

Virtuoso Interactive and Assisted Routing User Guide

**Product Version IC23.1
September 2023**

© 2023 Cadence Design Systems, Inc.

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.

Cadence is committed to using respectful language in our code and communications. We are also active in the removal and replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

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

1

<u>Setting up the Interactive Routing Environment</u>	1
<u>License Requirements of Interactive Routing Features</u>	2
<u>Configuring Wire Editing Options</u>	3
<u>Layout Editor Options Form</u>	4
<u>Stretch Options</u>	6
<u>Tapping Wires</u>	6
<u>Tapping Internal Nets</u>	8
<u>Tapping Vias</u>	9
<u>Tapping Results Based on validVias and validLayers</u>	9
<u>Creating Halos</u>	10
<u>Adding the lxStickyNet Property to Wires</u>	10
<u>Support for Voltage-Dependent Rules</u>	11
<u>Support for Area-based Rules</u>	18
<u>minWidth</u>	19
<u>allowedWidthRange</u>	20
<u>allowedLengthRange</u>	23
<u>rectShapeDir</u>	24
<u>minSpacing</u>	26
<u>minArea</u>	27
<u>Via Rules</u>	28
<u>Using Mouse Button Controls</u>	32
<u>Startup Warnings</u>	33

2

<u>Technology Requirements</u>	35
<u>Technology File Requirements</u>	35
<u>Specifying Layer Order</u>	35
<u>Specifying Routing Layer Directions</u>	36
<u>Specifying Via Definitions</u>	37
<u>Specifying Valid Layers and Valid Vias</u>	38

Virtuoso Interactive and Assisted Routing User Guide

<u>Limiting Routing on a layer</u>	38
<u>Via Usage Model</u>	39
<u>Specifying Track Patterns and Width Spacing Patterns</u>	39
<u>Supporting Colored Trim Metal Layers</u>	40
<u>Using Layer-Purpose Pairs</u>	40
<u>Poly Pins over an Implant Layer</u>	40
<u>Support For 45 nm Rules</u>	40
<u>Supported Constraints with Supported Parameters</u>	41
<u>Constraint Groups</u>	41
<u>Foundry Constraint Group</u>	42
<u>Application Constraint Groups</u>	43
<u>Specifying the Default Constraint Group for Routing</u>	43
<u>virtuosoDefaultExtractorSetup</u>	43
<u>virtuosoDefaultSetup</u>	44
<u>LEFDefaultRouteSpec</u>	45
<u>virtuosoDefaultTaper</u>	45
<u>User-Defined Constraint Groups</u>	47
<u>Storing Constraint Groups</u>	48
<u>Constraint Group Lookup Precedence</u>	48
<u>Scalar Versus Table Spacing Rule Precedence</u>	51
<u>Supported Constraints and Parameters</u>	52
<u>Supported Constraints and Parameters (Virtuoso Advanced Node for Layout Standard)</u>	55
<u>Supported Constraints and Parameters (Virtuoso Advanced Node for Layout Only)</u>	55

3

<u>Working with Wires</u>	61
<u>Introduction to Wires</u>	61
<u>Types of Wires</u>	62
<u>Wire Elements</u>	64
<u>Wire Objects</u>	66
<u>Wire Parameters</u>	73
<u>Creating Wires</u>	77
<u>Creating a Wire</u>	77
<u>Creating a Diagonal Wire</u>	83

Virtuoso Interactive and Assisted Routing User Guide

<u>Creating a Floating Wire</u>	84
<u>Creating a Wire with 45 Degree Segments</u>	84
<u>Creating an Odd Width Wire</u>	87
<u>Creating Wires Snapped to Track Patterns</u>	88
<u>Editing Wires</u>	94
<u>Editing Wire Properties</u>	94
<u>Editing Via Properties</u>	96
<u>Editing Cut Patterns</u>	96
<u>Multiple Wire Editing</u>	97
<u>Editing Wire Width</u>	98
<u>Editing Layer-Purpose Pair</u>	121
<u>Splitting Wires and Vias</u>	132
<u>Using Keep Wires Connected to Split Command</u>	134
<u>Support for Local Interconnect (Li/MOL) Layers</u>	164
<u>Wire Editing with Li/MOL Layers</u>	164
<u>Enhanced Snapping in Wire Editing</u>	173
<u>Smart Snapping in Interactive and Assisted Routing Commands</u>	178
<u>Disabling Layer Tapping, Via Insertion, and Smart Snapping</u>	186
<u>Support for Mosaics in Create Wire</u>	187
<u>Support for cdsGenVia for Different Purposes</u>	187
<u>Setting Dynamic Measurement for the Area Calculation</u>	188
<u>Support for Pixel Based Gravity</u>	191
<u>Displaying Wire Summary</u>	191

