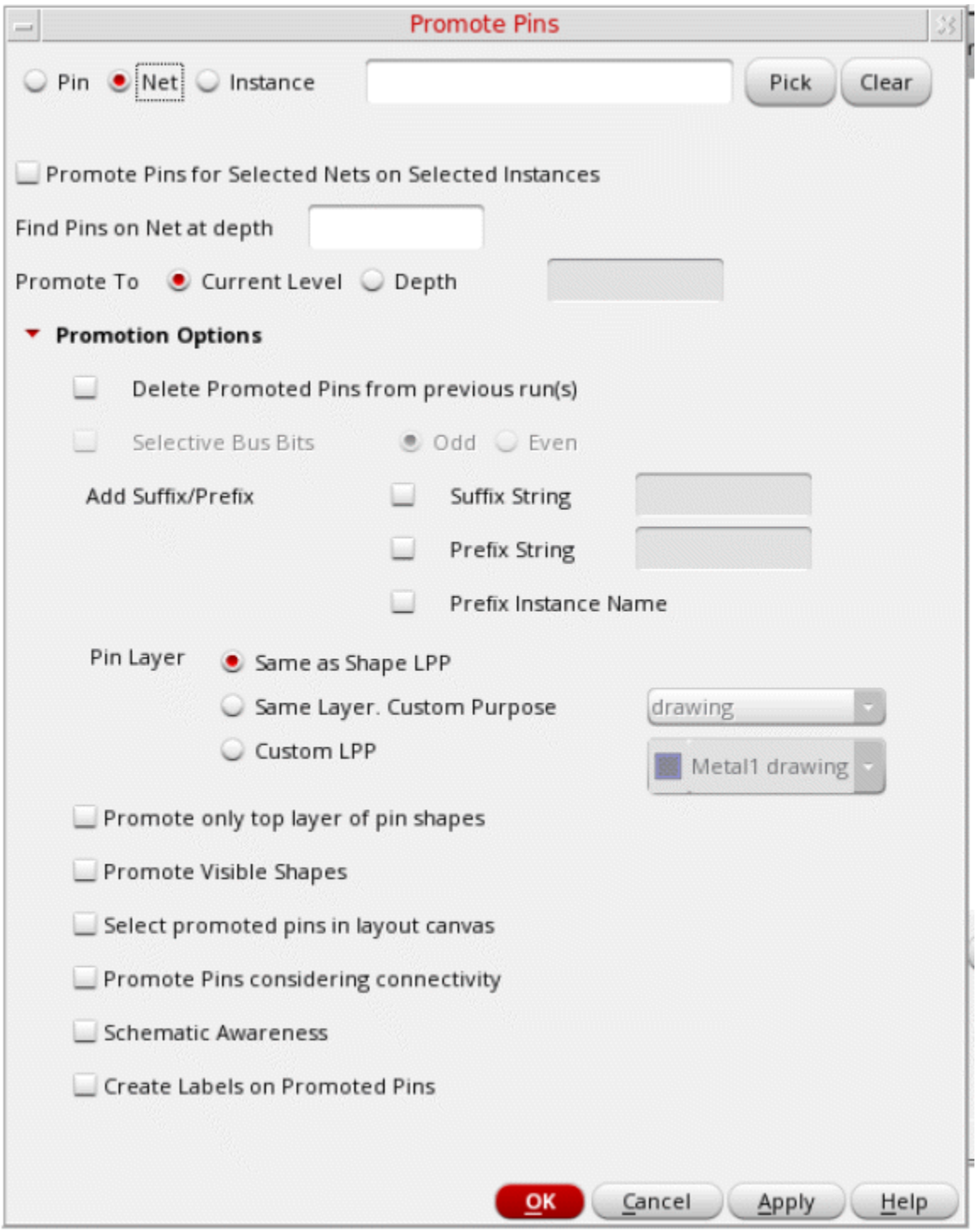


To promote pins in the Pin Tool:

1. Open the Pin Browser.

© 2020
All Rights Reserved.

2. Choose *Create – Promote Pins*. The Promote Pins form appears.



3. Select a mode in which you want to run promote pins.

© 2020 All Rights Reserved.

Pin: Lets you specify a space-separated list of hierarchical pin names for promotion. The *Pin* mode supports regular expressions. For example, `I0/in_*` indicates pins with names that start with `in_` and are in the `I0` instance, while `I0/.*` indicates all pins in the `I0` instance. You can promote bus pins by specifying the collapsed bus name. For example, `I0/in<0:3>` can be specified to promote bus pins `in<0>`, `in<1>`, `in<2>`, and `in<3>` from the `I0` instance.

Net: Lets you specify a space-separated list of top-level net names. Alternatively, you can select the required nets in the Navigator assistant and then click *Pick* to populate the net names in the adjacent text box. All the lower-level pins that are specified in *Find Pins on Net at depth* and connected to a top-level net are promoted. The *Net* mode does not support regular expressions.

Instance: Lets you specify a space-separated list of hierarchical instance names. Alternatively, you can select the required instances in the Navigator assistant and then click *Pick* to populate these names in the adjacent text box. All the pins inside the instance are promoted to a higher level.

At any point, click *Clear* to clear the list.

Environment Variables: promotePinNetChoice, promotePinNetList

Note: The *Promote Pins* command supports promotion of pins in mosaics.

4. *Promote Pins for Selected Nets on Selected Instances* is applicable only in the *Net* mode. With the required nets specified, select this option to filter the pins further by selecting one or more instances. Only those pins in the specified nets that belong to the selected instances are selected for promotion.

   Note: If *Promote Pins for Selected Nets on Selected Instances* is selected and there are no instances selected in the Navigator assistant or on the canvas, no pins are promoted.

   Environment Variable: promotePinOnNetForSelInsts

5. In the *Find Pins on Net at depth* field, specify the depth to which the tool must traverse to search for pins. For example, if the mode is set to *Net*, the net name is set to `gnd`, and *Find Pins on Net at depth* is set to `2`, the command traverses the design hierarchy to find pins on net `gnd` at level 2 to identify pins for promotion.

   Note: This option is available only in the *Net* mode.

   Environment Variables: promotePinOnNetPickLevel

6. Set *Promote To* to either the level specified in *Current Level* or to the depth specified in *Depth*. The specified depth must be at a lower level than the current level and must be in the editable mode so that the selected pins can be copied to it.

© 2020 All Rights Reserved.

Environment Variables: promoteToLevel, promoteToChoice

7. Click the expand button next to *Promotion Options* to display options to control the settings of the promoted pins. The settings in this section are optional. If not specified, pins are promoted using the default settings.

8. Select *Delete Promoted Pins from previous run(s)* to delete the pins that were promoted in the previous runs of the command.

   Environment Variable: promoteDeleteOldPins

9. Select *Selective Bus Bits* and specify whether *Odd* or *Even* bus pins must be promoted.

   Note: This option is available only in the *Pin* mode.

   When a bus is selected for promotion, by default, all its constituent bus pins are promoted. Use this option to restrict promotion to specific bus pins.

   Environment Variables: promoteSelectiveBusBits, promoteOddEvenChoice

10. Choose *Add Suffix/Prefix* and specify the required *Suffix String*, *Prefix String*, or both. The specified prefix and suffix are added to the promoted pin names.

    Note: In the *Net* mode, the suffix and prefix are added only to the promoted pin names, and not to the constituent net or term names.

    Environment Variables: promotePrefixChoice, promotePrefixString, promoteSuffixChoice, promoteSuffixString

11. Select *Prefix Instance Name* to add the pin source paths as prefix to the promoted pin names. For example, when you promote `pinA` from `I6/I2/pinA`, the new pin is named as `I6_I2_pinA`.

12. In the *Pin Layer* section, select one of the following options to specify the LPP to which the pins must be promoted:

    - *Same as shape LPP* (Default): Promotes pins to the LPP on which the source pins are located.
    - *Same layer, Custom Purpose*: Uses the same layer as the source pins, but lets you specify the purpose.
    - *Custom LPP*: Lets you specify both the layer and purpose to which pins must be promoted.

    Environment Variables: promoteLayerChoice, promoteCustomPurpose, promoteCustomLPP

© 2020 All Rights Reserved.

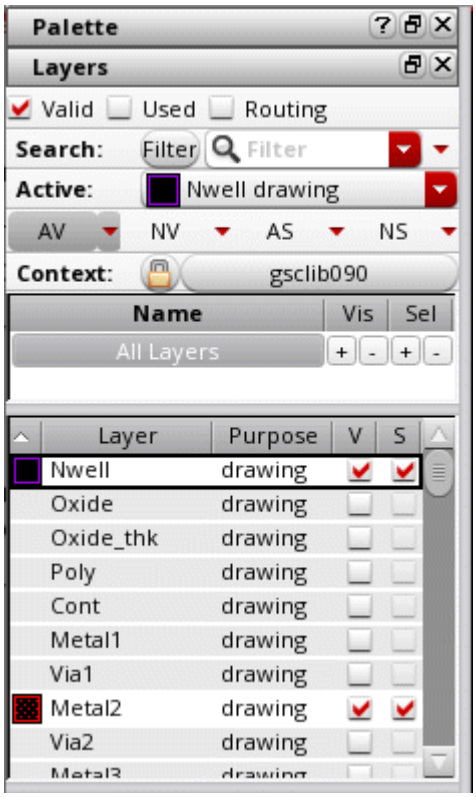
13. (Optional) Select *Promote only top layer of pin* to promote only those pin figures that are available on the top layer.

Environment Variable: promoteTopLayerFig

14. Select *Promote Visible Shapes* to promote pins from only those layers for which visibility has been turned on in the Palette assistant. For example, to promote only those pins that are present on layer `Metal2`:

a. Click *NV* in the Palette assistant to turn off the visibility of all layers.

b. Turn on the visibility of layer `Metal2`.



c. Run the Promote Pins command with *Promote Visible Shapes* selected.

Environment Variables: promoteVisibleLPPs

15. Select *Select promoted pins in layout canvas* to automatically select the promoted pins on the canvas. This option is available only when either *Promote To* is set to *Current Level* or *Find Pins on Net at depth* is set to `0`.

Environment Variable: promoteSelectInCanvas

© 2020 All Rights Reserved.

16. The *Promote Pins considering connectivity* option is available only in the *Pin* and *Instance* modes. Select this option to honor any existing connectivity of pins to top-level nets during the promotion of these pins. The connections are retained in the promoted pins.

    When *Promote Pins considering connectivity* is not selected, the promoted pins are connected to new nets that have the same net names as that of the lower level pin. It is recommended that you select this option to avoid any potential violations in Layout XL.

    Environment Variable: promotePinConnectivity

17. Select *Schematic Awareness* to promote only those pins that have a corresponding schematic counterpart at the top level.

    Environment Variable: promoteSchematicAware

18. Select *Create Labels on Promoted Pins* to create pin labels for promoted pins.

    Environment Variable: promoteCreateLabelsOnPins

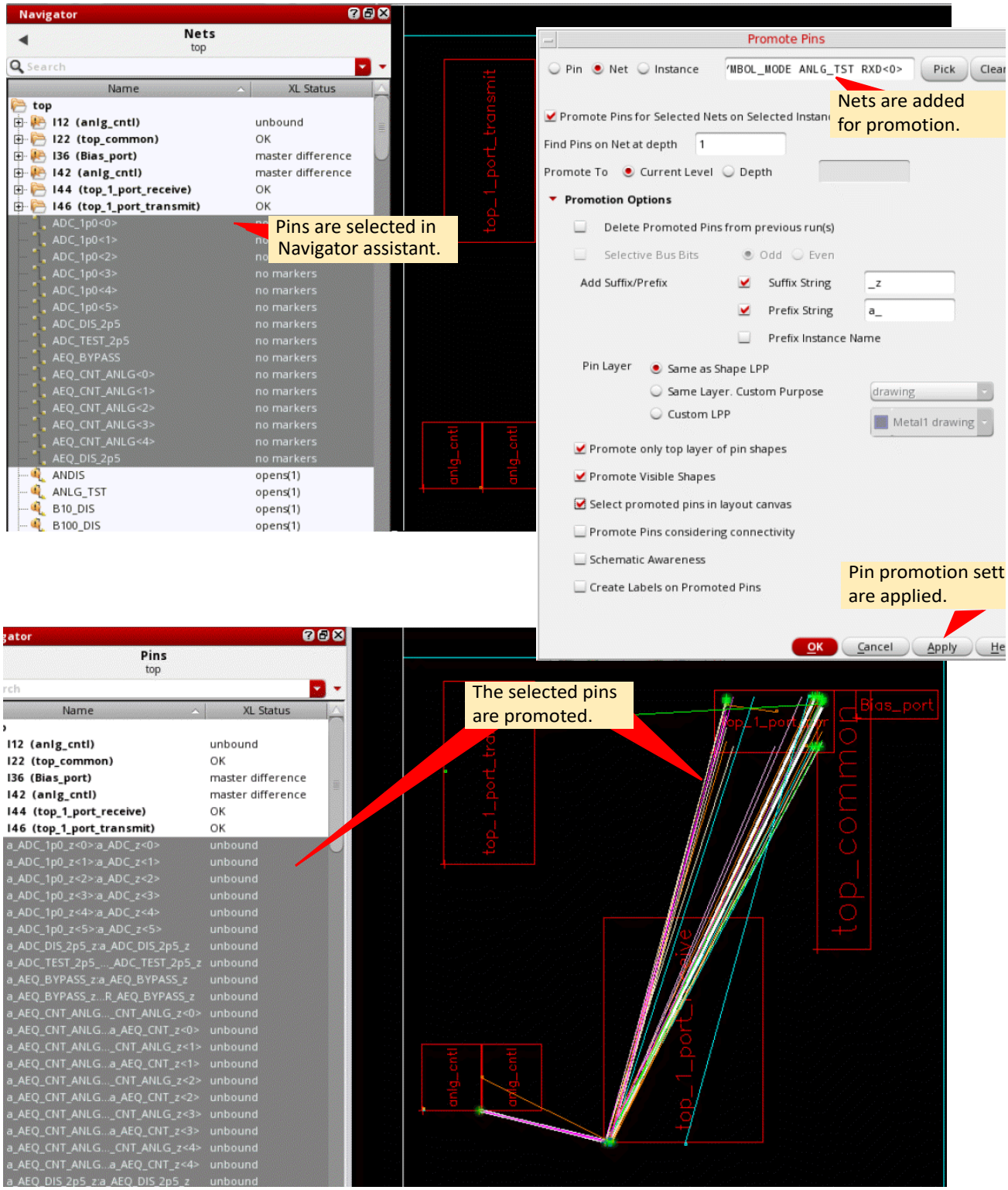
19. Click *Apply*.

Note: If any pin in the lower level has the must-connect-all-pins property set, the Promote Pins command copies the property along with the pin from the lower level to the current level. For more information about the must-connect-all-pins property, see Working With Pin Connectivity Models.

Note: When there are multiple instances of the same master in a design, a prefix of the instance name is added to the pin name. For example, consider instances `I1`, `I2`, and `I3` that share the same master. When `Pin A` from the master is promoted, the instances are renamed as `I1_A`, `I2_A`, and `I3_A`.

© 2020 All Rights Reserved.

The following images depict an example of pin promotion by selecting nets.



© 2020
All Rights Reserved.

Resizing Pins

You can create customized SKILL hooks to resize pins. The Resize Pins form lets you register and run such SKILL hook functions, also called the pin resize estimator SKILL functions, on all or selected pins in the cellview.

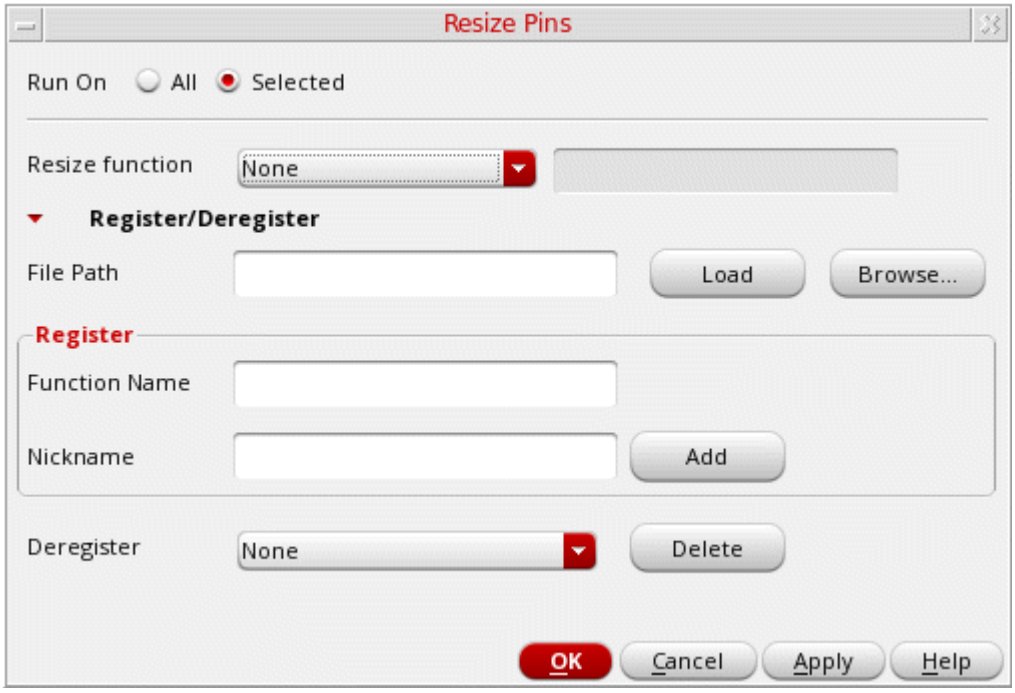
Here is a sample SKILL hook procedure to resize pins:

```
procedure( _pkfun1( mystr )
    let( ()
    info("In pkfunc mystr: %s \n" mystr)
))

procedure( pkResize1( topCv pinList)
    let( ( inBlockPinList curBlock curPinList curPin outPinSizeList outBlockList
outPinList width height )

    info("####Entering pkResize1 topCv:%L  pinList-%L \n" topCv->cellName pinList)
    outPinList = list()
    foreach(inBlockPinList pinList  ;( inst  pin1 pin2 ....)
        curBlock = car(inBlockPinList)
        curPinList = cdr(inBlockPinList)
        outBlockList = list()

        foreach(fig curPinList
            bBox = fig~>bBox
            width = caadr(bBox) - caar(bBox)
            height = cadadr(bBox) - cadar(bBox)
            outPinSizeList = list(fig 2*width 2*height)
            outBlockList=cons( outPinSizeList outBlockList)
            )
        outBlockList = cons(curBlock outBlockList)
        ;info("outBlockList: %L \n" outBlockList)
        outPinList = cons( outBlockList outPinList)
        ); end of foreach
    outPinList
))
procedure( pkResize2( topCv pinList)
    let( ( inBlockPinList curBlock curPinList curPin outPinSizeList outBlockList
outPinList width height )
```


© 2020 All Rights Reserved.

```
    info("####Entering pkResize2 topCv:%L  pinList-%L \n" topCv->cellName pinList)
    outPinList = list()
    foreach(inBlockPinList pinList  ;( inst  pin1 pin2 ....)
        curBlock = car(inBlockPinList)
        curPinList = cdr(inBlockPinList)
        outBlockList = list()
        foreach(fig curPinList
            bBox = fig~>bBox
            width = caadr(bBox) - caar(bBox)
            height = cadadr(bBox) - cadar(bBox)
            outPinSizeList = list(fig 1.5*width 1.5*height)
            outBlockList=cons( outPinSizeList outBlockList)
            )
        outBlockList = cons(curBlock outBlockList)
        ;info("outBlockList: %L \n" outBlockList)
        outPinList = cons( outBlockList outPinList)
        ); end of foreach
    outPinList
))
```

Use the Resize Pins form to register, run, or deregister the pin resize estimator SKILL functions. Choose *Place – Resize Pins* to display the Resize Pins form.



© 2020
All Rights Reserved.

Running a Pin Resize Estimator SKILL Function

1. Specify whether the procedure must be run on *All* or *Selected* pins.

   Note: In case of *Selected* pins, you must select the pins before launching the Resize Pins form.

2. Select the *Resize function* to be run. *Resize function* lists all registered pin resize functions.

   Note: *None* indicates that there are no registered functions.

3. Click *Apply* to run the selected function. The pins in the design view are resized.
4. Close the Resize Pins form.
5. Click *Refresh* in the Pin Tool to update the pin attributes in the Pin Browser.

Registering a Pin Resize Estimator SKILL Function

Use the options in the Register/Deregister section to perform the following steps:

1. Specify the *File Path* of the SKILL hook procedure file that contains the pin resize definitions. Alternatively, click *Browse* to select the required file and close the window.
2. Click *Load* to load the procedure.
3. Specify a *Function Name* and *Nickname*.
4. Click *Add* to register the function. The function is listed in the *Resize function* list, from where you can run it.
5. Click *Apply* to run the procedure.
6. Close the Resize Pins form.
7. Click *Refresh* in the Pin Tool to update the pin attributes in the Pin Browser.

Deregistering a Pin Resize Estimator SKILL Function

Use the options in the Register/DeRegister section to perform the following steps:

© 2020 All Rights Reserved.

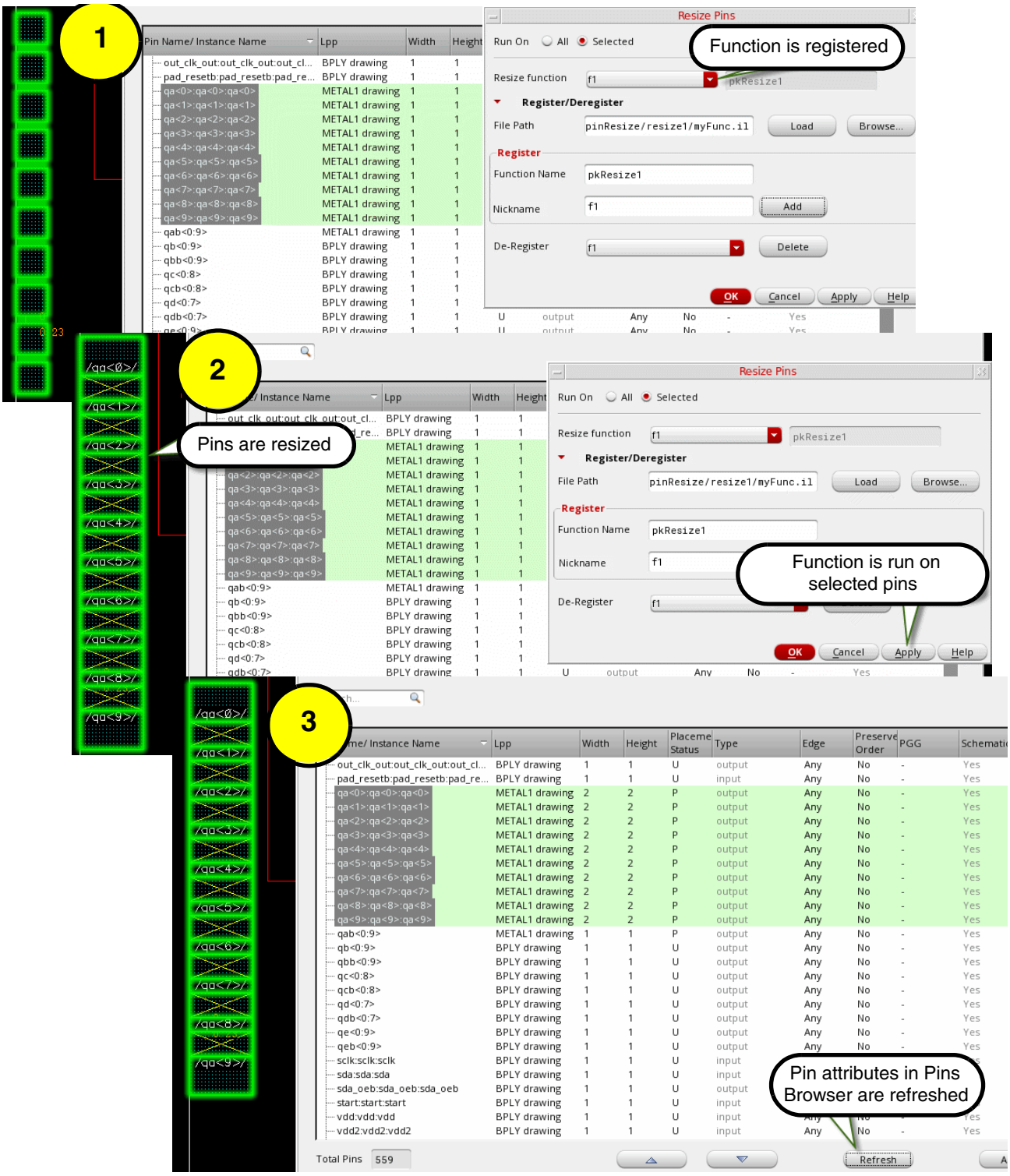
1. Select the pin resize estimator SKILL function to be deregistered from the Deregister drop-down list.
2. Click *Delete*.
3. Close the Resize Pins form.
4. Click *Refresh* in the Pin Tool to update the pin attributes in the Pin Browser.

Related SKILL Functions:

- vfpPtAddPinResizeEstimator
- vfpPtGetPinResizeEstimators
- vfpPtLoadPinResizeFile
- vfpPtRemovePinResizeEstimator
- vfpPtRunPinResizeEstimator

© 2020 All Rights Reserved.

The following images show how the above steps are performed:



© 2020
All Rights Reserved.

You can use the Resize Pins form to deregister procedures. To do this, select the corresponding function name from the *Deregister* drop-down list and click *Delete*.

© 2020 All Rights Reserved.

Making Pins Electrically Aware

(ICADVM20.1 EXL Only) In custom layout designs, while it is important for pins and net shapes to adhere to DRC rules, it is also important to ensure that they adhere to the electrical requirements of the designs. Pins should be sized keeping in mind the current carrying capacity of the nets.

The Pin Tool uses the capabilities of Layout Electrically Aware Design (EAD) to derive the electrical requirements of the design. Layout EAD, in turn, uses the electrical data produced by simulation in ADE Assembler and the parasitic information extracted from the layout view to analyze the design against predefined electromigration (EM) checks. For more information about running EM checks in Layout EAD, see Performing Electromigration Checks and Fixing Violations.

After deriving the electrical requirements for the design, the current specifications on a per-terminal basis and the permissible widths of pins and shapes are calculated.

The Electrically Aware Pin Sizing form lets you identify pins that do not meet electrical requirements and automatically fix these violations.

© 2020 All Rights Reserved.

The following list provides a quick snapshot of the overall flow:

Layout EAD

- Fetches and reads simulation results from the schematic for the current layout.
- Stores the current requirements for each net.
- Computes the required electrical width for each pin on each net.

Electrically Aware Pin Sizing Form

- Parses the layout to identify the pins and shapes that do not meet the required electrical width.
- Highlights the violating pins in the layout.
- Shows error markers in the Annotation Browser.
- Shows the present dimensions and the desired electrical dimensions for the violating pins on the layout canvas.
- Provides a preview of the potential fixes in the layout canvas.
- Automatically fixes all the violations.

Important

This feature checks out the EXL base license and 12 GXL tokens.

The Electrically Aware Pin Sizing form is divided into two sections to do the following tasks:

- Finding and Viewing Under-Sized Pins
- Resizing Under-Sized Pins

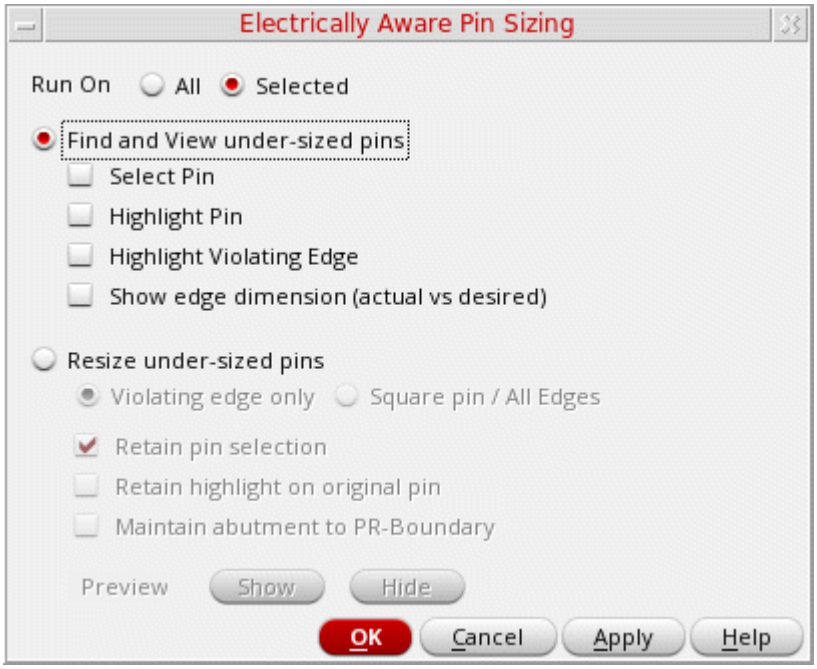
Finding and Viewing Under-Sized Pins

To find and view violating pins:

1. Choose *Place – Electrically Aware Pin Sizing*.

© 2020 All Rights Reserved.

The Electrically Aware Pin Sizing form appears.



2. Select one of the following *Run On* options to specify the pins on which the command is to be run:

   - *All*: Runs the command on all top-level and level-1 pins in the design.
   - *Selected*: Runs the command on the selected pins. All selection methods are supported, for example, selection on canvas, in the Navigator assistant, and in Pin Tool.

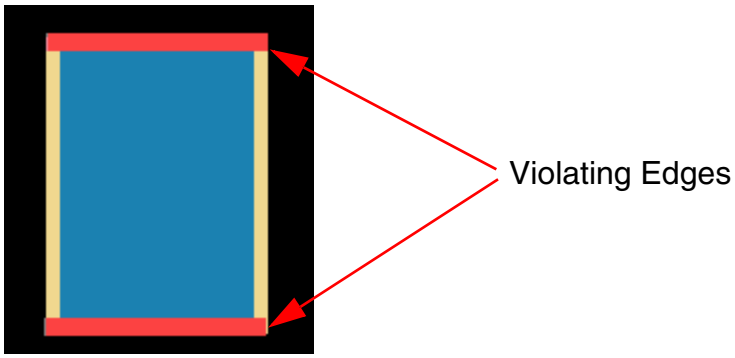
3. Select *Find and View under-sized pins*.

4. Select the required options for violation detection:

   - *Select Pin*: Selects all the detected under-sized pins. You can perform further actions on the selected pins, for example, you can move or resize them. Any prior selections are removed.
   - *Highlight Pin*: Highlights the under-sized pins for better visibility. Any prior highlights are removed.
   - *Highlight Violating Edge*: Highlights only the violating edge of under-sized pins.

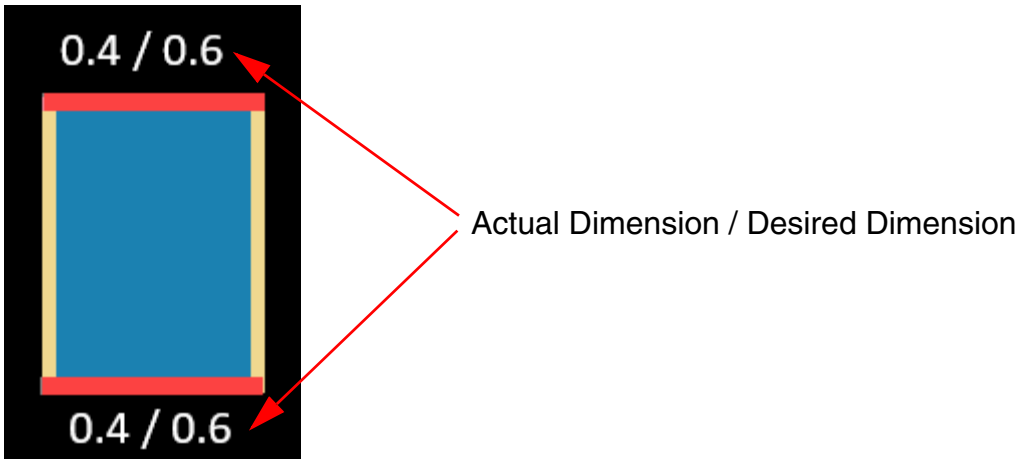
     In the following example, the top and bottom edges being shorter, are the violating edges. These edges are highlighted in red.

© 2020 All Rights Reserved.



- *Show edge dimension (actual vs desired)*: Displays the current edge dimension and the desired electrically-aware dimension of the violating edge.

  In the following example, the violating edge dimensions are displayed. `0.4` is the current pin width, whereas `0.6` is the desired pin width.



5. Click *OK*.

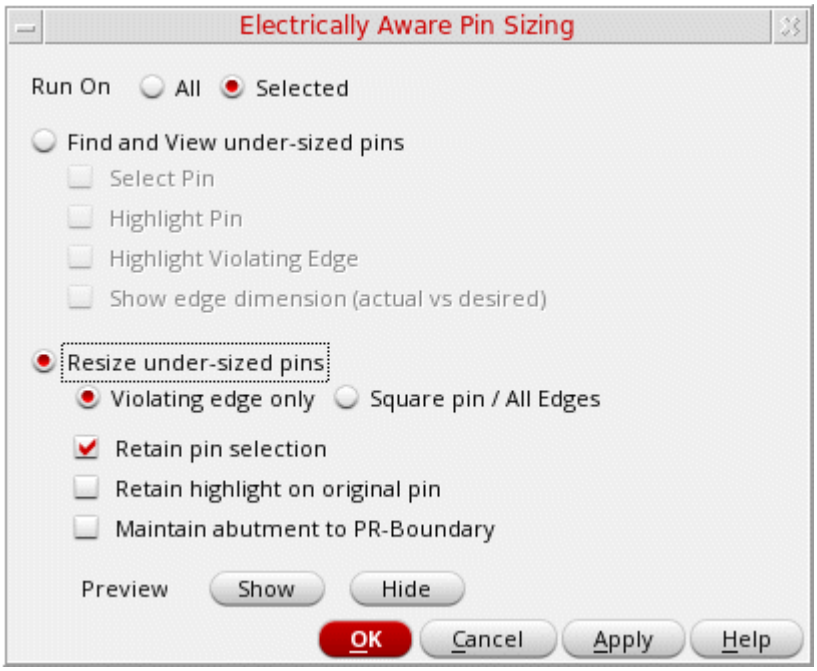
All violating pins are listed in the Annotation Browser.

After viewing the violating pins, you can use the options in the *Resize under-sized pins* section of the Electrically Aware Pin Sizing form to re-size these pins and fix the violations. For more information, see Resizing Under-Sized Pins.

© 2020 All Rights Reserved.

Resizing Under-Sized Pins

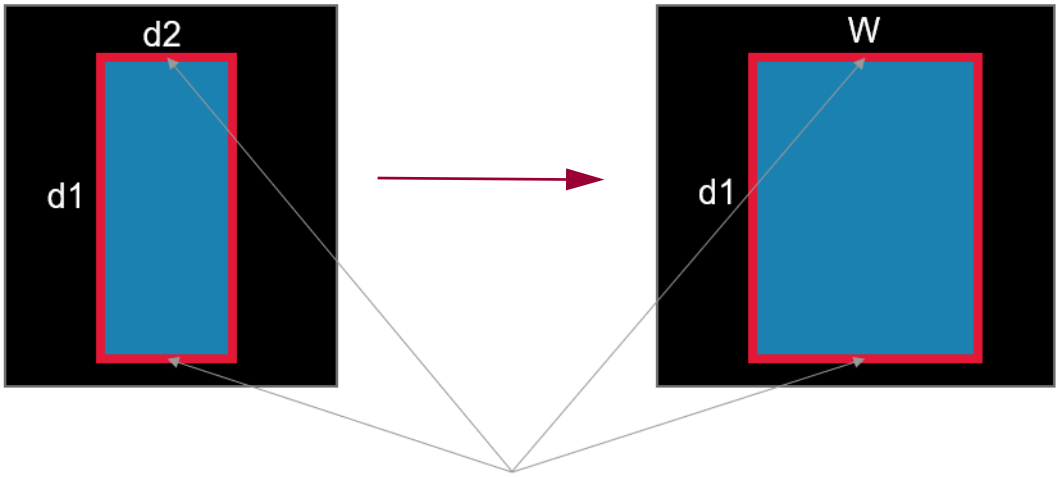
Use the options in the Electrically Aware Pin Sizing form to resize pins that do not meet the electrical requirements of the design. You can either first find and view the violating pins or directly resize them.



To resize violating pins:

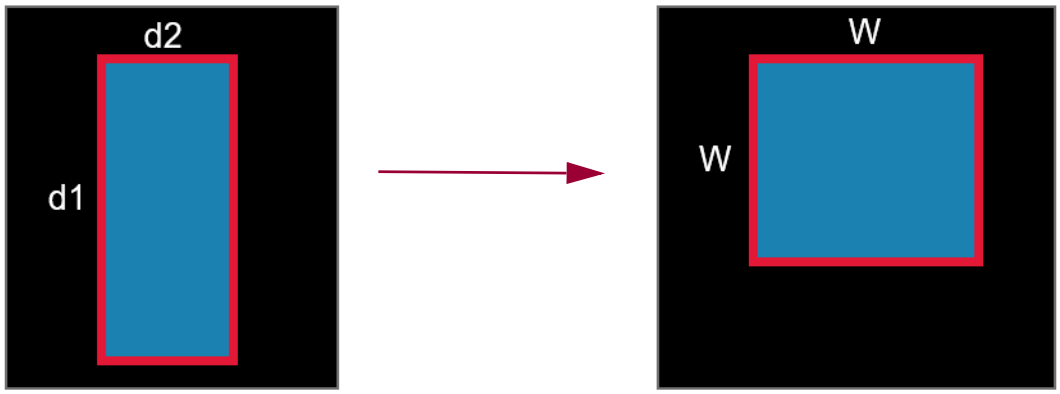
1. Select one of the following *Run On* options to specify the pins on which the command has to run:
   - *All*: Runs the command on all top-level and level-1 pins in the design.
   - *Selected*: Runs the command on the selected pins. All selection methods are supported, for example, selection on canvas, in the Navigator assistant, and in Pin Tool.
2. Select *Resize under-sized pins*.
3. Select one of the following options:
   - *Violating edge only*: Resizes only the violating pin edges to the desired electrically-aware width. All other edges remain unaffected.

© 2020 All Rights Reserved.



Only the violating (under-sized) edge d2 is resized
The longer edge d1 remains unaffected

- *Square pin / All Edges*: Resizes all edges to create square pins. For example, all edges of a rectangular pin are resized to the desired electrically-aware width (W) to create a square pin.



All edges are resized to the recommended
electrically-aware pin width (W)

4. Select *Retain pin selection* to retain the selection of pins after they are resized.
5. Select *Retain highlight on original pin* to retain the highlights on the pins that were selected for resize.
6. Select *Maintain abutment to PR-Boundary* to ensure that pins that are initially abutted to the PR boundary retain the abutment even after they are resized. Pins that are not initially abutted remain at the same position by maintaining the center of the pins at the same location.

© 2020 All Rights Reserved.

7. Select *Show* in the *Preview* section to see a preview of the pins after they are resized before updating the layout.
8. Verify that the changes are as per your requirements.
9. Click *Hide* to close the preview.
10. Click *OK*.

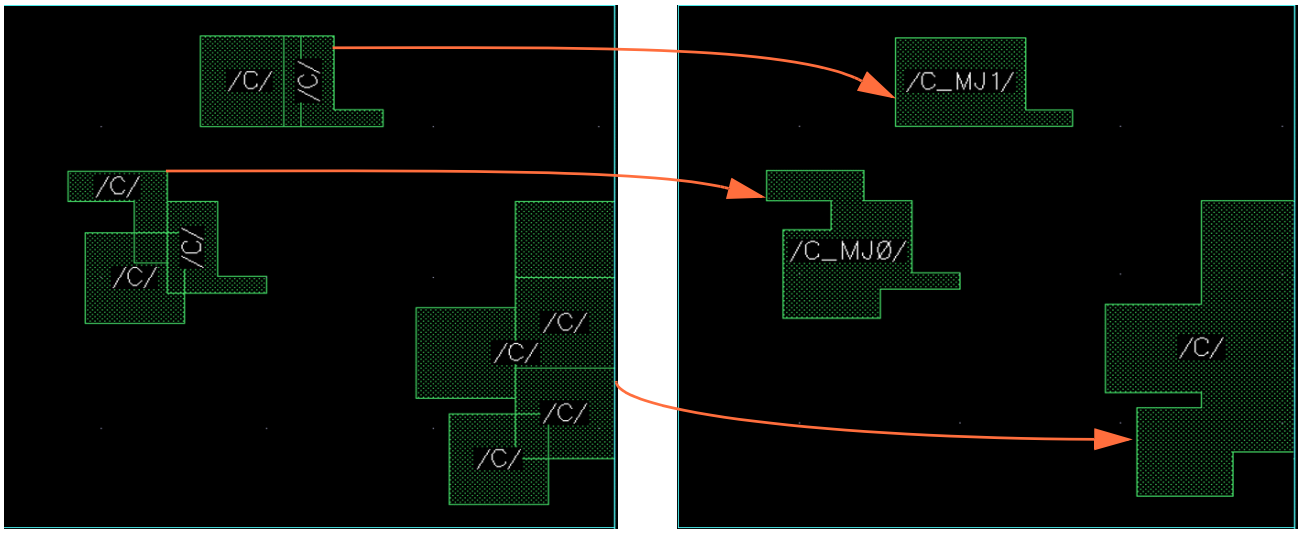
Pins in the design are resized as per your specifications.

© 2020 All Rights Reserved.

Merging Pins

You can merge two or more overlapping rectangular, rectilinear, or path pin shapes into a single rectilinear shape. Pin shapes are merged based on their connectivity. This implies that overlapping pin shapes that are on the same layer and are connected to the same net can be merged. Non-overlapping pin shapes that cannot be merged form islands. The must-connect-all-pins property, also referred to as the must-join property, is applied to these islands and to the merged shapes.

The following example shows how overlapping pins that belong to the same layer and are connected to the same net, net `C`, are merged.



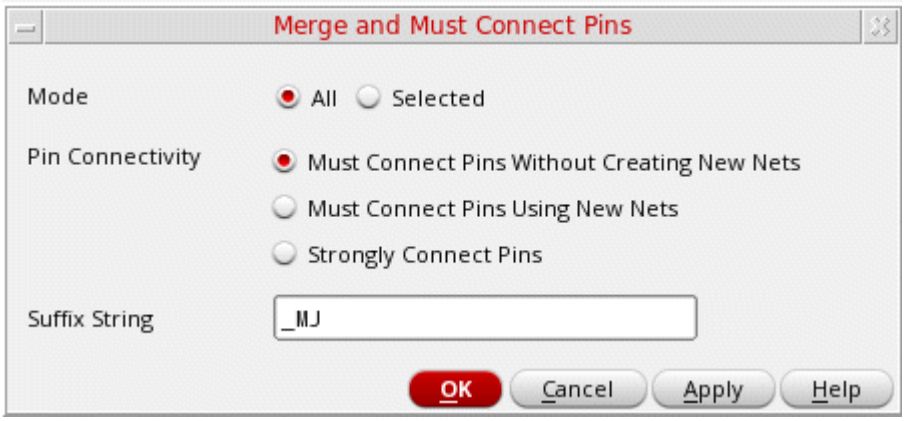
The merged pin shapes derive their names from the name of the connecting terminal. For example, when pin shapes `C:C:C`, `C:C:C1`, `C:C:C2`, and `C:C:C3` are merged, the resultant pin shape is named `C:C:C`. In the absence of a pin shape with the same name as the terminal, for example, `C:C:C`, the name of any pin shape, for example, `C:C:C1`, is considered.

You can either apply the `must connect all pins` setting to the existing terminals or create new nets and terminals for each island and then apply the `must connect all pins` setting. You can also specify a suffix string for the new terminals. The default value is `_MJ`.

To merge pins by using the Pin Tool:

© 2020 All Rights Reserved.

1. Select *Tools – Merge and Must Connect Pins*. The Merge and Must Connect Pins form appears.



2. Select a *Mode*:
   - *All* applies to all pin shapes.
   - *Selected* applies to the selected pin shapes.

   Related Environment Variable: mergePinsSelMode

3. Set *Pin Connectivity* to one of the following values:
   - *Must Connect Pins Without Creating New Nets* (default): The connectivity of the pin shapes is retained. The `must connect all pins` setting is applied to the existing terminals.
   - *Must Connect Pins Using New Nets*: New nets, terminals, and pins are created for each pin shape island. The new terminals are connected and are placed together in a `must connect` group.
   - *Strongly Connect Pins*: A single connection point on a highly conductive metal is used to provide the required connectivity.

   Related Environment Variable: mergePinsConnMode

4. Specify the *Suffix String* to be added to the new nets, terminals, and pins that have been created on the merged pin islands. By default, the suffix added is `_MJ`. This setting is applied only when *Must connect pins using new nets* is selected.

   Related Environment Variable: mergePinsSuffix

5. Click *OK*.

© 2020 All Rights Reserved.

Overlapping pins are merged and the *Must Connect Terms* setting is applied to the resultant non-overlapping pin shapes as specified.

© 2020 All Rights Reserved.

Planning and Optimizing Pins

You can access the Pin Planner and Pin Optimizer from the Pin Tool.

- Choose *Place – Pin Placement* to display the *Pin Planner* tab of the Pin Placement form. Use the options on this tab to assign or refine pin constraints, such as the side constraint (Left, Right, Top, Bottom), placement status (Fixed, Locked), and pin order.

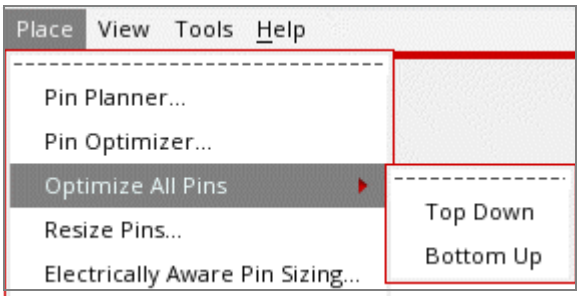
  You can then use the Pin Browser in the Pin Tool to refine the pin attributes, such as pin layer and size. For more information, see Pin Planning.

- Choose *Place – Pin Optimization* to display the *Pin Optimization* tab of the Pin Placement form. Run Pin Optimizer to position pins of blocks in a manner that helps obtain the shortest possible net length at a particular level in the design. For more information, see Pin Optimization.

- (ICADVM20.1 EXL Only) Choose *Place – Optimize All Pins* to optimize pins. You can run the command in the following modes:

  - *Top Down*: Runs pin optimizer on the top-level first, followed by the lower levels. Top-level pins are optimized with respect to level-1 pins, level-1 pins are optimized with respect to level-2 pins, and so on.

  - *Bottom Up*: Runs pin optimizer on the bottom-most level first, followed by the higher levels.



© 2020
All Rights Reserved.

© 2020 All Rights Reserved.

10

Setting Floorplan Options

This chapter covers the following topics:

- Floorplan Global Options
- Block Annotations

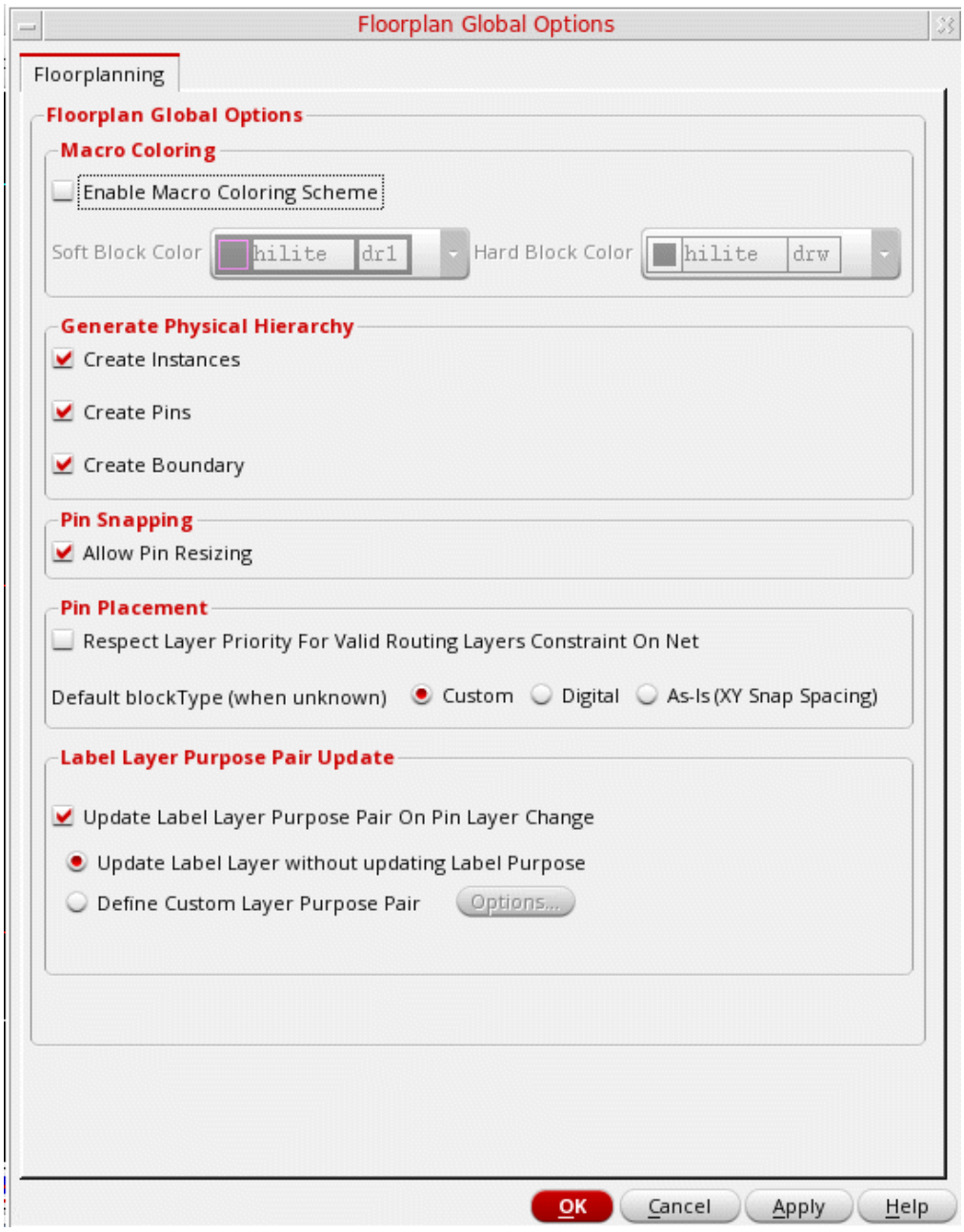
Important

(ICADVM20.1 Only) The Floorplan Global Options form is not available in Layout EXL. These settings are available in the Design Planning and Options form.

© 2020 All Rights Reserved.

Floorplan Global Options

The *Floorplan – Floorplan Global Options* menu item enables you to customize the layout design environment. Selecting this menu item displays the Floorplan Global Options form (Layout GXL Options form in IC6.1.8). You can use this form to set global options for either the current cellview or the current session.



© 2020 All Rights Reserved.

Use the options in this form to perform the following tasks:

- Color Coding Hard and Soft Blocks
- Controlling Re-Generation of Instances, Pins, and PR Boundary
- Governing Pin Resize During Pin Snapping
- Respecting Layer Priority for Valid Routing Layers
- Specifying the Default Block Type
- Updating Label Layer Purpose Pairs

Color Coding Hard and Soft Blocks

Use the following color coding options in the *Macro Coloring* section to assign different highlight colors for hard and soft blocks so that they are visually distinguishable in the layout canvas:

1. Select the *Enable macro coloring scheme* check box to enable color coding of hard and soft blocks.

   Corresponding environment variable: `colorOrNot`

2. Select a highlight color for soft blocks from the *Soft block color* drop-down list.

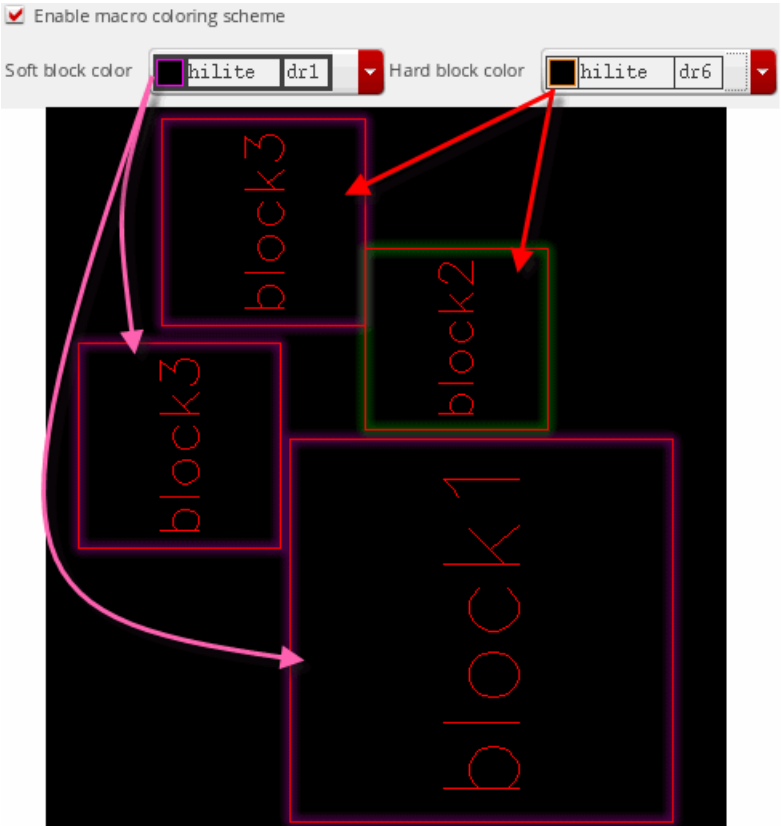
   Corresponding environment variable: `softBlockColor`

3. Select a highlight color for hard blocks from the *Hard block color* drop-down list.

   Corresponding environment variable: `hardBlockColor`

© 2020 All Rights Reserved.

4. Click *OK* or *Apply* to view the changes in the layout canvas.



The soft and hard blocks are highlighted in different colors, as shown above.

These visualization settings are not restricted to the current design. These settings are automatically applied to all layout cellviews that were open when you defined the color settings. To reset the settings, use either the options in the form or their corresponding environment variables.

Controlling Re-Generation of Instances, Pins, and PR Boundary

Use the *Create Instances*, *Create Pins*, and *Create Boundary* check boxes from the *Generate Physical Hierarchy* section to specify whether instances, pins, and the PR boundary need to be automatically re-created when the physical hierarchy is generated or the components and nets are updated.

These options provide a GUI alternative to using the `initCreateInstances`, `initCreatePins` and `initCreateBoundary` environment variables.

© 2020 All Rights Reserved.

To know more about the *Generate Physical Hierarchy* command, see Generating a Physical Hierarchy.

To know more about the *Update Components and Nets* command, see Updating Components and Nets.

Governing Pin Resize During Pin Snapping

To specify whether pins need to be resized during pin snapping, select the *Allow Pin Resizing* option from the *Pin Snapping* section. For more information about pin snapping, see Snapping Soft Block Pins to Grid.

Corresponding environment variable: `allowPinResizing`

Respecting Layer Priority for Valid Routing Layers

During pin optimization, pins are re-positioned and re-layered to obtain the shortest possible net length. The *Respect layer priority for valid routing layers constraint on net* option in the *Pin Placement* section specifies the layer priority to be followed while placing pins during pin optimization.

When this option is not selected (default state), the Pin Optimizer places pins on layers that are closest to the connected pin.

Example: Pin A is connected to top-level pin B. Valid layers from net PRO for pin A are (M1, M2, M5, M7). Pin B is located on layer M2.

Note: For information about net PROs, see Support for Net PROs in Virtuoso Floorplanner.

The Pin Optimizer first tries to place pin A on M2 to enable the router to connect the two pins without any via insertions. If there are no slots available on M2, the Pin Optimizer tries to place pin A on M1, which is the layer closest to M2.

When the *Respect layer priority for valid routing layers constraint on net* option is selected, pins are optimized based on their layer priority. Layers are prioritized in the order in which they are listed.

Corresponding environment variable: layerPriority

For more information about Pin Optimization, see Pin Optimization.

© 2020 All Rights Reserved.

Specifying the Default Block Type

Floorplanner supports two types of blocks - custom and digital. Soft blocks can be assigned to either of these block types. Use the *Default blockType (when unknown)* option to specify the default block type for soft blocks when the assigned block type is not custom or digital. The default value is *Custom*.

The block type determines the pin and PR boundary placement during floorplanning, as described below:

- **Custom** - Pins and PR boundary are placed on the manufacturing grid.
- **Digital** - Pins are placed on routing grid, and PR boundary is placed on the placement grid.
- **As-Is (XY Snap Spacing)** - Pins and PR boundary are placed on the grid defined by the *X Snap Spacing* and *Y Snap Spacing* options in the *Grid Controls* section of the Display Options form. For information about these options, see Setting Up Grid Controls in *Virtuoso Layout Suite L User Guide*.

For information about setting the block type, see Setting the Block Type.

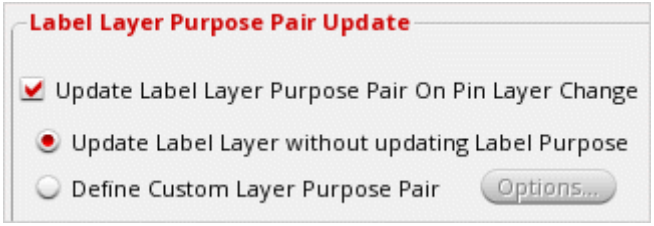
Corresponding environment variable: defaultBlockTypeWhenUnknown

SKILL function to set the placement grid information: dbSetPlacementGrid

© 2020 All Rights Reserved.

Updating Label Layer Purpose Pairs

When you run certain pin-related commands, such as the Pin Planner and Pin Optimizer, pins in the design are re-layered. Use the options in the *Label Layer Purpose Pair Update* section of the Floorplan Global Options from to control the re-layering of the associated pin labels.



To control the re-layering of pin labels, in the *Label Layer Purpose Pair Update* section, *Update Label Layer Purpose Pair On Pin Layer Change* is selected by default.

Define the label re-layering behavior by selecting one of the following options:

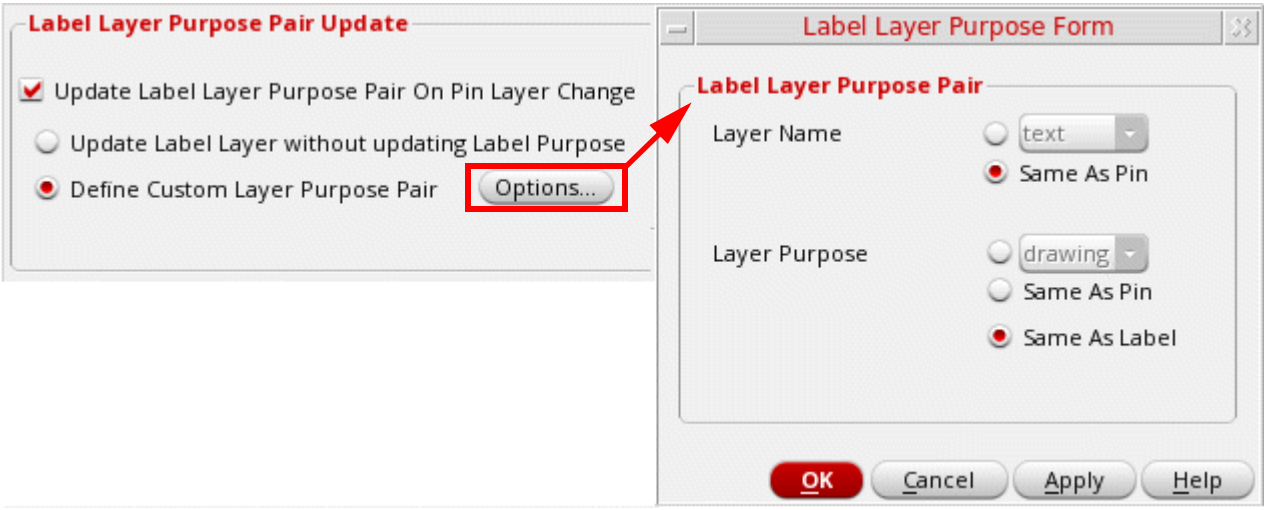
- *Update Label Layer without updating Label Purpose* (default): Re-layers labels to match the pins, but their purposes remain unchanged.

  Related Environment Variable: labelLayerPurposePair

- *Define Custom Layer Purpose Pair*: Lets you define the layer and purpose to which labels are to be moved. To specify custom layers and purposes:

  a. Select *Define Custom Layer Purpose Pair*.

  b. Click *Options* to display the Label Layer Purpose form.



© 2020
All Rights Reserved.

c. Select one of the following *Layer Name* options to specify the layers to which the labels are to be moved:

- *Custom layer*: Specifies the layer to which labels are to be moved. Irrespective of the pin layer, all labels are moved to the selected layer.

  Related Environment Variable: `pinTextLayer`

- *Same as Pin*: Moves labels to the same layers as the pins.

  Related Environment Variable: `pinTextSameLayer`

d. Select one of the following *Layer Purpose* options to specify the purposes to be assigned to labels when their associated pins are re-layered:

- Custom purpose: Specifies the purpose to be assigned to labels when the associated pins are re-layered.

  Related Environment Variable: `pinTextPurpose`

- *Same as Pin*: Matches the label purpose to that of the pins after they are re-layered.

  Related Environment Variable: `pinTextSamePurpose`

- *Same as Label*: Retains the original label purpose.

  Related Environment Variable: sameLabelPurpose

When pins are re-layered with *Update Label Layer Purpose Pair On Pin Layer Change* not selected, labels remain unchanged.

e. Click *OK*.

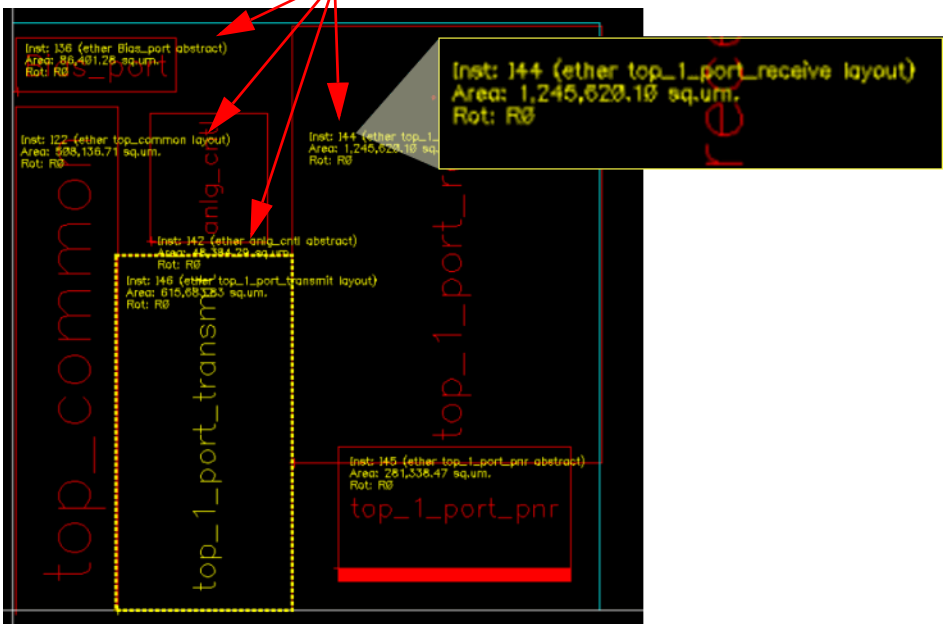
Note: When both `pinTextSamePurpose` and sameLabelPurpose are set to `t`, `sameLabelPurpose` takes precedence. This is irrespective of the `pinTextPurpose` setting.

© 2020 All Rights Reserved.

Block Annotations

During floorplanning, you can turn on block annotations to view textual information about the blocks. Block annotations are aligned to the top-left corner of the PR boundary of the blocks, and include the block instance names, master cellview information, area, and orientation values.

Block Annotations

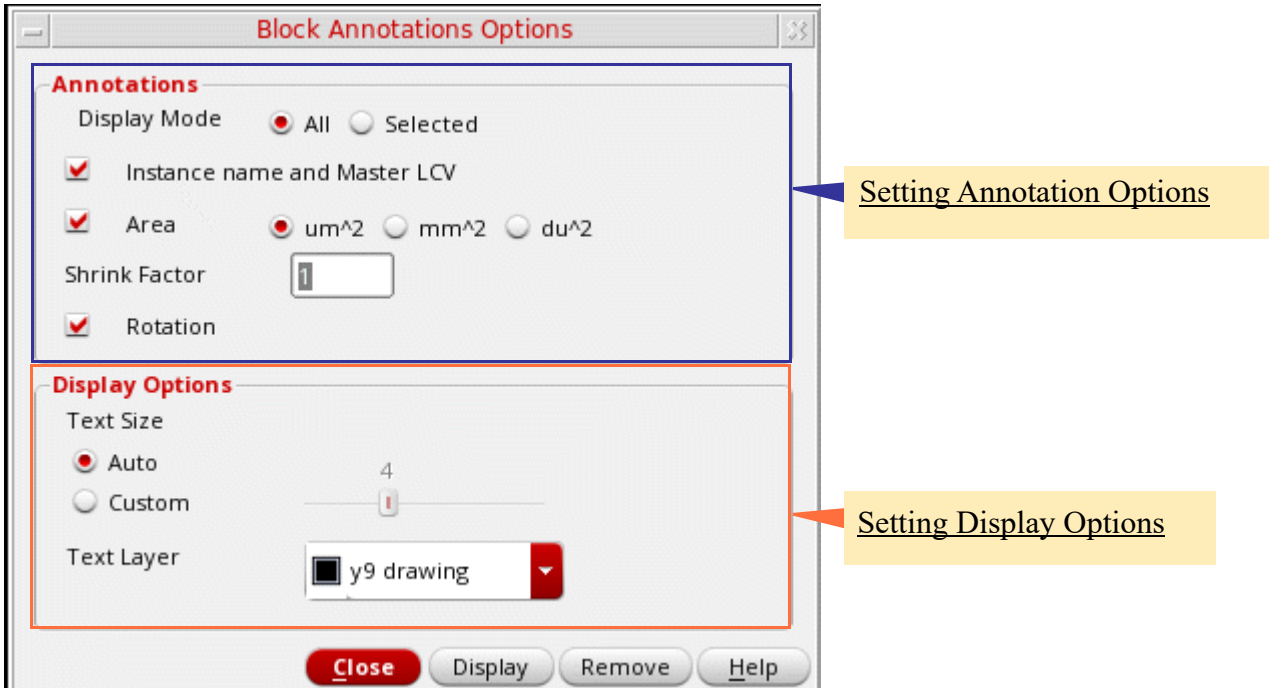


Any change made to the block is dynamically updated in the annotated information. For example, if a block is moved, the information text is also moved to the new location. Similarly, when a block shape is changed by stretching or chopping the block, the annotation values are dynamically updated.

To turn on block annotations, choose *Floorplan – Block Annotations.* The Block Annotations Options form is displayed. Use the options in this form to customize the

© 2020 All Rights Reserved.

information displayed in the block annotations. The Block Annotations form is available for both edit and read-only cellviews.



Important

In Layout EXL, choose *Plan – View – Block Annotations*.

Setting Annotation Options

Use the options in the *Annotations* section to specify the information that is to be included in the block annotations.

1. Set the *Display Mode* to specify whether the block annotations are to be displayed for *All* or *Selected* blocks.
2. Select *Instance name and Master LCV* to display the relevant information in the block annotations.
3. Select *Area* to display the total block area within the PR boundary. Select the required units:
   - `um^2`: Square micrometers.
   - `mm^2`: Square millimeters.
   - `du^2`: Square database unit.

© 2020 All Rights Reserved.

4. *Shrink Factor* specifies the factor by which each edge has to be shrunk. Specify a *Shrink Factor* to display the shrunk area. The shrunk area is only displayed in the annotation without changing any values in the design.
5. Select *Rotation* to include the block orientation information in the block annotations.

Setting Display Options

Use the options in the *Display Options* section to customize the following text settings:

- **Text Size:** Specifies the size of block annotations. Select one of the following options:
  - *Auto:* Calculates the annotation size such that all annotations are inside the instance boundary.
  - *Custom:* Lets you drag the slider to the required value.

  Click *Display* to apply the size of annotations to the layout design.
- **Text Color:** Select a *Text Layer* based on which the block annotation text is to be formatted.

Click *Display* to view and *Remove* to hide block annotations in the layout canvas.

Note: Clicking Close will not remove the block annotations.

Environment Variables

- blockAnnotationArea
- blockAnnotationAreaUnits
- blockAnnotationInstName
- blockAnnotationRotation
- blockAnnotationSelMode
- blockAnnotationTextLPP
- blockAnnotationTextSize
- blockAnnotationShrinkFactor

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

11

Updating Pin Labels and Text Displays

During pin optimization, pins are re-positioned to obtain the shortest possible net length. However, the Pin Optimizer does not update pin labels and text displays for clear visualization. This may result in positioning of pin labels and text displays outside the prBoundary causing overlaps and inapt visualization, defeating the purpose of labeling.

Therefore, after running the Pin Optimizer, use the *Floorplan > Label Update* command to update pin labels and text displays for clearer visualization.

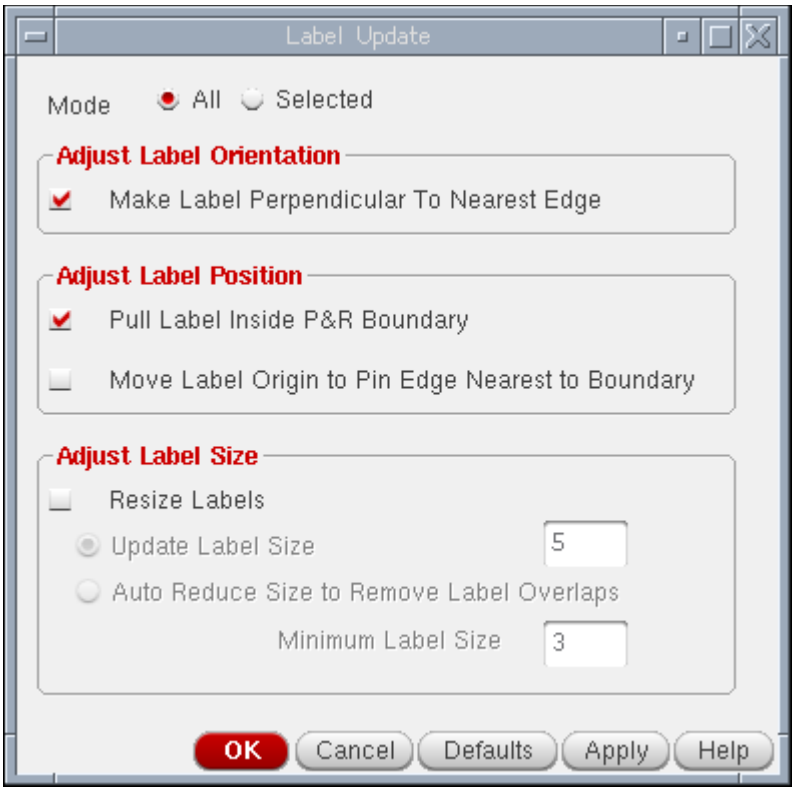
Important

In Layout EXL, choose *Plan – Pin Planning – Label Update*.

© 2020 All Rights Reserved.

Label Update GUI Options

The *Floorplan > Label Update* command displays the Label Update form.



Caution

The Floorplan > Label Update command is disabled when the top-level cellview is in the read-only mode.

The Label Update form has the following options for updating pin labels and text displays:

- Mode: Choose one of the following modes:
  - All: Runs on all soft block pin labels and text displays in the current design.

    Note: Soft blocks that are marked as read-only are make editable and a warning message is displayed in the CIW.
  - Selected: Runs only on the selected pin labels and text displays. You can select:
    - Top-level and level-1 pins
    - Individual soft blocks

© 2020 All Rights Reserved.

- Individual top-level pin labels and text displays

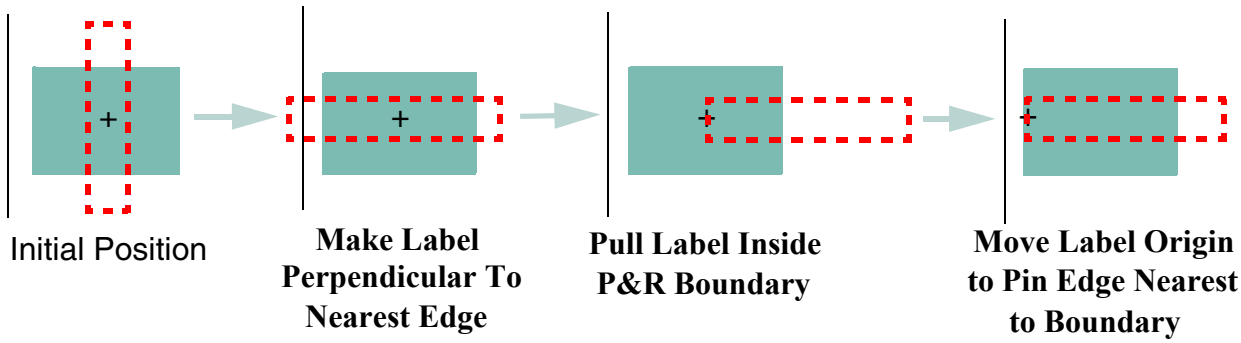
- Adjust Label Orientation:
  - Make Label Perpendicular To Nearest Edge: Makes labels and text displays perpendicular to the nearest prBoundary edge. The nearest prBoundary edge is determined based on the minimum distance of the label's parent pin center from the prBoundary.

    Note: Orientation remains unchanged for labels and text displays that are already perpendicular to the nearest prBoundary edge.

- Adjust Label Position: Includes the following options:
  - Pull Label Inside P&R Boundary: After adjusting the label orientation, some labels and text displays may protrude outside the prBoundary. Select such labels and text displays and choose the *Pull Label Inside P&R Boundary* option. When you choose this option, the tool tries different permutation and combinations for justification and orientation to place these labels and text displays inside the prBoundary. The label or text display origins, however, remain unaltered.

    Note: If all labels and text displays could not be successfully pulled inside the prBoundary, an appropriate report is displayed in the CIW.

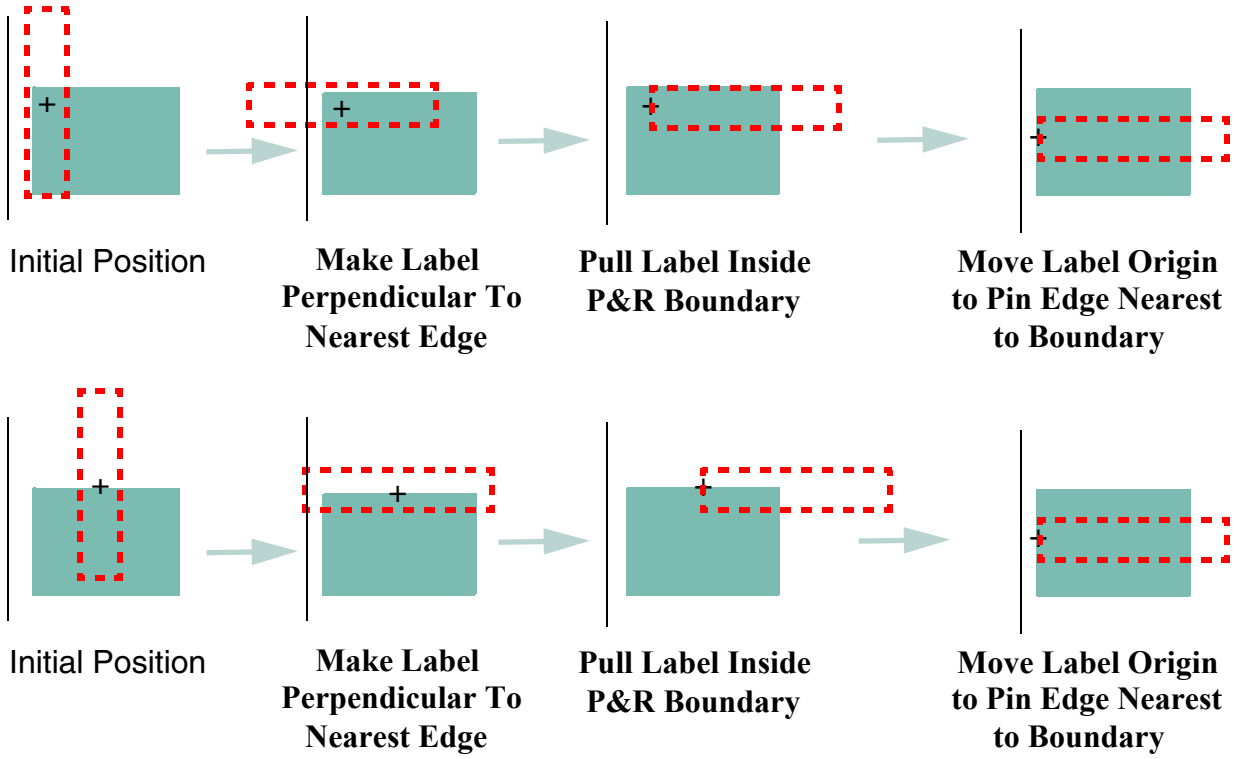
  - Move Label Origin to Pin Edge Nearest to Boundary: Irrespective of their initial origins, on selecting *Make Label Perpendicular To Nearest Edge*, all labels and text displays are moved to the center of the nearest pin edge as illustrated in the following figures:



Tip

The *Move Label Origin to Pin Edge Nearest to Boundary* option is not selected by default.

© 2020 All Rights Reserved.



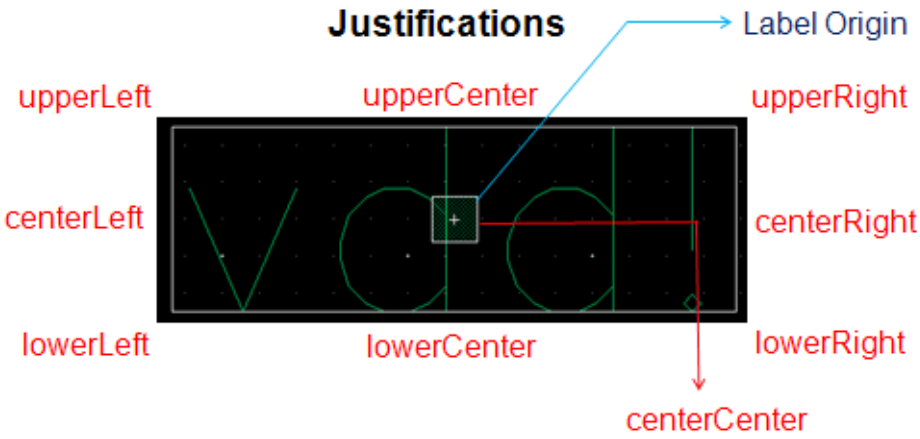
Depending on the access direction of the nearest prBoundary edge from the label parent pin, the orientation and justification of labels is set based on the following table:

Edge	Orientation / Justification
Left	R0/centerLeft
Right	R0/centerRight
Top	R90/centerRight
Bottom	R90/centerLeft

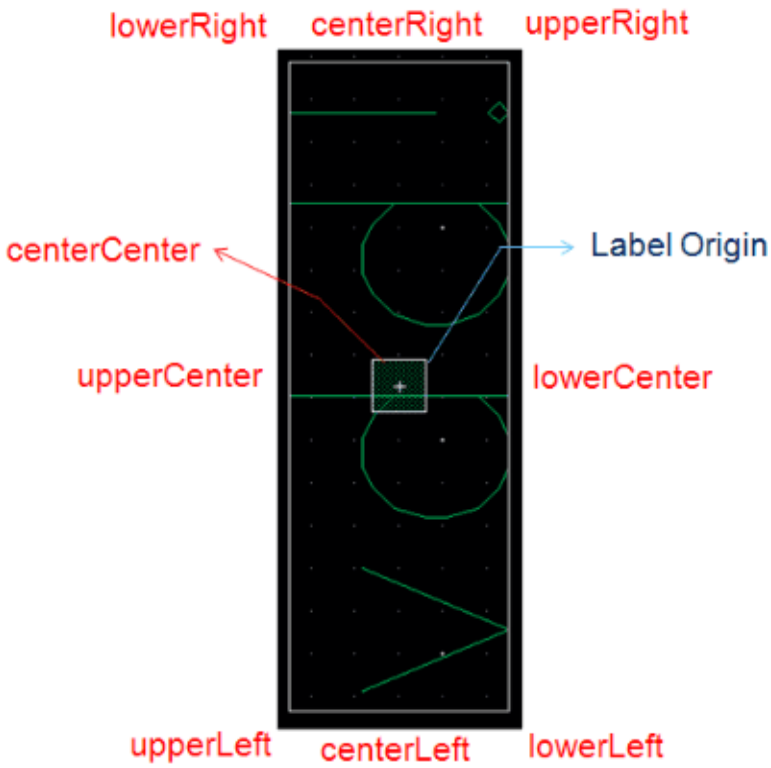
© 2020 All Rights Reserved.

The following images depict the various orientations and justifications:

Orientation = R0
Justifications



Orientation = R90
Justifications



The Adjust Label Orientation and Adjust Label Position options will not run for labels and text displays for which:

© 2020
All Rights Reserved.

- The label or text display origin is already outside the prBoundary.
- The label or text display parent is outside prBoundary.

In the above situations, appropriate message is printed in the report.

- Adjust Label Size: Includes the following options to *Resize* Labels:
  - Update Label Size: Choose this option and specify the label size in the adjoining box.

    The *Update Label Size* option will not run in the following situations.

    - On floating labels and text displays that are not associated with any parent pin.
    - When the label or text display parent is not pin.
    - When labels or text displays for the instance pins are present on the top-level instead of within the same cellview as the pin.

    In the above situations, appropriate message is printed in the report.
  - Auto Reduce Label Size: Choose this option when label or text display overlaps exist even after making the above settings. This option intelligently computes the overlaps and reduces the size of the overlapping labels or text displays so as to remove any remaining overlaps. The size of labels or text displays is reduced upto a minimum threshold value as specified in the adjoining text box.

When multiple options in the Label Update form are selected, the following order of execution is followed for best results:

1. *Update Label Size*
2. *Make Label Perpendicular to Nearest Edge*
3. *Move Label Origin to Pin Edge Nearest to Boundary*
4. *Pull Label Inside P&R Boundary*
5. *Auto Reduce Size to Remove Overlaps*

© 2020 All Rights Reserved.

12

Other Floorplanning Commands

This chapter introduces you to the following additional Floorplanning commands that are used to develop a layout from a schematic.

- Edit Soft Blocks
- Load Physical View
- Create Soft Pins
- Snap Pins
- Analyze Connectivity
- Create Feedthrough Terminal Pins
- Creating Boundary Pins on Selected Shapes
- Level-1 Editing
- Pin Connectivity Setting
- Set Celltype
- Pin Alignments

These commands are not available in the Floorplanner menu. The commands discussed here are a part of Virtuoso Layout XL suite and are covered in detail in the Virtuoso Layout XL user guide.

Important

In Layout EXL, these commands are available in the *Plan* menu.

Edit Soft Blocks

The Edit Soft Blocks command lets you modify the soft blocks in your design. Use the *Floorplan – Soft Block Update – Edit Soft Blocks* command to modify the attributes of

© 2020 All Rights Reserved.

individual soft blocks to take account of placement and connectivity requirements in your design. You can also access this command from *Edit – Edit Soft Blocks*. This command is recommended if you want to make changes that do not require the physical hierarchy to be regenerated. For more information on how to edit soft blocks, refer to the Virtuoso Layout Suite XL User Guide.

Load Physical View

The Load Physical View command is used to import the predefined information from a source physical cellview into a current cellview. To do this, select Floorplan – Load Physical View. You can also access this command from File – Load Physical View. For more information on how to load physical view into current layout cellview, refer to the Virtuoso Layout Suite XL User Guide.

Create Soft Pins

After the physical hierarchy is configured, if you need to improve the connectivity of your design, you can create additional soft pins on your soft blocks. The *FloorPlan – Connectivity – Create Soft Pins* command lets you create, delete, and update soft pins for your soft blocks. You can also access this command from Create – Soft Pins. To know more on how to create soft pins, refer to the Virtuoso Layout Suite XL User Guide.

© 2020 All Rights Reserved.

Snap Pins

The Snap Pins command is used to snap both top-level and level-1 pins to the grid appropriate to the block type. To snap all soft block pins, select *Floorplan – Connectivity – Snap Pins*. You can also access this command from *Connectivity – Pins – Snap Pins*.

At advanced nodes, this command provides additional options to snap pins to WSP grids.

For more information on snapping soft block pins, refer to the Virtuoso Layout Suite XL User Guide.

© 2020 All Rights Reserved.

Analyze Connectivity

Use the *Analyze* command to calculate the number of connections between each pair of selected instances, draw flight lines between each connected pair, and indicate the number of connections by displaying a numerical value on the flight lines. This is a command that generates information about connectivity and other useful design information. It is useful to find out the connectivity (the number of connections) of a block with other blocks, so that during manual floorplanning, you can keep the blocks that have more connectivity between them closer to each other.

To analyze the connectivity between instances:

1. Open the Analyze Connectivity form.

   For this command, only the pre-selection mode is supported. You can select multiple blocks whose connectivity you want to examine. Use one of the following methods to open the Analyze Connectivity form:

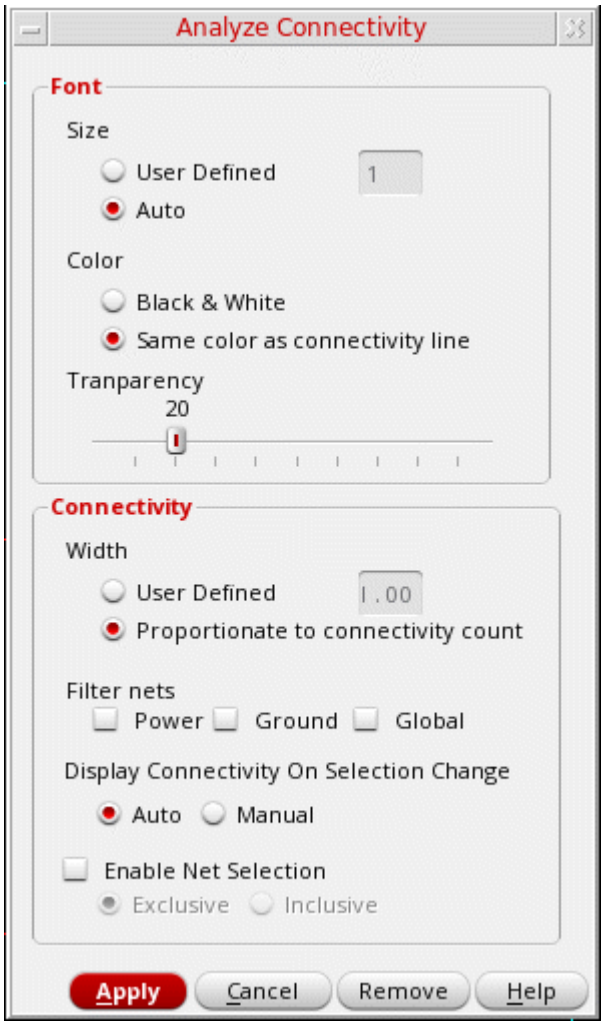
   - Choose *Floorplan – Connectivity – Analyze.*

     Note: In Layout EXL, choose *Plan – View – Analyze Connectivity*.

   - Choose *Connectivity – Analyze.*
   - Click the Analyze Connectivity button on the Design Planner toolbar.

© 2020 All Rights Reserved.

For more information, see Design Planner Toolbar.

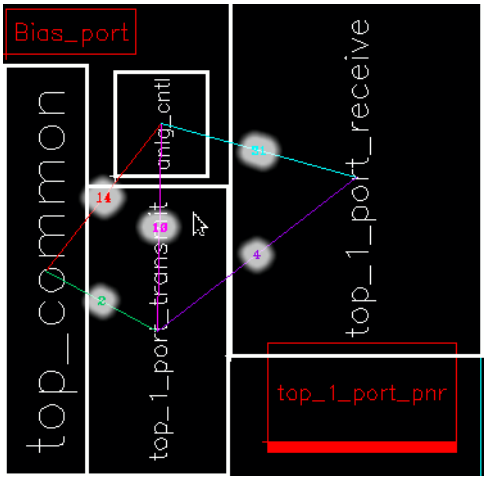


2. In the *Font* section, specify the size of the flight lines. Set *Size* to one of the following values:

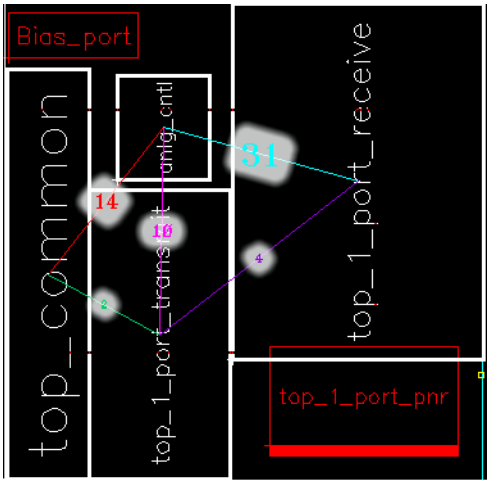
- *User Defined*: Specify the absolute font size. If the value is `0`, flight lines are displayed when you click *Analyze*, but numbers are not displayed.

© 2020 All Rights Reserved.

- *Auto* (default): Adjusts the font size based on the number of connections. For example, if there are 47 connections between two devices, its label would have a larger font size than the label between devices with 14 connections.

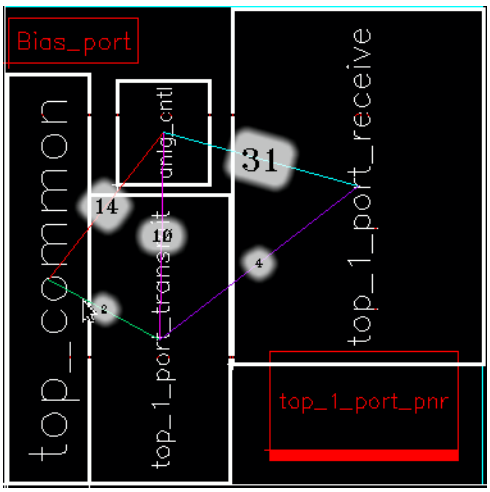


Size = User Defined (40)
All values are in the specified font size.

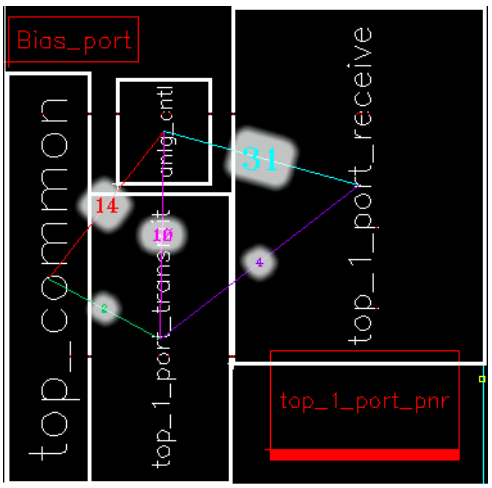


Size = Auto
Font size is as per the number of connections.

3. Specify the color of the text that is displayed on the flight lines. Set *Color* to either *Black & White* or *Same color as connectivity line* (default).



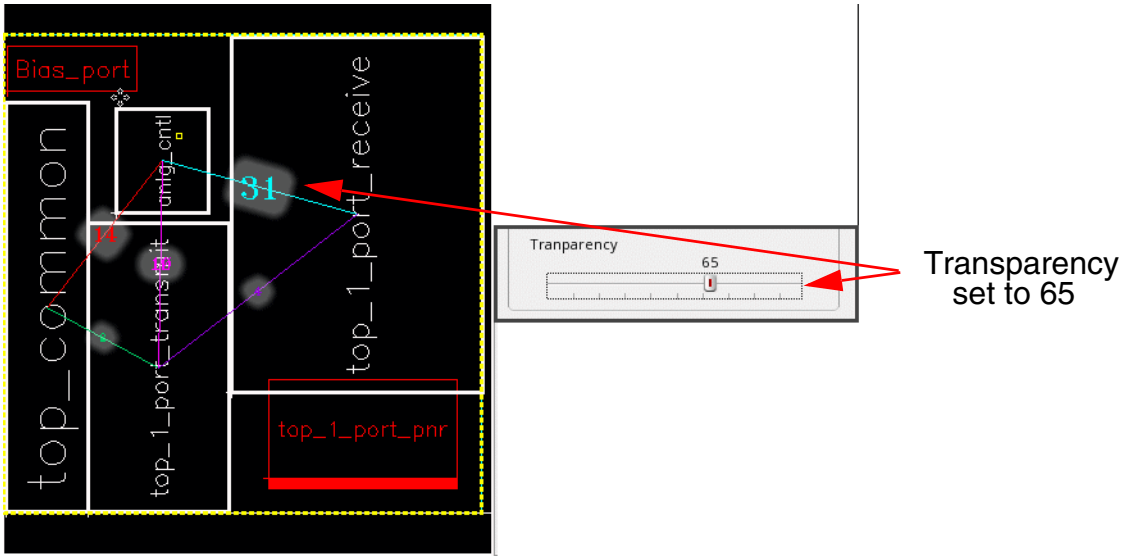
Color = Black & White



Color = Same color as connectivity line

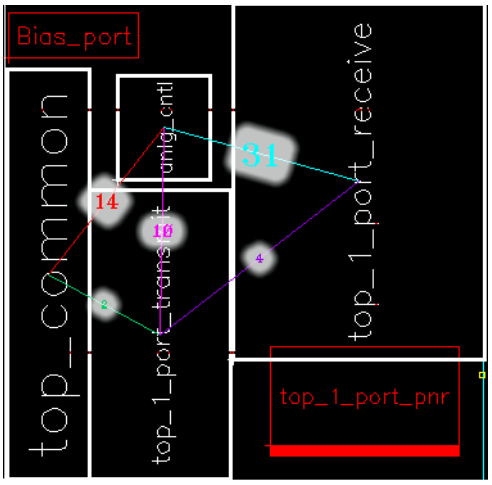
© 2020
All Rights Reserved.

4. Adjust the transparency of the text on the screen. Drag the *Transparency* slider to specify a value. In the following example, the transparency value is increased to `65`.

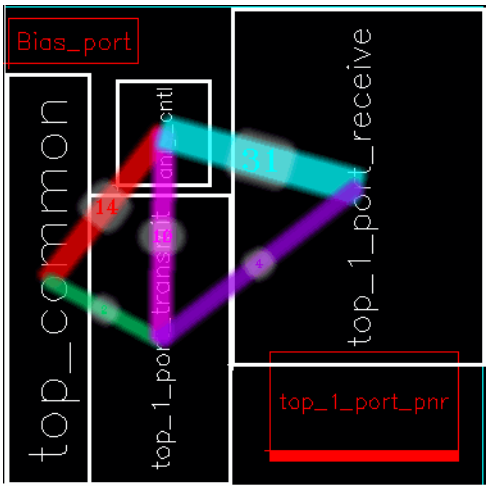


5. In the *Connectivity* section, specify the width of the flight lines by setting *Width* to one of the following values:

- *User Defined*: Lets you specify the width.
- *Proportionate to connectivity count*: Adjusts the line width depending on the number of connections.