4

<u>Working with Vias</u>	193
<u>Specifying a Valid Via List</u>	193
<u>Specifying Via Parameters Calculation Mode</u>	194
<u>Changing Layers and Selecting Vias</u>	194
<u>Inserting Vias Automatically</u>	203
<u>Fanning Out Wires to Vias</u>	208
<u>Using Via Assistance</u>	208
<u>Creating an Array of Vias</u>	209
<u>Stacking Vias</u>	209
<u>Inserting Multi-Layer Stack Via</u>	210

Virtuoso Interactive and Assisted Routing User Guide

<u>Minimum Area Rules</u>	213
<u>Changing Via Cut Color</u>	213

5

Working with Buses 217

<u>Creating a Bus Constraint</u>	218
<u>Autopicking Bus Bits</u>	220
<u>Creating a Bus</u>	223
<u>Restarting a Bus From Existing Routes</u>	229
<u>Using the Control Wire</u>	230
<u>Editing a Bus</u>	236
<u>Avoiding Shorts with Existing Blockages for Create Bus</u>	242
<u>Supporting Multi-Layer Bus</u>	242
<u>Creating a Multi-Layer Bus from Existing Pins</u>	243
<u>Creating a Multi-Layer Bus from an Empty Space</u>	251
<u>Creating a Stacked Multi-Layer Bus from a Single Pin</u>	255
<u>Creating a Stacked Multi-Layer Bus from Multiple Pins</u>	256
<u>Defining a Custom Layer Pattern</u>	258
<u>Transitioning Capabilities in a Multi-Layer Bus</u>	262
<u>Stacked Wires within Multi-Layer Bus</u>	269
<u>Transitioning Capabilities in a Wire Stack within a Multi-Layer Bus</u>	274
<u>Customizing Multi-Layer Bus Transition</u>	279

6

Working with Stranded Wires 289

<u>Creating a Stranded Wire</u>	290
<u>Creating a Stranded Wire from an Existing Pin</u>	296
<u>Creating a Stranded Wire from an Existing Wire</u>	300
<u>Creating a Stranded Wire from Tieout</u>	305
<u>Creating a Stranded Wire from a Rectangle or PathSeg Covering a Pin</u>	307
<u>Inserting Vias Automatically</u>	307
<u>Tapering in Stranded Wire</u>	310
<u>Enabling Bridge Vias Using WSPs</u>	315
<u>Avoiding Shorts with Existing Blockages</u>	321
<u>Filtering Objects by Net Name</u>	329

7

<u>Using Assisted Routing Commands</u>	333
<u>Point to Point Routing</u>	333
<u>Setting Point to Point Wires to Match Pins</u>	336
<u>Supporting Track-based Routing</u>	337
<u>Finishing a Trunk</u>	337
.....	337

8

<u>Editing Routed Connections</u>	339
<u>Stretching Wires and Vias</u>	339
<u>Variables that Control Stretch</u>	340
<u>Controlling the Angle of Wires</u>	341
<u>Using the Stretch Command</u>	341
<u>Preserving Paths</u>	343
<u>Stretching Wires Connected to Pins</u>	345
<u>Stretching Wires Connected to Vias</u>	345
<u>Stretching Wires Across Tracks</u>	346
<u>Moving Wires</u>	347
<u>Copying Wires</u>	347
<u>Re-routing a Wire</u>	348
<u>Reshaping Wires</u>	348
<u>Removing Loops From Wires</u>	352
<u>Merging Wires</u>	354
<u>Collinear Abutment or Overlap of PathSegs</u>	354
<u>Coincident Abutment or Overlap</u>	354
<u>Merging a Pathseg with a Path</u>	356
<u>Self-intersecting Objects</u>	356
<u>Deleting Wires</u>	358
<u>Copy Route Command</u>	359
<u>Setting Copy Route Options</u>	360
<u>Copying Pin-to-Pin Routes</u>	361
<u>Copying Pin-to-Wire Routes</u>	367
<u>Handling Wire Copies for EM Analysis</u>	370