Width = User Defined (1.00)



Width = Proportionate to connectivity count

© 2020
All Rights Reserved.

6. Select *Filter nets* to hide the connections to the selected type of nets - *Power*, *Ground*, and *Global*.
7. Set *Display Connectivity* to one of the following values:
   - *Auto*: Enables dynamic selection of objects such as instances, PR boundary, and figGroups. Connectivity information is refreshed dynamically when you select a different object.
   - *Manual*: With this option selected, connectivity information does not change dynamically.

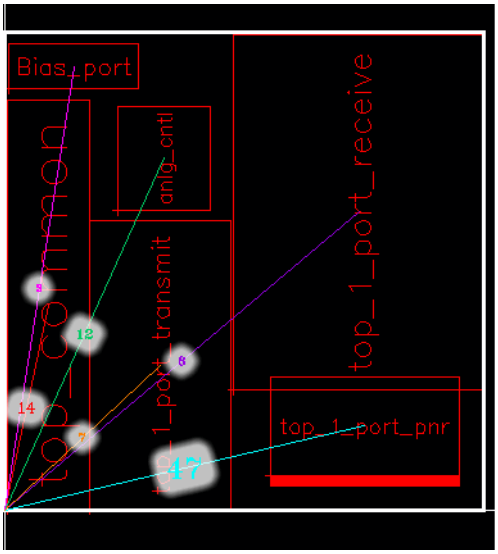
     Click either *Apply* or *Remove* to remove previous highlights before you select new objects. Clicking *Apply* commits the current settings, whereas clicking *Remove* removes all existing highlights.
8. Select *Enable Net Selection* to enable cross-selection of nets between the design canvas and the Navigator assistant. Therefore, when the Analyze Connectivity command is run, the nets highlighted in the design are automatically selected in the Navigator assistant. Use the following options to control net selection:
   - *Exclusive* (default): Highlights only those nets that exclusively connect to the selected blocks. Connections to all other blocks are excluded.
   - *Inclusive*: Highlights all nets that are connected to the selected blocks.
9. Click *Apply* to make changes.

   You can also click *Cancel* to ignore the changes or *Remove* to delete all the flight lines and associated information within the selected boundary or between the selected instances.

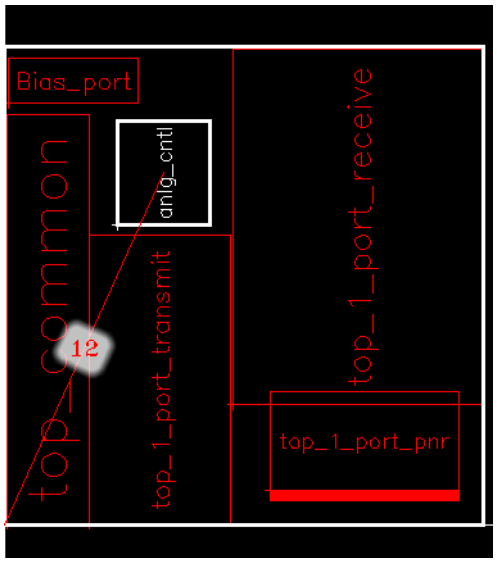
Instead of selecting individual instances, you can also:

© 2020 All Rights Reserved.

- Select the PR boundary and analyze its connectivity: Connections from the design to all the instances in the design are considered, as shown below:

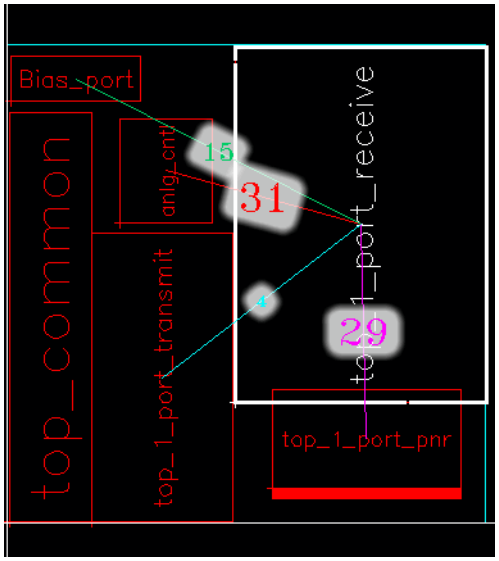


- Select the PR boundary and one or more instance and analyze their connectivity: Connections between the design and the selected instances are considered, as shown below:



© 2020
All Rights Reserved.

- Select a figGroup and analyze its connectivity: All connections to the selected figGroup are considered, as shown below:



If an instance, figGroup, or PR boundary was not selected before choosing the *Analyze* command, the following message appears in the CIW.

```
*WARNING* VFP-4005: Could not analyze connectivity because neither the PR boundary
nor any instance was selected. Select the PR boundary or the required instances,
and rerun the command.
```

If you select only a single instance, *Analyze* displays all the instances and figGroups connected to that instance.

Certain designs may have a high number of connections between nets and `insTerms`. Before analyzing the connectivity in such designs, ensure that the number of connections of each net to `instTerms` is within the maximum terminals per net threshold value. If the value exceeds the threshold value, the corresponding net is ignored during computation of connectivity, and an INFO message is populated in the CIW for the corresponding net.

The `adpEnvPinsPerNetThreshold` variable defines the maximum terminals per net threshold value. Use the `adpnlSetEnv` API to set the `adpEnvPinsPerNetThreshold` variable such that the maximum number of terminals per net threshold value is higher than the number of connections of nets to `instTerms` in your design.

© 2020 All Rights Reserved.

Create Feedthrough Terminal Pins

To create additional feedthrough pins in order to improve the connectivity of your design, the *Create – Feedthrough Terminal Pins* command is used. To know more on how to create feedthrough pins, refer to the *Virtuoso Layout Suite XL User Guide*.

Creating Boundary Pins on Selected Shapes

The `vfpCreateBoundaryPinsForSelectedShapes` SKILL API is used to create boundary pins on selected shapes. You need to select the shapes and run the API to create all boundary pins automatically, in one go. You specify the distance from the PRBoundary at which pins need to be created on the selected shapes. This distance will impact the size of pin created.

If the selected shapes are not associated with nets, only the boundary shapes are generated; no corresponding pin object is created.

Level-1 Editing

The Level-1 Editing allows you to edit the soft block without descending to hierarchy and also allows to edit soft blocks by looking at adjacent blocks. The commands used to do Level-1 Editing are as follows.

- Move
- Stretch
- Reshape
- Chop

For more information on Level-1 editing commands and supported objects, refer to the *Virtuoso Layout Suite XL User Guide*.

Pin Connectivity Setting

The Pin Connectivity Setting command allows you to create, change, and delete strong and weak pins, and create must-connect pins. You can change a weak pin to a strong pin or vice-versa, or create must connect groups. To do this, select *Connectivity – Pins – Pin Connectivity Setting* command. For more information on this, refer to the *Virtuoso Layout Suite XL User Guide*.

© 2020 All Rights Reserved.

Set Celltype

To set the celltype of blocks, click *Tools -> Set Cell Type* in the CIW. This command provides various features to search and set celltype for multiple cells. For more information, refer to Setting the Cell Type.

Pin Alignments

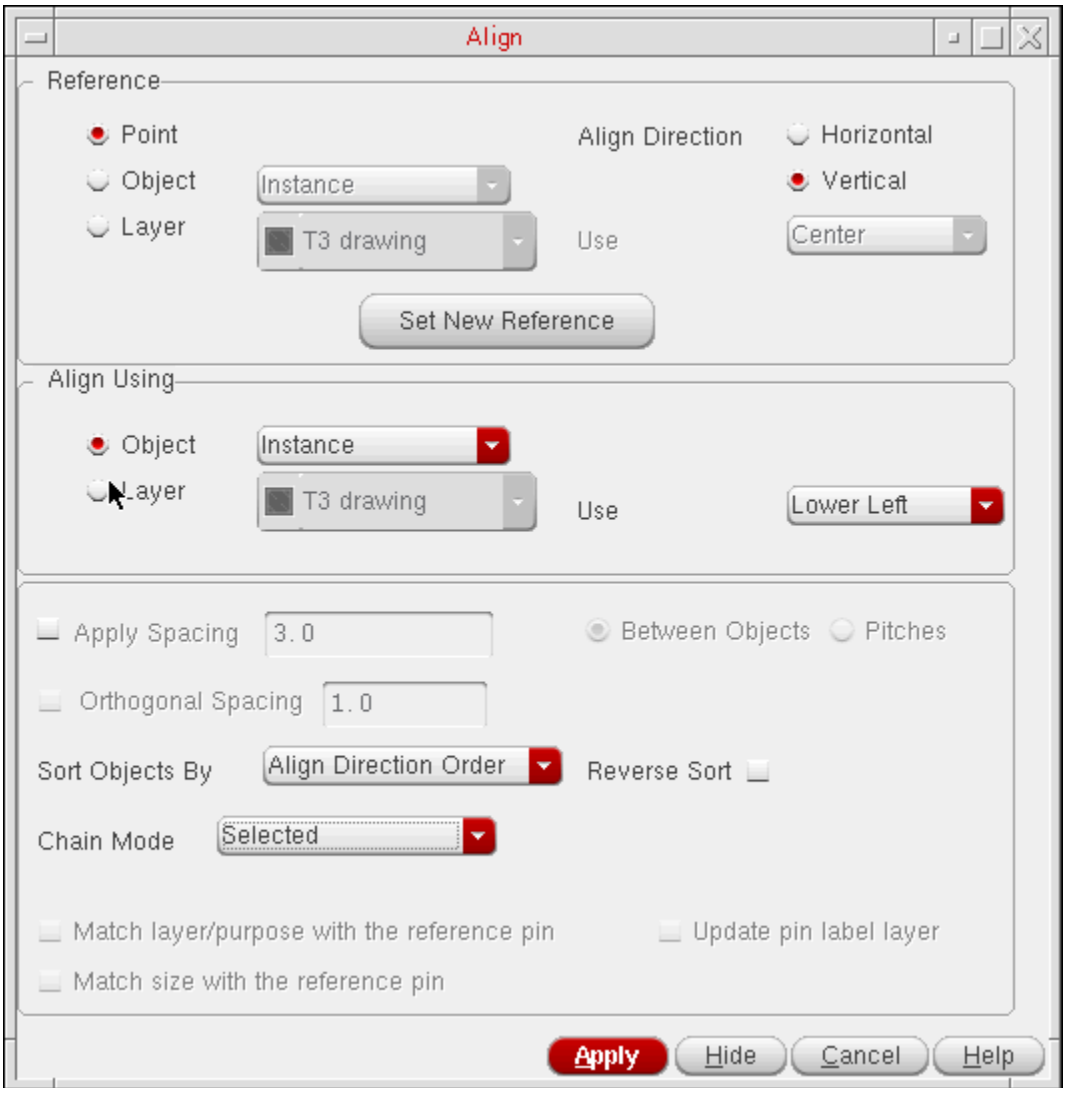
You align pins between blocks to get the shortest net length. The pin aligner is connectivity based and no specific pin selection is required, thereby, allowing multiple selection. The pin aligner aligns a list of target pins with respect to the position of a list of reference pins. Reference pins are the pins whose positions are used as the reference for alignment.

Reference pins can belong either to a block or to the top-level design. The block may be a soft block or a hard block because the pins of the reference block are not moved. Target pins are the pins that are moved by the alignment command. The target pins can either belong to a soft block or to the top-level design.

Use the *Edit - Advanced - Align* command to align pins. Alternatively, you can use the `vfpAlignPins` SKILL function. When you use the *Edit - Advanced - Align* command, the soft block master becomes editable automatically. To align pins, perform the following steps:

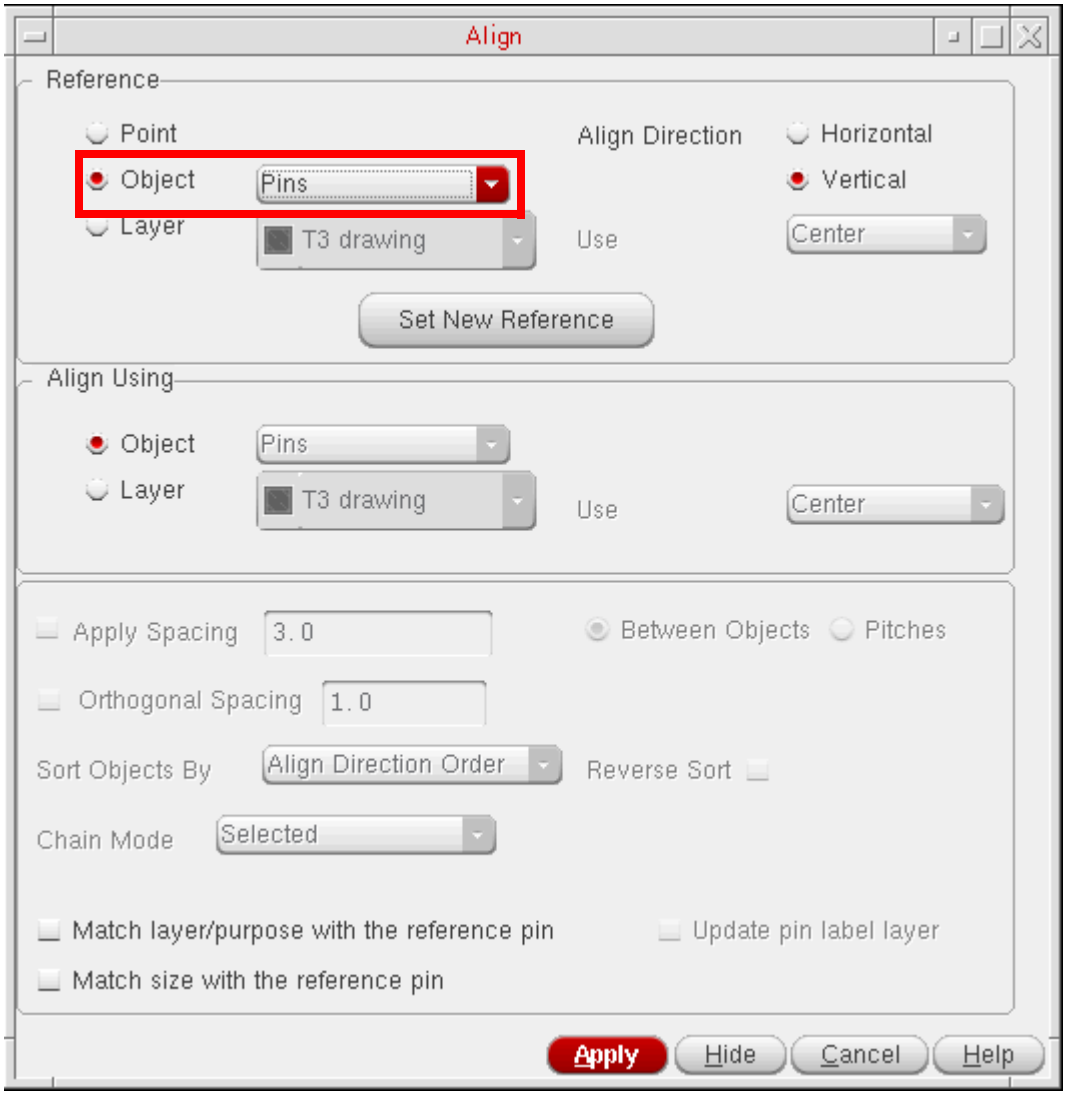
© 2020 All Rights Reserved.

1. Choose the *Edit – Advanced - Align* command. The following form is displayed.



© 2020
All Rights Reserved.

2. Select the *Object* option in the *Reference* section. Select *Pins* from the field next to the *Object* option.

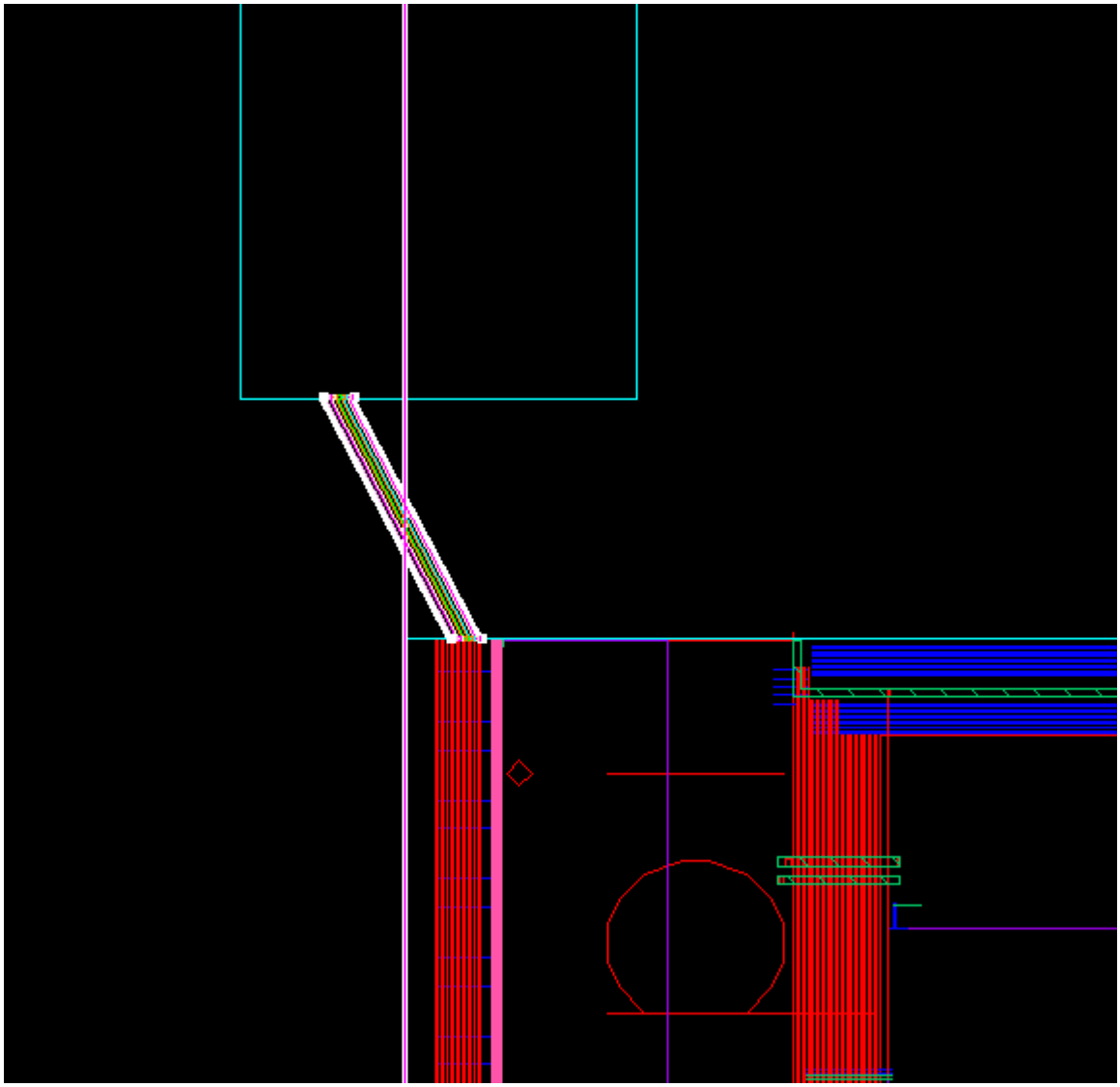


The following prompt appears:

```
Select reference pins/instances/top-level PRBoundary for pin-alignment:
```


© 2020
All Rights Reserved.

3. To select the reference pins, click the corresponding blocks. You can select multiple blocks such as the top-level PRBoundary, a soft block, or a hard block as reference blocks. The pins in these blocks are selected as the reference pins.



You can select pins in different reference blocks as the reference pins. The pins in the target block will be aligned with respect to these reference pins in different reference blocks.

Note: If you select the same pin on multiple instances of the same master, only one of the pins is considered and the remaining ones are ignored.

4. Click the middle mouse button to switch to the target selection mode. A new message prompt appears:

© 2020 All Rights Reserved.

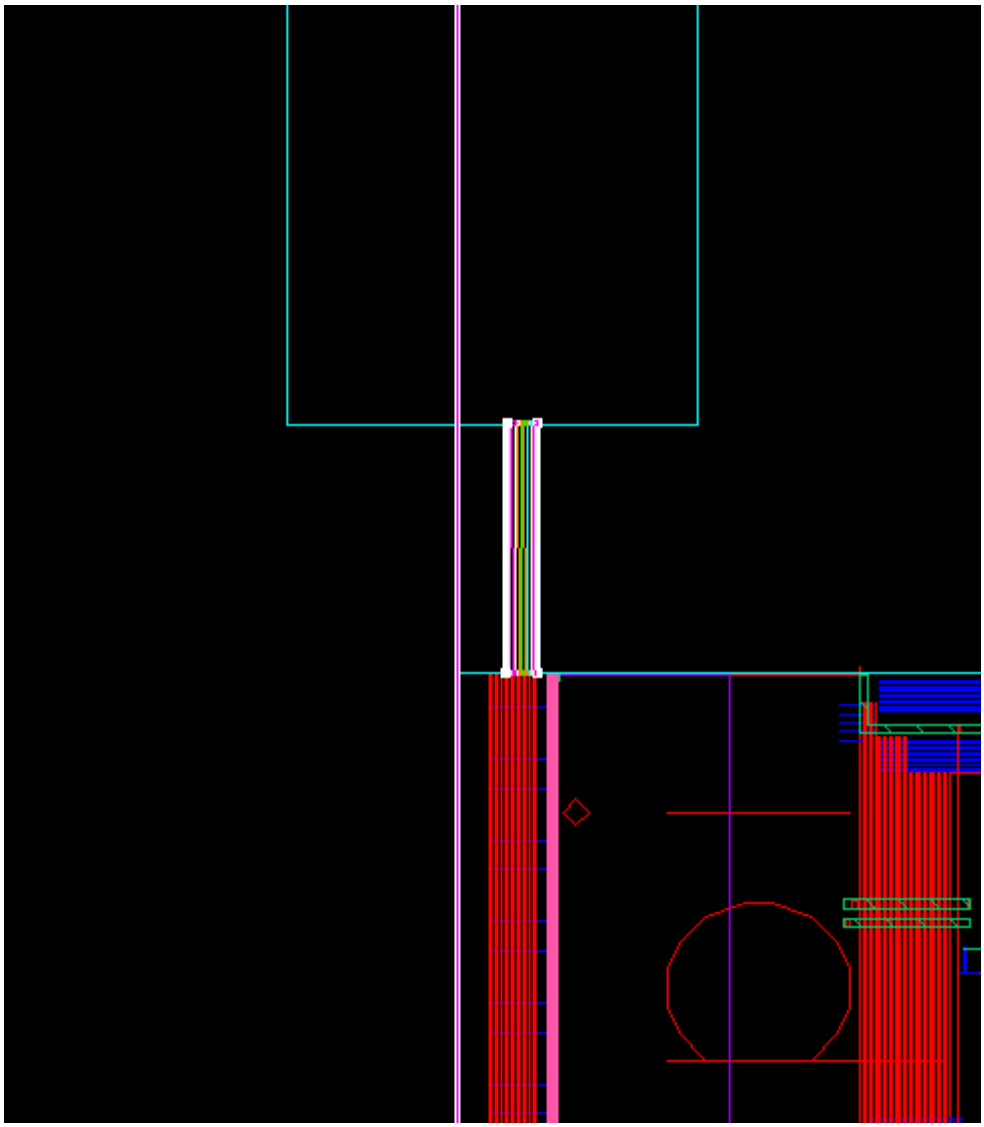
```
Select target pins/instances/top-level PRBoundary for pin-alignment:
```

5. To select the target pins, click the corresponding blocks.You can select multiple soft blocks, top-level pins, top-level PRBoundary or soft-block pins.

6. Click the middle mouse button or the *Apply* button in the *Align* form to mark the end of target selection.

   The pin aligner aligns the target pins with respect to the selected reference pins. The following message prompt is displayed:

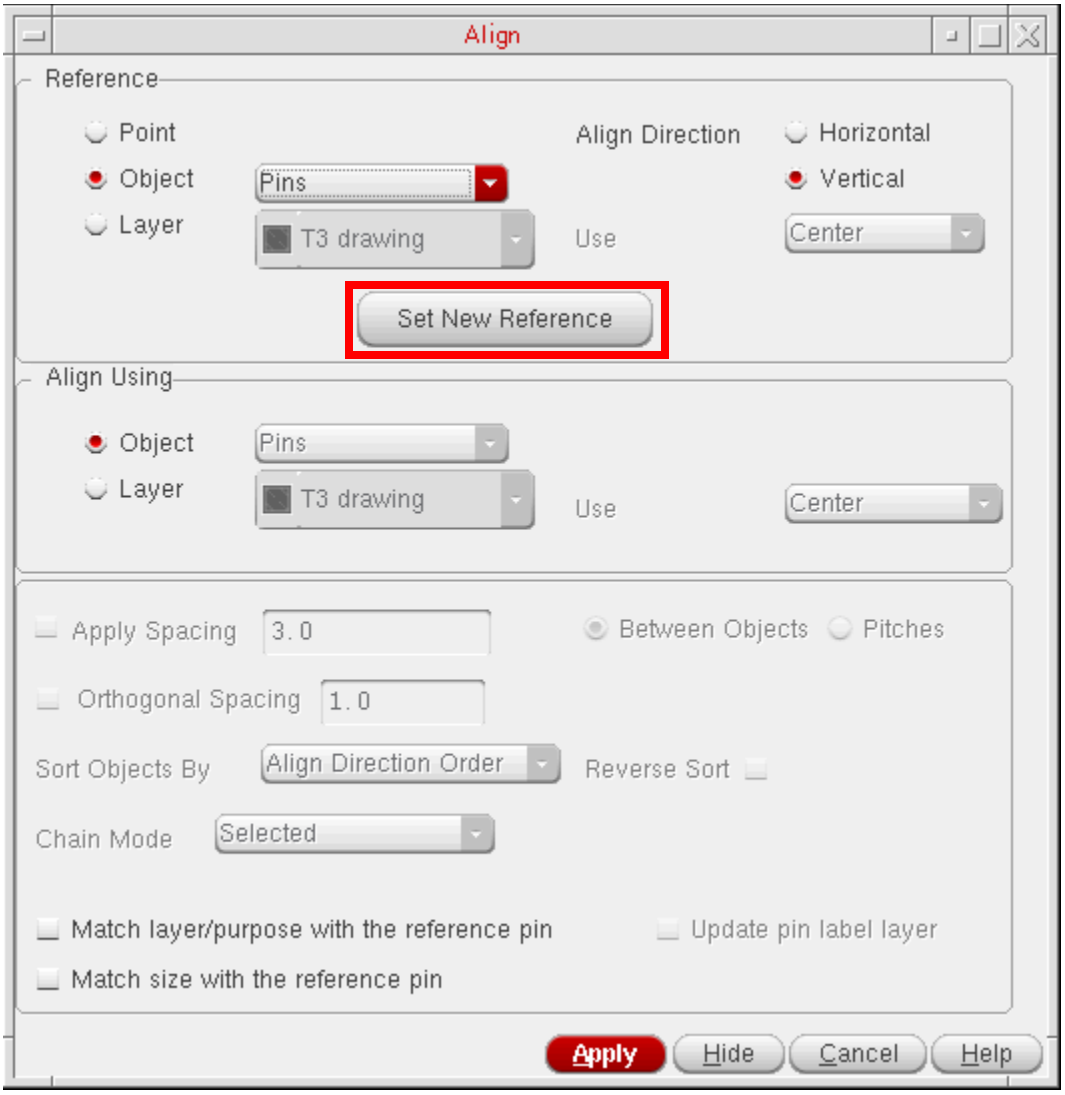
```
Select target pins/instances/top-level PRBoundary for pin-alignment:
```



7. You can now select another target object to align with the same reference.

© 2020 All Rights Reserved.

You can also click the *Set New Reference* button on the Align form to specify a new reference for aligning objects. This command resets the form and enables you to start again by selecting a new reference.



Note: You can use the `Esc` key to cancel the alignment command.

8. If you want to change the target pin layer/purpose of the pin to the reference pin layer purpose, select the *Match layer/purpose with the reference pin* check box. For example, if the target pin is on `Metal2 drawing` layer purpose and the reference pin is on `Metal1 drawing`, then selecting this option will change the target pin layer to `Metal1 drawing`.

9. When the Match layer/purpose with the reference pin option is selected, the Update pin label layer option is also selected by default. In this state, the pin labels are placed on the

© 2020 All Rights Reserved.

same layer as the target pins. So, when a pin is moved to a different layer, the pin label is also moved to the same layer.

However, this option will work only if the purposes of the reference and target pins match.

Example 1:

Reference pin LPP= (Metal1 drawing)

Target pin LPP = (Metal2 drawing) - ✓ exists

Pin label's layer will be updated to the target layer along with the pin.

Example 2:

Reference pin LPP= (Metal1 drawing)

Target pin LPP = (Metal2 drawing) – **X** does not exist

Pin label's layer will be not be updated. Only the pin layer will be updated.

10. If you want to change the target pin size to that of the reference pin, select the *Match size with the reference pin* check box. For example, if the target pin size is 0.6 x 0.6 and the reference pin size is 1.0 x 1.0 then selecting this option will change the target pin size to 0.6 x 0.6.

Note: Ensure that the target pin size and the minWidth of the target layer do not lead to a DRC violation on minWidth. Also, pin alignment will not make DRC checks.

After aligning the pins, the pin aligner performs a post-process check to determine any overlaps:

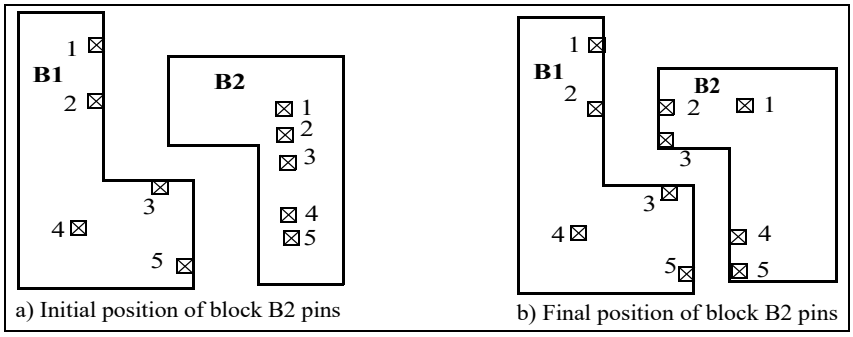
- Within the target pins
- With other shapes that are part of the design, but are not the target pins

If any overlap exists, then the pin aligner displays a warning message indicating the previous location, layer, term name, and new location.

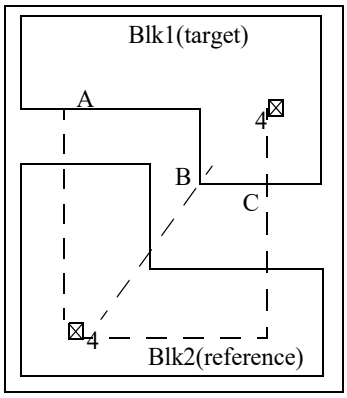
For example, in the figure below, pins `1` to `5` of block `B1` are connected to pins `1` to `5` of block `B2`, respectively. If you select all the pins of block `B1` as the reference pins and the block `B2` as the target block, the pins of block `B2` are moved as shown in the figure marked `b`. Note that pin `1` of block `B2` is not moved at all because no edge is available to get it aligned. Any target pin connected to a reference pin is moved to the first edge of the target block encountered by an imaginary line orthogonal to the reference pin edge.

© 2020 All Rights Reserved.

If none of the imaginary orthogonal lines touch the target block, the pin will not be moved (for example, pin `1` in block `B2`).



If a reference pin is not on the edge or is on the corner (and has two edges) then out of two orthogonal imaginary lines, the one that hits the target edge point (see example below) is considered and the connected target pin is moved to this point. This may not be the closest point. In this figure, Blk2 is the reference and Blk1 the target. Pin 4 needs to be aligned. If imaginary orthogonal lines are drawn from the reference pin 4 to the target, three possible positions can be identified for pin 4 on the target block. these are marked as A, B and C. The target pin can be moved to any of these positions. Pin aligner chooses position A for the target pin for the best alignment although positions B and C could have helped reduce the net length.



Fixed pins (with status “firm” or “locked”) are not moved by the pin aligner. A message is given in CIW:

```
*Warning* Can not align pin <name> of instance <name>, as it is a <status> pin
```

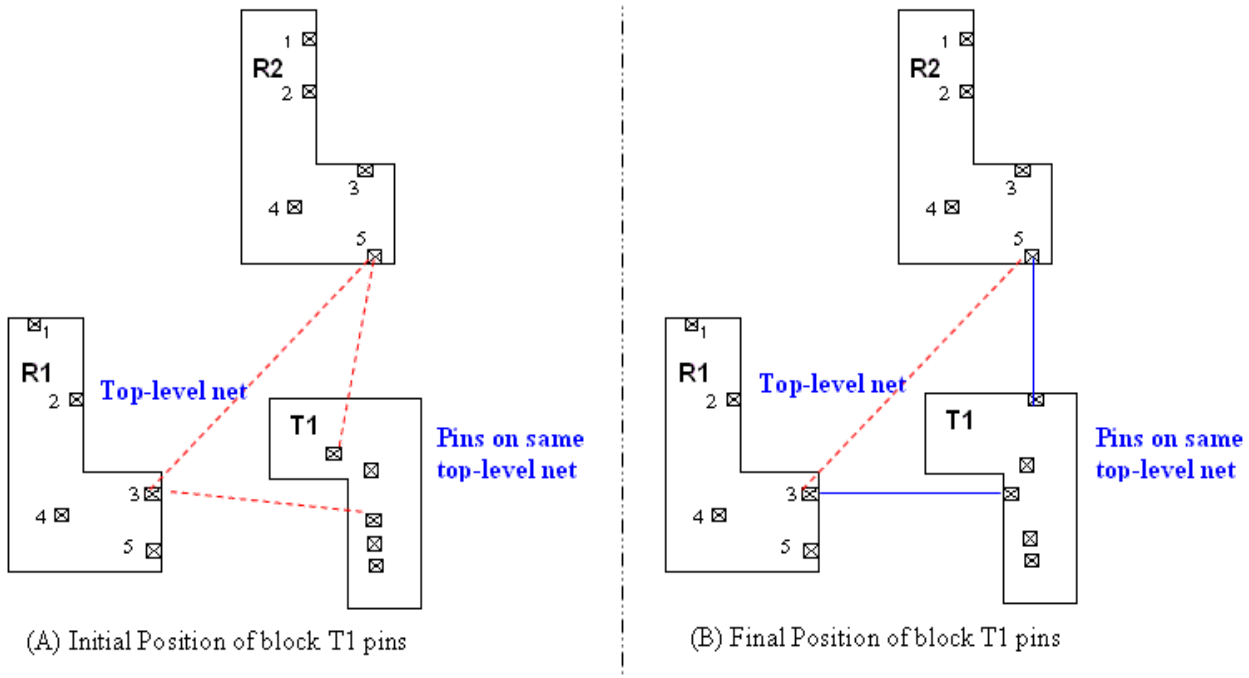
The Pin Alignment command supports multiple reference pins connected to multiple target pins on the same net. It also considers if multiple target pins can be aligned to the same reference pin based on the closest edge criteria.

© 2020 All Rights Reserved.

There are various cases in pin alignment some of which are described as follows. In the figures shown for various cases, the reference and target pins are shown as block pins but they can be top-level pins as well.

- Case I - Reference in Different blocks, Target in the Same Block

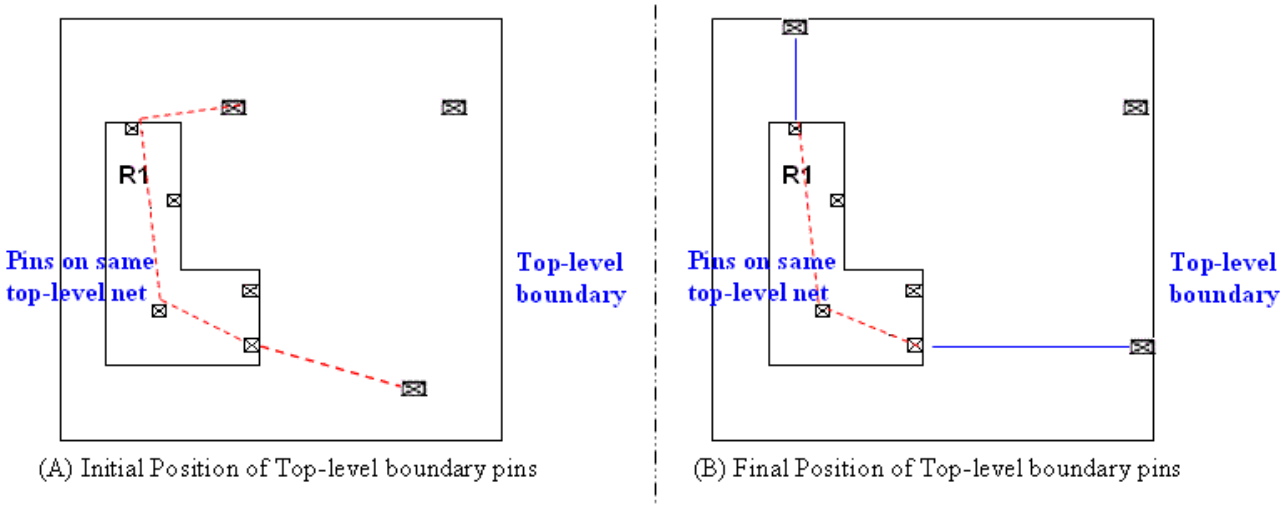
In this case, you select the pins in different blocks as the reference pins, which are connected to multiple pins in the same target block. The target pins are aligned as shown in the following figure.



- Case II - Reference in the Same Block, Target in Same Block

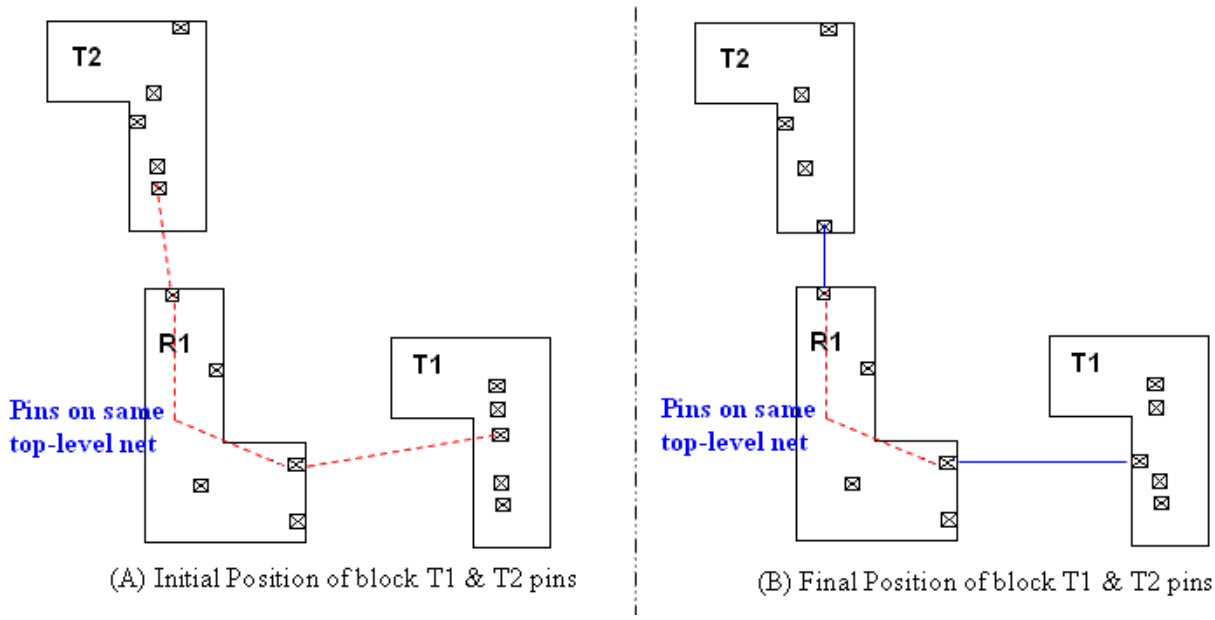
© 2020
All Rights Reserved.

In this case, you select multiple pins in the same block as reference pins, which are connected to multiple top-level pins. The target pins are aligned as shown in the following figure.



Case III - Reference in Same Block, Target in Different Blocks

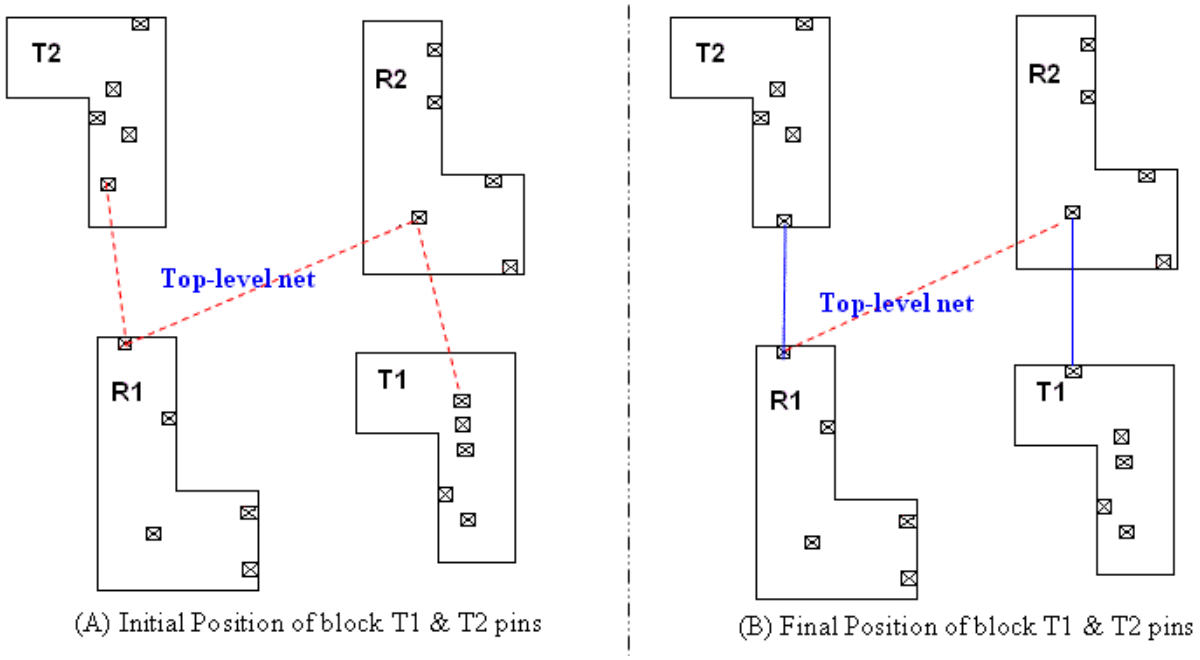
In this case, you select multiple pins in the same block as reference pins, which are connected to multiple pins in different target blocks. The target pins are aligned as shown in the following figure.



© 2020
All Rights Reserved.

Case IV - Reference in Different Blocks, Target in Different Blocks

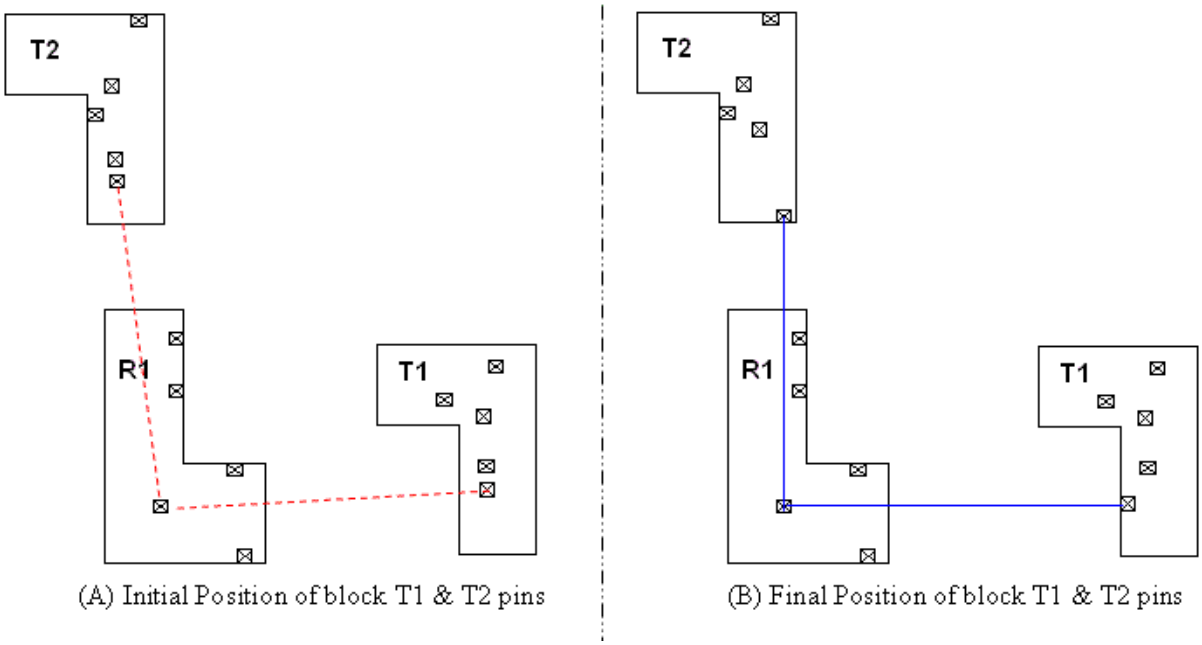
In this case, you select the pins in different blocks as reference pins, which are connected to multiple pins in different target blocks. The target pins are aligned as shown in the following figure.



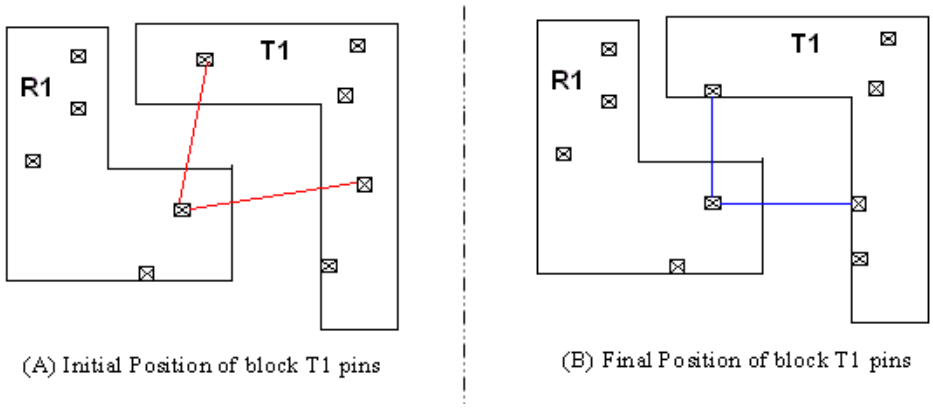
Case V - Multiple Target Pins Aligned with same Reference Pin

© 2020 All Rights Reserved.

In this case, you select a pin in a reference block, which is connected to multiple pins in the same or different target block(s). The target pins in different block(s) are aligned as shown in the following figure.



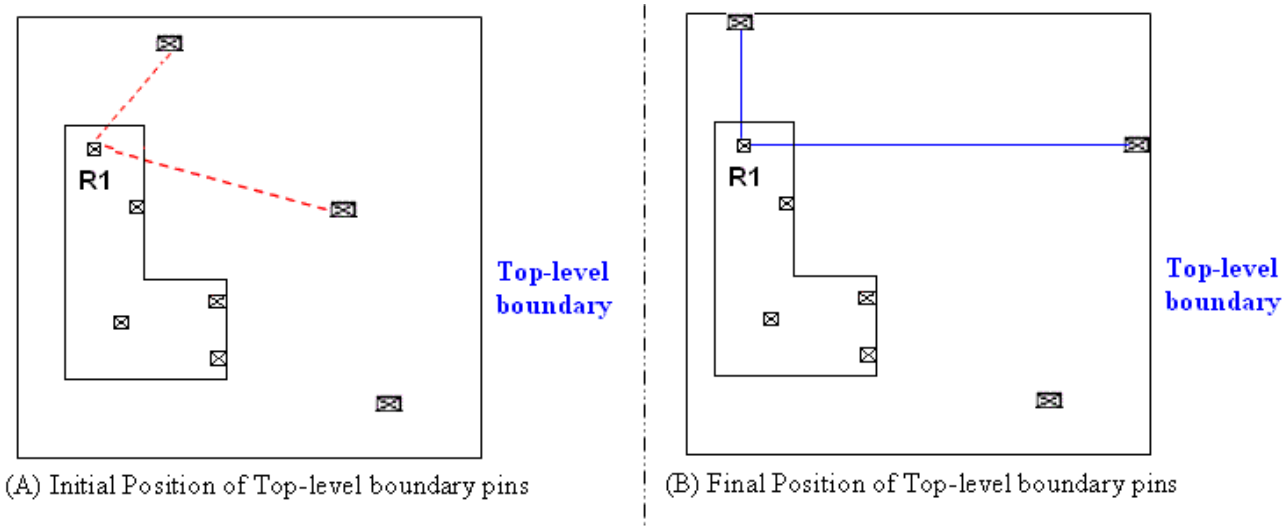
Note: The target pins can belong to same block of different blocks. The following figure shows the multiple pins in the same target block.



Case VI - Multiple Top-level Pins Aligned with same Reference Pin

© 2020
All Rights Reserved.

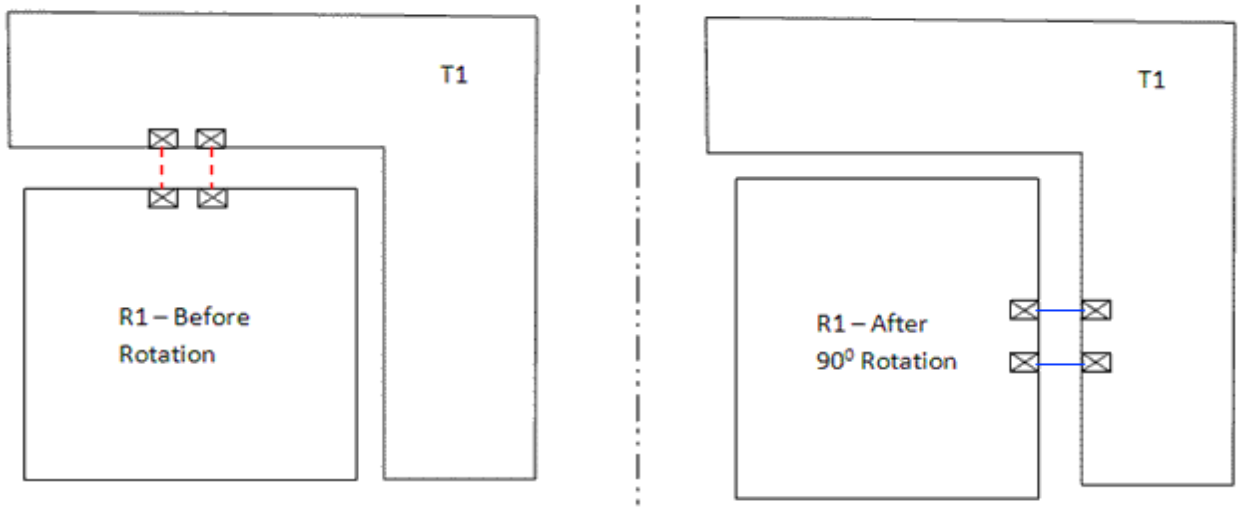
In this case, you select a single pin in the reference block, which is connected to multiple top-level pins in the prBoundary. The pins at the top-level are aligned as shown in the following figure.



(A) Initial Position of Top-level boundary pins (B) Final Position of Top-level boundary pins

Case VII - Alignment of pins when the block having Reference Pins is Rotated

In this case, when the block having reference pins is rotated by 90o, the pin aligner automatically and intelligently aligns the pins in the target block with the pins in the reference block to get the shortest net length.



© 2020 All Rights Reserved.

The pin aligner ignores any selected item that is not a pin, a block or the top level PRBoundary. A message is given in CIW:

```
*Warning* Pin Alignment is not supported for object of type <type>
```

If the target block is not a soft block, its pins are not aligned. A message is given in CIW:

```
*Warning* Can not align pins of instance <name> as it is not a soft block.
```

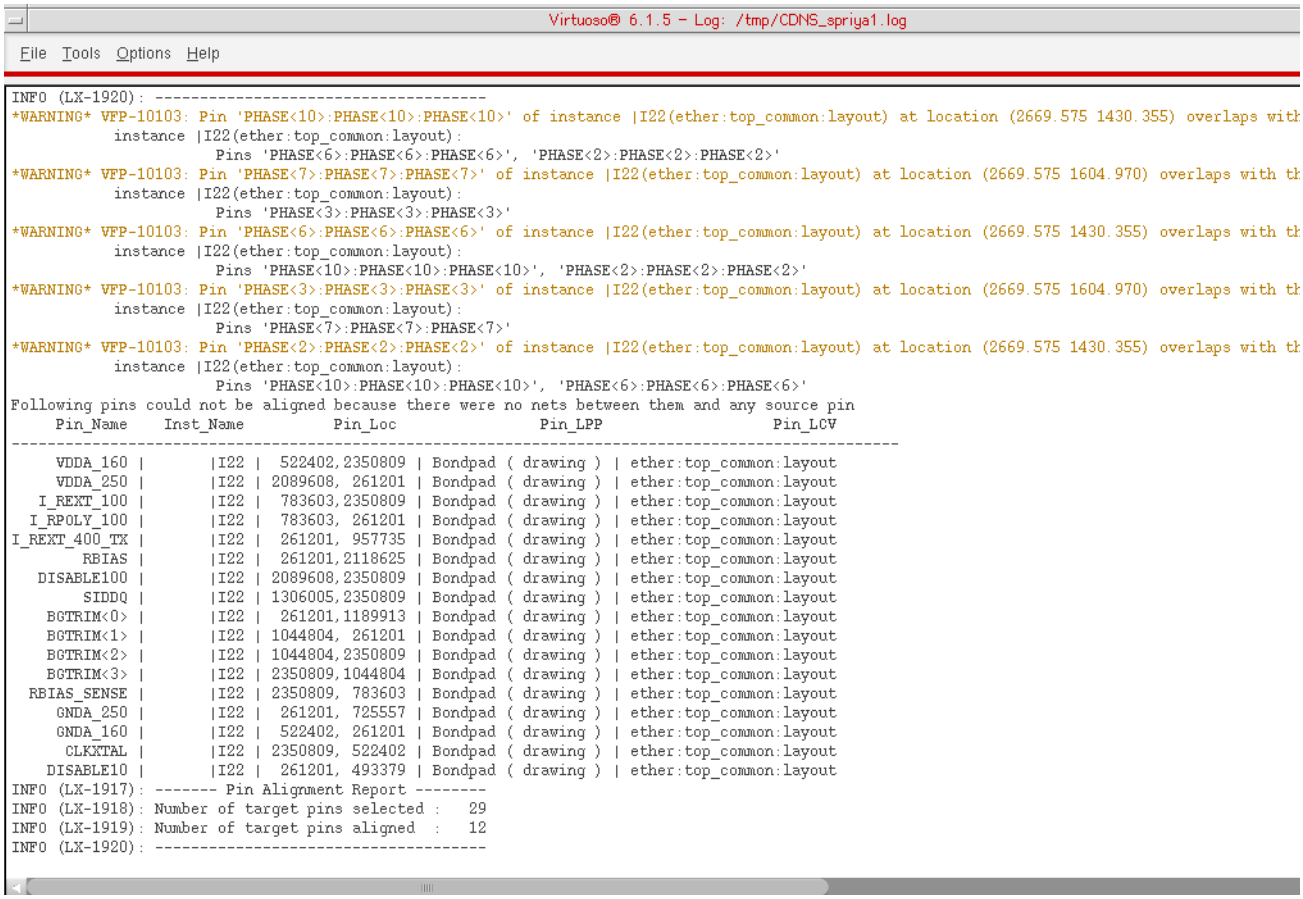
If the target soft block does not have a PRBoundary, its pins are not aligned. A message is given in CIW:

```
*Warning* Can not align pins of instance %s as it does not have PRBoundary.
```

If the block to which the target pin belongs is outside the top-level design boundary, then the pin is aligned but not snapped to the grid. A message is given in CIW:

```
*Warning* Can not snap pin %s of instance %s as it is outside the design boundary.
```

After aligning the pins, a Pin Alignment report is displayed in the CIW. The Pin Alignment report provides a detailed listing of target pins that are not aligned, along with the reason for not aligning them.



© 2020 All Rights Reserved.

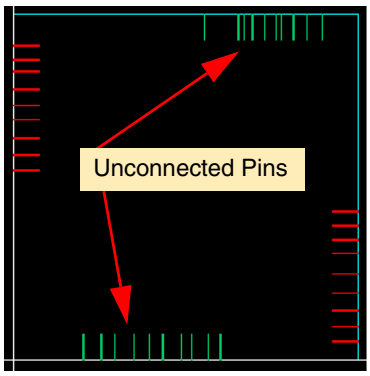
Aligning Unconnected Bus Pins

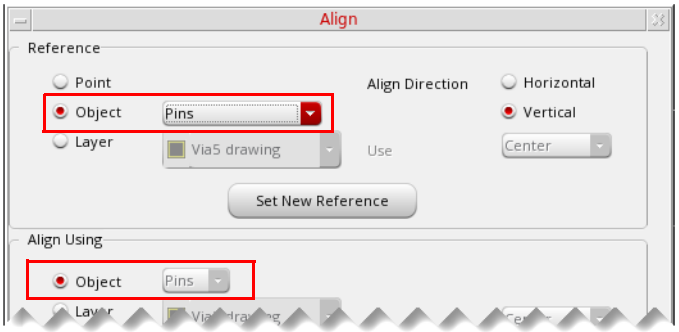
The pin aligner lets you align unconnected bus pins. The selected target bus pins are aligned to the selected reference bus pins.

This feature is not available by default. To enable connectivity-independent alignment of bus pins, set the environment variable ignoreConnectivityBusPinsAlign to `t`.

To align unconnected bus pins:

1. Choose *Edit – Advanced – Align*. The Align form appears.
2. Set *Reference* to *Object* and select *Pins* from the *Object* list.
3. Set *Align Using* to *Object*.
4. Select *Pins* from the *Object* list.

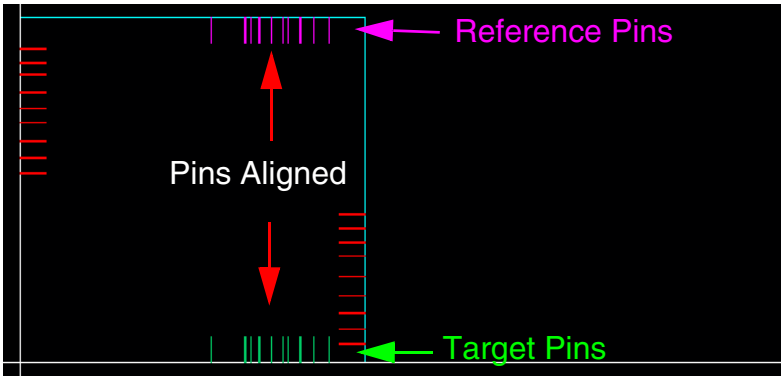




5. Click *Hide* in the Align form.
6. Select the reference bus pins on the design canvas.
7. Click the middle mouse button to switch to the target selection mode.
8. Select the target bus pins.

© 2020
All Rights Reserved.

9. To mark the end of target selection, click the middle mouse button. Alternatively, return to the Align form and click *Apply*.

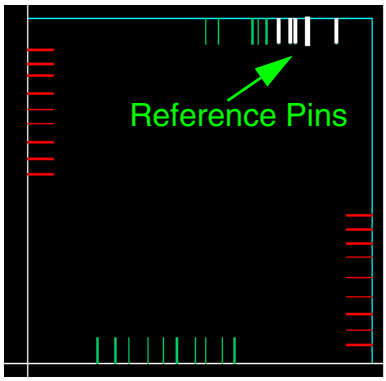


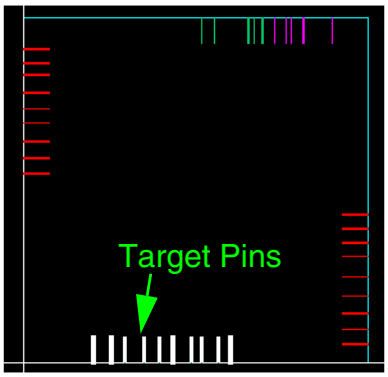
The target pins are aligned to the reference pins.

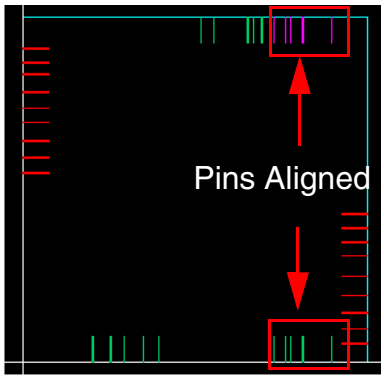
Mapping between the reference and target pins is based on their minimum and maximum bit numbers. For example, if you select the reference as `net1<4:9>` and target as `net2<0:5>`, the pin aligner derives the following mapping and aligns the pins accordingly:

```
net1<4> -> net2<0>
net1<5> -> net2<1>
net1<6> -> net2<2>
net1<7> -> net2<3>
net1<8> -> net2<4>
net1<9> -> net2<5>
```

If the number of selected reference and target bus pins do not match, the pin aligner aligns only those bus pin pairs for which both reference and target pins are selected.







© 2020
All Rights Reserved.

© 2020 All Rights Reserved.

A

Area Estimation Framework

The area estimation framework enables you to register your own area estimators or use previously registered area estimators.

This chapter includes the following topics:

- Register Area Estimators
- Other Utility Functions
- Sample Area Estimation Procedures

You can either directly type the area estimator functions in the CIW or define them in the `.cdsinit` file. For information about updating the `.cdsinit` file, see Modifying the `.cdsinit` File.

Register Area Estimators

Use the `leRegAreaEstimator` SKILL function to register area estimators.

```
leRegAreaEstimator (s_functionSymbol;
                    t_functionAlias;
                    l_legalObjectTypes;
                    [l_parameterNameValuePairs];
                    [t_runMode];
                    );
                    => t / nil
```

Specify the following parameters:

- *`s_functionSymbol`*: Area estimation function name.
- *`t_functionAlias`*: Alias name, which is a display name for the area estimation function. The alias name will be displayed in a cyclic list in the related forms.
- *`l_legalObjectTypes`*: Legal object types for the function, which can be used as filters to list relevant area estimators. Valid Values: `` `cellview ``, `` `cluster ``, `` `group ``, `` `inst ``

© 2020 All Rights Reserved.

- *l_parameterNameValuePairs*: List of parameter name-parameter default value pairs, which might be required internally by the area estimator for area calculation formulae.
- *t_runMode*: The run mode, which specifies the call mechanism. Valid Values: "direct", "deferred".
  - direct: The information required for area estimator to run is present and the area value can be viewed. For example: The input is a schematic cellview ID. The area estimator traverses the entire hierarchy to output an area.
  - deferred: You can set the area estimator and apply the parameters. This information is stored, to be called later when all the information is available. For example: The input is a physical layout cellview ID. Area calculation is deferred until the Generate From Source netlisting step. Once the physical hierarchy is available, area estimation is called.

Important

The sequence of arguments for registering an area estimator function must the same as defined in the area estimator function.

For more information, see leRegAreaEstimator.

Example 1:

Here, the areaEstUsingLayoutPRBoundary area estimator function is registered with alias estAreaUsingLayoutPR, cellView object type, parameter multFactor with default value 10, and the run mode set to direct.

```
leRegAreaEstimator('areaEstUsingLayoutPRBoundary
                  "estAreaUsingLayoutPR"
                  '('cellView)
                  '((multFactor 10))
                  "direct"
                  );
```

Example 2:

Here, the areaEstUsingLayoutBBox area estimator function is registered without any parameter.

```
leRegAreaEstimator('areaEstUsingLayoutBBox
                    "estAreaUsingLayoutBBox"
                    '('cellView)
                    nil
```

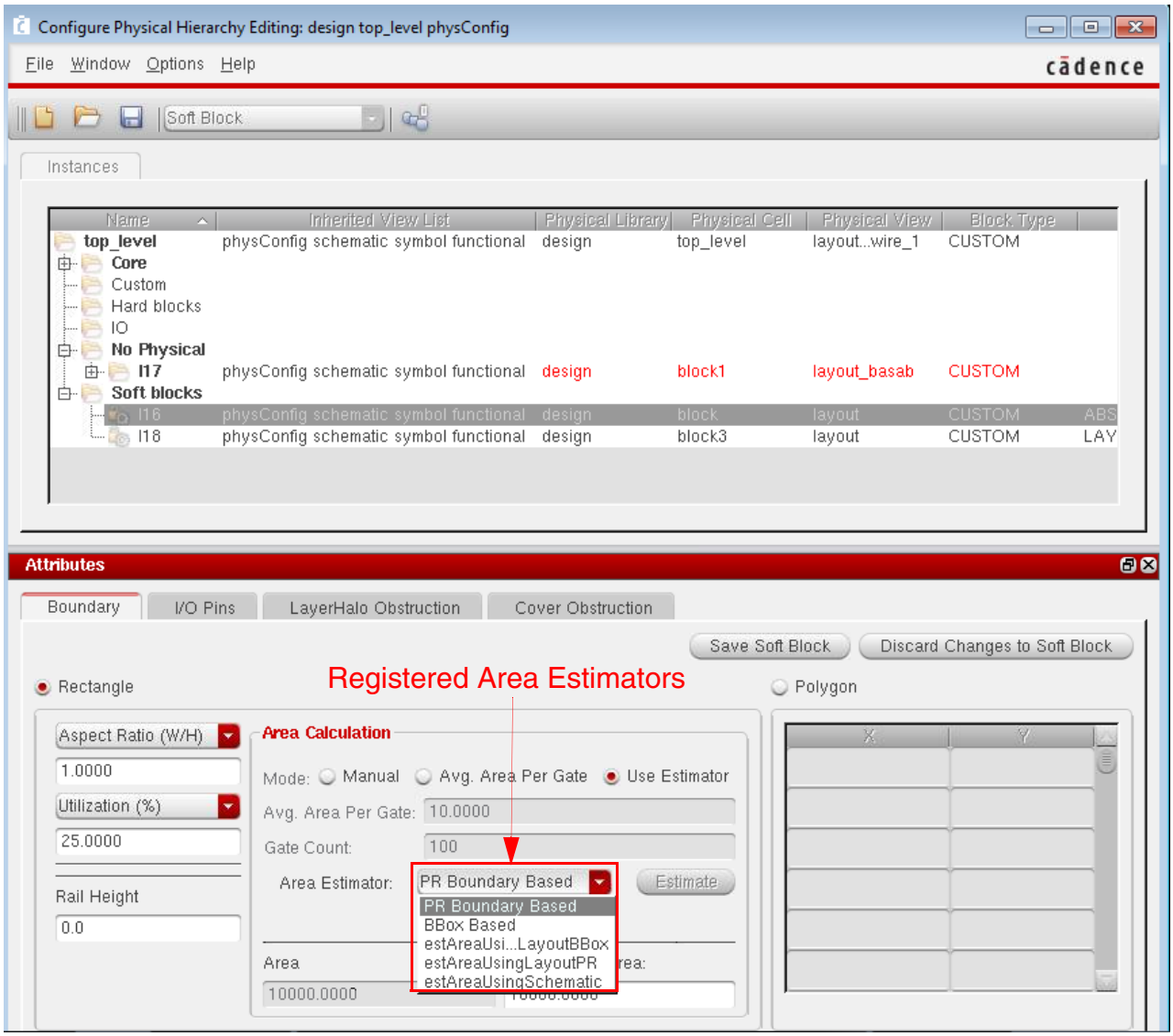

© 2020 All Rights Reserved.

```
                  "direct"
                );
```

You can define area estimation functions in the `.cdsinit` file using the `leRegAreaEstimator` function. When Virtuoso is launched, these area estimation functions are automatically registered.

© 2020 All Rights Reserved.

The registered area estimator registration functions are displayed with their alias names in the *Area Estimator* drop-down list in the Configure Physical Hierarchy form.



© 2020
All Rights Reserved.

For more information about the form, see Configure Physical Hierarchy.

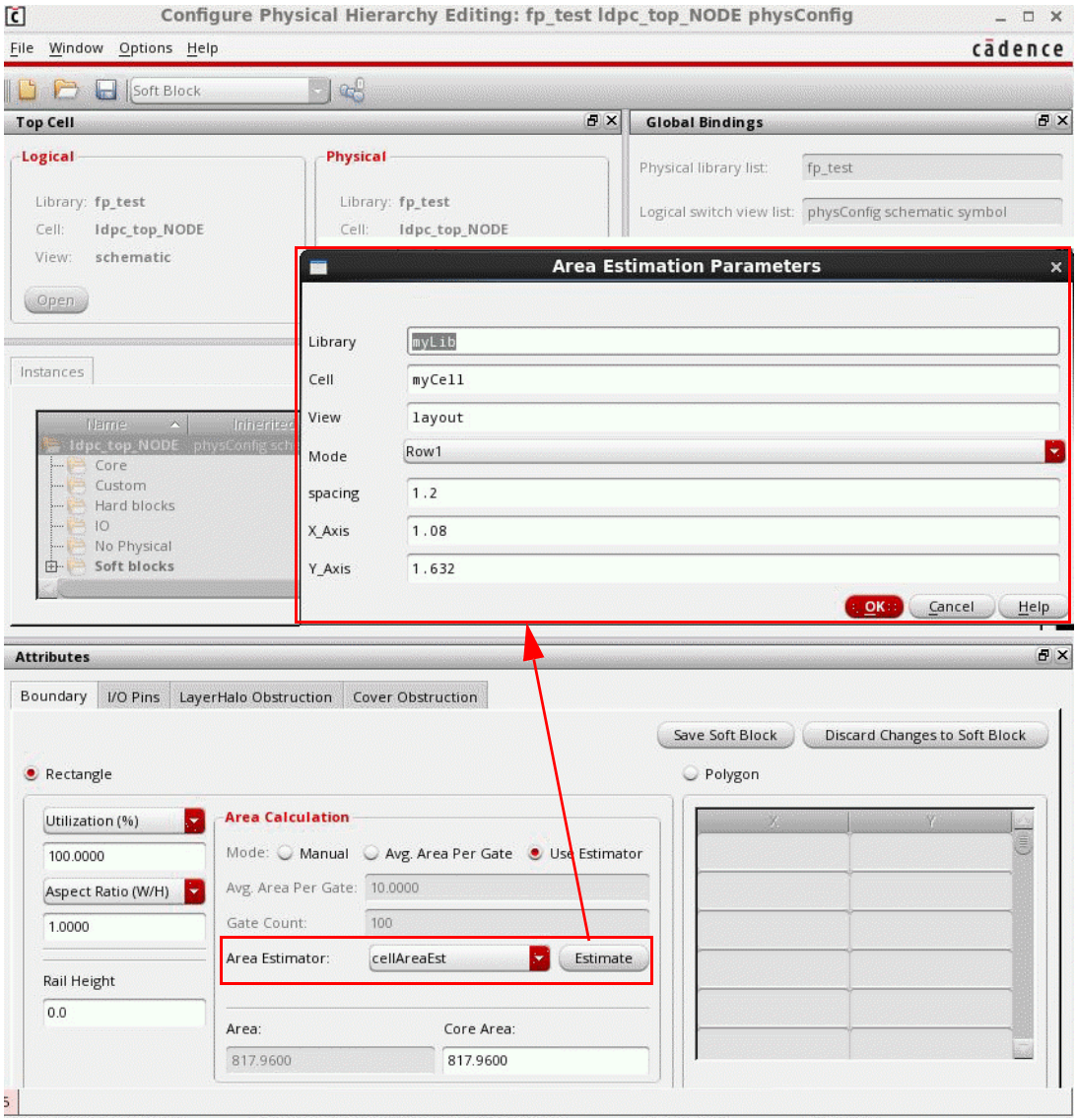
Example 3

In the following example, the `areaEstSchematic` area estimator function is registered with alias `stdCellAreaEst` and the `cellView` object type. Parameters include `Mode`, which is of the type list.

```
leRegAreaEstimator('areaEstSchematic
                    "cellAreaEst"
                    '('cellView)
                    '(
                        (Library     "myLib")
                        (Cell          "myCell")
                        (View        "layout")
                        (Mode         '("Row1" "Row2" "Row3")) // list support
                        (spacing       1.2)
                        (X_Axis        1.08)
                        (Y_Axis        1.632)
                    )
                    "direct")
```


© 2020 All Rights Reserved.

The registered area estimator registration function is displayed in the *Area Estimator* drop-down list in the Configure Physical Hierarchy form. Click *Estimate* to display the Area Estimation Parameters form.



Example 4

In the following example, the `myClusterAreaEstimator` area estimator function is first defined, and then registered with the `cluster` object type.

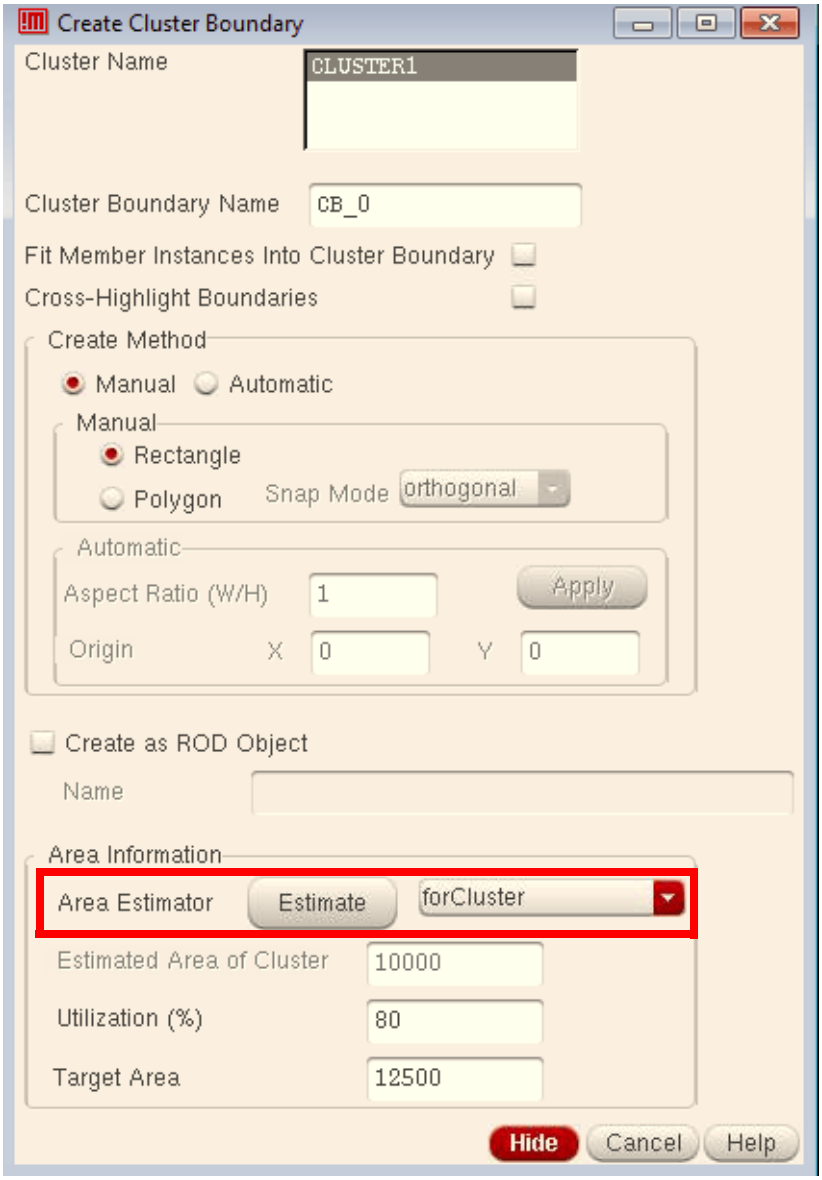
```
procedure(myClusterAreaEstimator(clusterId
         @optional (factor 10.0) (num 4) (text
```


© 2020 All Rights Reserved.

```
            "myFunc"))
          100
          )
leRegAreaEstimator('myClusterAreaEstimator
                    "forCluster"
                    '('cluster)
                    '((factor 10.0) (num 4) (text
                        "myFunc"))
                    "direct"
                    )
```


© 2020 All Rights Reserved.

Area estimator functions that are registered with the cluster object type are displayed in the `Area Estimator` cyclic list in the Create Cluster Boundary form. Use the options in this form dynamically fill the various parameters.



For more information about the form, see Create Cluster Boundary Form.

Format/Signature of Area Estimator

The format / signature of an area estimator should have the following important characteristics:

© 2020 All Rights Reserved.

```
areaEstimatorName ( d_objectId
                    @optional   (param1 defVal1) (param2 defVal2) ..
                    )
                    => areaVal / nil
```

- `d_objectId`: A valid database object ID, such as a cellview, instance, or cluster ID
- `@optional (param1 defVal1)...`: An optional list of name value pairs, where name is the parameter name and value is the default value, such as int, float, text, list, and boolean. On calling the area estimator function, the parameters in the list can be filled dynamically through a simple UI which will be displayed before execution of area estimator.

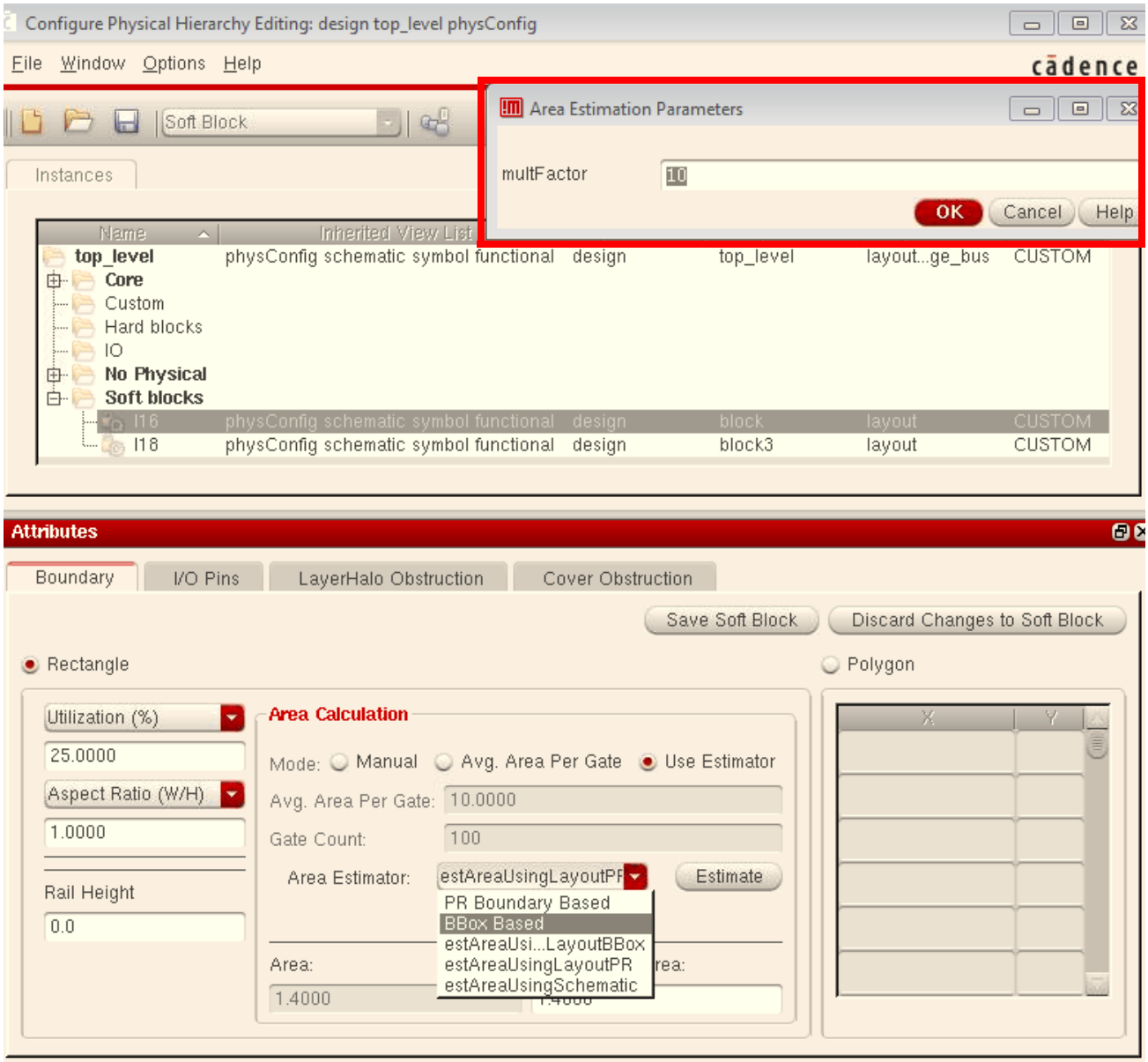
  Example 1:

  Consider the following registered area estimator function, for which the parameter list is defined in the estimator SKILL function and is also registered through `leRegAreaEstimator([l_parameterNameValuePairs];)`:

  ```
  leRegAreaEstimator('areaEstUsingLayoutPRBoundary
                     "estAreaUsingLayoutPR"
                     '('cellView)
                     '((multFactor 10))
                     "direct"
                   );
  => t | nil
  ```


© 2020 All Rights Reserved.

When this area estimator function is called and the *Estimate* button is clicked in the Configure Physical Hierarchy form, a UI is displayed, where the parameters can be entered dynamically.



The return value is either a floating number or `nil`.

© 2020 All Rights Reserved.

Example 2:

Consider the following area estimator function defined as follows:

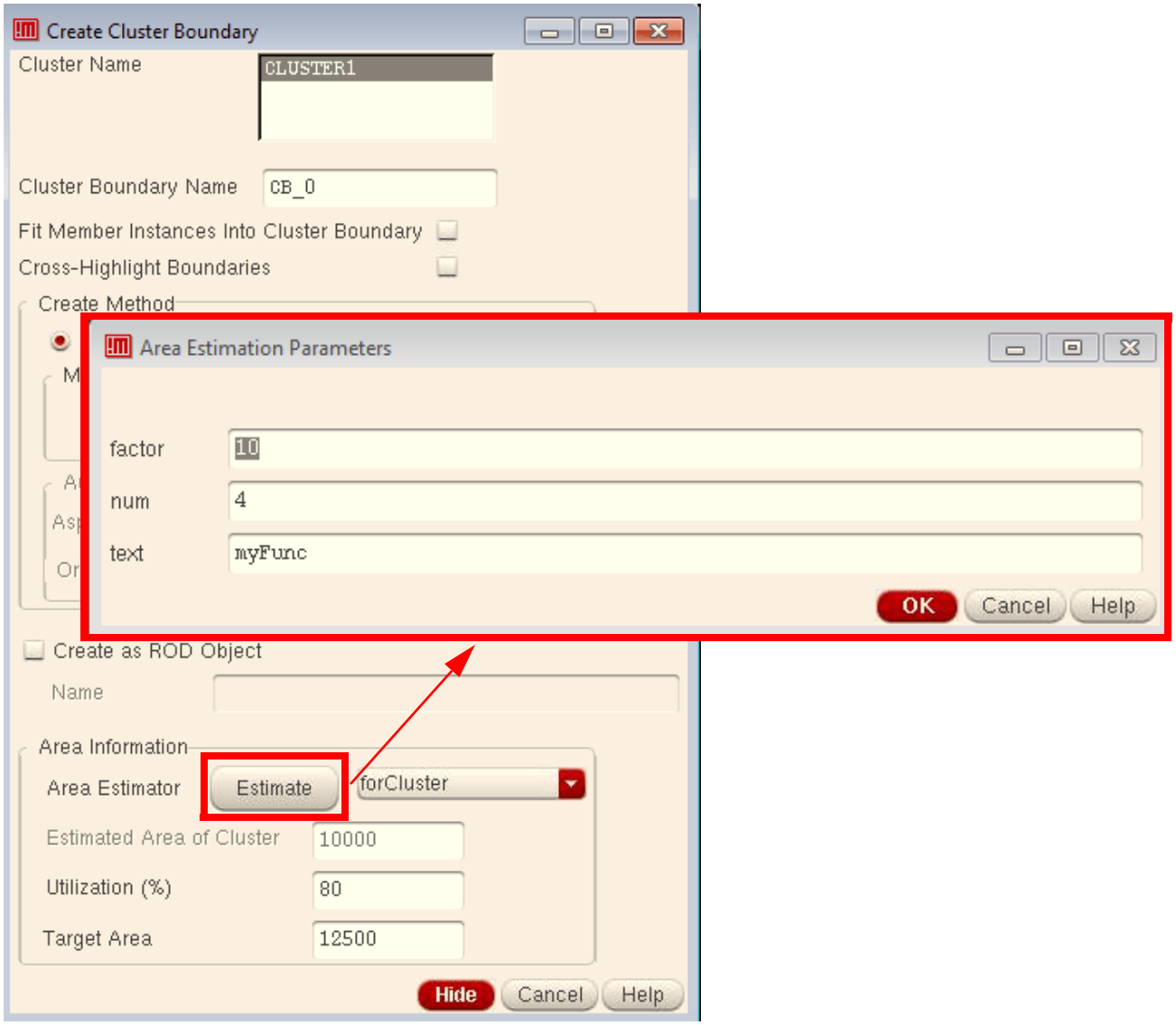
```
procedure(myClusterAreaEstimator(clusterId
                                 @optional (factor 10.0) (num 4)(text
                                    "myFunc"))
                                 100
                                )
```

The function is registered using:

```
leRegAreaEstimator('myClusterAreaEstimator
                   "forCluster"
                   '('cluster)
                   '((factor 10.0) (num 4) (text "myFunc"))
                   "direct"
                   )
```


© 2020 All Rights Reserved.

After registering the function, click the *Estimate* button in the Create Cluster Boundary form. A UI is displayed, where the parameters can be entered dynamically.



© 2020
All Rights Reserved.

Other Utility Functions

Use the `leUnRegAreaEstimator` function to unregister area estimation functions for one or more object types. For information about the syntax and arguments, see `leUnRegAreaEstimator`.

Example

To unregister an area estimation function with `"simpleAreaEst"` alias name, which appears in the *Area Estimator* drop-down list for the `cluster` object type, use:

```
leUnRegAliasEstimator( "simpleAreaEst" ('cluster) )
```

Displaying Registered Area Estimators

leGetAreaEstimationDisplayNames

Use the `leGetAreaEstimationDisplayNames` function to retrieve the area estimation function alias name list defined for the specified object types. For information about the syntax and arguments, see `leGetAreaEstimationDisplayNames`.

Example

To retrieve the alias name list for area estimation functions with the `cluster` object type, use:

```
funcCallbackSymbol = leGetAreaEstimationDisplayNames( ('cluster) )
```

leGetAreaEstimatorFunction

Use the `leGetAreaEstimatorFunction` function to retrieve the function symbol of the area estimation function registered for the specified alias name and object type. For information about the syntax and arguments, see `leGetAreaEstimatorFunction`.

Example

```
funcCallbackSymbol = leGetAreaEstimatorFunction( "simpleAreaEst" ('cluster) )
```

leGetAreaEstimatorParamList

Use the `leGetAreaEstimatorParamList` function to retrieve the parameters of the area estimation function registered for the specified alias name and object type. For information about the syntax and arguments, see `leGetAreaEstimatorParamList`.

Example

© 2020 All Rights Reserved.

```
funcParam List = leGetAreaEstimatorParamList( "simpleAreaEst" ("cluster") )
```

leGetAreaEstimatorRunMode

Use the `leGetAreaEstimatorRunMode` function to retrieve the Run Mode list of the area estimation function registered for alias name and object types.

Example

```
funcParam List = leGetAreaEstimatorParamList( "simpleAreaEst" ("cluster") )
```

The following SKILL script fragment uses the `leGetAreaEstimatorFunction`, `leGetAreaEstimationDisplayNames`, `leGetAreaEstimatorFunction`, `leGetAreaEstimatorParamListz`, and `leGetAreaEstimatorRunMode` functions.

```
foreach( d leGetAreaEstimationDisplayNames('('cellView))
         printf("Display Name = \"%s\",
                 CallBack Name = \"%s\",
                 Parameter List = %L,
                 Run Mode = \"%s\"\n"
         leGetAreaEstimatorFunction(d '('cellView))
         leGetAreaEstimatorParamList(d '('cellView))
         leGetAreaEstimatorRunMode(d '('cellView)))
        )
```


© 2020 All Rights Reserved.

Sample Area Estimation Procedures

Procedure 1:

```
procedure(areaEstUsingLayoutPRBoundary(
             cv
             @optional (multFactor 10.0) (pinArea 0.14) (numPins 10))
             let( ((area 0.0) ele master)
             foreach( ele cv~>instances
             master = ele~>master
             if(master~>prBoundary then
                area = area + master~>prBoundary~>area
             else
                area = area + abs((caadr(master~>bBox) - caar(master~>bBox))
                * (cadadr(master~>bBox) - cadar(master~>bBox)))
                )
             )
             area = area * abs(multFactor) + numPins * pinArea
             )
         )
```

Procedure 2:

```
procedure(areaEstUsingLayoutBBox(cv)
         let( ((area 0.0) ele master)
         foreach( ele cv~>instances
         master = ele~>master
         area = area + abs((caadr(master~>bBox) - caar(master~>bBox))
         * (cadadr(master~>bBox) - cadar(master~>bBox)))
         )

         area
         )
         )
```


© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

B

Block Override Support

Floorplanning engine supports overridden terminals and instance terminals in block domain. The Floorplanning commands do not create overriding terminals or instance terminals. However these overridden terminals and instance terminals can be present in OpenAccess data coming from Innovus™ Digital Implementation System (Innovus) or any other application using OA. Therefore, the support is required for interoperability between Innovus and Virtuoso.

The behavior of commands in Floorplanning is given as follows:

- The commands processing pins and pin figures on overridden terminals and instance terminals, of master instance having pins and pin figures, should consider overridden terminal and instance terminal as a normal block domain terminal or instance terminal, without any difference in the behavior. For example, the Pin Optimize command should optimize pins on overridden terminal and instance terminal.
- The commands involving change of sigType should be careful that nets connected to overriding terminals and instance terminals can only be of sigType “power”, “ground”, "tieHi", "tieLo", and "tieOff”.
- The commands processing (deleting) empty nets should be careful because a net may be empty only in the block domain and may have a terminal/instance terminal in module domain. Therefore, the net is not empty as corresponding terminals and instance terminals are hidden in the block domain.

Commands related to terminals and instance terminals are as follows:

- Pin Optimize
- Pin Align
- Push Into Blocks
- Load Physical View
- Create Soft Pins
- Edit Soft Blocks

© 2020 All Rights Reserved.

- Set Pin Connectivity
- Pin Snap

Concept of Overriding Terminals and Instance Terminals

The “tieHigh”, “tieLow”, “tieOff”, “power”, “ground”, and “signal” nets specified in verilog can be implemented in block domain using any physical only net. The example for this is given as follows:

```
/ Verilog file for cell "top_level" view "layout"
// Language Version: 2001
module top_level (
    A,
    B,
    C,
    D,
    Z
    );
    input A;
    input B;
    input C;
    input D;
    input _IN;
    output Z;
    output _OUT;
    wire net1;
    assign _OUT = 1’b1;
    assign _IN = 1’b0;
    block3 I1 ( .A(A), .B(B), .C(C), .Y(net1), .Z(1’b1));
    block I2 ( .A(net1), .B(1’b1), .C(D), .Z(Z), .„nd! (1’b0), .\vdd! (1’b1));
endmodule // top_level
```

In the mentioned example, observe the following in the connectivity:

- The terminals _IN and _OUT are connected to tie0 and tie1nets, respectively.
- The instance terminals corresponding to I1/Z and I2/B are connected to tie.
- The instance terminals corresponding to I2/vdd! and I2/gnd! are connected to tie1 and tie0 nets, respectively

© 2020 All Rights Reserved.

It is significant to understand that verilog designer has no clue how 1’b1 (tieHi) and 1’b0 (tieLow) nets will be implemented in the layout or block domain. It depends on the block designer to choose any physical only net to implement 1’b1 and 1’b0. For example, the following array of nets can be used to implement.

- _IN and I2/gnd! (tie0) to be implemented by net VSS (physical only).
- _OUT and I2/vdd! (tie1) to be implemented by net VDD (physical only).
- I1/Z and I2/B (tie1) to be implemented by net INTERNALVDD (physical only).

The following changes in the block domain will have to be made.

- Hide _IN _OUT terminals.
- Hide instance terminals I2/gnd!, I2/vdd!, I1/Z and I2/B.
- Create overriding terminals and overriding instance terminals with same names to implement them using the above mentioned physical only nets.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

C

Floorplan Property File

You can define the soft block properties, such as boundary, pins, and halo blockage for a selected instance on the *Instances* tab in Configure Physical Hierarchy form. These properties are used by the Generate Physical Hierarchy command to create soft blocks for the instances in the physical hierarchy. The properties defined for a soft block are retained in one session. Once, the Virtuoso session is closed, the defined properties are erased from the memory. In such cases, Floorplan property file is used for the persistent storage of the soft block properties, which can be reloaded when you work with a new Virtuoso session.

You can save the soft block properties in a property file by selecting *File -> Save Floorplan File* in the Configure Physical Hierarchy form.

The syntax of the Floorplan property file is as follows:

```
Top-Level
(
Master properties.....
          Boundary tab properties....
               (Terminal properties.............
                    Term props1....
                    Term props2....
                    ...........
                    Term propsN....
               )
               (Halo properties............)
)
```


© 2020 All Rights Reserved.

The following figure displays a sample of the Floorplan property file

Sample Floorplan Property File

sample_fp.txt - Notepad

File Edit Format View Help

```
DESIGN libName "design" cellName "top_level" viewName "schematic" versionName "" timeStamp "Aug 20 11:50:04 2009"

MASTER PROP

libName "design" cellName "block1" viewName "layout" schLibName "design" schCellName "block1"
schViewName "schematic" inhViewList "physConfig schematic symbol" inhStopViewList "layout abstract"
stopString "layout abstract" hierPath "/I17" versionName "" blockType "custom" symAbst nil block t
abgen t gateCount 100 avgAreaPerGate 10.0 areaMode "manual" polygonList nil areaCallback nil area nil
areaUtil nil aspRatio nil boundaryHeight 150.6 boundaryWidth 150.6 cyclic2 4 cyclic1 3 boundaryPolyType nil

termProps ((nil termCreate t netName "A"
    termPurpose "drawing" addedPin nil termDirection
    "input" termSigType "signal" termCriticality 10
    termNum 1 termHeight 20.1 termWidth
    20.1 termLayer "metal1" termName "A"
    )
    (nil termCreate t netName "B"
    termPurpose "drawing" addedPin nil termDirection
    "input" termSigType "signal" termCriticality 10
    termNum 1 termHeight 20.1 termWidth
    20.1 termLayer "metal1" termName "B"
    )
    (nil termCreate t netName "C"
    termPurpose "drawing" addedPin nil termDirection
    "input" termSigType "signal" termCriticality 10
    termNum 1 termHeight 20.1 termWidth
    20.1 termLayer "metal1" termName "C"
    )
    (nil termCreate t netName "gnd!"
    termPurpose "drawing" addedPin nil termDirection
    "inputOutput" termSigType "signal" termCriticality 0
    termNum 1 termHeight 0.6 termWidth
    0.6 termLayer "ndiff" termName "gnd!"
    )
    (nil termCreate t netName "vdd!"
    termPurpose "drawing" addedPin nil termDirection
    "inputOutput" termSigType "signal" termCriticality 0
    termNum 1 termHeight 0.6 termWidth
    0.6 termLayer "ndiff" termName "vdd!"
    )
    (nil termCreate t netName "Z"
    termPurpose "drawing" addedPin nil termDirection
    "output" termSigType "signal" termCriticality 10
    termNum 1 termHeight 20.1 termWidth
    20.1 termLayer "metal1" termName "Z"
    )
) haloProps ((nil haloLayer nil haloBlockageType "placement"
    haloLeftOffset 0.0 haloRightOffset 0.0 haloBottomOffset
    0.0 haloTopOffset 0.0 haloType "placement"
    )
) ioPinCreateLabel nil generateSoftBlock t
```

The various attributes of property file are defined in the following sections.