9

Managing Metal Density	373
Metal Density Toolbar	373
Performing Metal Density Checks for Shapes	379
Deleting Fillets	390
Specification Cells	391
Fillet Creation (Virtuoso RF Solution Only)	396
Specifying the Fillet Options	398
Support for Slot Shapes in DRD and EAD	399

10

Interactive Colored Routing	401
Interactive Coloring for Create Wire	403
Creating a Colored Wire	403
Inserting Via on Colored Wires	406
Tapping Colored Objects	410
Interactive Coloring for Create Bus	417
Creating a Bus in an Empty Space	418
Creating a Bus by Tapping Pins	425
Creating a Bus While Working with Colored WSPs	434
Performing DRC Colored Checks	437
Interactive Coloring for Create Stranded Wire	438
Specifying the Color Mode in Stranded Wire	438
Changing the Color of Stranded Wire	440
Changing the Stranded Wire Color Mode	442
Creating a Stranded Wire by Tapping Pins	444
Creating a Stranded Wire While Working with Colored WSPs	447
Avoiding Shorts with Existing Blockages on Colored WSPs	449
Supporting End-to-End Spacing for Colored Shapes	454
Interactive Coloring for the Stretch Command with Colored WSPs	455
Color Locks While Editing a Wire	455
Inserting TrimMetal Layers in Interactive Colored Routing	457
Supporting Colored TrimMetal Layers	460
Supporting Colored TrimMetal Layers for the Stretch Command	462

Virtuoso Interactive and Assisted Routing User Guide

<u>Supporting Colored TrimMetal Layers for Create Bus</u>	464
<u>Supporting Patch Metal Shapes</u>	464

11

Supporting Width Spacing Patterns

<u>Width Spacing Pattern Display in Wire Editing</u>	470
<u>Dynamic Display</u>	470
<u>Alpha Blending</u>	472
<u>Customizing Display of WSP Tracks</u>	473
<u>allowedWireTypes Support</u>	474
<u>Snapping Behavior in WSP During Wire Editing</u>	474
<u>Enabling Wire Snapping</u>	474
<u>Snapping of Wire in Width Spacing Patterns</u>	476
<u>Support for snapPattern singleTrackCenter</u>	479
<u>Automatic Via Insertion during Wire Snapping</u>	482
<u>Snapping on WSP for Assisted Routing Commands</u>	486
<u>Working with Width Spacing Patterns During Wire Editing</u>	487
<u>Switching Patterns Dynamically</u>	488
<u>Crossing Region Boundaries</u>	490
<u>Wire Editor Gravity</u>	492
<u>Automatic Switching of the Active Pattern Across Regions</u>	494
<u>Working with Selected Sets of Wires</u>	496
<u>Nested WSP Regions</u>	497
<u>Region R1 Partially Overlaps Another Region R2</u>	497
<u>Two Regions Overlap With the Exact Same Boundary</u>	498
<u>Region R1 Partially Overlaps Another Region R2 With the Same Width Spacing Pattern</u>	499
<u>Region R1 is Fully Enclosed in Another Region R2</u>	499
<u>Supporting pathSegs with Different WSPs across Regions</u>	500
<u>Wire Editor Hierarchy Support</u>	503
<u>Pattern Flipping Support</u>	511

12

Supporting Virtuoso RF

<u>Supporting Curved Paths</u>	513

Virtuoso Interactive and Assisted Routing User Guide

<u>Creating a Packaging Wire</u>	515
<u>Inserting Via and Changing Layer</u>	517
<u>Stretching a Curved Path</u>	519
<u>Automatic Snapping Using Stretch</u>	519
<u>Curved Path Snapping Behavior in Packaging Wire</u>	522
<u>Tapping of Curved Paths</u>	525
<u>Creating a Packaging Bus</u>	526
<u>Snapping Behavior of Curved Paths Using Create Packaging Bus</u>	531
<u>Supporting Package Constraints</u>	533
<u>Area-based Rules in Create Package Wire</u>	534
<u>minWidth</u>	534
<u>maxWidth</u>	534
<u>minSpacing</u>	535
<u>Area-based Rules in Create Packaging Bus</u>	536
<u>minWidth</u>	536
<u>maxWidth</u>	537
<u>minSpacing</u>	537
<u>Push and Shove in Interactive Routing</u>	538
<u>Using Push and Shove while Creating a Wire</u>	538
<u>Using Push and Shove while Stretching a Wire</u>	542
<u>Support for Area-based Rules in the Push-and-Shove Feature</u>	545
<u>Support for Shielding Constraints in the Push-and-Shove Feature</u>	546

A

<u>Interactive and Assisted Routing Forms</u>	549
<u>Create Bus Form</u>	550
<u>Edit Bus Form</u>	556
<u>Create Wire Form</u>	558
<u>Create Stranded Wire Form</u>	563
<u>Point to Point Form</u>	567
<u>Select Via Form</u>	568
<u>Metal Density Options Form</u>	575
<u>Layers</u>	575
<u>Slot Geometry</u>	576
<u>Slotting Style</u>	577

Virtuoso Interactive and Assisted Routing User Guide

<u>Slot Consistency</u>	581
<u>Fillet</u>	581
<u>Slotting Button</u>	583
<u>Create Packaging Bus Form</u>	585
<u>Copy Route Form</u>	587
<u>General</u>	587
<u>Options</u>	591

B

Interactive and Assisted Routing Shortcut Menu Commands.

593

<u>Create Bus Shortcut Menu Commands</u>	594
<u>Edit Bus Shortcut Menu Commands</u>	606
<u>Create Single Wire Shortcut Menu Commands</u>	607
<u>Reshape Wire Shortcut Menu Commands</u>	617
<u>Split Wire Shortcut Menu Commands</u>	619
<u>Create Stranded Wire Shortcut Menu Commands</u>	621
<u>Point to Point Shortcut Menu Commands</u>	628
<u>Stretch Shortcut Menu Commands</u>	631

C

Interactive and Assisted Routing Environment Variables ..

<u>Variables to Customize the Interactive and Assisted Routing Settings</u>	636
<u>adjustEditedViasParams</u>	639
<u>adjustEditedViasParamsScope</u>	640
<u>allLayersWrongWay</u>	641
<u>allLayersWrongWayCustomValue</u>	642
<u>allowLoops</u>	643
<u>allowOffTrackForCrossRegion</u>	644
<u>autoDisableProbeDuringIARouting</u>	645
<u>autoLengthControl</u>	646
<u>autoTap</u>	647
<u>autoTapActiveLayer</u>	648
<u>autoTerminate</u>	649

Virtuoso Interactive and Assisted Routing User Guide

<u>autoUpdateFlightlines</u>	650
<u>autoViaAlignment</u>	651
<u>autoViaAlignmentColinearTransitionBehavior</u>	652
<u>autoViaAlignmentOrthoTransitionBehavior</u>	653
<u>blockageAvoidanceSearchThreshold</u>	654
<u>blockageUseMinWidth</u>	655
<u>checkOffGridEditedWires</u>	656
<u>constraintGroupCustomFilterMode</u>	657
<u>copyRoutingAllowCrossings</u>	658
<u>copyRoutingPreserveSpacing</u>	659
<u>createBusAllowsMultiLayerSelection</u>	660
<u>createWireMode</u>	661
<u>createWireStitchAware</u>	662
<u>displayMinAreaMarker</u>	663
<u>displayOrthogonalGridEndPoints</u>	664
<u>dynamicDisplayEnabled</u>	665
<u>enableDynamicPatternSwitching</u>	666
<u>enableFinishRoutingCmds</u>	667
<u>enableWidthOverride</u>	668
<u>enforceStretchWidthConstraints</u>	669
<u>fanOutToVias</u>	670
<u>flightLinesLimit</u>	671
<u>flightLinesPolygonPointsLimit</u>	672
<u>flightLinesShapesLimit</u>	673
<u>gatherAndSpreadSpacing</u>	674
<u>ignoreSameMaskColor</u>	675
<u>keepWiresConnected</u>	676
<u>layerTapPickWireVia</u>	677
<u>lockUnselectedVias</u>	678
<u>maxNumConnWireElements</u>	680
<u>mixedSignalWireEditingEnvironment</u>	681
<u>moveConnectedInstancesAndWires</u>	682
<u>moveConnectedObjectType</u>	683
<u>p2pSeedStyle</u>	684
<u>pathSegBeginExt</u>	685
<u>pathSegBeginStyle</u>	686