- Master Properties
- Terminal Properties

© 2020 All Rights Reserved.

- Halo Properties

Master Properties

The master properties are defined in the CPH form when you select the soft block option. The properties that are defined are as displayed in the following figure.

```
MASTER PROP

libName "design" cellName "block1" viewName "layout" schLibName "design" schCellName "block1"
schViewName "schematic" inhViewList "physConfig schematic symbol" inhStopViewList "layout abstract"
stopString "layout abstract" hierPath "/I17" versionName "" blockType "custom" symAbst nil block t
abgen t gateCount 100 avgAreaPerGate 10.0 areaMode "manual" polygonList nil areaCallback nil area nil
areaUtil nil aspRatio nil boundaryHeight 150.6 boundaryWidth 150.6 cyclic2 4 cyclic1 3 boundaryPolyType nil
```

Boundary tab Properties

The Master properties are defined as follows:

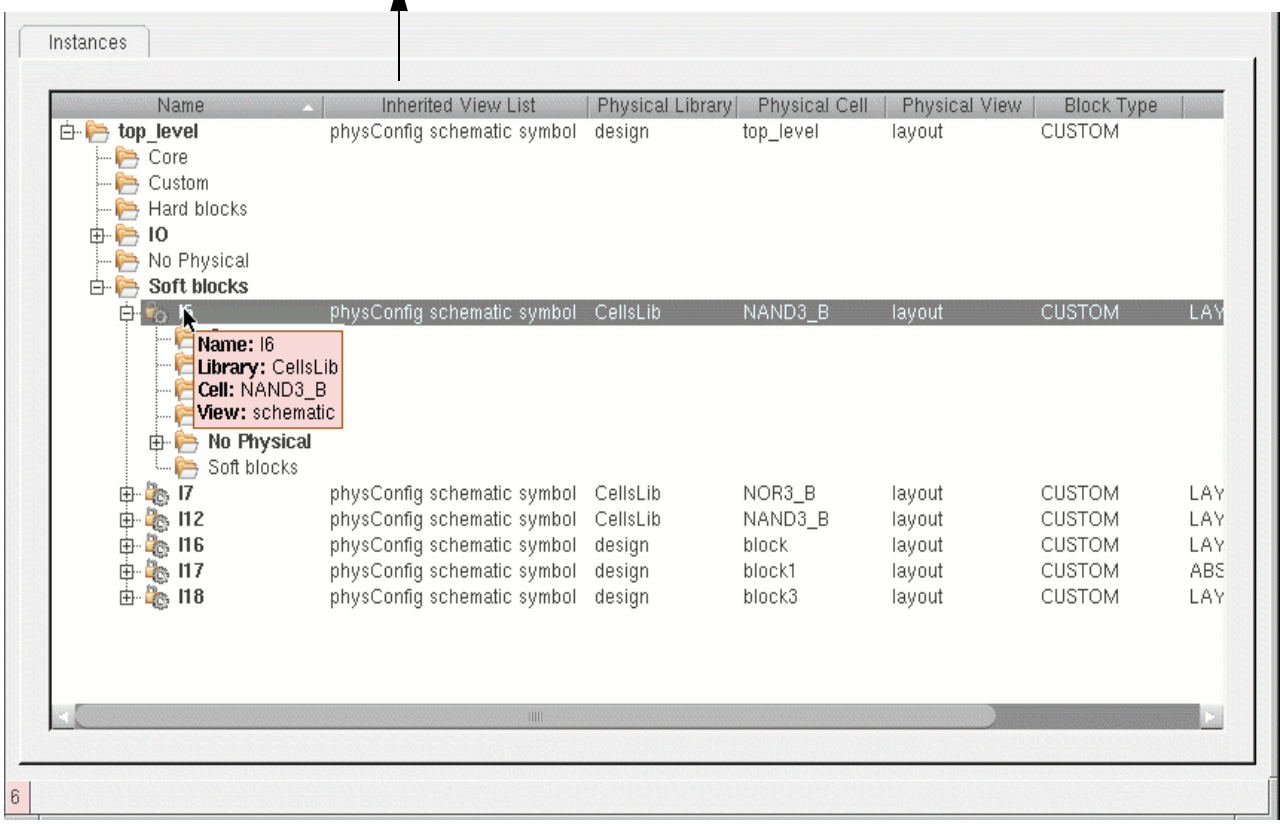
- Binding Properties
- Global Variables for Soft Block Generation
- Boundary Tab Properties

© 2020 All Rights Reserved.

Binding Properties

The binding properties provide the binding information for the soft block. The binding properties are defined in the CPH form as shown in the following figure.

Binding properties in CPH form



The following table lists the binding properties along with their description.

Property Name	GUI Option Name	Description
`libName`	Physical Library	Name of the layout library to be generated
`cellName`	Physical Cell	Name of the layout cell to be generated
`viewName`	Physical View	Name of the layout view to be generated
`schLibName`	Library (tooltip)	Name of the schematic library used to read properties.
`schCellName`	Cell (tooltip)	Name of the schematic cell used to read properties

© 2020
All Rights Reserved.

`schViewName`	View (tooltip)	Name of the schematic view used to read properties
`inhViewList`	Inherited View List	Specifies the design views that will be used to descend into the hierarchical design to find stop views.
`inhStopViewList`	Inherited Stop List	While descending the hierarchical design it stops if it finds any of the views, such as layout or abstract. The view list then populates the CPH form with its properties.
`stopString`	NA	These views are searched for IO pad cells to create a pin list from the main list, which is filtered. **Note:** This property is kept for legacy reasons to support commands for backward compatibility in Virtuoso and will be deprecated in the next release.
`hierPath`	Name (the complete hierarchy path)	This is the complete hier path of an instance with respect to the top-level cellview.
`versionName`	NA	The default value for this is an empty string (" "). **Note:** This property is kept for legacy reasons to support commands for backward compatibility in Virtuoso and will be deprecated in the next release.
`blockType`	Block Type	Describes the type of block, whether "CUSTOM" or "DIGITAL".

Note: The three properties `symAbst`, `block`, and `abgen` are boolean properties, which control the soft abstract generation of the block. If `symAbst` is set to `true` and `block` and `abgen` are set to `false`, then a soft abstract is created. However, if `symAbst` is set to `false` and `block` and `abgen` are set to `true`, then a soft abstract is not created.

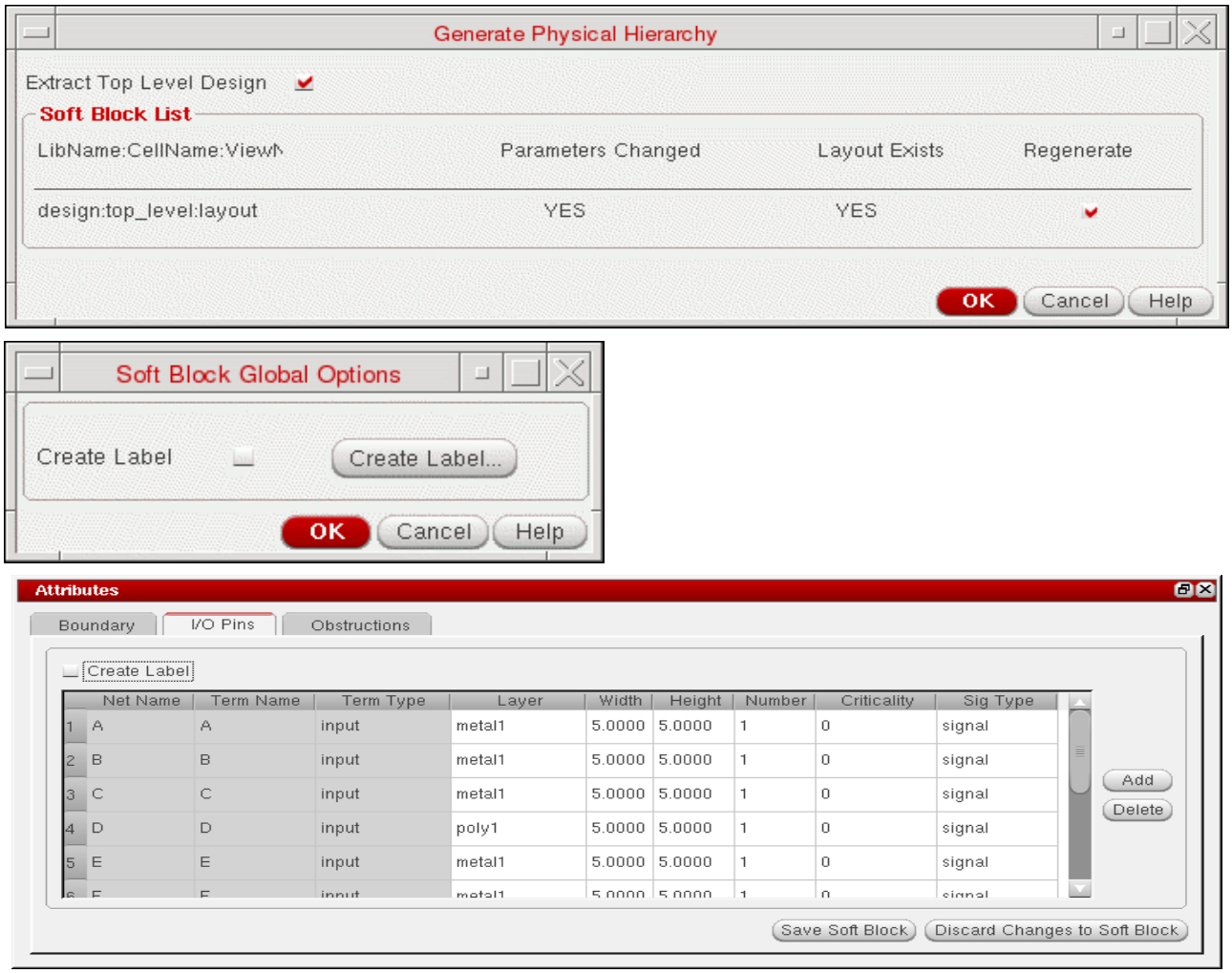
`symAbst`	Type	Specifies whether or not we have to create a soft abstract of the block. This property is mostly specified in cases where the instance in schematic is symbol.

© 2020 All Rights Reserved.

`block`	Type	If this property is specified as NIL, then it creates a soft abstract. **Note:** This property would be deprecated in the next release.
`abgen`	Type	If this property is specified as NIL, then it creates a soft abstract. **Note:** This property would be deprecated in the next release.

Global Variables for Soft Block Generation

The global variables for soft block generation are defined as shown in the following figure:



© 2020
All Rights Reserved.

The following table lists the global variables that are to be defined for soft block generation.

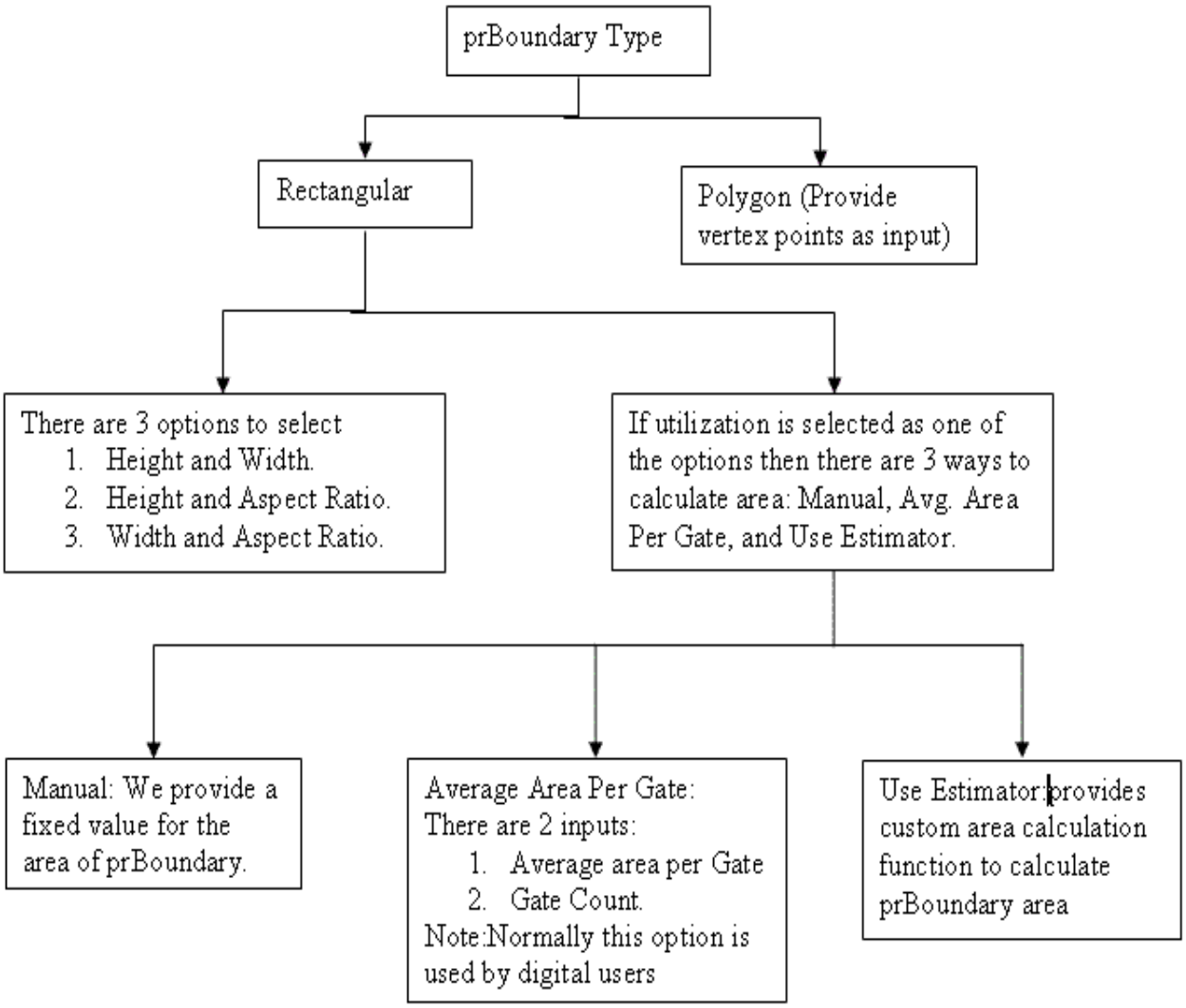
Property Name	GUI Option Name	Description
`ioPinCreateLabel`	Create Label	This is a boolean value. If set to true, it creates labels along with the pins.
`generateSoftBlock`	Regenerate	This is a boolean value. If set to true, it generates/regenerates the soft block with the defined set of properties.

Boundary Tab Properties

The boundary tab properties are defined to customize the prBoundary created in the layout. For more information, refer to Figure 1.

© 2020 All Rights Reserved.

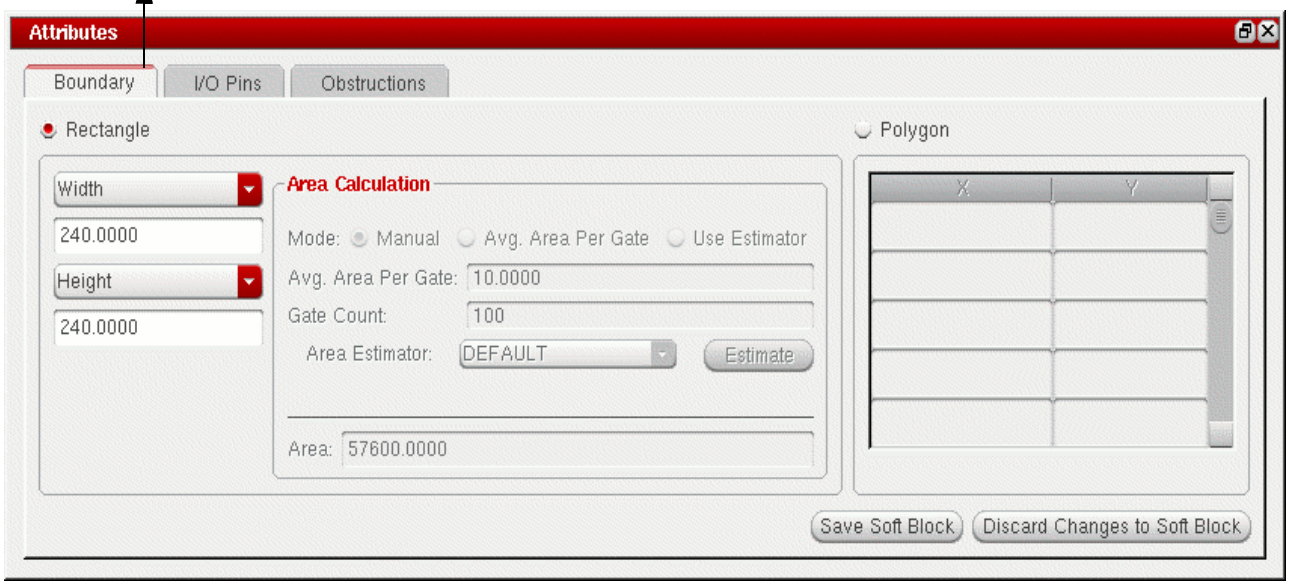
Figure 1



© 2020 All Rights Reserved.

The boundary tab properties are defined in the Boundary tabbed page of the CPH form as shown in the following figure.

Boundary Properties in CPH form



The following table lists the boundary properties along with their description.

Property Name	GUI Option Name	Description
`boundaryPolyType`	radio buttons in the Boundary tabbed page	Select the type of boundary to be created. The boundary type can be '*rectangular*' or '*polygon*'.
`cyclic1`	combo box to select among the various options	This property allows you to specify one of the area calculation mode. The four options in this property are: Height, Width, Area Utilization, and Aspect ratio. Each option has a value associated with it, which ranges from 1 to 4, where 1 represents Area Utilization, 2 is for Aspect Ratio, 3 for Width, and 4 is for Height.

© 2020
All Rights Reserved.

`cyclic2`	combo box to select among the various options	This property allows you to specify one of the area calculation mode. The four options in this property are: Height, Width, Area Utilization, and Aspect ratio. Each option has a value associated with it, which ranges from 1 to 4, where 1 represents Area Utilization, 2 is for Aspect Ratio, 3 for Width, and 4 is for Height.
`boundaryHeight`	value of boundary height in any cyclic field	Specifies the height of the prBoundary.
`boundaryWidth`	value of boundary width in any cyclic field	Specifies the width of the prBoundary.
`areaUtil`	value of area utilization in any cyclic field	Defines the value of area utilization. For example, if utilization is 25%, then accordingly it will calculate the prBoundary area.
`aspRatio`	value of aspect ratio in any cyclic field	This is the ratio of width/height values. For example, if aspect ratio is 2, then Width = 2*Height.
`areaMode`	Mode (‘Manual’, ‘Avg Area Per Gate’, Use Estimator)	Describes the area calculation mode, whether Manual, areaPerGate, useEstimator. By default, the area mode is selected as Manual. This is populated in the Boundary tab of the CPH form. For more information, refer to Figure 1.
`area`	Area field	This property is used to provide the fixed value of prBoundary area.
`gatecount`	Gate Count field	This option is mostly used by digital users. It specifies the number of gates. This is in unison with avgAreaPerGate.
`avgAreaPerGate`	Avg. Area Per Gate field	This option is usually used by digital users. You can specify the value for gate area. This is in unison with gatecount.
`areaCallBack`	Area Estimator cyclic field	The name of the area estimator function, which is used to calculate prBoundary. For more information on area estimator, refer to the Virtuoso Layout Suite XL User Guide.

© 2020 All Rights Reserved.

`polygonList`	X and Y coordinate columns	The list of polygon points for prBoundary. This is populated in the Boundary tab of the CPH form.

© 2020 All Rights Reserved.

Terminal Properties

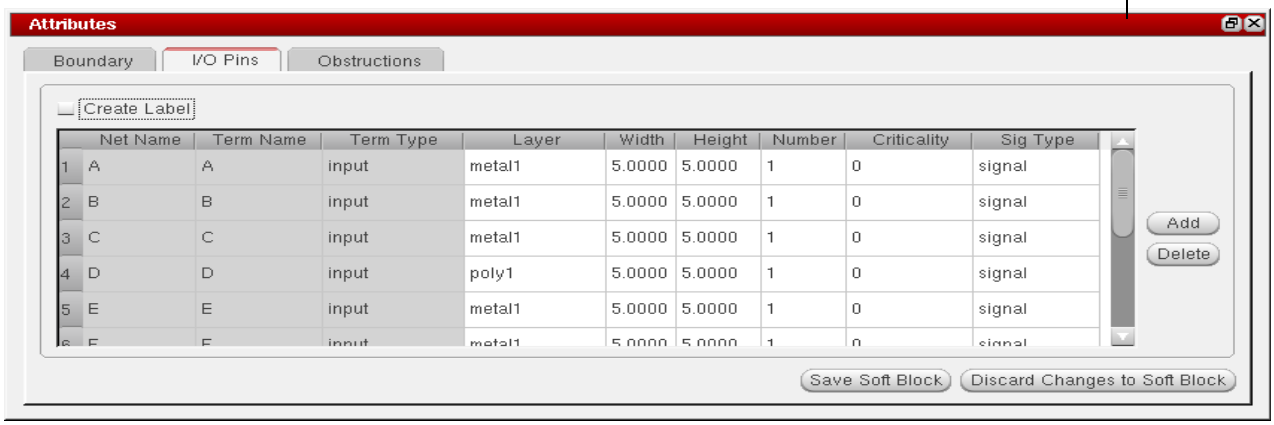
The terminal properties describes the properties that are to be defined for a terminal. The properties that are defined are as displayed in the following diagram.

```
termProps ((nil termCreate t netName "A"
    termPurpose "drawing" addedPin nil termDirection
    "input" termSigType "signal" termCriticality 10
    termNum 1 termHeight 20.1 termWidth
    20.1 termLayer "metal1" termName "A"
    )
    (nil termCreate t netName "B"
    termPurpose "drawing" addedPin nil termDirection
    "input" termSigType "signal" termCriticality 10
    termNum 1 termHeight 20.1 termWidth
    20.1 termLayer "metal1" termName "B"
    )
    (nil termCreate t netName "C"
    termPurpose "drawing" addedPin nil termDirection
    "input" termSigType "signal" termCriticality 10
    termNum 1 termHeight 20.1 termWidth
    20.1 termLayer "metal1" termName "C"
    )
    (nil termCreate t netName "gnd!"
    termPurpose "drawing" addedPin nil termDirection
    "inputOutput" termSigType "signal" termCriticality 0
    termNum 1 termHeight 0.6 termWidth
    0.6 termLayer "ndiff" termName "gnd!"
    )
    (nil termCreate t netName "vdd!"
    termPurpose "drawing" addedPin nil termDirection
    "inputOutput" termSigType "signal" termCriticality 0
    termNum 1 termHeight 0.6 termWidth
    0.6 termLayer "ndiff" termName "vdd!"
    )
    (nil termCreate t netName "Z"
    termPurpose "drawing" addedPin nil termDirection
    "output" termSigType "signal" termCriticality 10
    termNum 1 termHeight 20.1 termWidth
    20.1 termLayer "metal1" termName "Z"
    )
```

Properties of different terminals

Terminal Properties in Property File

Terminal properties in CPH form



© 2020 All Rights Reserved.

The following table lists the terminal properties along with their description

Property Name	GUI Option Name	Description
`termCreate`	NA	This is a boolean value, which indicates whether or not to create a terminal.
`netName`	Net Name	Name of the net.
`termName`	Term Name	The name of the terminal.
`termLayer`	Layer and initIOPinLayer environment variable	The name of the layer on which the pin figure has to be created. The valid values are the layers present in the technology data.
`termPurpose`	Layer and initIOPinLayer environment variable	This specifies the layer purpose on which the pin figure has to be created. The valid values are the purposes present in the technology data.
`termDirection`	Term Type	Describes the direction of the terminal. The valid values are: “input”, “output”, “inputOutput”, “switch”, “jumper”, and “unused”.
`termSigType`	Sig Type	Determines the signal type of the terminal. The valid values for the signal type are: "signal", “ground”, "supply", "clock", "analog", "tieOff", "tieHi", "tieLo”, "scan", and "reset".
`termCriticality`	Criticality	Determines the criticality of the terminal. The valid value ranges from -128 to 128.
`termNum`	Number	Defines the number of pins to be created on a terminal. The valid range is from 1 to 20.
`termHeight`	Height	Defines the height of the pin to be created on a terminal. The value values are all positive integer values.
`termWidth`	Width	Defines the width of the pin to be created on a terminal. The value values are all positive integer values.

© 2020 All Rights Reserved.

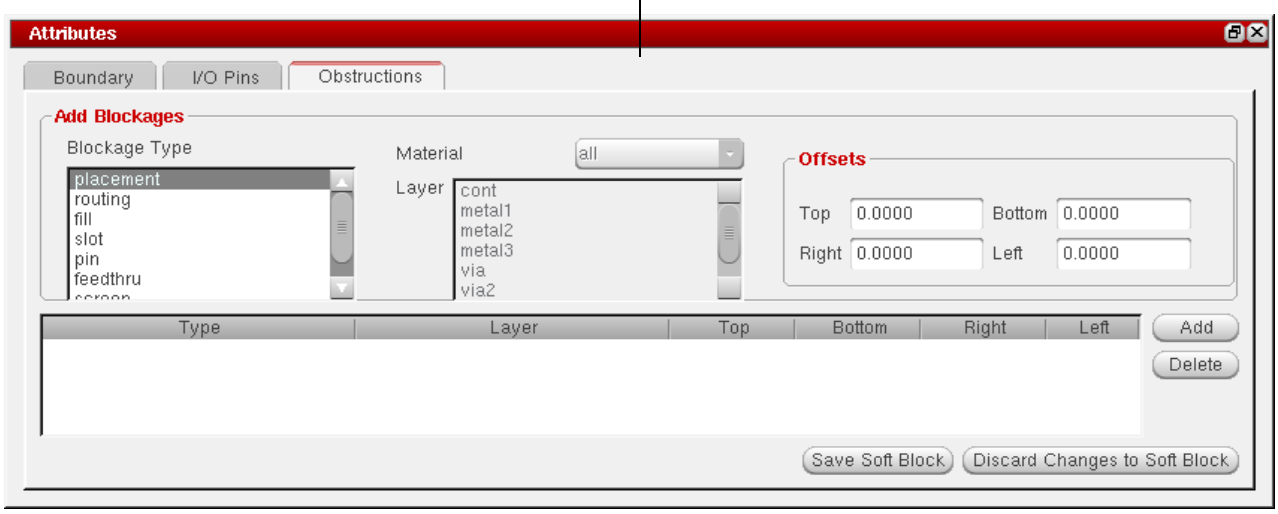
Halo Properties

The halo properties describes the properties that are to be defined for creating halo blockages in the block. The properties that are defined are as displayed in the following diagram.

```
haloProps ((nil haloLayer "metal1" haloBlockageType "routing"
  haloLeftOffset 0.0 haloRightOffset 0.0 haloBottomOffset
  0.0 haloTopOffset 0.0 haloType "layer"
  )
  (nil haloLayer "metal2" haloBlockageType "routing"
  haloLeftOffset 0.0 haloRightOffset 0.0 haloBottomOffset
  0.0 haloTopOffset 0.0 haloType "layer"
  )
```

Halo properties in Property File

Halo properties in CPH form



The following table lists the halo properties along with their description

Property Name	GUI Option Name	Description
`haloLayer`	Layer	The name of the layer on which halo blockages are to be created.

© 2020
All Rights Reserved.

`haloBlockageType`	Blockage Type	Describes the type of blockage. The valid values for the blockage type are: “placement”, “routing”, “fill”, “slot”, “pin”, “feedthru”, and “screen”. The default value is “routing”. For placement blockage, the halo layer is NIL.
`haloLeftOffset`	Offsets Left	Describes the distance of a halo blockage from the left of the prboundary. The valid value is either 0 or a positive float value.
`haloRightOffset`	Offsets Right	Describes the distance of a halo blockage from the right of the prboundary. The valid value is either 0 or a positive float value.
`haloBottomOffset`	Offset Bottom	Describes the distance of a halo blockage from the bottom of the prboundary. The valid value is either 0 or a positive float value.
`haloTopOffset`	Offset Top	Describes the distance of a halo blockage from the top of the prboundary. The valid value is either 0 or a positive float value.
`haloType`	Blockage Type	Describes the type of halo. The valid values for the halo type are placement and layer halo.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

D

Pin Constraints File

A pin constraints file is a simple SKILL file with constraints entered in a SKILL list. The constraints are entered in a tabular format so that they are readable and editable.

```
pinCons = list(

;; pinName        edge    pitch   status ...
;; -------        -----   -----   ---------

;; Top level
'("qa<0:9>:.*"      "left"    1.35  "floating" )
```

The SKILL list in the pin constraints file includes:

- Pin Name
- Edge Constraint
- Pitch Constraint
- Status Constraint

The pin constraints file also includes:

- An API to load the constraints in the Constraint Manager.
- A Sanity Checker that parses the pin constraints file and reports errors, if any.

Optimized Creation of Pin Constraints

The creation of pin constraints has been optimized such that only one alignment constraint is created for a given side and pitch value. Further, for each design, a single fixed and locked constraint is created.

© 2020 All Rights Reserved.

Pin Name

Pin name identifies the pins for which constraints need to be applied. Valid formats for specifying pin names are listed below:

Format	Description	Examples

Regular expressions	Use regular expressions to specify pin names. Wildcard characters and hierarchy delimiter “/” are also recognized.
	Format:
	`'[ Instance_Name / ] Terminal_Name [ : Pin_Name ] [ : PinFig_Name ]" | "[ Instance_Name / ] " .* "`
	Important
	Regular expressions with the “<“ and “>” characters are not recognized. Only use the “:” character to specify a range.
		`'("A:A" "left" 0.01 "locked")`
		`'("A/.*" "left" 0.01 "locked")`
		The following expression will not be recognized:
		`'("X<.*" "right" 0.1 "floating" )`
		`'("X<[0-1]>" "right" 0.1 "floating" )`
		`'("X<[0-1]>:X<[0-2]" "right" 0.1 "floating" )`

Numeric Pins	This method is used for pins with numeric names.	`'("3.1:4.1" "right" 0.1 "floating" )`

© 2020 All Rights Reserved.

Format	Description	Examples

Terminal names and pin names	Terminal and pin names are named OA objects. Therefore, you can assign constraints to:
	■ **Terminal names**
	Constraints assigned to terminal names are applicable to all the pins and pinFig objects corresponding to the terminal.
	■ **Terminal names and pin names**
	Constraints are applicable only to the specified pin in the specified terminal.
	Format:
	`'("Terminal_Name:Pin_Name" "Pin_Edge" Pin_Pitch "Pin_Status")`
		`'("A" "Top" 0.01 "locked")`
		`'("A:A1" "Top" 0.01 "locked")`
		`'("A:A2" "Top" 0.01 "locked")`

Individual PinFigs	This method is used for pin figures that are placed together to implement a feed through connection. You can enter constraints corresponding to each pin shape. **Format:** `'("Terminal_Name:Pin_Name:PinFig_Name" "Pin_Edge" Pin_Pitch "Pin_Status")`	`'("A:A:A1" "left" 0.01 "locked")` `'("A:A:A1.1" "right" 0.01 "locked")`

© 2020 All Rights Reserved.

Format	Description	Examples
Pin names for level-1 pins	When you specify instance pins, they correspond to the instance master's `Terminal_Name:Pin_Name:PinFig_Name`. Therefore, ensure that you specify the instance name while defining constraints for level-1 or soft block instance pins. **Format:** `'("Instance_Name/Terminal_Name:Pin_Name:PinFig_Name" "Pin_Edge" Pin_Pitch "Pin_Status")`	`'("I1/A:A:A1" "left" 0.01 "locked")`

Wildcard characters and hierarchy delimiters	To specify the constraints on all terminals and pins for a given instance, enter only the instance name and a `".*"` after the hierarchy delimiter `"/"`.
	To specify the constraints on the top-level pins, enter `".*"`.
	You can use `".*"` as a wildcard in regular expressions for names.
	You can also specify regular expressions in pin or pinFig names.
		`'("I1/.*" "left" 0.01 "locked")`
		`'(".*" "left" 0.01 "locked")`
		`'("ADDRESS_.*" "left" 0.01 "locked")`
		`'(".*:PIN246.*" "left" 0.01 "locked")`
		`'(".*:PIN246.*:PinFIG1.*" "left" 0.01 "locked")`
		`'("D[1-3]:P.*" "right" 0.1 "floating" )`

Vector single and vector bit notations	You can include vector single and vector bit notations `A<0:4>:A<0:4>` in pin names. Such vector notations are automatically expanded to their corresponding vector bits.	`'("A<0:4>:A<0:4>" "Top" 0.01 "locked")`

© 2020 All Rights Reserved.

If there are duplicate entries for a pin, only the last entry is honored. All previous entries are ignored.

Example 1

```
pinCons = list(
; pinName        edge    pitch   status ...
; ---            -----   -----   ----
 '("Z"     "right"    0.1  "floating" )
 '("Z"     "right"    0.1  "floating" )
 '("Z"     "right"    0.1  "floating" )

 '("Z"     "right"    0.1  "floating" )
 '("Z"     "left"    0.1  "floating" )
 '("Z"     "top"    0.1  "floating" )
 )
```

In the above example, there are multiple assignments for pin Z. However, only the last assignment is honored.

Example 2

```
pinCons = list(
; pinName        edge    pitch   status ...
; ---            -----   -----   ----
'("D.*:.*:.*"     "right"    0.1  "floating" )
'("D.*:.*:P__10"     "left"    0.1  "floating" )
)
```

In this example, the first entry assigns all pins in the given range to the right edge. The second entry refines the previous entry by specifying all pins on right side except one pin on left. The last entry is honored.

© 2020 All Rights Reserved.

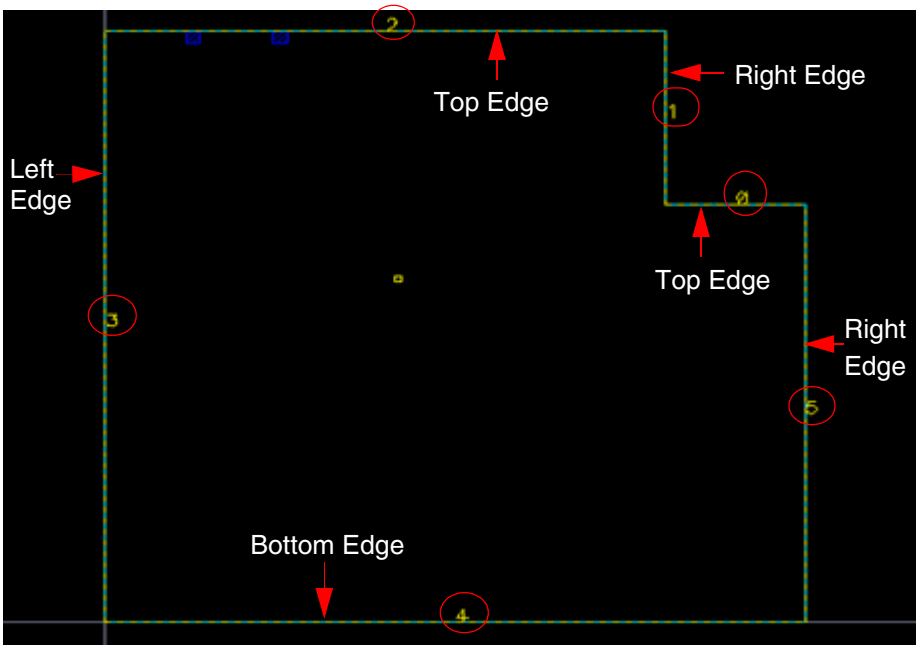
Edge Constraint

The edge constraint specifies the side on which the pin should be placed. Valid values for this constraint are:

`"left" | "right" | "top" | "bottom" | "edge_number" | "any"`

For shapes with rectangular PRBoundaries, the `"left"`, `"right"`, `"top"`, and `"bottom"` strings are supported. For shapes with rectilinear PRBoundaries boundaries, specify either the edge side or the `"edge_number"`. Edge numbers are assigned in the increasing order starting with zero, which is assigned to the edge at the right-top vertex, followed by the adjoining edge in the anti-clockwise direction.

An edge side for rectilinear boundaries may correspond to one or more edges. In the following example, the "`Top`" edge side can correspond to edge numbers `0` or `2`. Therefore, the pin planner has a choice of edges.



Note: The `"center"` and `"origin"` side parameter values of alignment constraints are not supported because they are not respected by the Pin Optimizer.

Use the `"any"` side specification for a pin that can be placed on any available edge. When the `"any"` side is specified, the "`orderOnly`" option is automatically turned on. Therefore, pin order is respected while placing the pins.

Note: In case of pins with a single pinFig, the alignment constraint is created on the pin, and not on the pinFig—even if the constraint is specified on the pinFig.

© 2020 All Rights Reserved.

Pitch Constraint

The pitch constraint defines the minimum edge-to-edge spacing between pins and other components. The format for specifying the pitch is as follows:

```
Unsigned Number > mfgGridResolution  | -1
```

Pitch values greater than the manufacturing grid resolution are valid. If you do not want to specify any pitch value, enter -1 in the table. This will be treated as the pitch value not being set.

Important

During pin optimization, "floating" and “placed” pins are positioned such that the pitch of fixed/locked pins is honored. For more information about these pin statuses, see Status Constraint.

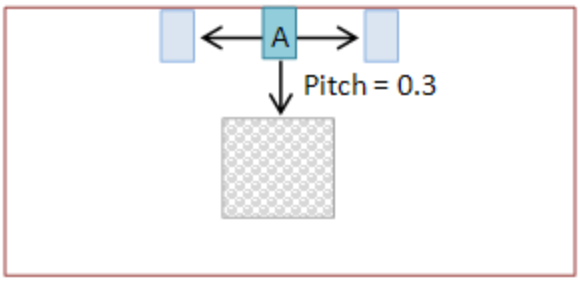
Caution

If different pitch values are defined for a pin and a corresponding pinfig, then the pinFig pitch value is respected.

The following examples illustrate how the pitch value is calculated in different situations.

Example 1:

This is a simple example, where the pitch for pin A is 0.3. A pitch can have value zero. Assumption is that the pitch for the neighboring pins or objects is zero. Pins are placed such that adjacent pins or objects are separated by a minimum distance of 0.3, which is the pin pitch value.

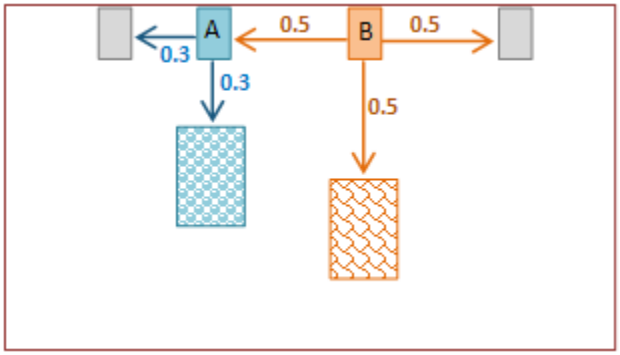


© 2020 All Rights Reserved.

Example 2:

Consider pins A and B. The pitch for pin A is 0.3, whereas the pitch for pin B is 0.5. Here, objects are placed at a minimum distance of 0.3 from pin A and 0.5 from pin B. However, when determining the distance between these two pins, the higher pitch is honored.

Pins A and B are placed at a distance of 0.5, which is the higher pin pitch value.

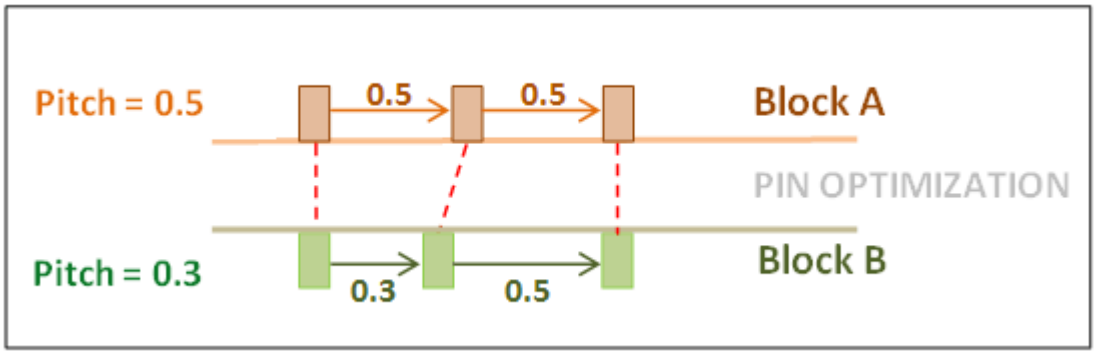


Example 3:

Consider pins in two separate blocks, Block A and Block B. The pitch for pins in these blocks is different. During pin optimization, the pins are placed such that:

- The pitch constraint value is honored
- The minimum net length is achieved

In the following image, notice that after pin optimization, the pins in Block B are separated by a minimum distance of 0.3, which is the pitch value. The spacing is further increased to ensure that the minimum net length is achieved.



Note: If bus pins are assigned different pin pitch values, then the values are honored without breaking the bus order or splitting the group.

© 2020
All Rights Reserved.

Status Constraint

The status constraint defines the placement status of the pin. Valid values are `"fixed"/ "firm"`, `"locked"`, and `"floating"`.

Pins with the `"fixed"` or `"firm"` status can be moved by interactive commands. The `"locked"` status indicates that the corresponding pin cannot be moved by any commands. The placement of such pins should not be changed. Specify `"floating"` for pins with no status.

Sanity Checker

The pin constraints file integrates a simple sanity checker to perform a basic sanity check of all the pin names and constraints that are included in the pin constraints file. If there are erroneous entries, then the Pin Optimizer does not respect the constraint. The sanity checker helps ensure that there are no erroneous entries in the pin constraints file.

The sanity checker performs the following checks for pin names:

- Validity of the naming format
- Existence of the specified instance, terminal, pin and pinFig
- Non-existence of duplicate entries for a pin name

  Note: The sanity checker only reports the existence of duplicate entries. It does not perform additional checking to check the constraints.
- Existence of multiple instances of the same master

  Note: The sanity checker will only report such instances.

In addition, the sanity checker performs the following checks to ensure correctness of the constraints:

- Validity of the boundary edge in case of a rectilinear prBoundary
- Validity of the boundary side
- Checking whether the pin pitch has a higher value than the manufacturing grid or has the value of -1

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

E

Constraint Groups

A constraint group lets you specify different sets of design constraints to be applied under different circumstances. This allows the flexibility to experiment with less or more stringent process rules at different stages of the design process.

You can group together any set of constraints into a constraint group, which is then applied to any design associated with the technology library. For more information, see “Technology File Constraint Groups and Constraints” in the *Virtuoso Technology Data Constraints Reference*.

There are three types of constraint groups that you can create:

- Foundry constraint group (`foundry`)
- Application-specific constraint group
- User-defined constraint group

There can only be one `foundry` constraint group, but there can be any number of application specific constraint groups and user-defined constraint groups for routing specific nets or net types.

The foundry, application-specific, and user-defined constraint groups may be created automatically during a translation process such as `cdb2oa` or `lef2oa`, and are based on the information in either the CDB techfile or LEF/DEF files. Other constraint groups may be created by your CAD group, and still others created by the individual layout designer. Some can be stored in a technology database and/or on the design itself.

In Virtuoso Layout Site, you typically have the following constraint groups as a required minimum.

- Foundry: The `foundry` constraints group has the absolute minimum rules which must be adhered to manufacture the design.
- Setup constraints group: The `Setup` constraints group typically has the `virtuosoDefaultExtractorSetup` as the default value. Other possible values are `virtuosoDefaultSetup` and `LEFDefaultRouteSpec`.

© 2020 All Rights Reserved.

Foundry Constraint Group

The `foundry` constraint group represents the absolute minimum rules (i.e. `minWidth`/`minSpacing`, etc.,) which must be adhered to in order to manufacture the design. These are commonly referred to as the base process rules and are stored only in the technology database (and not in the design database).

Creation and Usage

- Creating the `foundry` constraint group

The `foundry` constraint group is created and populated with information during `cdb2oa` translation or `lef2oa` translation. The translators get the information from sections within the CDB techfile and the LEF file.

Alternatively, you can create the `foundry` constraint group by loading an ASCII technology file containing the `foundry` constraint group information by using the *Tools -> Technology File Manager* option in CIW.

- Information in the `foundry` constraint group

The `foundry` constraint group contains all the information necessary to manufacture the design. Everything from `minSpacing`/`minWidth` for layers, `minEnclosure`, `minStep`, to `via` rules, to electrical constraints such as antenna rules.

- Using the `foundry` constraint group in Virtuoso

The `foundry` constraint group is the final stop along the constraint precedence lookup trail. If a constraint override is not defined elsewhere, the tools use the value(s) in the `foundry` constraint group.

The `foundry` constraint group is usually not writable for most users, but can be viewed by using the Process Rule Editor or by taking an ASCII dump. You can launch the Process Rule Editor by clicking the RMB in the Constraint Manager assistant or by clicking the PRE icon from the toolbar in the Constraint Manager assistant. You can set the scope to Technology and view the foundry constraint group in this dialog box. For details, refer to Process Rule Editor in the *Virtuoso Unified Custom Constraints User Guide*.

The virtuosoDefaultExtractorSetup Group

The `virtuosoDefaultExtractorSetup` constraint group can be stored within the technology database or directly on the design itself (i.e. lib/cell/view).

© 2020 All Rights Reserved.

Creation and Usage

- Creating the virtuosoDefaultExtractorSetup Constraint Group

The `virtuosoDefaultExtractorSetup` constraint group is created during `cdb2oa` translation.This constraint group may be created by loading an ASCII technology file containing the constraint group section, or through the Process Rule Editor.

- Information in the `virtuosoDefaultExtractorSetup` Constraint Group

The `virtuosoDefaultExtractorSetup` constraint group includes layers needed for extracting connectivity, plus poly, all metal layers, and cut layers in the valid layers section.

- Using the `virtuosoDefaultExtractorSetup` Constraint Group in Virtuoso

A constraint group for extraction is required to specify the extractable layers.

Connectivity extraction gets the constraint group name from the value of the `setupConstraintGroup` variable in your `.cdsenv` file. The default value is `virtuosoDefaultExtractorSetup`. You can change the name of the constraint group by resetting the value.

If the `setupConstraintGroup` variable has no value set, the extractor prompts you for a constraint group to use for extraction.

The virtuosoDefaultSetup Group

The purpose of the `virtuosoDefaultSetup` constraint group is to supply a default set of constraints for both interactive and automatic routing.

Creation and Usage

- Creating the `virtuosoDefaultSetup` Constraint Group

The `virtuosoDefaultSetup` constraint group is created during the `cdb2oa` translation using one of the following methods.

If the CDB techfile includes the `iccRules` section, the `cdb2oa` translator automatically translates the rules from the `iccRules` section to create the `virtuosoDefaultSetup` constraint group.

If the ASCII VCAR rule file is available during `cdb2oa` translation, use the `-iccrulesfile` option of the `cdb2oa` translator to create the `virtuosoDefaultSetup` constraint group. The syntax is the following.

© 2020 All Rights Reserved.

```
cdb2oa -iccrulesfile <vcarRuleFile>
```

- Information in the `virtuosoDefaultSetup` Constraint Group

The `virtuosoDefaultSetup` constraint group contains `validVias` and `validLayers` for routing (no cut layers), and possibly other rules such as `minWidth` and `minSpacing` for the layers, overriding the foundry rules.

- Using the `virtuosoDefaultSetup` Constraint Group in Virtuoso

The `virtuosoDefaultSetup` constraint group is typically used to define a full set of `validLayers` and `validVias` for the purpose of default interactive and automatic routing.

For example, if any particular signal is selected for routing and does not have a constraint group assigned to it, the constraints in the `virtuosoDefaultSetup` constraint group apply. The contents of `validLayers` in `virtuosoDefaultSetup` are usually a subset of those defined in the `virtuosoDefaultExtractorSetup` constraint group because, while you may want to extract connectivity for non-metal layers, you do not want interactive and automatic routing to use the non-metal layers.

The LEFDefaultRouteSpec Group

The `LEFDefaultRouteSpec` constraint group contains rule definitions typically used in digital standard cell applications or custom/mixed-signal applications that require metal layers only.

Creation and Usage

- Creating the `LEFDefaultRouteSpec` Constraint Group

  The `LEFDefaultRouteSpec` constraint group is specific to a LEF/DEF design flow and is created during `lef2oa` translation. Since a LEF file serves as the library source for technology information, the lef2oa translator puts all the technology information into the `LEFDefaultRouteSpec` constraint group.

  The valid layers, valid via definitions, and valid rules also become part of the `LEFDefaultRouteSpec` constraint group.

  Note: The `lef2oa` translator translates any `NONDEFAULTRULES` in LEF as constraint groups in the OpenAccess tech database.

- Information in the `LEFDefaultRouteSpec` Constraint Group

  The `LEFDefaultRouteSpec` contains valid layer and valid via definitions. The via definitions are typically standard vias and several custom vias used by other routers such

© 2020 All Rights Reserved.

as NanoRoute in the Innovus environment. The `LEFDefaultRouteSpec` also contains any other non-default rules such as wide metal rules.

- Using the `LEFDefaultRouteSpec` Constraint Group in Virtuoso

You can select the `LEFDefaultRouteSpec` as the default constraint group for interactive or automatic routing in the Wire Editing section of the Layout Editor Options form (Options > Editor...).

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

F

Migration from Preview to Floorplanner

From the IC6.1.0 release, Virtuoso Preview has been replaced by Virtuoso Floorplanner on OpenAccess. Both Preview and Floorplanner contain automatic and interactive floorplanning tools and technologies that help designers develop a layout from a schematic in a methodical manner. However, there are differences in the functionality and use models of both tools. The following sections describe the use model differences between Virtuoso Preview IC5.1.41 and Virtuoso Floorplanner IC6.1.x.

Note: This chapter does not describe the database change (CDB -> OA), but only the Floorplanner use model changes resulting from the database change.

Preview-Floorplanner Menu and Feature Differences

The following features and menus have changed in Floorplanner:

- Configuring the Physical Hierarchy
- Setting the Cell Type
- Generating the Physical Hierarchy
- Editing Soft Blocks
- Enabling Floorplanner
- Generating Design Objects
- Controlling the Visibility and Selectability of Objects and Layers
- Using the Menu Commands

© 2020 All Rights Reserved.

Configuring the Physical Hierarchy

The Hierarchy Browser in Preview has been replaced with the Configure Physical Hierarchy (CPH) functionality in IC6.1.x. CPH is a physical hierarchy configuration utility that lets you specify how a layout implementation is generated from a specific schematic design. The CPH window has three modes of operation: Hierarchy Configuration, Component Types, and Soft Blocks. Use the Soft Block mode to configure bindings for the soft blocks to be generated in the layout canvas.

Tip

Choose *Floorplan – Generate Physical Hierarchy* command to generate the physical hierarchy based on the configuration defined in CPH.

This section describes the following differences between the Hierarchy Browser and the CPH functionalities:

- Launching the GUI
- Creating Soft Blocks

Launching the GUI

Important

(ICADVM20.1 Only) In Layout EXL, floorplan commands are available under the *Plan* menu.

The following table describes the various methods to launch the two interfaces:

Virtuoso Preview	Virtuoso Floorplanner
You open the Hierarchy Browser from: ■ A logical view (schematic or netlist) using *Floorplan – Hierarchy Browser*. ■ A physical view (`autoLayout`) using *Tools – Floorplan – Hierarchy Browser*. ■ The CIW using *File – Export – PR Flatten*.	You launch the Configure Physical Hierarchy window from either the Schematic or the Layout views. To launch the Configure Physical Hierarchy window, with Layout XL or a higher tier enabled, select *Launch – Configure Physical Hierarchy* from either the schematic or the layout window menu bar.

© 2020 All Rights Reserved.

Creating Soft Blocks

In Preview, you use the Create Soft Blocks form to create soft blocks from selected regions, groups, or instances.

In IC6.1.x, you can define soft blocks in *Configure Physical Hierarchy – Soft Block* mode, and then use either the *Floorplan – Generate Physical Hierarchy* or the *Generate Abstract From Symbol* command to generate the soft blocks in the layout canvas.

For more information about CPH, see Configuring the Physical Hierarchy in the Virtuoso layout suite documentation.

Setting the Cell Type

The Set Cell Type form in IC6.1.x is the equivalent of specific properties required in Preview on the abstract views. It is similar to the `prCellType` and `prCellClass` instance master properties. The Set Cell Type form is used to set different celltypes. The following table lists the various celltypes supported by Floorplanner and their corresponding Virtuoso Design Environment attributes:.

Instances	Virtuoso Design Environment Attributes
I/O (PADs)	Pads, padSpacer, padAreaIO, corner
Core (Standard Cells)	Core, coreSpacer, coreAntenna, coreWellTap
Hard Macro	CoverBump, Cover, blockRing, block
Soft Layout	SoftMacro
Soft Abstract	blockBlackBox
Cells (Pcells and Custom Cells)	none

With integration with the Layout XL environment, the *Custom Cells* option is added, which corresponds to Pcells in Layout XL.

For more information, see Setting the Cell Type in the Virtuoso layout suite documentation.

Generating the Physical Hierarchy

The Generate Physical Hierarchy interface has changed in IC6.1.x with integration with the Layout XL environment. The tool now uses the Layout XL netlister. *Generate Physical*

© 2020 All Rights Reserved.

Hierarchy generates the physical hierarchy based on the configuration defined in the Configure Physical Hierarchy window. It also creates soft layouts and soft abstracts in the layout canvas corresponding to the configuration view generated by CPH. *Generate Physical Hierarchy* runs in batch mode and uses two main engines: *Generate All From Source* (for all soft layouts) and *Generate Abstract From Symbol* (to generate soft abstracts).

For more information, see Generating a Physical Hierarchy.

Editing Soft Blocks

In Preview, the Edit Block form provides options to change the boundaries of a soft block to reflect the aspect ratio and area utilization.

In IC6.1.x, you use the Edit Soft Blocks form to modify the attributes of individual soft blocks, while taking into account the placement and connectivity requirements in the design. Using the Edit Soft Blocks command, you can modify:

- The type of soft block to be created.
- The height, width, area, or shape of the PR boundary.
- The length, width, layer, and signal type of soft pins.
- Routing and placement obstructions inside the block.

For more information, see Editing Soft Blocks in the Virtuoso layout suite documentation.

Enabling Floorplanner

The *Floorplan* menu is enabled by default in Layout XL and higher tiers.

Important

(ICADVM20.1 Only) In Layout EXL, floorplan commands are available under the *Plan* menu.

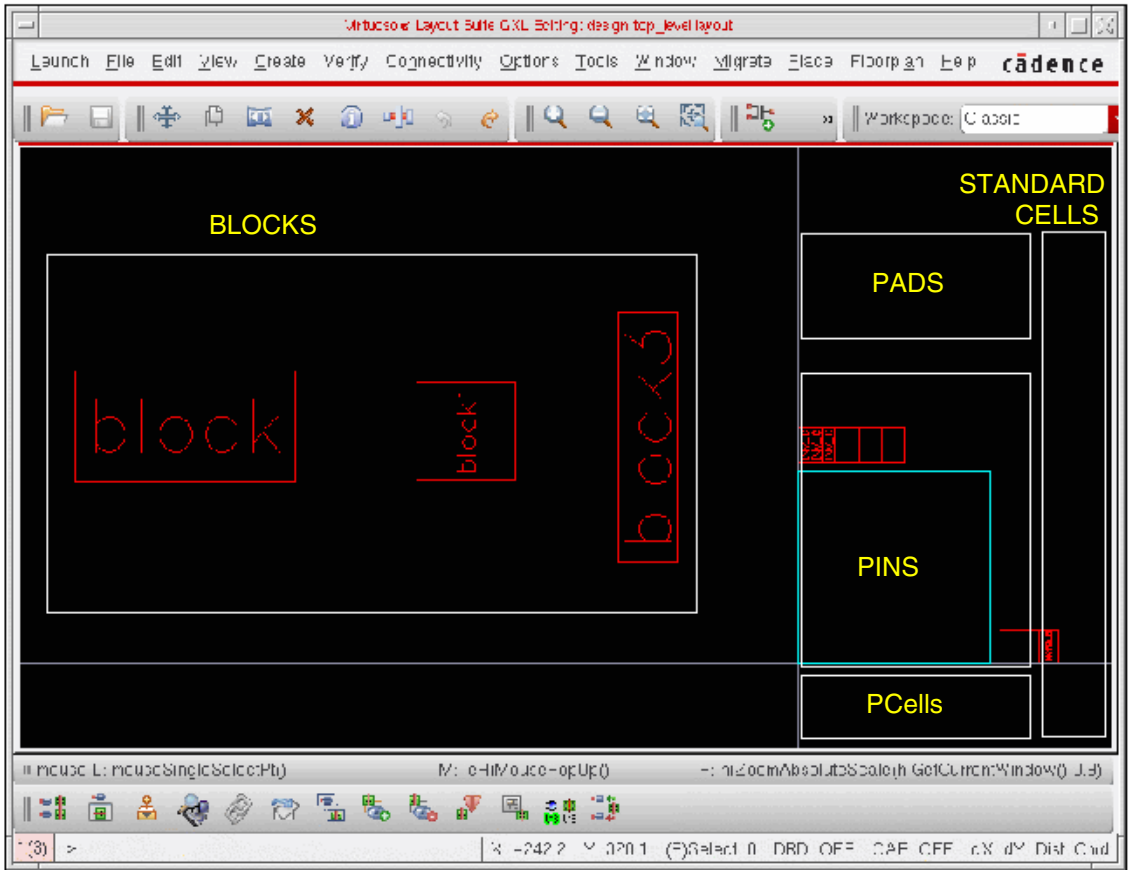
© 2020 All Rights Reserved.

Generating Design Objects

The process of generating design objects now provides the following additional features:

- **Auto Initialization of Design**

  After data generation, Floorplanner initializes the design and arranges its components according to their cell types. Floorplanner places components of the same cell type in different areas in the design as shown below:



- **Soft Layout Generation**

  All the views generated by Floorplanner are layout views. The cell type depends on the *Type* setting in CPH. When set to *LAYOUT*, the tool adds the `softMacro` cell type to make it soft. The soft layout is generated by *Generate All From Source* in batch mode. After generating the soft layout, only the PR boundary and the pins are visible, and all other components and their connectivity are hidden at the top level. You can descend into the layout view to make the other components visible.

- **Soft Abstract Generation**

© 2020
All Rights Reserved.

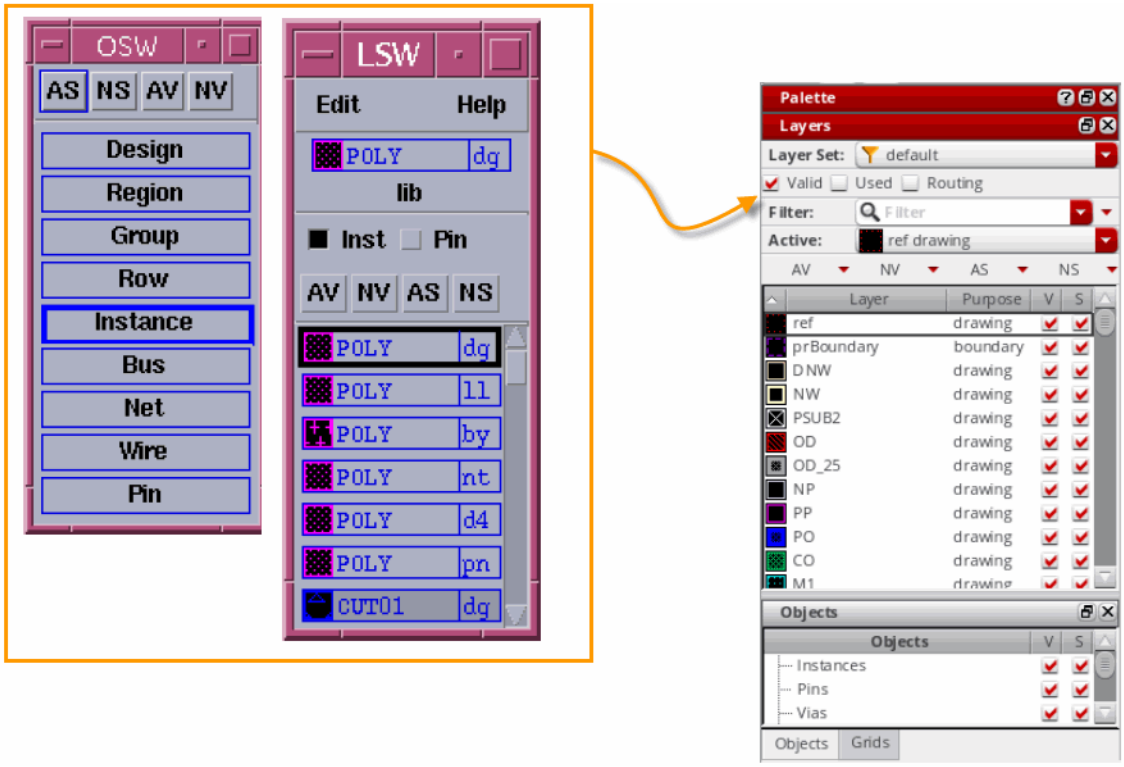
All the views generated by Floorplanner are layout views. The cell type depends on the *Type* setting in CPH. When set to *ABSTRACT*, the tool adds the `blockBlackBox` cell type to make it soft. Soft abstract is a layout view with only the PR boundary and pins, but no connectivity and components.

Controlling the Visibility and Selectability of Objects and Layers

In Preview, the following windows are used to control the visibility and selectability settings of objects and layers:

- **Object Selection Window (OSW):** Controls the selectability and visibility of floorplanning objects in the design.
- **Layer Selection Window (LSW):** Controls the layer visibility.

These windows are now replaced by the *Palette* assistant, which is a single assistant container that encapsulates the *Layers*, *Objects*, and *Grids* panels. You can hide, dock, or undock the entire *Palette* or each of the three panels individually. The following graphic shows the differences between the OSW, LSW, and Palette panels:



© 2020 All Rights Reserved.

The above change impacts the use model of some commands such as level-1 editing and net selection. For more information, see Palette Assistant in the Virtuoso layout suite documentation.

Using the Menu Commands

The table below summarizes the changes in the following Preview menu commands:

- Design Menu
- Create Menu
- Edit Menu
- Floorplan Menu
- Analyze Menu

Virtuoso Preview	Virtuoso Floorplanner
Design Menu	
Design – Save Saves the current level and also one level down for the `autoAbstract` and the `autoLayout`.	*File – Save* Uses the VLS use model, which saves only the current level and not one level down.
Design – Save As Saves the current level of hierarchy and one level down for the soft blocks in a different view. The new top-level view is automatically remastered with the new soft blocks view.	*File – Save As* Specifies the new view name for the top level and lower levels. The new top-level is remastered with the new soft blocks view. For binding with the VLS XL environment, the new view must be added to the Physical View List on the *Extraction* tab of the *Options – Layout XL* window.
Design – Discard Edits Works on the current level and on soft blocks instantiated at the current level. A confirmation window displays information about the instances that are edited and that have been discarded.	*File – Discard Edits* Discards edits. A confirmation window is displayed, but there is no information about discarded instances.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Design – Make Editable/Make Read Only Makes the current view and the soft block view, which are instantiated in the current view, either read-only or editable. By default, the soft block view is editable.	*File – Make Editable/Make Read Only* This command affects only the current level, and not the soft blocks. To make a hierarchy editable, choose *Edit – Hierarchy – Make Editable*.
Design – Show Selected Displays a new window with details of the selected objects.	*View – Show Selected Set* Same as Preview.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Create Menu	
Create – Region Creates a region in a specific LPP. You must specify the name, *Aspect Ratio*, and *Area Utilization* for the LPP. You can then move the region to the required location.	*Create – P&R objects – Cluster Boundary* or *Cluster Interface* Cluster boundary is the equivalent object in the new environment. A cluster boundary can exist only if a cluster exists. A cluster is the equivalent to a group. You first need to create a cluster (*Create – P&R objects – Cluster*) before creating a cluster boundary. For more information, see Creating Cluster Boundary the Virtuoso layout suite documentation.
Create – Group Creates a rectangular shape, which corresponds to a group, on the specified LPP at the bottom of the boundary. You must specify the name, *Aspect Ratio*, and *Area Utilization*.	*Create – P&R objects – Cluster* The equivalent object in the new environment is a cluster, which has no physical representation in Virtuoso. The only way to select a cluster is using the *Create – Cluster* form. You can cross-highlight the cluster members and the cluster boundaries. You do not need to specify any area utilization or aspect ratio because the members determine the behavior of the cluster.
Create – Soft Block Creates a soft block that includes the selected instances. In addition, a new cell is created in the library, and therefore a new level of hierarchy is added. The connectivity is propagated inside the block. Some pins are created in the new instance and the top-level connectivity is updated accordingly.	*Edit – Hierarchy – Make Cell* It is the equivalent in terms of feature, but the use model is different in terms of results. The tool uses the same infrastructure for Floorplanner and VLS XL. Floorplanner has inherited some limitations in terms of hierarchy modification. The binder cannot handle hierarchy modifications like *Flatten* and *Make Cell*. Therefore, the above command creates a new cell, but the binder is not updated with the new hierarchy.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Create – Abstract for Subblock Provides the following options: ■ **Run Abgen:** Runs Abgen automatically with the specified set of rules and user-defined metal layers. ■ **Create Abstract:** Starts Abgen in the stand-alone mode with the selected cell already listed in the stand-alone interface.	*Tools – Abstract Generation* Runs Abstract Generator in batch mode with the selected set of cells. If no cells are selected, it runs on the current cellview. Abstract Generator runs directly with the techfile definitions, and therefore does not require additional files for the technology rules.
Create – Row Generates a row in the form of a shape on the specified LPP. You must specify the *Row Type*, *Placement Class*, *Row Orientation*, *Row Distribution Mode*, *Channel Spacing* or *Factor* or the *Number of Row*, and the row *Flip* setting.	*Create – P&R objects – Row* Generates a row object. The interface has been modified, but the information required is the same as Preview. An interface for the Custom Placement Area has been added. The Dressing Template part is specific to Custom Placement Area. For more information, see Creating Rows the Virtuoso layout suite documentation.
Create – Bus Groups the selected nets. Select the required nets in the canvas and specify a bus name. You can select, query, and edit buses. Pin Optimizer and the Block Placer consider buses as constraints.	Constraint Manager Assistant – *Bus* You define bus constraints. A bus constraint is a group of two or more nets. To define a bus constraint using the Constraint Manager, select the required nets in the *Navigator* window and click *Bus* from the constraints list in Constraint Manager.
Create – Soft Pin Creates a soft pin by creating a duplicate of the selected pin. Also lets you add feedthrough pins either at the top-level or in the soft block.	*Create – Soft Pin* Creates soft pins for soft blocks. Unlike Preview, you can create multiple soft pins.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Create – Bus Pin Adds a feedthrough pin based on a predefined bus.	*Create – Feedthrough Terminal Pins* with the filter *Net Class* or *Bus* Creates feedthrough pins. You can choose the number of pins to add. The new net is propagated to the soft block netlist and is compatible with VLS XL.
Create – Blockage Creates a blockage on the specified LPP. You must specify the blockage type and draw the shape on the canvas.	*Create – P&R Objects – Blockage* The tool uses the `Blockage` object to represent blockages. The interface has been modified, but the information required to create a blockage is the same – *Blockage type* and *Layer* – if needed. Additional methods for creating blockages (automatic and manual) are available in the new environment. You can also attach blockages to instances or clusters.
Create – Wire Creates a wire from an unrouted string. Use this command to manually route critical nets or power supply nets, or to edit connections after detailed routing.	*Create – Wire* The use model is the same as Preview. The only difference is that the tool uses `oaPathSeg` to represent wires.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Edit Menu	
Edit – Move Lets you move regions, groups, rows, instances, and soft pins.	*Edit – Move* No change in use model.
Edit – Snap Resizes soft block boundaries to a multiple of the specified placement grid and snaps soft block pins to the specified routing grid.	Available through the `F3` option form of the *Move* command. Snaps moved objects to grids and pins to the PR boundary. Also lets you specify whether pins must be resized during snapping.
Edit – Align Rows Aligns the edges of the selected rows with the point you select with the mouse.	*Edit – Quick Align* and *Edit – Advanced – Align* The two commands can be used to perform similar tasks, but they differ in their usage. Use these commands to align objects to a point, an edge, a layer, or a target axis.
Edit – Properties Lets you view and edit properties of objects in the current design.	Property Editor Assistant You use the Property Editor assistant to view and edit properties of components in the design.
Edit – Search Displays the Search For form, which lets you search for objects in the design and then highlight or select these objects.	Individual P&R Objects-Related Forms After consolidation with VLS, the option is integrated with the P&R objects in this interface such as clusters, soft blocks, cluster boundaries, and soft pins.
Edit – Block – Edit Changes the boundaries of a soft block to reflect the aspect ratio and area utilization required for the block.	*Edit – Edit Soft Blocks* Modifies the attributes of individual soft blocks taking into account the placement and connectivity requirements of the design.
Edit – Block – Adjust Lets you move, grow, push, and plow soft blocks in any direction according to certain grid steps.	*Floorplan – Soft Block Update – Adjust Blocks* Adjusts the existing floorplan by abutting and pushing the soft blocks in the design.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Edit – Block – Align Aligns the selected blocks. You specify one block as the anchor block. The command aligns the second block with the anchor block.	*Edit – Advanced – Align* Aligns objects or shapes in a cellview. In the new environment, you have more flexibility in terms of the type of alignment. You can align objects to a point, another object, or a layer. You can also align a layer with another layer or object.
Edit – Block – Adjust Pins Adjusts symbolic pins in soft blocks by snapping them to grids depending on their sides.	*Connectivity – Pins – Snap* Snaps both top-level and level-1 pins to grids appropriate to the block type being edited. Forms related to the *Move*, *Reshape*, *Stretch*, and *Chop* commands have options that let you snap boundaries and pins automatically to the grid during interactive editing.
Edit – Region Lets you change the *Aspect Ratio* and *Area Utilization*, and re-compute the region boundary.	No corresponding command You cannot edit the cluster boundary parameters. You must delete the cluster boundary and re-create it as per your requirements.
Edit – Group Lets you change the *Aspect Ratio* and *Area Utilization*, and re-compute the boundary.	*Create – P&R objects – Cluster* Lets you select and edit clusters. Aspect ratio and area utilization options are not available because a cluster is a free form.
Edit – Member Lets you add, remove, fit, and unfit members in a region or group.	No corresponding option In the new environment, a cluster boundary is linked to a cluster, and therefore cannot be dissociated. All members of a cluster are members of the cluster boundary. To change the cluster members, use *Create – P&R Objects – Cluster*. The *Fit* command is available in the interface of *Create – P&R objects – Cluster Boundary*.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Edit – Break Soft Block Flattens the soft block and instantiates all its instances at the current level. The connectivity is updated and all associated pins are deleted.	*Edit – Hierarchy – Flatten* This is the equivalent command in the new environment in terms of feature, but the use model is different in terms of the results. With integration with VLS XL, the tool uses the same infrastructure for Floorplanner and VLS XL. Floorplanner has inherited some limitations in terms of hierarchy modification. The binder cannot handle hierarchy modification such as *Flatten* and *Make Cell*. The above command flattens the selected cell, but the binder is not updated with the new hierarchy. The instances from the soft block do not have the connectivity.
Edit – Bus Pins Edits the existing bus pins. You can also edit or delete the associated side constraints and layers.	Navigator Assistant - *Bus Constraint* You need to create a Bus constraint in the Navigator assistant on the connected nets. The pins inherit the constraint automatically.
Edit – Wire Lets you edit wires. Wires are routed nets. The *Wire* menu provides commands to split and move wire segments, stretch rails, and change routing layers. To change the width of a wire, use the *Edit – Properties* command.	Any *Edit* command There is no specific command for editing wires in the new floorplanner interface. You can use any *Edit* command on existing wires.
Edit – Select Lets you select or deselect floorplan objects.	*Edit – Select* The *Edit – Select – Select All* and *Edit – Select – Deselect All* commands are the same as in Preview. For the other commands, such as *Edit – Select – By Name*, and *By Property*, use the *Tools – Find/Replace* interface. The *Connected To* option is not available in the new environment.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Edit – Delete Removes the selected region, group, row, bus, or soft pin from the database. This command works only on floorplan objects, and does not modify the netlist.	*Edit – Delete* The feature and use model is the same as in Preview.
Edit – Undo Cancels the last interactive command that you used and returns the selected objects to their previous statuses and locations.	*Edit – Undo* The feature and use model is the same as in Preview.
Edit – Redo Reissues the last command you canceled with *Undo*.	*Edit – Redo* The feature and use model is the same as in Preview.
Edit – Netlist Displays the Edit Netlist Toolbox that lets you make changes to the netlist. You can change connections and instances in the netlist, read an ECO file, or compare the current view with another view.	No corresponding option The new environment is not based on netlists, and therefore it does not include any option to edit netlists. It is directly linked to the schematic. You can directly modify the schematic by using *Connectivity – Update – Components and Nets*.
Edit – Soft Pin operations Controls the direction in which you can move a soft pin.	No corresponding option With migration to OA, symbolic pins are not available in the new environment. This is replaced with geometric pins as in VLS-L.
Edit – Soft Pins – Snap Mode Controls the direction in which you can move a soft pin.	*Connectivity – Pins – Snap* Snaps both top-level and level-1 pins to the grid appropriate to the type of block being edited. The *Move*, *Reshape*, *Stretch*, and *Chop* command interfaces have options that let you snap boundaries and pins automatically to grids during interactive editing.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Edit – Soft Pins – Align Aligns the selected pins. You select the reference pins and then the target pins. These pins are automatically aligned based on their connectivity.	*Edit – Advanced – Align* Aligns objects or shapes in a cellview. All alignment commands in VLS-L and Floorplanner have been consolidated. In the new environment, you have more flexibility in terms of the type of alignment. You can align objects to a point, another object, or a layer. You can also align a layer to another layer or other objects.
Edit – Soft Pins – Respace Respaces the selected soft pins along the block boundary. You must specify the spacing size, the type of spacing (*Edge to Edge* or *Center to Center*), and the start side (*Left/Top* or *Right/Bottom*).	*Place – Pin Placement* (*Pin Spacing* section) The *Pin Planner* tab in the Pin Placement form includes options to respace pins on the same edge. You can also generate spacing constraints. For more information, see Pin Planning.
Edit – Soft Pins – Optimize/Float Optimizes, fixes, or floats all or selected soft pins. The pin optimizer tries to position pins to get the shortest possible net length at the top level.	*Floorplan – Pin Optimization* Positions soft block pins in a manner that helps obtain the shortest possible net length at a particular level in the design. The objects that can be selected for pin optimization are: top-level and soft block pins, top-level PR boundary, soft block instances, and nets. For more information, see Pin Optimization.
Edit – Soft Pins – Pin Overlap Checks the design for soft pins that overlap other soft pins and displays markers on the shorted pins.	*Floorplan – Pin Checker* (ICADVM20.1 Only) Pin Checker lets you perform several checks on pins, including pin overlaps, and reports the results. For more information, see Using the Pin Checker.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Edit – Soft Block Lets you perform block operations by using the *Chop*, *Reshape*, and *Stretch* commands.	*Edit – Stretch, Edit – Basic – Chop, Edit – Advanced – Reshape* A number of Virtuoso Layout Suite commands have been enhanced to support the Level-1 Editing mode. To use these commands, you must first enable the Level-1 Editing mode. You can then move, stretch, reshape, or chop objects in this mode. You can also find and replace nets, discard edits, and save hierarchical cellviews under a different name. For more information about enabling the Level-1 Editing mode and using the various editing options, see Editing Soft Blocks in the Virtuoso layout suite documentation.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Floorplan Menu	
Floorplan – Reinitialize Estimates and creates a design boundary and a simple floorplan with the design objects applicable to the place and route tool. If you have an existing floorplan, you can remove the data and reset the locations of design objects. Initialization creates I/O rows and standard cell rows.	*Floorplan – Reinitialize* The use model is the same, but the interface has been modified according to the new objects that are supported. For more information, see Reinitializing the Design.
Floorplan – I/O Place Places and spaces the I/Os in the design.	*Floorplan – I/O Placer* Includes the following commands to optimize I/O positions: ■ Row Creation ■ PAD Placement ■ Filler Cells ■ Corner Cells For more information, see I/O Planning and Placement.
Floorplan – I/O Constraint File Lets you use an I/O constraint file to place I/O cells in an initialized design. The file lists the I/O cells associated with each side of the chip design and gives the orientation of each I/O cell.	No corresponding option available The constraints file is no longer supported. You need to define your I/O constraints in the Constraint Manager.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Floorplan – Partition Automatically creates placement groups or blocks using a ratio-cut algorithm.	No corresponding option available The automatic partitioner is not available. Due to integration with the Layout XL environment, the tool has some restrictions in terms of flexibility on the schematic and layout correspondence. The only way to partition is to use groups, which is now Cluster. (Refer to the *Create – Groups* commands.)
Floorplan – Block Place Automatically places hard blocks, soft blocks, and I/Os in pure block designs. It minimizes the wire length and chip area, while preserving user-specified placement constraints. The command can adjust soft blocks within the specified aspect ratio to maximize chip density.	*Floorplan – Block Placer* The functionality remains the same. In the new use model, the block placer reads the constraints specified in the Constraint Manager. You can also specify additional constraints in the Block Placer form, which the block placer respects. The block placer takes into account the blockages in a design and does not place blocks on them. For more information, see Placing Blocks.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Floorplan – Create / Update Auto Layouts Updates a soft block `autoLayout` view with the following information from the higher level `autoLayout` view: ■ Boundary and pin locations ■ Over-the-block pre-routing ■ Placement and routing blockages ■ Placement plan under the soft block instances ■ Routing grid pattern ■ Manufacturing grid pattern	No corresponding option available The tool does not use `autoLayout` because it works directly with the layout, and not `autoAbstract`, as in Preview. The equivalent of `autoLayout` is a layout with cell type `BlockBlackBox`.
Floorplan – Floorplan File (Preview IC5.1.41) / Template File (IC6.1.x) Lets you read, write, and edit the floorplan constraints and placement status information in a given floorplan file.	*File – Load Physical View* You no longer need to edit an ASCII template file manually. Use the above capability to import into the current cellview predefined information from another layout cellview. This approach lets you: ■ Use the full range of functionality available through the VLS GUI and the public SKILL APIs to create and edit the source cellview. ■ Extend the functionality to accommodate new database objects as they are developed. ■ Use any existing layout cellview as the source cellview.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Floorplan – Replace View Changes the view name of the library masters that the specified instances refer to. This command also lets you replace all place and route abstract views with their corresponding layout views for design verification.	*Tools – Remaster Instances* The above command provides more flexibility because you can switch an instance to another instance with a different library, cell, or view name. The Terminal Checker is also available. For more information, see Re-mastering Instance Views.
Floorplan – Edit Routing Grid Defines global and detailed routing grids.	*Create – P&R Objects – Track Pattern Editor* Creates tracks in both preferred and non-preferred layer directions. For each routing layer, you can define or edit the routing grid. The above command can only modify the detailed routing grid, but not the global grid. For more information, see Creating Track Patterns.
Floorplan – Special Route Makes the selected nets and pins special. Special nets and pins are located in the `SPECIALNETS` section of the DEF file and are routed by the `SROUTE` commands.	No corresponding option available. There is no specific interface for special routing in the new environment. You can do it through the *Edit Properties* window by changing the Net Signal Type to `Power` or `Ground`. To unset it, you have to do the same, but you need to change the Signal Type to `Signal`. To report all nets with a specific Signal Type, use *Tools – Find/Replace Net – Signal Type* = `Power/Ground`. You also need to set the `routeMethod` to `geometric` on one `instTerm` of this net through SKILL commands.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Floorplan – Distribute Instances Removes cell overlaps and redistributes cells to remove routing congestion.	*Place – Custom Digital – Auto Placement* (Virtuoso Custom Digital Placer) Use Virtuoso Custom Digital Placer to place standard cells in rows.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Analyze Menu	
Analyze – FP Objects Displays a window that contains physical analysis information about the selected design, region, group, and instance. The information in the window varies depending on the type of object selected.	*File – Summary* There is no change in the use model.
Analyze – Connectivity Displays the number of nets connecting the selected blocks, regions, groups, and instances in the floorplan.	*Connectivity – Analyze* There is no change in the use model.
Analyze – Ruler Creates a ruler to measure objects and distances between objects in a cellview. You can also draw a segmented ruler to measure dimensions such as half perimeters, which require measurement on more than one axis.	*Tools – Create Rulers* There is no change in the use model.
Analyze – Clear Rulers Deletes the rulers that are currently displayed in the design window.	*Tools – Clear All Rulers* There is no change in the use model.
Analyze – Show Markers Displays a text window that shows the violations in the view.	*Verify – Markers – Find* There is no change in the use model.
Analyze – Clear Marker Clears errors that are highlighted in the design.	*Verify – Markers – Delete All* There is no change in the use model.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Analyze – Check – Cell Overlap Checks the design for overlaps between cell bounding boxes.	No corresponding option available The command is now fully interactive. The check is done automatically in the VLS environment. The difference is in terms of the use model. Overlaps between a PR boundary and a placement blockage are notified, but overlaps between two PR boundaries are not notified.
Analyze – Check – Soft Pins Overlap Checks the design for soft pins that overlap other soft pins.	No corresponding option available The command is not available, but the interactive checker is available to report shorts. Also, the Pin Optimizer checks for pin overlaps.
Analyze – Check – Soft Pins Spacing Checks the design for soft pins that violate the minimum spacing rules.	No corresponding option available The command is no longer available, but the interactive checker performs the check. Also, Pin Optimizer checks for pin overlaps and the validity of the spacing value on each layer.
Analyze – Check – Region Checks regions for size and boundary design violations.	No corresponding option available The command is no longer available. The Constraint Management System includes an interactive checker, which shows violations on the Cluster constraints.

© 2020 All Rights Reserved.

Virtuoso Preview	Virtuoso Floorplanner
Soft Block Blockages Menu	
Soft Block – Blockages In Preview, there is a specific way to put blockages over a soft block. The blockages are stored in the master and are not selectable at the top level. To edit a blockage, select the required soft block and then select *Edit – Properties*.	*Launch – Configure Physical Hierarchy – Soft Block* mode *– Attributes – LayerHalo Obstructions* *Edit – Soft Blocks – Obstructions* In Floorplanner, you can create halos, which are blockages that are stored in the master, but are not editable from the top. As compared to Preview, there are several blockage types available that you can choose from. Also, you can set the blockage offsets from all four sides of the soft block.

© 2020 All Rights Reserved.

© 2020 All Rights Reserved.

G

Virtuoso Flooplanner Forms

This chapter lists the controls in the Virtuoso Floorplanner forms.

Note: Many of the options described in this section have a corresponding environment variable. For a full listing of these environment variables, see Layout XL Floorplan Environment Variables.

- Analyze Connectivity
- Auto-Create Pins
- Auto Generate Hierarchy
- Block Annotations
- Block Placer
- Compare Pin Group Guides
- Corner Pad Placement
- Create I/O Row
- Create Pin From Label
- Create Pin From Net Shapes
- Electrically Aware Pin Sizing
- Floorplan Global Options
- Generate Physical Hierarchy
- Insert Filler
- I/O Pad Placer
- Label Layer Purpose
- Label Update
- Load Solutions

© 2020 All Rights Reserved.

- Merge and Must Connect Pins
- Pin Checker (ICADVM20.1 Only)
- Pin Group Guide
- Pin Group Guide - Place as Schematic/Symbol
- Pin Placement
- Pin Spacing Advanced Options
- Pin Offset Options

Promote Pins

Push Pre-Routes

Reinitialize Floorplan

Resize Pins

Snap Pins

© 2020 All Rights Reserved.

Analyze Connectivity

Use the **Analyze Connectivity** form to display information about connectivity within and between blocks. This information is useful during design floorplanning.

Font section specifies the settings of the labels displayed on the flight lines.

Size specifies the font size of the labels. The available options are:

User Defined specifies an absolute font size.

Auto calculates the font size depending on the number of connections.

Color specifies the font color of the labels. The available options are **Black & White** and **Same color as connectivity line**.

Transparency specifies the transparency of the labels displayed on the flight lines.

Connectivity section specifies the settings of flight lines.

Width sets the width of the flight lines to either **User Defined** or **Proportionate to connectivity count**.

Filter nets hides connections to the selected type of nets—**Power**, **Ground**, and **Global**.

Display Connectivity On Selection Change specifies the mode in which connectivity information is to be displayed when a block is selected. The available options are:

Auto enables dynamic selection of objects and the connectivity information is refreshed automatically.

Manual does nor change connectivity information dynamically.

Enable Net Selection cross-selection of nets between the design canvas and the Navigator assistant.. The available net selection options are:

Exclusive Highlights only those nets that exclusively connect to the selected blocks. Connections to all other blocks are excluded.

Inclusive highlights all nets that are connected to the selected blocks.

Related Topics

Analyze Connectivity

© 2020 All Rights Reserved.

Auto-Create Pins

Use the **Auto-Create Pins** form to search for net shapes in a design and create boundary and buried pins automatically on these net shapes.

Shape Search section provides options to search for net shapes in the design.

Net in specifies the location in which net shapes are to be searched. Select either **CellView** to search all nets in the current cellview or **Selected** to search only in the selected nets.

Consider Net Shape on Routing Layers Only restricts the search for net shapes to the routing layers.

Consider Net Shape on Selected Purpose specifies the purpose on which net shapes are to be searched.

Environment Variables: modeOption

Generate Option section specifies generic pin generation options.

Schematic Aware specifies whether the tool must search for corresponding shapes in the schematic view before creating pins.

Delete Pins deletes the existing pins in the layout canvas before generating new ones.

Environment Variables: deletePinCheckbox, deletePinSelection

Auto Created deletes only those pins and associated metal shapes that have been generated by the Auto-Create Pins form.

All deletes all pins in the current cellview.

Check for Duplicate Pins checks for the presence of any similar pins in the design, and if found, does not create a duplicate pin.

Signal Pin Generation section specifies the following signal pin settings.

Boundary Shape Pin specifies that pins are to be created only for the shapes that are touching the PR boundary. Use the following settings to specify the number of signal pins to be generated when multiple boundary shapes belong to the same net:

Single creates a single boundary pin on the highest metal layer.

Multiple lets you specify the layer in which multiple boundary pins are to be created. The available options are **Top Layer**, which refers to the highest metal layer, and **User Layer**, where you can specify the layer.

© 2020 All Rights Reserved.

Environment Variable: boundarySquareShapePin

Buried Whole Shape Pin specified that pins are to be created only for the shapes that are either completely inside the PR boundary or are straddling the PR boundary.

Single creates a single buried pin on the highest metal layer.

Multiple lets you specify the layer in which multiple buried pins are to be created. The available options are **Top Layer**, which refers to the highest metal layer, and **User Layer**, where you can specify the layer.

Environment Variables: buriedWholeShapePin, buriedMultipleMode

Power Pin Generation section specifies the following power pin settings. Similar to signal shapes, you can choose to create pins for boundary shapes and buried shapes.

Pin Attributes section specifies the following attributes for the new pins to be generated.

Boundary Signal Pin Size lets you specify the size of the auto-created pins.

Auto creates square pins. Only rectangular and square path, pathsegs, and polygonal shapes are recognized for pin creation.

Whole Shape creates pins of the same shape as the corresponding net shapes. For example, square, rectangle, L-shaped, and path pins are supported.

Environment Variable: pinSizeSelection

Pin Layer specifies the layer on which pins are to be generated. The available options are **Same as Shape LPP**, **Same Layer, Custom Purpose**, and **Custom LPP**.

Environment Variables: pinLayers, customPurposeField, customLPPField

Pin Connectivity section lets you select one of the following pin connectivity models for the pins that are generated.

Strongly Connected specifies that the router must connect the pins internally, within the device.

Must join specifies that the router must connect the pins externally, at a higher level.

Must Connect specifies that the router must connect to all pins of the selected terminal.

Environment Variables: pinConnectivityMode

Create Text Label for Pins creates text labels for the pins that are generated.

(ICADVM20.1 Only) **Enable Coloring** uses the multiple patterning color engine to assign colors to the new pins based on their positions relative to the WSP tracks.

© 2020 All Rights Reserved.

Create Metal Shapes Under Pin creates a metal shape under each auto-created level-1 (soft block) pin.

Path creates a pathSeg metal shape.

Rectangle creates a rectangular metal shape.

Show displays a preview of how the settings would be applied to the design.

Hide closes the preview.

Related Topics

Creating Pins Automatically

© 2020 All Rights Reserved.

Auto Generate Hierarchy

The Auto-Generate Hierarchy lets you quickly generate a physical hierarchy by applying common parameters. You can define common boundary and pin parameters for the blocks you want to generate. You can specify the shape and size of the boundary that is created when the soft block is generated.

Stop View specifies the required physical view names to be used to determine the corresponding physical views.

Switch View specifies the names to be used to create a physical view for blocks if the stop views are absent in the hierarchy.

Create View specifies the name to be used for the new layout view created when no matching stop views are found. **Environment Variables:** autoGenHierCreateView

physconfig View specifies the name of the physical configuration view that provides information about the schematic-layout pairs. **Environment Variables:** autoGenHierPhysConfig

Boundary Attributes section provides options to define the boundary.

> **Use Area Estimator** calculates the boundary based on the following area estimator functions:
>
> **PR Boundary Based** sums up the polygonal area of the individual instances to derive the overall area estimation.
>
> **BBox Based** sums up the minimum bounding box area of the individual instances to derive the overall area estimation.
>
> **Add** lets you register custom area estimation functions and use them.
>
> **Aspect Ratio** specifies the width-to-length ratio of the boundary. The default value is 1, which specifies a square boundary.
>
> **Area Utilization** specifies the percentage of area within the boundary that must be filled. The default is 25%. **Environment Variables:** autoGenHierDefaultArea

Pin Attributes section specifies the following common settings for all the soft block pins:

> **Layer Purpose Pair** specifies the LPP on which the pins must be generated.
>
> **Height** specifies the pin height.
>
> **Width** specifies the pin width.

© 2020 All Rights Reserved.

Create Labels specifies whether labels are to be generated along with the soft block pins.

Initialize Instances Outside Boundary generates all the instances and pins outside the PR boundary, without any overlap. The placement statuses of the instances are set to none.

(ICADVM20.1 EXL Only) **Hierarchy Controls** section defines settings for hierarchical instances.

Generate Hierarchy to Level defines the level up to which the hierarchy must be generated. The default (maximum) value is 31. **Environment Variables:** autoGenHierDepth

Last Level Generation specifies the state in which the last level of hierarchy is to be generated. The available options are:

Black Box generates the last hierarchical level as a black box containing only the PR boundary and pins.

Flatten generates the PR boundary and pins in the last level. If any additional stop views are found inside the hierarchy, those levels are flattened and displayed.

Show Unplaced Instances on Descend specifies whether unplaced instances are to be displayed when you descend into the hierarchy. The available options are:

Yes (default) shows all unplaced instances.

No hides all unplaced instances.

Select displays a popup message each time you descend into a hierarchy, which lets you specify whether unplaced instances are to be displayed or hidden.

Environment Variable: `showUnplacedInstances`

(ICADVM20.1 EXL Only) **Place Instances and Pins** section specifies the following pin placement settings:

Place Instances like Schematic places the generated soft block instances inside the PR boundary as in the corresponding schematic view.

Run Pin Optimization runs Pin Optimizer hierarchically on the generated blocks. You can select one of the following modes:

Top Down runs Pin Optimizer to derive soft block pin positions hierarchically from level-0 until the leaf level.

Bottom Up follows the same behavior as described in Top Down, but runs in the opposite direction.

© 2020 All Rights Reserved.

Environment Variables: autoGenHierRunPinOpt, autoGenHierPinOptMode

Related Topics

Analyze Connectivity

© 2020 All Rights Reserved.

Block Annotations

Use the options in the Block Annotations Options form to customize the information displayed in the block annotations. The Block Annotations form is available for both edit and read-only cellviews.

Annotations section specifies the information that is to be included in the block annotations.

Display Mode specifies whether the block annotations are to be displayed for **All** or **Selected** blocks. Environment variable: blockAnnotationSelMode

Instance name and Master LCV displays relevant information in the block annotations. Environment variable: blockAnnotationInstName

Environment Variable: blockAnnotationTextLPP

Area displays the total block area within the PR boundary. Select the required units:

um^2: Square micrometers

mm^2: Square millimeters

du^2: Square database unit

Environment variables: blockAnnotationArea, blockAnnotationAreaUnits

Shrink Factor specifies the factor by which each edge has to be shrunk. Environment Variable: blockAnnotationShrinkFactor

Rotation specifies the block orientation information. Environment variable: blockAnnotationRotation

Display Options section lets you customize the following text settings:

Text Size sets the size of block annotations to one of the following values:

Auto calculates the annotation size such that all annotations are inside the instance boundary.

Custom provides a slider to drag to the required value.

Environment variable: blockAnnotationTextSize

Related Topics

Block Annotations

© 2020 All Rights Reserved.

Block Placer

Use the **Block Placer** form to automatically place all hard and soft blocks in a design and minimize the wire length and chip area.

General tab lets you define the chip dimensions, cost order, and other options.

Chip Dimensions section has the following options:

Fixed Die specifies that the block placer cannot alter the chip dimensions. This option is selected by default if the Boundary Area constraint is unavailable in the design.

Minimum I/O to Core Distance specifies the distance between the IOBox and the coreBox (the spacing between the pads and the blocks).

Minimum Boundary to Instance Distance specifies the minimum distance between the design boundary and instance edge. This option is available only when there are no IO pads in the design.

Specify Chip Dimension Range specifies the chip size with respect to one of the following:

Width and Height activates the *PRBoundary Width Range* and *PRBoundary Width Range* options.

Area and Aspect Ratio activates the *PRBoundary Area Range* and *PRBoundary Aspect Ratio (W/H) Range* options.

Cost Order section specifies the order that the block placer has to follow to minimize cost. The cost with the highest priority is minimized first.

Default Cost Order restores the original cost order.

Options section specifies the following placement options:

Default Minimum Distance between Instances specifies the default minimum distance between instances.