Virtuoso Interactive and Assisted Routing User Guide

<u>pathSegEndExt</u>	687
<u>pathSegEndStyle</u>	688
<u>patternFillAlpha</u>	689
<u>patternGravity</u>	690
<u>patternGravityLayerTransitionType</u>	691
<u>patternOutlineAlpha</u>	692
<u>patternRenderingAperture</u>	693
<u>pushAndShoveMode</u>	694
<u>pushAndShoveSmooth</u>	696
<u>pushAndShoveCheckerStopLevel</u>	697
<u>setTaperMode</u>	698
<u>showAlignmentMarkers</u>	699
<u>snapResizeWireLabels</u>	700
<u>snapToPin</u>	701
<u>snapToViaCenter</u>	702
<u>snapWireGrid</u>	703
<u>spaceFuncName</u>	704
<u>taperToPinWidth</u>	705
<u>taperToPinWidthStyle</u>	706
<u>tapPreference</u>	707
<u>useMergeBusAndWire</u>	708
<u>useOverlapRegionForVia</u>	709
<u>viaAlignment</u>	710
<u>viaCheckDRC</u>	711
<u>viaCutColorMode</u>	713
<u>viaCutColorPatterns</u>	714
<u>viaExtendEnclosureBeyondOverlap</u>	715
<u>viaFillEnclosureOverlap</u>	716
<u>viaMaxNumCuts</u>	717
<u>viaPattern</u>	718
<u>weAllowOddWidth</u>	719
<u>weAreaBasedRulesEnabled</u>	720
<u>weAreaBasedRulesHierLevel</u>	721
<u>weAreaBasedRulesWidthMode</u>	722
<u>weARWidthMode</u>	723
<u>weARTapWideEdge</u>	724

Virtuoso Interactive and Assisted Routing User Guide

<u>weARTapWidthOnPins</u>	725
<u>weARTapWidthOnWires</u>	726
<u>weAutoDropVia</u>	727
<u>weAutoDropViaForOverlaps</u>	728
<u>weAutoMergeWires</u>	729
<u>weAutoWidthModeAfterVias</u>	730
<u>weBusColorMode</u>	731
<u>weBusNumBits</u>	733
<u>weChangeWireDirectionAfterVia</u>	734
<u>weCoverPin</u>	735
<u>weCreateBusConstraint</u>	736
<u>weCustomLayerPattern</u>	737
<u>weCWStretchLastPointModifier</u>	738
<u>weCWWidthMode</u>	739
<u>weEnableDanglingTargetsFL</u>	740
<u>weEnableXLCheckoutInL</u>	741
<u>weEnforceDirFromRoutingPattern</u>	742
<u>weFlightLinesHierLevel</u>	743
<u>weGuidedRouteGatherAtBend</u>	744
<u>weHalo</u>	745
<u>weHaloDashStyle</u>	747
<u>weHaloEnabled</u>	748
<u>weHaloEOLSpacing</u>	749
<u>weHaloTrueColor</u>	750
<u>weInstancePinPattern</u>	751
<u>weInteractiveShielding</u>	752
<u>weKeepSelectViaOrder</u>	753
<u>weMemoryOptimization</u>	754
<u>weMfgGridWidthIncrement</u>	756
<u>weNoSmartSnap</u>	757
<u>wePostCriticWires</u>	758
<u>weShowCtrlWireAlignMarkerOnly</u>	759
<u>weShowHints</u>	760
<u>weSmartSnappingDepth</u>	761
<u>weStrandedAlignCollinearMode</u>	762
<u>weStrandedColorMode</u>	763