Optimize Top-Level Pins specifies whether the block placer has to optimize the top-level pins after block placement.

Effort specifies the effort that block placer should make before arriving at the possible solutions.

Enable Multi Threading reduces the time taken by the block placer to place a large number of macros.

© 2020 All Rights Reserved.

Enable Congestion Based Refinement is enabled only if congestion data is available. When this field is enabled, the congestion value is taken into account during block placement.

Number of Iterations specifies the maximum number of iterations that can be performed between the congestion-based refinement block placer and the Global Router to bring congestion within the limits.

Constraints tab specifies the constraints that the block placer has to use to converge on a solution.

Soft Block Constraints tab specifies the side constraints for soft blocks.

Soft Block Size Constraints table lists all the soft blocks in the design cellview.

Area specifies the area of the selected soft block.

Minimum AR and **Maximum AR** specify the minimum and maximum aspect ratio of the selected soft block.

Update All applies changes to all soft blocks.

Update Selected applies changes to only the selected soft blocks.

Reload Constraints from Constraint Manager reloads constraint values from the Constraint Manager.

Instance Constraints tab specifies the instance constraints for soft blocks.

Instance To Instance Edge Distance section specifies the distance constraints between any two instances.

Instance to Instance Distance table is populated from the Instance to Instance Distance constraints present in the Constraint Manager.

Instance1 and **Instance2** list all the instances in the design. You can select the instances on which the minimum and maximum distance constraint are to be applied.

Minimum and **Maximum** specify the minimum and maximum edge-to-edge distance values.

Instance To Boundary Distance Constraints section specifies the distance constraints between an instance and the design boundary.

Instance to Boundary Distance table lists all the Instance to Boundary Distance constraints present in the Constraint Manager.

© 2020 All Rights Reserved.

Instance Name lists all instance names.

Minimum Distance and **Maximum Distance** specify the minimum and maximum instance-to-boundary distance values.

Boundary Edge specifies the edge to which the constraint is to be applied.

Reload Constraints from Constraint Manager reloads constraint values from the Constraint Manager.

Relative Constraints section specifies the order and alignment constraints to be applied.

Order/Alignment table specifies the alignment constraints to be applied.

Reference Instance specifies the reference instance for calculating the order and alignment.

Alignment Side specifies the edge for which the constraint is to be applied.

Relative Position specifies the relative position of the target instance with respect to the reference instance.

All Instances lists all the instances in the design except the fixed and locked instances. You can select multiple instances in this field and move them to *Target Instances* by using the up arrow button.

Add creates a new constraint.

Update modifies the selected constraint.

Delete deletes the selected constraint.

Related Topics

Place Blocks

© 2020 All Rights Reserved.

Compare Pin Group Guides

The **Compare Pin Group Guides** form lets you compare the pin group guides in two different cellviews.

Library Name specifies the source library that contains the required pin group guide for comparison.

Cell Name specifies the source cell that contains the required pin group guide for comparison.

View Name specifies the source view that contains the required pin group guide for comparison.

Related Topics

Comparing Pin Group Guides

© 2020 All Rights Reserved.

Corner Pad Placement

Use the **Corner Pad Placement** form to insert corner cells in the four corners of the design.

Upper Left, **Upper Right**, **Lower Left**, and **Lower Right** refer to the four corners of the design in which corner cells are to be inserted.

Instance Name specifies the instance to be inserted.

Orientation specifies the orientation of the selected instance.

Related Topics

Corner Cells

© 2020 All Rights Reserved.

Create I/O Row

Use the **Create I/O Row form** to create IO rows of PAD type siteDefs available in technology.

Rows Per Side specifies the number of I/O rows to be created. The available options are **One Row** and **Two Rows**.

Edges lists the edges along which rows are to be created.

Row *number* specifies the row for which parameters are to be applied.

Site Pattern lists the patterns defined in the technology file.

Site Orient is the relative orientation of the pad inside the row.

Related Topics

Row Creation

© 2020 All Rights Reserved.

Create Pin From Label

Use the **Create Pin From Label** form to create pins from labels and text displays in the layout cellview.

Scope section defines the scope for seracing labels

Label in specifies the scope for selecting labels. The available options are:

Cellview considers all top-level labels in the current cellview as target labels.

Selected considers the selected labels in the current cellview as target labels.

LPP Filter restricts the search only to the specified layer-purpose pairs. All other LPPs are ignored.

Consider Instance Labels includes instance labels in the search results.

Shape Search section defines options for searching for labels.

Search Depth specifies the level up to which shapes must be searched. For example, if *Search Depth* is set to 4, then levels 0 through 4 are searched for the required shapes. The default value is 0, which refers to the top-level (level 0).

Select Shape specifies the shape that must be used for pin creation - First or Largest. First indicates that the first shape that matches the specified criteria must be used. The search is stopped as soon as a suitable shape is found. When set to Largest, all matching shapes are retrieved; then the shape with the largest area is identified and used.

Use Shape On specifies the layer property based on which the search results must be filtered out—**Metal or Poly Layer** or **Same Label Layer**.

Apply Search Depth for Instance Labels uses the specified *Search Depth* for instance labels.

Create Pin if Connectivity Exists creates pins only for the labels that are associated with existing nets, terms, or pins in the current cellview. All other labels are ignored.

Pin Attributes section defines settings for the pins to be generated.

Pin Size specifies the size and shape of the pins to be generated. The available options are:

Auto creates square pins inside the shape. The smaller dimension (width, height) is chosen as the pin side of the square pins. Pins are created along the edge of the PR boundary, so that they are accessible for defining connectivity.

© 2020 All Rights Reserved.

Whole Shape creates pins of the same shape as the labels.

Custom (X and Y values) lets you specify the size of the pins by specifying the width and height values. The same size is applied to all output pins.

Pin Layer specifies the layer on which pins are to be generated. The available options are:

Same as Shape LPP creates pins on the same layer as the overlapping shape.

Same Layer Custom Purpose uses the same layer as the overlapping shape, but lets you specify the purpose.

Custom LPP lets you specify both the layer and purpose on which pins must be created.

Attach Label to the Pin attach the labels to their corresponding pins.

Virtual Connect

Create Whole Shape Pin for Labels in the **Power** section creates power pins of the same shape as their corresponding labels. Click **Define** to display a list of labels. Select the labels for which power pins must be created.

Show in the **Preview** section displays a preview of the pins that would be created.

Hide closes the preview.

Related Topics

Creating Pins from Labels

© 2020 All Rights Reserved.

Create Pin From Net Shapes

Use the **Create Pin From Net Shapes** form to create pins from existing net shapes in the layout cellview. Only top-level pins can be created using this method.

Net in specifies whether net shapes are to be searched in all or selected nets in the current cellview. The available options are **CellView** and **Selected**.

Consider Net Shape on Routing Layers Only restricts the search to the net shapes that are located on routing layers.

Create Pin on Top Layer Only creates pins only on the top layer shape.

Create Pin on Connected Shapes restricts the search to those net shapes that are connected to top-level nets.

Consider Level-1 Connected Shapes includes net shapes that are connected to level-1 nets in the search. This option is available only when the design has level-1 shapes.

Check for Duplicate Pins checks for existing pins with the same names to ensure that duplicates are not created.

Schematic Aware ensures that pins are created only for those nets that have corresponding terminals in the schematic view. Pins are not created for internal nets.

Create Label creates labels over pins.

Options displays the Set Pin Label Text Style form, which can be used to define pin label settings.

Pin Layer section specifies the LPP on which the pins must be created. The available options are:

Same as Shape LPP creates pins on the layers on which the net shapes are located.

Same Layer, Custom Purpose uses the same layer as the net shape, but lets you specify the purpose.Custom LPP specifies the layer and purpose on which pins must be created.

Show in the **Preview** section displays a visual depiction of how the pins would be created; actual pins are not created yet.

Hide closes the preview.

© 2020 All Rights Reserved.

Related Topics

Creating Pins from Net Shapes

© 2020 All Rights Reserved.

Electrically Aware Pin Sizing

(ICADVM20.1 EXL Only) Use the Electrically Aware Pin Sizing form to identify pins that do not meet electrical requirements and automatically fix the violations.

Run On specifies the pins on which the command is to be run.

All runs the command on all top-level and level-1 pins

Selected runs the command only on the selected pins.

Find and View under-sized pins identifies violating pins.

Select Pin selects all the detected under-sized pins. You can perform further actions on the selected pins, for example, you can move or resize them.

Highlight Pin highlights the under-sized pins for better visibility.

Highlight Violating Edge highlights only the violating edge of under-sized pins.

Show edge dimension (actual vs desired) displays the current edge dimension and the desired electrically-aware dimension of the violating edge.

Resize under-sized pins fixes the pin violations.

Violating edge only resizes only the violating pin edges to the desired electrically-aware width.

Retain pin selection retains the selection of pins after they are resized.

Retain highlight on original pin retains the highlights on the pins that were selected for resize.

Maintain abutment to PR-Boundary ensures that pins that are initially abutted to the PR boundary retain the abutment even after they are resized. Pins that are not initially abutted remain at the same position by maintaining the center of the pins at the same location.

Show in the Preview section displays a preview of the pins after they are resized before updating the layout.

Hide closes the preview.

Related Topics

Making Pins Electrically Aware

© 2020 All Rights Reserved.

Floorplan Global Options

Use the **Floorplan Global Options** form to set global options for either the current cellview or the current session.

Macro Coloring section provides options to assign different highlight colors for hard and soft blocks so that they are visually distinguishable in the layout canvas.

> **Enable macro coloring scheme** enables color coding of hard and soft blocks. **Environment variable:** `colorOrNot`
>
> **Soft block color** specifies the highlight colors for soft blocks. **Environment variable:** `softBlockColor`
>
> **Hard block color** specifies the highlight colors for hard blocks. **Environment variable:** `hardBlockColor`

Generate Physical Hierarchy section lets you specify which objects are to be automatically re-generated when the physical hierarchy is generated or the components and nets are updated.

> **Create Instances** specifies that all instances are to be re-generated. **Environment variable:** `initCreateInstances`
>
> **Create Pins** specifies that all pins are to be re-generated. **Environment variable:** `initCreatePins`
>
> **Create Boundary** specifies that the boundary is to be re-generated. **Environment variable:** `initCreateBoundary`

Allow Pin Resizing in the **Pin Snapping** section specifies whether pins are to be resized during pin snapping. **Environment variable:** `allowPinResizing`

Pin Placement section defines the following settings:

> **Respect layer priority for valid routing layers constraint on net** specifies that the layer priority must be followed while placing pins during pin optimization.
>
> **Default blockType (when unknown)** specifies the default block type for soft blocks when the assigned block type is not custom or digital. The available options are:
>
> > **Custom** places pins and PR boundary on the manufacturing grid.
> >
> > **Digital** places pins on the routing grid and the PR boundary on the placement grid.

© 2020 All Rights Reserved.

As-Is (XY Snap Spacing) specifies that the pins and PR boundary must placed on the grid defined by the *X Snap Spacing* and *Y Snap Spacing* options in the Grid Controls section of the Display Options form.

Environment variable: defaultBlockTypeWhenUnknown

Label Layer Purpose Pair Update section controls the re-layering of pin labels when their associated pins are re-layered.

Update Label Layer without updating Label Purpose (default) re-layers labels to match the pins, but their purposes remain unchanged. Environment variable: labelLayerPurposePair

Define Custom Layer Purpose Pair defines the layer and purpose to which labels are to be moved. To specify custom layers and purposes:

Options opens the Generate Physical Hierarchy form.

(ICADVM20.1 Only) **Enable Deep Hierarchy Editing of Soft Blocks** enables viewing and selection of soft-block PR boundary, instances, and pins from any hierarchical level.

Related Topics

Floorplan Global Options

© 2020 All Rights Reserved.

Generate Physical Hierarchy

Extract Top Level Design reads the value from the `extractAfterGenerateAll` environment variable and lets you specify whether the connectivity extraction for top block only and control the extraction for the remaining of the soft block is to be done through the environment variable.

Generate options specify which soft blocks are to be configured. The available options are: *all* and *none*.

Soft Block List provides information about the soft blocks to be generated in the layout canvas.

Related Topics

Physical Hierarchy Generation

© 2020 All Rights Reserved.

Insert Filler

Use the **Insert Filler Cells** form to insert filler cells in the empty spaces between PADs in IO rows.

Region Selection Mode section specifies the mode for running the command.

Entire Design specifies that the filler cells are to be inserted in all IO rows of the design.

Selected Region specifies that filler cells are to be inserted in only in the IO rows that are present in the selected region. Use one of the following methods to specify the region:

Draw lets you select an area of the design in the canvas. The values are automatically populated in the **Coordinates** field.

Coordinates defines the coordinates of the selected region. You can manually edit the values.

Zoom lets you zoom the selected region.

Select Filler Cells section specifies the cells to be used as filler cells.

Library Name, **Cell Name**, and **View Name** specify the cellview that contains the required cells. Click *Browse* to display the Library Browser window to select the required lib:cell:view names.

Add Filler adds the selected lib:cell:view name as a filler and displays it in the filler table.

Remove Filler deletes the selected lib:cell:view from the filler table.

Filler Instance Attribute section defines the instance name prefix and the placement status of filler cells.

Instance Name Prefix specifies a prefix for the physical-only filler instances.

Mark Fillers as Fixed specifies the placement status of the filler cells as `Fixed`.

Related Topics

Filler Cells

© 2020 All Rights Reserved.

I/O Pad Placer

Use the **IO PAD Placement** form to place PADS on IO rows.

Style section lists the supported styles.

Connectivity Based places PADs in the rows minimizing the net length between the PADs and the placed blocks in the design.

Non Connectivity Based distributes the PADs around the periphery of the design boundary without considering the connectivity between PADs and blocks placed in the design.

Spacing section defines the spacing between the PADs in a given row.

LEFT, **RIGHT**, **TOP**, and **BOTTOM** refer to the row sides.

None specifies that all the placement constraints are to be turned OFF by default and pads placed with spacing value `0`.

Even distributes PADs evenly (with same spacing between PADs) in the designated row.

Abut guides the placement engine to abut the PADs in the designated row. The selection of abutment will depend upon the row side.

Custom specifies a custom spacing value. The placement engine honors this value while placing PADs.

Related Topics

PAD Placement

© 2020 All Rights Reserved.

Label Layer Purpose

Use the **Label Layer Purpose** form to specify the layer and purpose to which labels are to be moved when their associated pins are moved.

Layer Name specifies the name of the layer to which the labels are to be moved. The available options are:

Custom layer specifies the layer to which labels are to be moved. Irrespective of the pin layer, all labels are moved to the selected layer. **Environment variable:** `pinTextLayer`

Same as Pin moves labels to the same layers as the pins. **Environment variable:** `pinTextSameLayer`

Layer Purpose specifies the purpose of the layer to which the labels are to be moved. The available options are:

Custom purpose specifies the purpose to be assigned to labels when the associated pins are re-layered. Environment variable: `pinTextPurpose`

Same as Pin matches the label purpose to that of the pins after they are re-layered. **Environment Variable:** `pinTextSamePurpose`

Same as Label retains the original label purpose. Environment variable: sameLabelPurpose

Related Topics

Updating Label Layer Purpose Pairs

© 2020 All Rights Reserved.

Label Update

Use the **Label Update** form to update pin labels and text displays for clearer visualization.

Mode specifies the pins on which the command has to run.

All runs the command on all soft block pin labels and text displays in the current design.

Selected runs the command only on the selected pin labels and text displays.

Make Label Perpendicular To Nearest Edge in the **Adjust Label Orientation** section makes labels and text displays perpendicular to the nearest prBoundary edge. The nearest prBoundary edge is determined based on the minimum distance of the label's parent pin center from the prBoundary.

Adjust Label Position section Includes the following options:

Pull Label Inside P&R Boundary places labels and text labels inside the PR boundary. The label or text display origins remain unaltered.

Move Label Origin to Pin Edge Nearest to Boundary moves all labels and text displays to the center of the nearest pin edge.

Adjust Label Size section includes the following options to resize labels:

Update Label Size specifies the label size.

Auto Reduce Label Size computes any existing label overlaps and reduces the size of the labels or text displays so as to remove any remaining overlaps.

Minimum Label Size specifies the minimum threshold value for reducing the size of labels.

Related Topics

Label Update GUI Options

© 2020 All Rights Reserved.

Load Solutions

Use the **Load Solutions** form to load a design solution that is generated by the block placer.

Load Solution lists the design solutions that are available in the directory specified by the BPDIR variable (or bpLogs directory within the testcase directory).

Related Topics

Load Solutions

© 2020 All Rights Reserved.

Merge and Must Connect Pins

You can use the **Merge and Must Connect Pins** form to merge two or more overlapping rectangular, rectilinear, or path pin shapes into a single rectilinear shape.

Mode specifies the pins to be selected for merge.

All applies to all pin shapes.

Selected applies to the selected pin shapes.

Related Environment Variable: mergePinsSelMode

Pin Connectivity specifies the connectivity settings for the merged pin.

Must Connect Pins Without Creating New Nets (default) specifies that the connectivity of the pin shapes is to be retained. The must connect all pins setting is applied to the existing terminals.

Must Connect Pins Using New Nets specifies that new nets, terminals, and pins are to be created for each pin shape island. The new terminals are connected and are placed together in a must connect group.

Strongly Connect Pins specifies that a single connection point on a highly conductive metal is to be used to provide the required connectivity.

Related Environment Variable: mergePinsConnMode

Suffix String specifies the suffix to be added to the new nets, terminals, and pins that have been created on the merged pin islands. The default value is _MJ, which is applied when Must connect pins using new nets is selected.

Related Environment Variable: mergePinsSuffix

Related Topics

Merging Pins

© 2020 All Rights Reserved.

Pin Checker (ICADVM20.1 Only)

Use the **Pin Checker** form to run various checks on pins and report the results.

Check Two Pin Nets section lists checks that are applicable only to the nets that are connected to two pins:

Mis-aligned checks whether the connected pins are correctly aligned.

Layer mismatch checks whether the connected pins are on the same layer.

Size mismatch checks whether the dimensions of the connected pins are the same.

Abut checks whether the connected pins are abutted to each other.

Overlap checks whether the connected pins overlap each other.

Check All Nets section lists checks for all nets. Violations are reported in the Pin Check Report displayed in the CIW.

Multi pin nets identifies nets that have multiple pins and lists their names in the Pin Check Report.

Pin Threshold restricts the check to nets that have the specified number of pins.

Pin layer direction mismatch with side identifies the pins for which pin layers do not have tracks in the specified pin side. This option is applicable only to pins that have side layer constraints.

Pin Check Report section lets you customize the information displayed in the Pin Check Report.

Summary generates a summary report about the checks performed by the Pin Checker.

Detailed generates a detailed report with information about nets as per the above settings.

Related Topics

Running the Pin Checker (ICADVM20.1 Only)

© 2020 All Rights Reserved.

Pin Group Guide

Use the **Pin Group Guide** form to create pin groups and guides interactively.

Group Name specifies the name of the new pin group guide.

New generates a unique group name.

Delete deletes the selected pin group guide.

Refresh loads default values from the database, technology file, and constraint.

Preserve Order specifies that the order of pins in the pin group constraint are to be honored and pins are to be optimized in that sequence.

Foreign Pins Restrictions section includes the following options:

> **Inclusive** honors foreign pins present within the pin group guide area. Pin group guides with different pins can overlap in the layout design.
>
> **Exclusive** does not honor foreign pins present within the pin group guide area.

Group Constraints section includes the following options:

> **Valid Layers** specifies valid layers for pin placement.
>
> **Layer Priority** specifies that pins are to be optimized in the order in which they are listed in *Valid Layers*.
>
> **Pin-Pin Spacing** specifies the spacing between adjacent pins on each layer.
>
> **Keepout Spacing** defines the area around the pin within which no object can be placed.
>
> **Pin Width** specifies the width of pins for each layer.

Pins tab provides the following options:

> **Top-Level** lists only top-level pins in the pin browser.
>
> **Soft Block** lists only the soft block pins in the pin browser.
>
> Pin browser enlists all the shape pins found in the current scope.
>
> Pin group table lists the pins added to the current pin group.

Guide tab provides the following options:

> **Create** generates a pin group guide as per the specifications.

© 2020 All Rights Reserved.

Partial Edge creates pin group guides that cover partial edges. You can specify the first and last points of the guide.

Full Edge creates pin group guides that cover the edges completely. You can click the required edges in the design canvas.

> **Offset** specifies the Start and End offset values of the pin group guide on the corresponding edges. This option is applicable only to full edge pin group guides.

Box lets you draw one or more areas in the layout window. Pin group guides are created on the PR boundary edges that intersect the selected boxes.

> **Split into edge guides during create** specifies that the top-level and soft block edges that intersect the box are to be created as the partial- or full-edges for creating pin group guides.

Free Area places pins in the free area within the PR boundary of the top level or instance level (level-1) from the top level.

Constraints Only only updates the constraints.

Preview updates the constraints runs pin planner.

Optimize updates the constraints runs pin optimizer.

Mode specifies whether the above operations are to be performed on *All* or *Selected* pins.

Related Topics

Creating a Pin Group Guide

© 2020 All Rights Reserved.

Pin Group Guide - Place as Schematic/Symbol

Use the **Pin Group Guide - Place as Schematic** form to generate pin group guides and place pins as per their relative positions in the schematic or symbol view.

Schematic specifies that the pins in the pin group guide are to be placed as per their counterparts in the corresponding schematic view.

Symbol specifies that the pins in the pin group guide are to be placed as per their counterparts in the corresponding symbol view.

Library Name, **Cell Name**, and **View Name** specify the cellview in which the pin group guide is to be generated.

Related Topics

Placing Pin Group Guides as per Schematic or Symbol

© 2020 All Rights Reserved.

Pin Placement

Use the **Pin Placement** form to place and optimize pins your design.

Pin Planner tab specifies how pins are to be placed in the design.

Edit lets you specify the scope for running the Pin Planner.

Top Level Pins applies the form settings only to the top-level pins in the design.

Auto-Switch to Level-1 automatically zooms to the selected soft block and switches to Level-1 Pins mode when you click a soft block in the design canvas. This option is off by default, which means that when you click a soft block, the soft block is selected, but you remain at the top level.

Level-1 Pins applies the form settings to pins of the selected soft block in the design. Select the required soft block from the drop-down list.

Note: If you have already selected a block in the canvas, that block is automatically selected in the *Pin Planner* tab. If you have more than one block selected in the canvas, the first block you selected is the one selected in the form.

Filter filters the pins listed in the form. Use a combination of the two pull-downs to filter the pins that are displayed. Alternatively, you can use the *By Name* field to filter the pins by name.

Edge filters pins based on the edge to which they are assigned. The available edge options are *All, Left*, *Top*, *Right*, *Bottom*, *Level-1 pin*, *Top-Level Route*, *To Hard Block Multi Pin*, and *Hard Block Multi Pin*.

Status or Type filters pins based on their placement status or pin types. The available options are *All*, *Placed*, *Unplaced*, *Locked*, *Fixed*, *Ordered*, *Unordered*, *I*, *O*, and *IO*.

By Name filters pins based on their names. The default regular expression for this filter is "**.***" (dot asterisk)**,** which implies that a name filter is not applied. Therefore, all the pins in the design are displayed. Other supported regular expressions are:

" " (Blank): Is equivalent to the default regular expression and it implies that the name filter is not being applied. Displays all the pins.

"in.*": Displays all the pins that have names starting with "in".

".*in.*": Displays all the pins that contain "in" in their name.

".*in": Displays all the pins that have names ending in "in".

- The "in" in the valid regular expressions above can be replaced with any other expression, such as "VDD".

© 2020 All Rights Reserved.

- The filter string "***.***" is not a valid regular expression. Use a valid regular expression instead.
- The *By Name* field is case-sensitive.

Pin table lists all the shape pins found in the current scope. It does not list pins that are vias. The entries are expressed as *termName:pinName:figName* because the pin figure is the physical entity that you are placing. If a pin has more than two pin figures attached to it, but the pin figures have the same name, they are represented by a single entry in the pin table.

The other columns in the pin table are the *Edge* to which the pin is assigned, the *Order* in which the pins are placed, and the *Status*, *Type*, *Layer*, *Width*, *Height,* and *Connectivity* (ICADVM20.1 Only) of the pins. You can sort the table by any column by clicking the relevant column header. Click again to reverse the sort order.

Message provides a hint about your next action.

Attributes applies the following pin attributes to the pins in the pin table.

Edge specifies the edge on which the selected pins are to be placed. The options are *Left*, *Top*, *Right*, *Bottom*, *Level-1 pin*, *Top-Level Route*, *To Buried Pin* (ICADVM20.1 Only), *Level-1 Route*, *To Hard Block Multi Pin*, *Any*, and *As is*. The *As is* setting allows the edge settings of pins to remain intact when changing the *Order* or *Placement Status Constraint* properties.

Note: Depending on the block type, the pin edges are snapped to either the manufacturing grid or the routing grid. If the block type is custom, the pin edges are snapped to the manufacturing grid. If the block type is digital, the pin edges are snapped to the routing grid.

Order specifies the order in which the selected pins are to be placed. You can specify an order only if you have assigned the pins to an edge.

Note: To place the pins unordered, leave the field blank or type "-". To specify an order for the pins, enter a value.

Placement Status Constraint specifies the status to be assigned to the pins after they are placed.

Lock at Placed Location places pins at an appropriate locations on the boundary and then locks them at that location. A locked pin cannot be moved either by the automatic placement functions or manually by the user.

Fix at Placed Location places pins at the appropriate location on the boundary and fixes them at that location. A fixed pin cannot be moved by the automatic

© 2020 All Rights Reserved.

placement functions, but can be moved manually, for example, using the *Edit – Move* command.

No Constraint (Floating) specifies that the selected pins will neither be locked nor fixed after they are placed. A *no constraint*, floating pin can be moved freely by the automatic placement functions and by using manual editing functions.

As is does not change the status of the selected pins after placement. Use this option to preserve differing status settings on the selected pins.

Shape specifies the following pin shape attributes:

Layer specifies the layer on which the pin shapes are to be drawn. Layer purpose names are filtered dynamically as you type a value in the *Layer* field.

By default, the width, height, and layer of the pin selected in the pin table is displayed. When you change the layer, Pin Planner checks whether the pin width and height values are valid. These values cannot be lesser than the layer `minwidth`. Therefore, when you select a different layer, if the pin width and height values retrieved are equal to or greater than the layer `minwidth`, the values are displayed in the form. If the pin width and height values are less than the layer `minwidth`, the layer `minwidth` value is displayed in the corresponding field.

Width specifies the width of the pin shape to be generated.

Height specifies the height of the pin shape to be generated.

Re-Place Pins specifies which pins are to be updated with the new attributes when you click *Apply*.

All Edges specifies that pins on all edges can be moved by the placer.

Selected Edge specified that only the pins assigned to the selected edge can be moved. All other pins are fixed in their current position.

Selected Pins means that only the selected pins can be moved by the placer when you click *Apply*. All other pins are fixed in their current position.

Apply applies the specified pin attributes to the selected pins.

Pin Spacing specifies how the selected pins are to be spaced. Select the required pins, set the spacing attributes, and click one of the spacing buttons depending on which of the selected pins is to be assigned as the reference pin. Pins are spaced only by clicking a spacing button in this section and not by clicking *Apply* in the *Attributes* section.

Value specifies the distance by which the selected pins are to be spaced.

Spacing type specifies how the spacing value is to be applied.

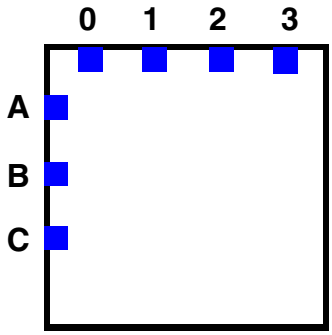
© 2020 All Rights Reserved.

center-center applies the spacing value to the center lines of each of the pin shapes to be spaced.

edge-edge applies the spacing value to the edges of the each of the pin shapes to be spaced.

Update Constraints automatically updates the constraints that are applicable to the design (for example, alignment constraints on the selected pins) based on the settings in the *Pin Spacing* section. If a pin to be spaced has the alignment constraint set, the placer reads the constraint to determine the appropriate access direction for the pin.

Space from LEFT/BOTTOM pin uses as its reference pin the leftmost pin on the top and bottom edges (**0** in the example below) and the bottom pin on the side edges (**C**).



Space from CENTER pin uses the center pins on all edges as reference. On the left edge in the example above, this is pin B. On the top edge, the pin spacer divides the total number of pins by 2 and uses the result as the reference pin (2).

Space from RIGHT/TOP pin uses the rightmost pin on the top and bottom edges (3) and the top pin on the side edges (A).

Place As In provides the following options:

Schematic places pins in the same relative positions as in the schematic view.

Symbol places pins in the same relative positions as in the schematic view.

Iterated Pins specifies the way iterated pins are to be listed.

Interleave interleaves the bits of the selected iterated pins by applying an order constraint that specifies how the individual bits are to be placed.

Uninterleave reverses the interleaving of the selected iterated pin bits.

Expand shows all the individual bits of iterated pins.

Collapse shows bus notations for iterated pins.

© 2020 All Rights Reserved.

All performs the *Expand* or *Collapse* function on all the pins in the list box.

Selected performs the *Expand* or *Collapse* function only on the selected pins.

Change Order lets you change the order of pins in the pin table.

Arrow left/down moves the selected pins down in the ordered list.

Arrow right/up moves the selected pins up in the ordered list.

Swap order inverts the order of two selected pins in the list.

Create lets you create horizontal and vertical rail pins.

VRail elongates the selected pins vertically from the top edge of the boundary to the bottom.

HRail elongates the selected pins horizontally from the left edge of the boundary to the right.

Zoom section lets you zoom the display in the canvas to the currently-selected pins.

Zoom ON enables the zoom functionality. This makes the slider active. You can then use the slider to set the desired zoom level. The acceptable zoom range is between 0 and 4. Level 0 provides the minimum zoom and level 4 provides the maximum zoom.

Alternatively, you can use the environment variable, `vpaAutoZoomEnable`, to enable the auto zoom functionality. This ensures that the zoom functionality is automatically enabled when you open the layout view. To set the desired zoom level, you can use the environment variable, `vpaAutoZoomLevel`.

If you set the auto zoom level to a value less than 0 when using the `vpaAutoZoomLevel` environment variable, the minimum zoom level will automatically default to 0. Likewise, if you set a maximum value higher than 4 using the environment variable, the maximum zoom value will automatically default to 4. The zoom range lies between 0 to 4 irrespective of the method you use to set the zoom level.

Pin Optimization tab lets you to position pins to obtain the shortest possible net length.

Mode specifies the pins to be optimized.

All optimizes all pins in the design.

Selected optimized only the selected pins. In this mode, you can select **High Effort** to get better results.

Environment variable: pinOptMode, selectedOnly

Pin Spacing section specifies the minimum spacing between the pins.

© 2020 All Rights Reserved.

Use Table Spacing scales up the spacing values as per the table spacing rules defined for different widths of the pins.

Auto indicates that the minimum spacing values are to be based on the line-line spacing mode.

Custom enables the **Options** and **Advanced Options** buttons. Click *Options* to display the Pin Spacing Options form, which lets you specify the pin spacing values for each layer. Click *Advanced Options* to display the Pin Spacing Advanced Options form.

Environment variable: customPinGrid

Pin Offset specifies the distance between the adjacent side and the center point of the first pin.

Auto places the first pin on a grid such that it is aligned to the corner pins of the adjacent side and that there is no spacing violation.

Custom enables the **Options** button. Click this button to display the Pin Offset Options form, which provides options to specify the pin offset values for all layers in the design. For more information, see Pin Offset.

Environment variables: offSetByGrids, offSetBySpacing, offSetMode

Target Pin Side specifies the **Side** of the block or the top-level PR boundary on which pins are to be placed. The valid values are: **Left**, **Right**, **Top**, and **Bottom**. For more information, see Target Pin Side. Environment variable: targetPinSide

Pin layer/side constraints provides options to switch the pin layer depending on the side of the block on which the pin is placed by Pin Optimizer. For more information, see Pin Layer/Side Constraints. Environment variables: enablePinLayerSideConstraints, bottomSidePinLayersConstraints, leftSidePinLayersConstraints, rightSidePinLayersConstraints, topSidePinLayersConstraints

Match Fixed Pins Layer/Size provides options to change the layer and purpose of the 'to be placed' pins to match the layer and purpose of the connected fixed pins. For more information, see Match Fixed Pins Layer/Size.

Place Rectangular Pin is applicable only to rectangular pins. It specifies the pin edge to be abutted to the PR boundary. The valid values are: **As Is**, **Short Edge**, and **Long Edge**. For more information, see Place Rectangular Pin. Environment variable: rectangularPinEdgeOnPRB

(ICADVM20.1 Only) Honor Current Pin Location snaps pins (selected or all) to their nearest WSP tracks by moving them by a minimum distance from their current position. For

© 2020 All Rights Reserved.

more information, see (ICADVM20.1 Only) Snapping Pins to WSP Tracks. Environment variable: honorCurrentPinLocation

(ICADVM20.1 Only) Congestion Aware runs Pin Optimizer in the congestion-aware mode. For more information, see Congestion-Aware Pin Placement (ICADVM20.1 EXL Only). Environment variable: enableCongestion

Optimize runs Pin Optimizer.

Related Topics

Pin Planning

Pin Optimization

© 2020 All Rights Reserved.

Pin Spacing Options

Use the **Pin Spacing Options** form to define pin spacing values for each layer.

Spacing type provides the following spacing options:

line-line uses the minimum spacing between two wires on a layer for spacing pins.

line-via uses the minimum spacing between a wire and a via on a layer for spacing pins.

via-via uses the minimum spacing between two vias on a layer for spacing pins.

specify enables you to enter pin spacing values for each layer in the current design manually.

Environment Variables: spacingType, spacingValue

Update All specifies the same pin spacing value for all layers.

Define Pin Spacing (Edge to Edge) specifies the pin spacing values for all layers in the design. Environment Variable: perLayerSpacingList

Related Topics

Pin Optimization

© 2020 All Rights Reserved.

Pin Spacing Advanced Options

Use the **Pin Spacing Advanced Options** form to define pin grids. A pin grid defines the points at which pins need to be placed. Each layer has a separate pin grid.

Choose width for Grid section specifies whether the width of the grid should correspond to the **Width of the Smallest Pin** in the design or to the **Min Width from Technology File** (default). Environment variable: widthForPinGrid

Pin Grid Adjustment Factor section provides options to specify the value by which the pin grid needs to be scaled for each layer. Specify the adjustment factor for each layer individually in the following fields.

Define Pin Grid Adjustment Factor specifies that the pin grid adjustment factor can be specified for individual layers. Environment variable: perLayerAdjustmentFactorList

Custom Pin Grid section specifies the absolute values for the width of the pin grid for each layer. Specify the value for each layer individually in the following fields.

Define Custom Pin Grid lets you specifies that the custom pin grid can be specified for each layer individually. Environment variable: perLayerCustomPinGridList

spacingMode, spacingType. spacingValue

Related Topics

Pin Optimization

© 2020 All Rights Reserved.

Pin Offset Options

Use the **Pin Offset Options** form to define pin offset values in terms of **Spacing**, **Grids**, or both.

Spacing specifies the absolute space between the cell boundary and the first pin. Specify the spacing value for each layer individually in the following fields. Environment variables: offSetBySpacing, perLayerOffsetBySpacingList

Grids specifies the number of grids to skip before placing the first pin. Manufacturing grids apply for custom pins, and routing grids apply for digital pins. Specify these values for each layer individually in the following fields. Environment variables: offSetByGrids, perLayerOffsetByMfgGridList, perLayerOffsetByRoutingGridList

Related Topics

Pin Optimization

© 2020 All Rights Reserved.

H

Layout XL Floorplan Environment Variables

This appendix provides information on the names, descriptions, and graphical user interface equivalents for Pin Optimizer environment variables.

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

Important

(ICADVM20.1 Only) In Layout EXL, *Floorplan* is available under the *Plan* menu.

List of Floorplanner Environment Variables

- autoGenHierCreateView
- autoGenHierPhysConfig
- autoGenHierDepth
- autoGenHierDefaultArea
- autoGenHierPinOptMode
- autoGenHierRunPinOpt
- blockAnnotationArea
- blockAnnotationAreaUnits
- blockAnnotationInstName
- blockAnnotationRotation
- blockAnnotationSelMode

© 2020 All Rights Reserved.

- blockAnnotationShrinkFactor
- blockAnnotationTextLPP
- blockAnnotationTextSize
- bottomSidePinLayersConstraints
- customPinGrid
- defaultBlockTypeWhenUnknown
- enableCongestion
- enableNoPinsBetweenBusPins
- enablePinLayerSideConstraints
- enableTrackSkipping
- honorCurrentPinLocation
- ignoreConnectivityBusPinsAlign
- labelLayerPurposePair
- layerChangeResize
- layerPriority
- leftSidePinLayersConstraints
- matchLabelLayerWithPin
- numberOfSkipTracks
- offSetByGrids
- offSetBySpacing
- offSetMode
- optimizeInstancesPartiallyOutsidePRB
- perLayerAdjustmentFactorList
- perLayerCustomPinGridList
- perLayerOffsetByMfgGridList
- perLayerOffsetByRoutingGridList
- perLayerOffsetBySpacingList

© 2020 All Rights Reserved.

- perLayerSpacingList
- pinGridAdjustmentFactor
- pinOptMode
- rectangularPinEdgeOnPRB
- retainLongPin
- rightSidePinLayersConstraints
- sameLabelPurpose
- selectedOnly
- spacingMode
- spacingType
- spacingValue
- tableSpacingRule
- targetPinSide
- topSidePinLayersConstraints
- widthForPinGrid

© 2020 All Rights Reserved.

Pin Tool Environment Variables

- closePinToolSubForms

Auto-Create Pins

- autoPinCreateFigGroup
- autoPinPathSpineAware
- boundarySquareShapePin
- buriedMultipleMode
- buriedWholeShapePin
- customLPPField
- customPurposeField
- deletePinCheckbox
- deletePinSelection
- modeOption
- pinConnectivityMode
- pinLayers
- pinSizeSelection
- purposeField

Merge Pin

- mergePinsConnMode
- mergePinsSelMode
- mergePinsSuffix
- pinResizeMode
- pinResizeAllEstimators
- pinResizeCurrentEstimator
- pinResizeFiles

© 2020 All Rights Reserved.

Promote Pins

- promoteCreateLabelsOnPins
- promoteCustomPurpose
- promoteCustomLPP
- promoteDeleteOldPins
- promoteLayerChoice
- promoteOddEvenChoice
- promotePinConnectivity
- promotePinNetChoice
- promotePinNetList
- promotePinOnNetForSelInsts
- promotePinOnNetPickLevel
- promotePrefixChoice
- promotePrefixString
- promoteSchematicAware
- promoteSelectiveBusBits
- promoteSelectInCanvas
- promoteSuffixChoice
- promoteSuffixString
- promoteToChoice
- promoteTopLayerFig
- promoteToLevel
- promoteVisibleLPPs

© 2020 All Rights Reserved.

autoGenHierCreateView

`layoutXL.floorplan autoGenHierCreateView string` *`string_lpp`*

Description

Specifies the name of the new layout view.

The default value is `layout`.

GUI Equivalent

Command	*Floorplan – Auto Generate Hierarchy*
Field	*Create View* (Auto-Generate Hierarchy)

Examples

```
envGetVal("layoutXL.floorplan" "autoGenHierCreateView")
envSetVal("layoutXL.floorplan" "autoGenHierCreateView" 'string "layout02")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

autoGenHierPhysConfig

`layoutXL.floorplan autoGenHierPhysConfig string` *`string_lpp`*

Description

Specifies the name of the physical configuration view that provides information about the schematic-layout pairs.

The default value is `physConfig`.

GUI Equivalent

Command	*Floorplan – Auto Generate Hierarchy*
Field	*PhysConfig View* (Auto-Generate Hierarchy)

Examples

```
envGetVal("layoutXL.floorplan" "autoGenHierPhysConfig")
envSetVal("layoutXL.floorplan" "autoGenHierPhysConfig" 'string "physConfig2")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

autoGenHierDepth

`layoutXL.floorplan autoGenHierDepth int` *`int_depth`*

Description

Specifies the level up to which the hierarchy must be generated.

The default value is `31`, which is the maximum value.

GUI Equivalent

Command	*Floorplan – Auto Generate Hierarchy*
Field	*Generate Hierarchy to Level* (Auto-Generate Hierarchy)

Examples

```
envGetVal("layoutXL.floorplan" "autoGenHierDepth")
envSetVal("layoutXL.floorplan" "autoGenHierDepth" 'int 25)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

autoGenHierDefaultArea

`layoutXL.floorplan autoGenHierDefaultArea float` *`float_area`*

Description

Specifies the area within the boundary that must be filled.

The default value is `100`.

GUI Equivalent

Command	*Floorplan – Auto Generate Hierarchy*
Field	*Area Utilization* (Auto-Generate Hierarchy)

Examples

```
envGetVal("layoutXL.floorplan" "autoGenHierDefaultArea")
envSetVal("layoutXL.floorplan" "autoGenHierDefaultArea" 'float 25)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

autoGenHierPinOptMode

```
layoutXL.floorplan autoGenHierPinOptMode cyclic { "Top Down" | "Bottom Up" }
```

Description

(ICADVM20.1 EXL Only) Specifies the mode for running Pin Optimizer hierarchically on the blocks that were generated using the Auto-Generate Hierarchy form. The valid values are:

- `Top Down`: Pin Optimizer is first run on the level-0 and level-1 soft blocks to get the best possible net length. Any existing pins, for example those of any level-1 hard blocks, are recognized and the soft block pins are aligned to these pins. Next, the pin positions of the level-1 soft blocks are used to derive the pin positions inside them. This process continues until the leaf level is reached.
- `Bottom Up`: Follows the same behavior as described in `Top Down`, but runs in the opposite direction. Pin Optimizer is first run on pins of the leaf level layouts.

This option is effective only when autoGenHierRunPinOpt is set to `t`.

The default value is `Top Down`.

GUI Equivalent

Command	*Floorplan – Auto Generate Hierarchy*
Field	*Run Pin Optimization* (Auto-Generate Hierarchy)

Examples

```
envGetVal("layoutXL.floorplan" "autoGenHierPinOptMode")
envSetVal("layoutXL.floorplan" "autoGenHierPinOptMode" 'cyclic "Bottom Up")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

autoGenHierRunPinOpt

```
layoutXL.floorplan autoGenHierRunPinOpt boolean { t | nil }
```

Description

(ICADVM20.1 EXL Only) Specifies whether Pin Optimizer must be run after automatically generating the physical hierarchy.

When the value is set to `t`, additionally, ensure that autoGenHierPinOptMode is set to the required mode.

The default value is `nil`.

GUI Equivalent

Command	*Floorplan – Auto Generate Hierarchy*
Field	*Run Pin Optimization* (Auto-Generate Hierarchy)

Examples

```
envGetVal("layoutXL.floorplan" "autoGenHierRunPinOpt")
envSetVal("layoutXL.floorplan" "autoGenHierRunPinOpt" 'boolean t)
envSetVal("layoutXL.floorplan" "autoGenHierRunPinOpt" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

blockAnnotationArea

```
layoutXL.floorplan blockAnnotationArea boolean { t | nil }
```

Description

Displays information about the total block area in the block annotation.

The default value is `t`.

GUI Equivalent

Command	*Floorplan – Block Annotations*
Field	*Area* (Block Annotations)

Examples

```
envGetVal("layoutXL.floorplan" "blockAnnotationArea")
envSetVal("layoutXL.floorplan" "blockAnnotationArea" 'boolean t)
envSetVal("layoutXL.floorplan" "blockAnnotationArea" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

blockAnnotationAreaUnits

`layoutXL.floorplan blockAnnotationAreaUnits int` *`area_units`*

Description

Specifies a value for the total block area for the block annotation.

The default value is `1`.

GUI Equivalent

Command	*Floorplan – Block Annotations*
Field	*Area* (Block Annotations)

Examples

```
envGetVal("layoutXL.floorplan" "blockAnnotationAreaUnits")
envSetVal("layoutXL.floorplan" "blockAnnotationAreaUnits" 'int 2)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

blockAnnotationInstName

```
layoutXL.floorplan blockAnnotationInstName boolean { t | nil }
```

Description

Displays the instance names in the block annotation.

The default value is `t`.

GUI Equivalent

Command	*Floorplan – Block Annotations*
Field	*Instance name and Master LCV* (Block Annotations)

Examples

```
envGetVal("layoutXL.floorplan" "blockAnnotationInstName")
envSetVal("layoutXL.floorplan" "blockAnnotationInstName" 'boolean t)
envSetVal("layoutXL.floorplan" "blockAnnotationInstName" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

blockAnnotationRotation

```
layoutXL.floorplan blockAnnotationRotation boolean { t | nil }
```

Description

Displays the block orientation information in the block annotation.

The default value is `t`.

GUI Equivalent

Command	*Floorplan – Block Annotations*
Field	*Rotation* (Block Annotations)

Examples

```
envGetVal("layoutXL.floorplan" "blockAnnotationRotation")
envSetVal("layoutXL.floorplan" "blockAnnotationRotation" 'boolean t)
envSetVal("layoutXL.floorplan" "blockAnnotationRotation" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

blockAnnotationSelMode

```
layoutXL.floorplan blockAnnotationSelMode boolean { t | nil }
```

Description

Sets the block annotation display mode to *Selected*.

The default value is `nil`.

GUI Equivalent

Command	*Floorplan – Block Annotations*
Field	*Display Mode – Selected* (Block Annotations)

Examples

```
envGetVal("layoutXL.floorplan" "blockAnnotationSelMode")
envSetVal("layoutXL.floorplan" "blockAnnotationSelMode" 'boolean t)
envSetVal("layoutXL.floorplan" "blockAnnotationSelMode" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

blockAnnotationShrinkFactor

`layoutXL.floorplan blockAnnotationShrinkFactor float` *`shrink_factor`*

Description

Specifies the factor by which each edge of the block annotation must be shrunk.

The default value is `1`.

GUI Equivalent

Command	*Floorplan – Block Annotations*
Field	*Shrink Factor* (Block Annotations)

Examples

```
envGetVal("layoutXL.floorplan" "blockAnnotationShrinkFactor")
envSetVal("layoutXL.floorplan" "blockAnnotationShrinkFactor" 'float 3)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

blockAnnotationTextLPP

`layoutXL.floorplan blockAnnotationTextLPP string` *`string_lpp`*

Description

Specifies the layer based on which the block annotation text must be formatted.

The default value is `y9 drawing`.

GUI Equivalent

Command	*Floorplan – Block Annotations*
Field	*Text Layer* (Block Annotations)

Examples

```
envGetVal("layoutXL.floorplan" "blockAnnotationTextLPP")
envSetVal("layoutXL.floorplan" "blockAnnotationTextLPP" 'string "M2 drawing")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

blockAnnotationTextSize

`layoutXL.floorplan blockAnnotationTextSize integer` *`string_lpp`*

Description

Specifies the size of the block annotation text.

The default value is `4`.

GUI Equivalent

Command	*Floorplan – Block Annotations*
Field	*Text Size* (Block Annotations)

Examples

```
envGetVal("layoutXL.floorplan" "blockAnnotationTextSize")
envSetVal("layoutXL.floorplan" "blockAnnotationTextSize" 'int 7)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

bottomSidePinLayersConstraints

`layoutXL.floorplan leftSidePinLayersConstraints string` *`list_of_pin_layers`*

Description

Specifies the list of pin layers to be assigned to the bottom side.

`list_of_pin_layers` is a list of pin layers to be assigned to the bottom side. The list must be enclosed in quotation marks; for example:

```
"metal1 metal2 metal4"
```

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin layer/side constraints* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "bottomSidePinLayersConstraints")
envSetVal("layoutXL.floorplan" "bottomSidePinLayersConstraints" 'string "metal2
metal3")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

customPinGrid

```
layoutXL.floorplan customPinGrid boolean { t | nil }
```

Description

Specifies whether custom pin grids can be defined by specifying absolute grid width values for each layer.