Virtuoso Interactive and Assisted Routing User Guide

<u>weStrandedLadderAtTurn</u>	765
<u>weStrandNum</u>	767
<u>weStrandSpacing</u>	768
<u>weStrandWidth</u>	769
<u>weTapWidthOnPins</u>	770
<u>weTapWidthOnVias</u>	771
<u>weTapWidthOnWires</u>	772
<u>weTrimBridgePurpose</u>	773
<u>weTrimInsertion</u>	774
<u>weTrimInsertionOnPRBoundary</u>	775
<u>weTrimPatchExtension</u>	777
<u>weTrimPatchPurpose</u>	778
<u>weTrimTrimPurpose</u>	779
<u>weWACmdCalculatedColor</u>	780
<u>weWAConflictColor</u>	781
<u>weWADefaultSeedCGName</u>	782
<u>weWASeedCGColor</u>	783
<u>weWAUserDefColor</u>	784
<u>wireConstraintGroup</u>	785
<u>wireOffset</u>	786
<u>wireTaperConstraintGroup</u>	787
<u>wireViaStackMode</u>	788
<u>wireWidth</u>	789
<u>Variables to Customize the Settings for Slotting</u>	790
<u>addFilletOnBondFingers</u>	792
<u>addFilletOnCircle</u>	793
<u>addFilletOnComplexShapes</u>	794
<u>addFilletOnOblong</u>	795
<u>addFilletOnOctagon</u>	796
<u>addFilletOnPaths</u>	797
<u>addFilletOnPadWithoutDrills</u>	798
<u>addFilletOnPins</u>	799
<u>addFilletOnRectangle</u>	800
<u>addFilletOnSquare</u>	801
<u>addFilletOnTconnections</u>	802
<u>addFilletOnVias</u>	803

Virtuoso Interactive and Assisted Routing User Guide

<u>areaDelimitedMode</u>	804
<u>curvedFillet</u>	806
<u>desiredAngle</u>	807
<u>densityAwareMode</u>	808
<u>directionalSlotPattern</u>	809
<u>enableSlotSpecCell</u>	811
<u>excludePins</u>	812
<u>floatingSlot</u>	813
<u>hideConvertToPolygon</u>	814
<u>hideReconstruct</u>	815
<u>lengthWidthRatio</u>	816
<u>lSpacing</u>	817
<u>mainSlotFigGroupName</u>	818
<u>material</u>	819
<u>maxAngle</u>	820
<u>maxLineWidth</u>	821
<u>maxOffset</u>	822
<u>maxSizeOnCircle</u>	823
<u>maxSizeOnComplexShapes</u>	824
<u>maxSizeOnOblong</u>	825
<u>maxSizeOnOctagon</u>	826
<u>maxSizeOnRectangle</u>	827
<u>maxSizeOnSquare</u>	828
<u>metalDensityLayers</u>	829
<u>minLineWidth</u>	830
<u>minPolygonEdgeLength</u>	831
<u>slottingContext</u>	832
<u>slotLength</u>	833
<u>slotOverBlockage</u>	834
<u>slotOverInstance</u>	835
<u>slotOrientationMode</u>	836
<u>slotOutOfGroup</u>	837
<u>slotSpecCell</u>	838
<u>slotStaggered</u>	839
<u>slotToEdge</u>	840
<u>slotVia</u>	841

Virtuoso Interactive and Assisted Routing User Guide

<u>slotWidth</u>	842
<u>taperedTraces</u>	843
<u>tapDesiredAngle</u>	844
<u>tapMaxOffset</u>	845
<u>tapMinSegAngle</u>	846
<u>touchBlockageMode</u>	847
<u>turnSlotPattern</u>	848
<u>unassignedShapes</u>	849
<u>useOneSlotSpecCell</u>	850
<u>usePaletteLayers</u>	851
<u>wSpacing</u>	852
<u>widthThreshold</u>	853
<u>Variables to Customize the Settings for Copy Route</u>	854
<u>copyRouteAllowCopiesWithoutMatchingPins</u>	855
<u>copyRouteBottomLayer</u>	856
<u>copyRouteClearPreviewWhenEsc</u>	857
<u>copyRouteCreateFigGroup</u>	858
<u>copyRouteEnableRoutingChannel</u>	859
<u>copyRouteEnableSpacingAtEnds</u>	860
<u>copyRouteEnableValidLayer</u>	861
<u>copyRouteEnableWidthAtEnds</u>	862
<u>copyRouteGenerateCopyWithErrors</u>	863
<u>copyRouteLayerPattern</u>	865
<u>copyRouteLayersMode</u>	866
<u>copyRouteNumCopiesInSpace</u>	869
<u>copyRoutePreserveCutClasses</u>	870
<u>copyRoutePreviewMode</u>	871
<u>copyRouteRoutingChannelLPP</u>	872
<u>copyRouteSpacingMode</u>	873
<u>copyRouteSpacingModeAtEnds</u>	875
<u>copyRouteSpacingOverrideAllLayers</u>	877
<u>copyRouteSpacingOverrideAllLayersAtEnds</u>	878
<u>copyRouteTopLayer</u>	879
<u>copyRouteTrimmingFixer</u>	880
<u>copyRouteWidthMode</u>	881
<u>copyRouteWidthModeAtEnds</u>	883

Virtuoso Interactive and Assisted Routing User Guide

<u>copyRouteWidthOverrideAllLayers</u>	885
<u>copyRouteWidthOverrideAllLayersAtEnds</u>	886

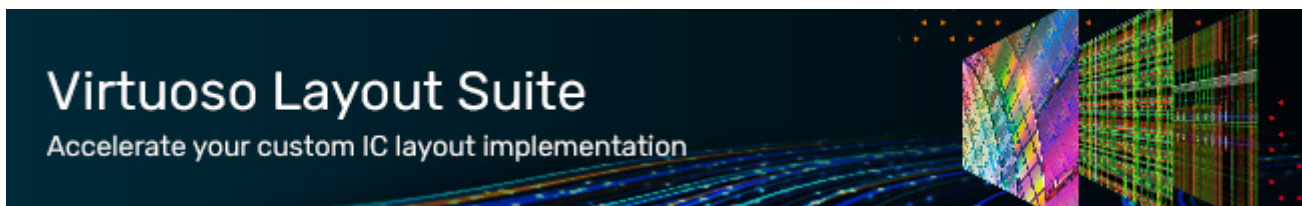
D

<u>Bindkey Keyboard Map</u>	887

E

<u>Routing Assistant - Interactive Mode</u>	889
<u>Routing Assistant Tabs</u>	889
<u>Routing Assistant Command Buttons</u>	912

Setting up the Interactive Routing Environment



If you choose not to use automatic routing, you can manually (or interactively) route the connections in your design. The interactive routing features complement automatic routing features. Interactive routing lets you do the following:

- complete critical nets before automatic routing.
- finish incomplete nets left after automatic routing.

The Virtuoso® Interactive and Assisted Routing capabilities enable you to route connections interactively within the Virtuoso environment. These capabilities provide efficient ways to interactively and automatically route connections in order to meet critical design constraints and rules. The interactive and assisted routing capabilities are fully enabled on all process nodes including the most advanced process technologies.

In this chapter we are going to explore the settings that can modify and change the behavior and functionalities of the interactive and assisted routing commands.

The chapter describes the following,

- Configuring Wire Editing Options
 - Layout Editor Options Form
 - Stretch Options
- Tapping Wires

Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

- ☐ [Tapping Internal Nets](#)
- ☐ [Tapping Vias](#)
- ☐ [Tapping Results Based on validVias and validLayers](#)
- [Adding the lxStickyNet Property to Wires](#)
- [Support for Voltage-Dependent Rules](#)
- [Support for Area-based Rules](#)
- [Using Mouse Button Controls](#)
- [Startup Warnings](#)

License Requirements of Interactive Routing Features

The Interactive wire editing features use the token-based license scheme under the Layout XL and higher tier licenses. The following table lists the features and the layout levels at which each is available.

Feature	VLS XL	VLS EXL
Create Geomteric Wire Note: The Create Geometric Wire command is available by default in CIW using a SKILL API leHiCreateGeometricWire . In addition, to add the Create Geometric Wire to the Create – Wiring menu, set the mixedSignalWireEditingEnvironment environment variable.	NA	NA
Create Wire	yes	yes
Create Stranded Wire	yes	yes
Stretch	yes	yes
Reshape	yes	yes
Split	yes	yes
Bus routing (Create Bus) Note: The Create multi-layer bus feature is unavailable in VLS L.	yes	yes
Point-to-Point (P2P)	yes	yes
Finish Trunk	yes	yes

Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

Feature	VLS XL	VLS EXL
DRD editing	yes	yes
Wire Assistant	yes	yes
Interactive SDR	NA	yes

* This license level offers a subset of feature functionality.

For information about licensing in the Virtuoso Studio design environment, see [*Virtuoso Software Licensing and Configuration Guide*](#).

Configuring Wire Editing Options

You can configure wire editing options to control the way the wire editor behaves as well as how it generates vias. You can set up wire editing options in:

- [Layout Editor Options Form](#)
- [Stretch Options](#)

Layout Editor Options Form

Choose *Options – Editor* to open the Point to Point Form. The fields relevant for creating and editing wires include options in the *Snapping*, *Tap*, *Wire Editing*, and *Halo* sections.