The default value is `nil`.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Advanced Options - Custom Pin Grid* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "customPinGrid")
envSetVal("layoutXL.floorplan" "customPinGrid" 'boolean t)
envSetVal("layoutXL.floorplan" "customPinGrid" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

defaultBlockTypeWhenUnknown

```
layoutXL.floorplan defaultBlockTypeWhenUnknown cyclic {"Custom" | "Digital" | "As-
     Is (XY Snap Spacing)"}
```

Description

Specifies the default block type for blocks with no block type specified.

The default value is `Custom`.

GUI Equivalent

Command	*Floorplan – Floorplan Global Options*
Field	*Default blockType (when unknown)*

Examples

```
envGetVal("layoutXL.floorplan" "defaultBlockTypeWhenUnknown")
envSetVal("layoutXL.floorplan" "defaultBlockTypeWhenUnknown" 'cyclic "Digital")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

enableCongestion

```
layoutXL.floorplan enableCongestion boolean { t | nil }
```

Description

(ICADVM20.1 Only) Specifies whether the channel congestion must be considered while optimizing pins.

The default value is `nil`.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Congestion Aware* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "enableCongestion")
envSetVal("layoutXL.floorplan" "enableCongestion" 'boolean t)
envSetVal("layoutXL.floorplan" "enableCongestion" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

enableNoPinsBetweenBusPins

```
layoutXL.floorplan enableNoPinsBetweenBusPins boolean { t | nil }
```

Description

Prevents placing other pins between bus pins. The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL.floorplan" "enableNoPinsBetweenBusPins")
envSetVal("layoutXL.floorplan" "enableNoPinsBetweenBusPins" 'boolean t)
envSetVal("layoutXL.floorplan" "enableNoPinsBetweenBusPins" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

enablePinLayerSideConstraints

```
layoutXL.floorplan enablePinLayerSideConstraints boolean { t | nil }
```

Description

When set to `t`, enables the *Respect pin layer/side constraints* option, which allows the pins to switch their layers depending on the side on which they are placed by the Pin Optimizer. Use the rightSidePinLayersConstraints, leftSidePinLayersConstraints, topSidePinLayersConstraints, and bottomSidePinLayersConstraints environment variables to specify the layers on the respective sides.

The default value is `nil`.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Respect pin layer/side constraints* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "enablePinLayerSideConstraints")
envSetVal("layoutXL.floorplan" "enablePinLayerSideConstraints" 'boolean t)
envSetVal("layoutXL.floorplan" "enablePinLayerSideConstraints" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

enableTrackSkipping

```
layoutXL.floorplan enableTrackSkipping boolean { t | nil }
```

Description

Honors the pin order along with the width and wire type values during pin placement. If adjacent tracks do not meet the required wire type or width requirements, then tracks are skipped until suitable tracks are found. Therefore, pins are placed as per the order specified.

The default value is `t`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL.floorplan" "enableTrackSkipping")
envSetVal("layoutXL.floorplan" "enableTrackSkipping" 'boolean t)
envSetVal("layoutXL.floorplan" "enableTrackSkipping" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

honorCurrentPinLocation

```
layoutXL.floorplan honorCurrentPinLocation boolean { t | nil }
```

Description

(ICADVM20.1 Only) Snaps pins to their nearest WSP tracks by moving them by a minimum distance from their current position. Pin constraints set on the selected pins are honored.

The default value is `nil`.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Honor Current Pin Location* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "honorCurrentPinLocation")
envSetVal("layoutXL.floorplan" "honorCurrentPinLocation" 'boolean t)
envSetVal("layoutXL.floorplan" "honorCurrentPinLocation" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

ignoreConnectivityBusPinsAlign

```
layoutXL.floorplan ignoreConnectivityBusPinsAlign boolean { t | nil }
```

Description

Specifies whether the pin aligner can be used to align unconnected bus pins.

The default value is `nil`, in which case unconnected pins cannot be aligned.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL.floorplan" "ignoreConnectivityBusPinsAlign")
envSetVal("layoutXL.floorplan" "ignoreConnectivityBusPinsAlign" 'boolean t)
envSetVal("layoutXL.floorplan" "ignoreConnectivityBusPinsAlign" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

labelLayerPurposePair

```
layoutXL.floorplan labelLayerPurposePair cyclic { "Auto" | "Custom" }
```

Description

Specifies the label LPP settings to be applied when pins are re-layered in the design. When set to `Auto`, labels are re-layered to match the pins, but their purposes remain unaltered. When set to `Custom`, you can specify the required layer and purpose preferences.

The default value is `Auto`.

GUI Equivalent

Command	*Floorplan – Floorplan Global Options*
Field	*Label Layer Purpose Pair Update* (Floorplan Global Options)

Examples

```
envGetVal("layoutXL.floorplan" "labelLayerPurposePair")
envSetVal("layoutXL.floorplan"  "labelLayerPurposePair" 'cyclic "Auto")
envSetVal("layoutXL.floorplan"  "labelLayerPurposePair" 'cyclic "Custom")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

layerChangeResize

```
layoutXL.floorplan layerChangeResize cyclic {"currentSize" | "lastLayerValue" |
     "minWidth"}
```

Description

Specifies the dimensions of pins when they are moved from one layer to another. The valid values are:

- `currentSize`: The current pin size is retained. However, the pin is resized if the `minWidth` of the destination layer is higher than the current pin size.
- `lastLayerValue`: This setting is applicable only to pins that were placed in the destination layer earlier within the current session. Their `lastLayerValue` is stored, and the pins are automatically resized to this value when moved back to these layers. If the value is not available, then the pin size is set to the higher of the two values - current pin size and the `minWidth` of the new layer.

  Note: The `lastLayerValue` is stored only for the current session.
- `minWidth`: The pin is resized to the `minWidth` of the destination layer.

The default value is `currentSize`.

GUI Equivalent

None

Examples

```
envSetVal("layoutXL.floorplan"  "layerChangeResize" 'cyclic "lastLayerValue")
envSetVal("layoutXL.floorplan"  "layerChangeResize" 'cyclic "minWidth")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

layerPriority

```
layoutXL.floorplan layerPriority boolean { t | nil }
```

Description

Specifies whether layer priority should be followed while placing pins during pin optimization. When set to `nil` (default), the Pin Optimizer places pins on layers that are closest to the connected pin.

GUI Equivalent

Command	*Floorplan – Floorplan Global Options*
Field	*Respect layer priority for valid routing layers constraint on net* (Floorplan Global Options)

Examples

```
envGetVal("layoutXL.floorplan" "layerPriority")
envSetVal("layoutXL.floorplan" "layerPriority" 'boolean t)
envSetVal("layoutXL.floorplan" "layerPriority" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

leftSidePinLayersConstraints

`layoutXL.floorplan leftSidePinLayersConstraints string` *`list_of_pin_layers`*

Description

Specifies the list of pin layers to be assigned to the left side.

`list_of_pin_layers` is a list of pin layers to be assigned to the left side. The list must be enclosed in quotation marks; for example,

```
"metal1 metal2 metal4"
```

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin layer/side constraints* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "leftSidePinLayersConstraints")
envSetVal("layoutXL.floorplan" "leftSidePinLayersConstraints" 'string "metal2
metal3")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

matchLabelLayerWithPin

```
layoutXL.floorplan matchLabelLayerWithPin boolean { t | nil }
```

Description

Specifies whether pin labels should be placed on the same layers as their corresponding pins. So if a pin is moved to a different layer, then its label is also re-layered to match the pin layer.

The default value is `t`.

GUI Equivalent

Command	*Floorplan – Floorplan Global Options*
Field	*Match label layer with pins* (Floorplan Global Options)

Examples

```
envGetVal("layoutXL.floorplan" "matchLabelLayerWithPin")
envSetVal("layoutXL.floorplan" "matchLabelLayerWithPin" 'boolean t)
envSetVal("layoutXL.floorplan" "matchLabelLayerWithPin" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

numberOfSkipTracks

`layoutXL.floorplan numberOfSkipTracks int` *`integer_number`*

Description

Specifies the number of tracks to be skipped during pin placement when enableTrackSkipping is set to t.

The default value is 0.0.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL.floorplan" "numberOfSkipTracks")
envSetVal("layoutXL.floorplan" "numberOfSkipTracks" 'int 1)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

offSetByGrids

```
layoutXL.floorplan offSetByGrids boolean { t | nil }
```

Description

Specifies whether the pin offset value can be defined in terms of the number of grids to be skipped before placing the first pin.

The default value is `nil`.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin Offset – Custom – Options – Grids* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "offSetByGrids")
envSetVal("layoutXL.floorplan" "offSetByGrids" 'boolean t)
envSetVal("layoutXL.floorplan" "offSetByGrids" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

offSetBySpacing

```
layoutXL.floorplan offSetBySpacing boolean { t | nil }
```

Description

Specifies whether the pin offset value can be defined as an absolute value that determines the space between the cell boundary and the first pin.

The default value is `nil`.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin Offset – Custom – Options – Spacing* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "offSetBySpacing")
envSetVal("layoutXL.floorplan" "offSetBySpacing" 'boolean t)
envSetVal("layoutXL.floorplan" "offSetBySpacing" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

offSetMode

```
layoutXL.floorplan offSetMode cyclic {"Auto" | "Custom"}
```

Description

Sets the pin offset mode to either `Auto` or `Custom`.

When set to `Auto`, the Pin Optimizer automatically places the first pin on a grid such that there is no spacing violation with the corner pins of the adjacent side. The Pin Optimizer also ensures that there is no wastage of grid space.

When set to `Custom`, you can manually set the pin offset values for all layers.

The default value is `Auto`.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin Offset – Auto/Custom* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "offSetMode")
envSetVal("layoutXL.floorplan" "offSetMode" 'cyclic "Auto")
envSetVal("layoutXL.floorplan" "offSetMode" 'cyclic "Custom")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

optimizeInstancesPartiallyOutsidePRB

```
layoutXL.floorplan optimizeInstancesPartiallyOutsidePRB boolean { t | nil }
```

Description

Optimizes instances that are partially outside the top-level PR boundary.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL.floorplan" "optimizeInstancesPartiallyOutsidePRB")
envSetVal("layoutXL.floorplan" "optimizeInstancesPartiallyOutsidePRB" 'boolean t)
envSetVal("layoutXL.floorplan" "optimizeInstancesPartiallyOutsidePRB" 'boolean
nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

perLayerAdjustmentFactorList

```
layoutXL.floorplan perLayerAdjustmentFactorList string (layerName
     AdjustmentFactor)
```

Description

Specifies the value by which the pin grid needs to be scaled for each layer. This environment variable can be used only if the pinGridAdjustmentFactor environment variable is set to `t`. The valid values are:

- *`layerName`* specifies the name of the layer.
- *`gridWidth`* is the value by which pin grid needs to be scaled for the specified layer.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin Spacing – Advanced Options – Pin Grid Adjustment Factor* (Pin Optimization)

Examples

```
envSetVal("layoutXL.floorplan" "perLayerAdjustmentFactorList" 'string "(metal1
4.2) (metal2 1.3)" )
```

```
envSetVal("layoutXL.floorplan" " perLayerAdjustmentFactorList" 'string "(metal1 1)
(metal2 0.5)" )
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

perLayerCustomPinGridList

`layoutXL.floorplan perLayerCustomPinGridList string (layerName gridWidth)`

Description

Specifies the absolute pin grid width value for each layer. By default, this option is turned off.

- *`layerName`* specifies the name of the layer.
- *`gridWidth`* is the absolute pin grid width value for the specified layer.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin Spacing – Advanced Options – Custom Pin Grid* (Pin Optimization)

Examples

```
envSetVal("layoutXL.floorplan" "perLayerCustomPinGridList" 'string "(metal1 5)
(metal2 2)" )
envSetVal("layoutXL.floorplan" "perLayerCustomPinGridList" 'string "(metal1 5)
(metal2 2)" )
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

perLayerOffsetByMfgGridList

`layoutXL.floorplan perLayerOffsetByMfgGridList string (layerName NumberOfGrids)`

Description

Specifies the number of manufacturing grids to skip before placing the first pin. By default, this option is turned off. Manufacturing grids apply only for custom pins. For digital pins, use perLayerOffsetByRoutingGridList.

- *`layerName`* specifies the name of the layer.
- *`gridWidth`* is the number of manufacturing grids to skip before placing the first pin on the specified layer.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin Offset – Custom – Options – Grids – Mfg (Custom Blocks)* (Pin Optimization)

Examples

```
envSetVal("layoutXL.floorplan" "perLayerOffsetByMfgGridList" 'string "(metal1 2)
(metal2 3)" )
```

```
envSetVal("layoutXL.floorplan" "perLayerOffsetByMfgGridList" 'string "(metal1 2)
(metal2 3)" )
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

perLayerOffsetByRoutingGridList

`layoutXL.floorplan perLayerOffsetByMfgGridList string (layerName NumberOfGrids)`

Description

Specifies the number of routing grids to skip before placing the first pin. By default, this option is turned off. Routing grids apply only for digital pins. For custom pins, use perLayerOffsetByMfgGridList.

- *`layerName`* specifies the name of the layer.
- *`gridWidth`* is the number of routing grids to skip before placing the first pin on the specified layer.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin Offset – Custom – Options – Grids – Routing (Digital Blocks)* (Pin Optimization)

Examples

```
envSetVal("layoutXL.floorplan" "perLayerOffsetByRoutingGridList" 'string "(metal1
3) (metal2 2)" )
```

```
envSetVal("layoutXL.floorplan" "perLayerOffsetByRoutingGridList" 'string "(metal1
3) (metal2 2)" )
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

perLayerOffsetBySpacingList

`layoutXL.floorplan perLayerOffsetBySpacingList string (`*`layerName SpacingValue`*`)`

Description

Specifies the absolute space between the cell boundary and the first pin for each layer. By default, this option is turned off.

- *`layerName`* specifies the name of the layer.
- *`SpacingValue`* is the absolute space between the cell boundary and the first pin for the specified layer.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin Offset – Custom – Options – Spacing* (Pin Optimization)

Examples

```
envSetVal("layoutXL.floorplan" "perLayerOffsetBySpacingList" 'string "(metal1 5)
(metal2 5)" )
```

```
envSetVal("layoutXL.floorplan" "perLayerOffsetBySpacingList" 'string "(metal1 5)
(metal2 5)" )
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

perLayerSpacingList

`layoutXL.floorplan perLayerSpacingList string (`*`layerName PinSpacing`*`)`

Description

Specifies the pin spacing value for each layer. Pin spacing values cannot be smaller than the `minSpacing` or the `tableSpacing` value specified in the technology file.

- *`layerName`* specifies the name of the layer.
- *`PinSpacing`* is the pin spacing value for each layer for the specified layer.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin Spacing – Custom – Options* (Pin Optimization)

Examples

```
envSetVal("layoutXL.floorplan" "perLayerSpacingList" 'string "(metal1 2) (metal2
1)" )
```

```
envSetVal("layoutXL.floorplan" "perLayerSpacingList" 'string "(metal1 2) (metal2
1)" )
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

pinGridAdjustmentFactor

```
layoutXL.floorplan pinGridAdjustmentFactor boolean { t | nil }
```

Description

Enables you to specify the value by which the pin grid needs to be scaled for each layer. When set to `t`, you can use the perLayerAdjustmentFactorList environment variable to specify the adjustment factor for each layer.

The default value is `nil`.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Advanced Options - Define Pin Grid Adjustment Factor* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "pinGridAdjustmentFactor")
envSetVal("layoutXL.floorplan" "pinGridAdjustmentFactor" 'boolean t)
envSetVal("layoutXL.floorplan" "pinGridAdjustmentFactor" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

pinOptMode

```
layoutXL.floorplan pinOptMode boolean { t | nil }
```

Description

Specifies whether you can set the mode for running Pin Optimizer.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL.floorplan" "pinOptMode")
envSetVal("layoutXL.floorplan" "pinOptMode" 'boolean t)
envSetVal("layoutXL.floorplan" "pinOptMode" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

rectangularPinEdgeOnPRB

```
layoutXL.floorplan rectangularPinEdgeOnPRB cyclic {"As Is" | "Short Edge" | "Long
     Edge"}
```

Description

Specifies the edge of the pin that needs to be abutted to the PR Boundary. The valid values are:

- `As Is` abuts the rectangular pin as-is to the PR Boundary, irrespective of whether the abutted edge is short or long. Depending on the original position of the pin and the PR Boundary side, either the long edge or the short edge of the pin is abutted to the PR Boundary.
- `Short Edge` abuts the short edge of the rectangular pin to the PR Boundary.
- `Long Edge` abuts the long edge of the rectangular pin to the PR Boundary.

The default value is `As Is`.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Place Rectangular Pin* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "rectangularPinEdgeOnPRB")
envSetVal("layoutXL.floorplan" "rectangularPinEdgeOnPRB" 'cyclic "As Is")
envSetVal("layoutXL.floorplan" "rectangularPinEdgeOnPRB" 'cyclic "Short Edge")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

retainLongPin

```
layoutXL.floorplan retainLongPin boolean { t | nil }
```

Description

Retains the dimensions of long, rectangular pins while snapping them to WSP/SP tracks.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL.floorplan" "retainLongPin")
envSetVal("layoutXL.floorplan" "retainLongPin" 'boolean t)
envSetVal("layoutXL.floorplan" "retainLongPin" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

rightSidePinLayersConstraints

`layoutXL.floorplan rightSidePinLayersConstraints string` *`list_of_pin_layers`*

Description

Specifies the list of pin layers to be assigned to the right side.

`list_of_pin_layers` specifies a list of pin layers to be assigned to the right side. The list must be enclosed in quotation marks; for example:

```
"metal1 metal2 metal4"
```

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin layer/side constraints* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "rightSidePinLayersConstraints")
envSetVal("layoutXL.floorplan" "rightSidePinLayersConstraints" 'string "metal1
metal4")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

sameLabelPurpose

```
layoutXL.floorplan sameLabelPurpose boolean { t | nil }
```

Description

Controls the assignment of label purposes when pins are re-layered. When set to `t`, label purposes remain unaffected during pin re-layering.

The default value is `nil`.

GUI Equivalent

Command	*Floorplan – Floorplan Global Options*
Field	*Define Custom Layer Purpose Pair – Options – Layer Purpose – Same As Label* (Floorplan Global Options)

Examples

```
envGetVal("layoutXL.floorplan" "sameLabelPurpose")
envSetVal("layoutXL.floorplan" "sameLabelPurpose" 'boolean t)
envSetVal("layoutXL.floorplan" "sameLabelPurpose" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

selectedOnly

```
layoutXL.floorplan selectedOnly boolean { t | nil }
```

Description

Specifies whether to run pin optimization on the selected objects only or all the objects. The valid values are `t` and `nil`. The default value is `nil`.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Mode: Selected* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "selectedOnly")
envSetVal("layoutXL.floorplan" "selectedOnly" 'boolean t)
envSetVal("layoutXL.floorplan" "selectedOnly" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

spacingMode

```
layoutXL.floorplan spacingMode cyclic {"Auto" | "Custom"}
```

Description

Sets the pin spacing mode to either `Auto` or `Custom`. When set to `Auto`, the Pin Optimizer automatically calculates the pin spacing value for all layers based on the width values of pins in each layer. When set to `Custom`, you can manually set the pin spacing values for each layer.

The default value is `Auto`.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin Spacing – Auto/Custom* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "spacingMode")
envSetVal("layoutXL.floorplan" "spacingMode" 'cyclic "Auto")
envSetVal("layoutXL.floorplan" "spacingMode" 'cyclic "Custom")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

spacingType

```
layoutXL.floorplan spacingType cyclic {"line-line" | "line-via" | "via-via" |
     "specify"}
```

Description

Specifies how to space out the pins using the relevant spacing values from the technology file. The valid values are:

- `line-line` uses the minimum spacing between two wires on a layer for spacing pins.
- `line-via` uses the minimum spacing between a wire and a via on a layer for spacing pins.
- `via-via` uses the minimum spacing between two vias on a layer for spacing pins.
- `specify` provides options to enter a minimum pin spacing values for each layer.

The default value is set to `line-line`.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Spacing type: line-line, line-via, via-via, specify* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "spacingType")
envSetVal("layoutXL.floorplan" "spacingType" 'cyclic "line-line")
envSetVal("layoutXL.floorplan" "spacingType" 'cyclic "line-via")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

spacingValue

`layoutXL.floorplan spacingValue float` *`floating_point_number`*

Description

The spacing value is specified if the *spacingType* is set to `specify`. This value is used to specify the minimum spacing between the pins.

`floating_point_number` is a floating point value specifying the minimum spacing between the pins.

The default value is 0.0.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*specify: text field* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "spacingValue")
envSetVal("layoutXL.floorplan" "spacingValue" 'float 0.25)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

tableSpacingRule

```
layoutXL.floorplan tableSpacingRule boolean { t | nil }
```

Description

Specifies whether to scale up the specified spacing values as per the table spacing rules. Valid values are `t` and `nil`. The default value is `nil`.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Use Table Spacing* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "tableSpacingRule")
envSetVal("layoutXL.floorplan" "tableSpacingRule" 'boolean t)
envSetVal("layoutXL.floorplan" "tableSpacingRule" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

targetPinSide

`layoutXL.floorplan targetPinSide string` *`list_of_target_sides`*

Description

Specifies the side(s) of the blocks and the PRBoundary on which the pins should be placed. The valid values are `left`, `right`, `top`, and `bottom`. By default all the four sides are considered. You can also specify a combination of one or more sides.

`list_of_target_sides` is a list of target sides for placing the pins. The list must be enclosed in quotation marks; for example:

```
"top bottom right left"
```

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Side: left, right, top, bottom* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "targetPinSide")
envSetVal("layoutXL.floorplan" "targetPinSide" 'string “left top”)
envSetVal("layoutXL.floorplan" "targetPinSide" 'string “right”)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

topSidePinLayersConstraints

`layoutXL.floorplan topSidePinLayersConstraints string` *`list_of_pin_layers`*

Description

Specifies the list of pin layers to be assigned to the top side.

`list_of_pin_layers` is a list of pin layers to be assigned to the top side. The list must be enclosed in quotation marks; for example,

```
"metal1 metal2 metal4"
```

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin layer/side constraints* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "topSidePinLayersConstraints")
envSetVal("layoutXL.floorplan" "topSidePinLayersConstraints" 'string "metal2
metal3")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

widthForPinGrid

```
layoutXL.floorplan widthForPinGrid cyclic {"minPinWidth" | "techMinWidth"}
```

Description

Specifies whether the width of the pin grid should be calculated from the value of the width of the smallest pin on a layer or should be picked from the technology file (default). The valid values are:

- `minPinWidth` specifies the width of the smallest pin on a layer.
- `techMinWidth` specifies the minimum pin width specified in the technology file.

GUI Equivalent

Command	*Floorplan – Pin Optimization*
Field	*Pin Spacing – Options – Advanced Options – Width of the Smallest Pin /Min Width from Technology File* (Pin Optimization)

Examples

```
envGetVal("layoutXL.floorplan" "widthForPinGrid")
envSetVal("layoutXL.floorplan" "widthForPinGrid" 'cyclic "minPinWidth")
envSetVal("layoutXL.floorplan" "widthForPinGrid" 'cyclic "techMinWidth")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

Pin Tool Environment Variables

boundaryMultipleMode

```
layoutXL.floorplan boundaryMultipleMode cyclic { "Top Layer" | "Use Layer" }
```

Description

Specifies the layers on which boundary pins are to be created. The valid values are:

- `Top Layer` (default): Creates multiple boundary pins on the highest metal layer.
- `User Layer`: Lets you specify the layers on which boundary pins are to be created.

GUI Equivalent

Command	*Pin Tool – Create – Auto-Create Pins*
Field	*Boundary Shape Pin – Multiple (*Creating Pins Automatically*)*

Examples

```
envGetVal("layoutXL.floorplan" "boundaryMultipleMode")
envSetVal("layoutXL.floorplan" "boundaryMultipleMode" 'cyclic "Use Layer")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

closePinToolSubForms

```
layoutXL.floorplan closePinToolSubForms boolean { t | nil }
```

Description

Specifies whether all open Pin Tool sub-forms are to be automatically closed when the Pin Tool is closed.

The default value is `t`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL.floorplan" "closePinToolSubForms")
envSetVal("layoutXL.floorplan" "closePinToolSubForms" 'boolean t)
envSetVal("layoutXL.floorplan" "closePinToolSubForms" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

Using the Pin Tool

© 2020 All Rights Reserved.

autoPinCreateFigGroup

```
layoutXL.floorplan autoPinCreateFigGroup boolean { t | nil }
```

Description

Specifies whether the Auto-Create Pins tool needs to create pins as groups or figGroups. When set to `t`, figGroups are created. When set to `nil`, groups are created.

The default value is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL.floorplan" "autoPinCreateFigGroup")
envSetVal("layoutXL.floorplan" "autoPinCreateFigGroup" 'boolean t)
envSetVal("layoutXL.floorplan" "autoPinCreateFigGroup" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

Creating Pins Automatically

© 2020 All Rights Reserved.

autoPinPathSpineAware

```
layoutXL.floorplan autoPinPathSpineAware boolean { t | nil }
```

Description

Determines whether signal pins created in the *Single* mode are path spine aware.

- When set to `t`, a single merged pin is created on the highest metal layer shape island. A metal layer shape island is made of overlapping shapes that belong to the same layer and net.
- When set to `nil`, a single pin is created on the highest metal layer shape island.

GUI Equivalent

None

Examples

```
envGetVal("layoutXL.floorplan" "autoPinPathSpineAware")
envSetVal("layoutXL.floorplan" "autoPinPathSpineAware" 'boolean t)
envSetVal("layoutXL.floorplan" "autoPinPathSpineAware" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

Creating Pins Automatically

© 2020 All Rights Reserved.

boundarySquareShapePin

```
layoutXL.floorplan boundarySquareShapePin cyclic { "Single" | "Multiple" }
```

Description

Specifies the mode for generating boundary pins. The valid values are:

- `Single` (default): Creates a single boundary pin on the highest metal layer.
- `Multiple`: Lets you specify a layer selection option using the boundaryMultipleMode environment variable.

GUI Equivalent

Command	*Pin Tool – Create – Auto-Create Pins*
Field	*Boundary Shape Pin (*Creating Pins Automatically*)*

Examples

```
envGetVal("layoutXL.floorplan" "boundarySquareShapePin")
envSetVal("layoutXL.floorplan" "boundarySquareShapePin" 'cyclic "Multiple")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

buriedMultipleMode

```
layoutXL.floorplan buriedMultipleMode cyclic { "Top Layer" | "Use Layer" }
```

Description

Specifies the layers on which buried pins are to be created. The valid values are:

- `Top Layer` (default): Creates multiple buried pins on the highest metal layer.
- `User Layer`: Lets you specify the layers on which buried pins are to be created.

GUI Equivalent

Command	*Pin Tool – Create – Auto-Create Pins*
Field	*Buried Whole Shape Pin – Multiple (*Creating Pins Automatically*)*

Examples

```
envGetVal("layoutXL.floorplan" "buriedMultipleMode")
envSetVal("layoutXL.floorplan" "buriedMultipleMode" 'cyclic "Use Layer")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

buriedWholeShapePin

```
layoutXL.floorplan buriedWholeShapePin cyclic { "Single" | "Multiple" }
```

Description

Specifies the number of buried pins to be generated when there are multiple buried shapes belonging to the same net. The valid values are:

- `Single` (default): Creates a single buried pin on the highest metal layer.
- `Multiple`: Lets you specify a layer selection option using the buriedMultipleMode environment variable.

GUI Equivalent

Command	*Pin Tool – Create – Auto-Create Pins*
Field	*Buried Whole Shape Pin (*Creating Pins Automatically*)*

Examples

```
envGetVal("layoutXL.floorplan" "buriedWholeShapePin")
envSetVal("layoutXL.floorplan" "buriedWholeShapePin" 'cyclic "Multiple")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

customLPPField

`layoutXL.floorplan customLPPField string` *`layer_name purpose_name`*

Description

Specifies the layer on which pins are to be generated.

GUI Equivalent

Command	*Pin Tool – Create – Auto-Create Pins*
Field	*Pin Layer – Custom LPP (*Creating Pins Automatically*)*

Examples

```
envGetVal("layoutXL.floorplan" "customLPPField")
envSetVal("layoutXL.floorplan" "customLPPField" 'string "Metal3 drawing")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

customPurposeField

`layoutXL.floorplan customPurposeField string` *`purpose_name`*

Description

Specifies the purpose for the layer in which pins are to be created. Pins are created on the same layer as the original shapes, but on the specified purpose.

GUI Equivalent

Command	*Pin Tool – Create – Auto-Create Pins*
Field	*Pin Layer – Same layer, Custom purpose (*Creating Pins Automatically*)*

Examples

```
envGetVal("layoutXL.floorplan" "customPurposeField")
envSetVal("layoutXL.floorplan" "customPurposeField" 'string "drawing")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

deletePinCheckbox

```
layoutXL.floorplan deletePinCheckbox boolean { t | nil }
```

Description

Specifies whether the existing pins that were created by the Auto-Create Pins tool are to be deleted before generating new ones. Use deletePinSelection to specify which pins are to be deleted.

The default value is `nil`, in which case the existing pins are not deleted.

GUI Equivalent

Command	*Pin Tool – Create – Auto-Create Pins*
Field	*Delete Pins (*Creating Pins Automatically*)*

Examples

```
envGetVal("layoutXL.floorplan" "deletePinCheckbox")
envSetVal("layoutXL.floorplan" "deletePinCheckbox" 'boolean t)
envSetVal("layoutXL.floorplan" "deletePinCheckbox" 'boolean nil)
```

Related Topics

Pin Tool Environment Variables

© 2020 All Rights Reserved.

deletePinSelection

```
layoutXL.floorplan deletePinSelection cyclic { "Auto Created" | "All" }
```

Description

Deletes the existing pins in the layout canvas before generating new ones to avoid creating pins over existing pins. The valid values are:

- `Auto Created` (default): Deletes only those pins and associated metal shapes that were generated automatically by using the Auto-Create Pins tool.
- `All`: Deletes all pins in the current cellview.

GUI Equivalent

Command	*Pin Tool – Create – Auto-Create Pins*
Field	*Delete Pins (*Creating Pins Automatically*)*

Examples

```
envGetVal("layoutXL.floorplan" "deletePinSelection")
envSetVal("layoutXL.floorplan" "deletePinSelection" 'cyclic "Auto Created")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

modeOption

```
layoutXL.floorplan modeOption cyclic { "CellView" | "Selected" }
```

Description

Specifies the location in which the net shapes must be searched. The valid values are:

- `CellView` (default): Searches for net shapes in all nets in the current cellview.
- `Selected`: Searches for net shapes only in the selected nets in the current cellview.

GUI Equivalent

Command	*Pin Tool – Create – Auto-Create Pins*
Field	*Net in (*Creating Pins Automatically*)*

Examples

```
envGetVal("layoutXL.floorplan" "modeOption")
envSetVal("layoutXL.floorplan" "modeOption" 'cyclic "Selected")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

pinConnectivityMode

```
layoutXL.floorplan pinConnectivityMode cyclic { "Strongly Connected" | "Must Join"
     | "Must Connect" }
```

Description

Specifies the way in which router must connect the pins that are generated by the Auto-Create Pins tool. The valid values are:

- `Strongly Connected` (default): Specifies that the router must connect the pins internally, within the device.
- `Must Join`: Specifies that the router must connect the pins externally, at a higher level.
- `Must Connect`: Specifies that the router must connect to all pins of the selected terminal.

GUI Equivalent

Command	*Pin Tool – Create – Auto-Create Pins*
Field	*Pin Connectivity (*Creating Pins Automatically*)*

Examples

```
envGetVal("layoutXL.floorplan" "pinConnectivityMode")
envSetVal("layoutXL.floorplan" "pinConnectivityMode" 'cyclic "Must Join")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

pinLayers

```
layoutXL.floorplan pinLayers cyclic { "Same as Shape LPP" | "Same Layer. Custom
     Purpose" | "Custom LPP" }
```

Description

Specifies the layers in which the Auto-Create Pins tool must generate pins. The valid values are:

- `Same as Shape LPP` (default): Creates pins on the same LPPs as the original net shapes.
- `Same Layer. Custom Purpose` (default): Lets you choose a different purpose for the same layer. Pins are created in the specified LPP.
- `Custom LPP`: Lets you choose a different LPP on which pins must be created.

GUI Equivalent

Command	*Pin Tool – Create – Auto-Create Pins*
Field	*Pin Layer (*Creating Pins Automatically*)*

Examples

```
envGetVal("layoutXL.floorplan" "pinLayers")
envSetVal("layoutXL.floorplan" "pinLayers" 'cyclic "Custom LPP")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

pinSizeSelection

```
layoutXL.floorplan pinSizeSelection cyclic { "Auto" | "Whole Shape" }
```

Description

Specifies the size of the pins that are to be created by the Auto-Create Pins tool. The valid values are:

- `Auto` (default): Creates square pins.
- `Whole Shape`: Creates pins of the same shape as the corresponding net shapes.

GUI Equivalent

Command	*Pin Tool – Create – Auto-Create Pins*
Field	*Boundary Signal Pin Size (*Creating Pins Automatically*)*

Examples

```
envGetVal("layoutXL.floorplan" "pinSizeSelection")
envSetVal("layoutXL.floorplan" "pinSizeSelection" 'cyclic "Whole Shape")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

purposeField

`layoutXL.floorplan purposeField string` *`purpose_name`*

Description

Restricts the search for net shapes to the routing layers that correspond to the specified purpose.

GUI Equivalent

Command	*Pin Tool – Create – Auto-Create Pins*
Field	*Consider net shape on selected purpose (*Creating Pins Automatically*)*

Examples

```
envGetVal("layoutXL.floorplan" "purposeField")
envSetVal("layoutXL.floorplan" "purposeField" 'string "drawing")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

mergePinsConnMode

`layoutXL.floorplan mergePinsConnMode int pinConnectMode`

Description

Specifies the mode in which pins are to be merged. The available options are:

- `0`: Refers to *Strongly Connect Pins* pin connectivity model, where a single connection point on a highly conductive metal is used to provide the required connectivity.
- `1`: Refers to *Must Connect Pins Using New Nets* pin connectivity model, where new nets, terminals, and pins are created for each pin shape island.
- `2`: Refers to the *Must Connect Pins Without Creating New Nets* pin connectivity model, where the connectivity of the pin shapes is retained.

The default value is `2`.

GUI Equivalent

Command	*Pin Tool – Tools – Merge and Must Connect Pins*
Field	*Pin Connectivity*(Merging Pins)

Examples

```
envGetVal("layoutXL.floorplan" "mergePinsConnMode")
envSetVal("layoutXL.floorplan" "mergePinsConnMode" 'int 0)
envSetVal("layoutXL.floorplan" "mergePinsConnMode" 'int 1)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

mergePinsSelMode

```
layoutXL.floorplan mergePinsSelMode boolean { t | nil }
```

Description

Specifies whether the selected pin shapes are to be merged.

The default value is `nil`, in which case all pin shapes in the design are selected for merge.

GUI Equivalent

Command	*Pin Tool – Tools – Merge and Must Connect Pins*
Field	*Mode – Selected* (Merging Pins)

Examples

```
envGetVal("layoutXL.floorplan" "mergePinsSelMode")
envSetVal("layoutXL.floorplan" "mergePinsSelMode" 'boolean t)
envSetVal("layoutXL.floorplan" "mergePinsSelMode" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

mergePinsSuffix

`layoutXL.floorplan mergePinsSuffix string` *`suffix`*

Description

Specifies the suffix to be added to the names of the new nets, terminals, and pins that have been created on the non-overlapping merged islands.

The default value is `"_MJ"`.

GUI Equivalent

Command	*Pin Tool – Tools – Merge and Must Connect Pins*
Field	*Suffix String* (Merging Pins)

Examples

```
envGetVal("layoutXL.floorplan" "mergePinsSuffix")
envSetVal("layoutXL.floorplan" "mergePinsSuffix" 'string "merged")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

pinResizeMode

```
layoutXL.floorplan pinResizeMode cyclic { "Selected" | "All" }
```

Description

Specifies whether the selected pin resize function must be run on all or selected pins. The default value is `Selected`.

GUI Equivalent

Command	*Pin Tool – Place – Resize Pins*
Field	*Run On (*Resizing Pins*)*

Examples

```
envGetVal("layoutXL.floorplan" "pinResizeMode")
envSetVal("layoutXL.floorplan" "pinResizeMode" 'cyclic "All")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

pinResizeAllEstimators

`layoutXL.floorplan pinResizeAllEstimators string` *`estimators`*

Description

Specifies a list of area estimator functions to be listed in the *Resize function* drop-down list.

The default value is `""`.

GUI Equivalent

Command	*Pin Tool – Create – Resize Pins*
Field	*Resize function (*Resizing Pins*)*

Examples

```
envGetVal("layoutXL.floorplan" "pinResizeAllEstimators")
envSetVal("layoutXL.floorplan" "pinResizeAllEstimators" 'string "functionA
functionB")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

pinResizeCurrentEstimator

`layoutXL.floorplan pinResizeCurrentEstimator string` *`current_estimator`*

Description

Specifies the area estimator function that is to be displayed as the selected area estimator function in the *Resize function* drop-down list.

The default value is `""`.

GUI Equivalent

Command	*Pin Tool – Create – Resize Pins*
Field	*Resize function (*Resizing Pins*)*

Examples

```
envGetVal("layoutXL.floorplan" "pinResizeCurrentEstimator")
envSetVal("layoutXL.floorplan" "pinResizeCurrentEstimator" 'string "functionA")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

pinResizeFiles

`layoutXL.floorplan pinResizeFiles string` *`file_path`*

Description

Specifies the File Path of the SKILL hook procedure file that contains the pin resize definitions.

The default value is `""`.

GUI Equivalent

Command	*Pin Tool – Create – Resize Pins*
Field	*File Path (*Resizing Pins*)*

Examples

```
envGetVal("layoutXL.floorplan" "pinResizeFiles")
envSetVal("layoutXL.floorplan" "pinResizeFiles" 'string "d:/resizeFn1.il")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteCreateLabelsOnPins

```
layoutXL.floorplan promoteCreateLabelsOnPins boolean { t | nil }
```

Description

Specifies whether pin labels must be created for promoted pins. The default value is `nil`.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Create Labels on Promoted Pins* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteCreateLabelsOnPins")
envSetVal("layoutXL.floorplan" "promoteCreateLabelsOnPins" 'boolean t)
envSetVal("layoutXL.floorplan" "promoteCreateLabelsOnPins" 'boolean nil)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteCustomPurpose

`layoutXL.floorplan promoteCustomPurpose string` *`purpose`*

Description

Specifies a custom purpose for the promoted pin LPP. The pins are promoted to an LPP with the same layer and custom purpose. For example, for a pin `clk` (inside an instance) on LPP `(metal 2 drawing)`, a custom purpose `pin` is specified. The promoted pin will be on LPP `(metal2 pin)`.

The default value is `nil`.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Pin Layer – Same Layer, Custom Purpose* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteCustomPurpose")
envSetVal("layoutXL.floorplan" "promoteCustomPurpose" 'string "drawing")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteCustomLPP

`layoutXL.floorplan promoteCustomLPP string` *`LPP`*

Description

Specifies the custom LPP to which pins must be promoted. The default value is `nil`.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Pin Layer – Custom LPP* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteCustomLPP")
envSetVal("layoutXL.floorplan" "promoteCustomLPP" 'string "Metal4 drawing")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteDeleteOldPins

```
layoutXL.floorplan promoteDeleteOldPins boolean { t | nil }
```

Description

Deletes the pins that were promoted earlier. The default value is `nil`.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Delete Promote Pins from previous run(s)* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteDeleteOldPins")
envSetVal("layoutXL.floorplan" "promoteDeleteOldPins" 'boolean t)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteLayerChoice

```
layoutXL.floorplan promoteLayerChoice cyclic { "Same as Shape LPP" | "Same Layer,
     Custom Purpose" | "Custom LPP" }
```

Description

Specifies the LPP selection mode for promoting pins. The default value is `Same as Shape LPP`.

When set to `Same Layer, Custom Purpose`, use promoteCustomPurpose to specify the purpose.

When set to `Custom LPP`, use promoteCustomLPP to specify the LPP.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Pin Layer* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteLayerChoice")
envSetVal("layoutXL.floorplan" "promoteLayerChoice" 'cyclic "Custom LPP")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteOddEvenChoice

```
layoutXL.floorplan promoteOddEvenChoice cyclic { "Odd" | "Even" }
```

Description

Specifies whether odd or even bus pins must be promoted. The default value is `Odd`.

This setting is valid only when promotePinNetChoice is set to `Pin` or `Instance`. This setting is applicable only when promoteSelectiveBusBits is set to `t`.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Selective Bus Bits* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteOddEvenChoice")
envSetVal("layoutXL.floorplan" "promoteOddEvenChoice" 'cyclic "Even")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promotePinConnectivity

```
layoutXL.floorplan promotePinConnectivity boolean { t | nil }
```

Description

Honors the connectivity of pins to top-level nets during the promotion of these pins. The connections are retained in the promoted pins. If not specified, the default value, `nil`, is considered.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Promote Pins considering connectivity* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promotePinConnectivity")
envSetVal("layoutXL.floorplan" "promotePinConnectivity" 'boolean t)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promotePinNetChoice

```
layoutXL.floorplan promotePinNetChoice cyclic { "Pin" | "Net" | "Instance" }
```

Description

Specifies a mode to run the promote pins command. The default value is `Pin`.

When set to `Net`, use promotePinNetList to specify a list of nets. Use promoteToLevel to specify the depth to which the tool must traverse to search for pins.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Pin, Net, Instance* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promotePinNetChoice")
envSetVal("layoutXL.floorplan" "promotePinNetChoice" 'cyclic "Net")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promotePinNetList

`layoutXL.floorplan promotePinNetList string` *`Nets`*

Description

Specifies a space-separated list of hierarchical pins, top-level nets, or instances that contain the pins to be promoted.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Pin, Net, Instance* (Text Box) (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promotePinNetList")
envSetVal("layoutXL.floorplan" "promotePinNetList" 'string "net1 net2 net3")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promotePinOnNetForSelInsts

```
layoutXL.floorplan promotePinOnNetForSelInsts boolean { t | nil }
```

Description

Filters the pins in the selected nets further by selecting one or more instances. Only those pins in the specified nets that belong to the selected instances are promoted.

This setting is valid only when promotePinNetChoice is set to `Net`. The default value is `nil`.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Promote Pins for Selected Nets on Selected Instances* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promotePinOnNetForSelInsts")
envSetVal("layoutXL.floorplan" "promotePinOnNetForSelInsts" 'boolean t)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promotePinOnNetPickLevel

`layoutXL.floorplan promotePinOnNetPickLevel string` *`Level`*

Description

Specifies the depth to which the tool must traverse to search for pins. For example, if set to `2`, the command traverses the design hierarchy to find pins on the selected net at level 2 to identify the pins for promotion.

This setting is valid only when promotePinNetChoice is set to `Net`.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Find Pins on Net at depth* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promotePinOnNetPickLevel")
envSetVal("layoutXL.floorplan" "promotePinOnNetPickLevel" 'string 2)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promotePrefixChoice

```
layoutXL.floorplan promotePrefixChoice boolean { t | nil }
```

Description

Specifies whether a prefix must be added to the promoted pin names. When set to `t`, use promotePrefixString to specify the prefix. The default value is `nil`.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Add Suffix/Prefix* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promotePrefixChoice")
envSetVal("layoutXL.floorplan" "promotePrefixChoice" 'boolean t)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promotePrefixString

`layoutXL.floorplan promotePrefixString string` *`prefix`*

Description

Specifies the prefix string to be added to the promoted pin names. The default value is blank, which means that there is no prefix. This setting is valid only when promotePrefixChoice is set to `t`.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Add Suffix/Prefix – Prefix String* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promotePrefixString")
envSetVal("layoutXL.floorplan" "promotePrefixString" 'string "a_")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteSchematicAware

```
layoutXL.floorplan promoteSchematicAware boolean { t | nil }
```

Description

Promotes only those pins that have a corresponding schematic counterpart at the higher level. If not specified, the default value, `nil`, is considered.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Schematic Awareness* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteSchematicAware")
envSetVal("layoutXL.floorplan" "promoteSchematicAware" 'boolean t)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteSelectiveBusBits

```
layoutXL.floorplan promoteSelectiveBusBits boolean { t | nil }
```

Description

Lets you specify whether only the odd or even bus pins must be promoted. The default value is `nil`, where all bus pins are promoted. When set to `t`, use promoteOddEvenChoice to restrict the selection to either odd or even bus pins.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Selective Bus Bits* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteSelectiveBusBits")
envSetVal("layoutXL.floorplan" "promoteSelectiveBusBits" 'boolean t)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteSelectInCanvas

```
layoutXL.floorplan promoteSelectInCanvas boolean { t | nil }
```

Description

Specifies that the promoted pins must be selected on the canvas. This option is available only when either promoteToChoice is set to `Current Level` or promotePinOnNetPickLevel is set to `0`. If not specified, the default value, `nil`, is considered.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Select promoted pins in layout canvas* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteSelectInCanvas")
envSetVal("layoutXL.floorplan" "promoteSelectInCanvas" 'boolean t)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteSuffixChoice

```
layoutXL.floorplan promoteSuffixChoice boolean { t | nil }
```

Description

Specifies whether a suffix must be added to the promoted pin names. When set to `t`, use promoteSuffixString to specify the suffix. The default value is `nil`.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Add Suffix/Prefix* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteSuffixChoice")
envSetVal("layoutXL.floorplan" "promoteSuffixChoice" 'boolean t)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteSuffixString

`layoutXL.floorplan promoteSuffixString string` *`suffix`*

Description

Specifies the suffix string to be added to the promoted pin names. The default value is blank, which means that there is no suffix added. This setting is valid only when promoteSuffixChoice is set to `t`.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Add Suffix/Prefix – Suffix String* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteSuffixString")
envSetVal("layoutXL.floorplan" "promoteSuffixString" 'string "_z")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteToChoice

```
layoutXL.floorplan promoteToChoice cyclic { "Current Level" | "Depth" }
```

Description

Specifies the level to which pins must be promoted. The default value is `Current Level`. `Depth` is a level lower than the current level.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Promote To* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteToChoice")
envSetVal("layoutXL.floorplan" "promoteToChoice" 'cyclic "Depth")
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteTopLayerFig

```
layoutXL.floorplan promoteTopLayerFig boolean { t | nil }
```

Description

Promotes only those pin figures that are available on the top layer. If not specified, the default value, `nil`, is considered.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Promote only top layer of pin shapes* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteTopLayerFig")
envSetVal("layoutXL.floorplan" "promoteTopLayerFig" 'boolean t)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteToLevel

`layoutXL.floorplan promoteToLevel string` *`Level`*

Description

Specifies the level to which pins must be promoted. The specified level must be lower than the current level. Also, the specified level must be in the editable mode so that the selected pins can be copied to it.

This setting is valid only when promoteToChoice is set to `Depth`.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Promote To – Depth* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteToLevel")
envSetVal("layoutXL.floorplan" "promoteToLevel" 'string 2)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.

promoteVisibleLPPs

```
layoutXL.floorplan promoteVisibleLPPs boolean { t | nil }
```

Description

Promotes pins from only those layers for which the visibility has been turned on in the Palette assistant.

If not specified, the default value, `nil`, is considered.

GUI Equivalent

Command	*Pin Tool – Create – Promote Pins*
Field	*Promote Visible Shapes* (Promoting Pins)

Examples

```
envGetVal("layoutXL.floorplan" "promoteVisibleLPPs")
envSetVal("layoutXL.floorplan" "promoteVisibleLPPs" 'boolean t)
```

Related Topics

List of Floorplanner Environment Variables

© 2020 All Rights Reserved.