The *Default Wire Constraint Group* drop-down list box in the *Wire Editing* section allows the selection of one of the available constraint groups, which may be defined at the technology or the design level. By default, the *Default Wire Constraint Group* cyclic field displays a list of all available constraint groups. However, when the `constraintGroupCustomFilterMode` environment variable is set to `t`, the *Default Wire Constraint Group* cyclic field displays a customized list of constraint groups. Only the constraint groups for which the custom API `constraintGroupCustomFilter` returns `t` are displayed. The constraint group defines the appropriate layers and vias for the application being used. Only the layers, vias, and any rules specified in the selected constraint group are used when creating wires. Selection of a constraint group in the *Default Wire Constraint*

Group drop-down list box specifies the application default constraint group for wire creation and editing commands.

Important

Existing wires are not adjusted based on changes to the constraint group in the *Wire Editing* section. The recommended flow is to set the constraint group at the beginning of a session and to not change the constraint group while you are working on a design. To vary the constraints on different nets, set desired constraints on specific nets by using the *Process Rule Editor*.

If you update the constraint group while working on a design, remove the existing results (delete all routing) that would be effected by the constraint group change and re-route.

To override constraints in the application default constraint group at the design level,

1. Create a new design constraint group.
2. Inherit the application constraint group in the design constraint group.
3. Add new constraints (the ones that you need to override) in front of the linked application constraint group.
4. Select the new design constraint group as the application default constraint group in the *Default Wire Constraint Group* drop-down list box.

All container objects in the database have a default constraint group associated with them. Creating a rule override directly on a net or on the design requires adding that rule to the default constraint group of the net or the design. CST is the constraint look-up mechanism to determine the precedence of constraints applied at various levels. The precedence order followed by the router is:

- a.net default constraint group
- b.global net default constraint group
- c.design default constraint group
- d.foundry constraint group

Related Topics

[Point to Point Form](#)

[Tapping Wires](#)

Stretch Options

In Virtuoso Layout Suite, logical connectivity is not taken into account when stretching wires. If DRD is on, DRD uses logical connectivity to flag or avoid shorts. Choose *Edit – Stretch* to open the [Metal Density Options Form](#).

For the stretch command, there is no initial check for constraints. The constraint check depends on whether the `constraintAwareEditing` environment variable is ON or OFF. If it is ON, you can create a bus and check the constraints on each net. However, if it OFF, create a set of wires and do not check the `minSpacing` constraints.

This is recommended for the interactive routing commands to improve the runtime and also the use model.

In addition, in the *Configure* tab of the Connectivity form, you can deselect the *Constraint-aware editing* option.

Related Topics

[Metal Density Options Form](#)

[Stretching Wires and Vias](#)

Tapping Wires

For a description of the Layout Editor Tap options, see [Point to Point Form](#).

For more information about selecting the starting and target object layers, without tapping an object, see [Smart Snapping in Interactive and Assisted Routing Commands](#).

When creating a wire, layers can be selected by clicking on an LPP in the Layers Assistant or by tapping on objects. You can select to tap the objects layer and/or attributes. In the Layout XL or higher applications, in addition to object attributes, an existing net name can be tapped. To disable the layer tapping, hold the `Shift` or the `Ctrl` key.

Note: If a setup constraint group is being used, this constraint group controls which layers can be tapped. Additional constraint groups that define interconnect layers must contain the same layers or a subset of the layers as those in the setup constraint group.

There are two different tap options, one is used when creating shapes and the other is used when creating wires.

■ *Options – Layout Editor – Auto Tap -- Wire*

Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

Turning on *Auto Tap – Wire* will automatically tap an object's layers and attributes when digitizing the first coordinate of a wire. Layers that can be tapped are determined from the technology file.

When two or more layers are tapped, the Choose Object to Tap From form is displayed. However, when one of the tapped layers is already selected in Layers Assistant, it prevents displaying the Choose Object to Tap From form. For more information, refer to [Point to Point Form](#).

When using the Layout L editor, available layers are determined from the `viaDefs` statement. When using the Layout XL editor and higher tiers, valid routing layers should be specified in a setup constraint group. When tapping an object, only layers specified in the setup constraint group can be tapped.

■ *Options – Layout Editor – Auto Tap – Shape*

Turning on *Auto Tap – Shape* will automatically tap an object's layers and attributes when digitizing the first coordinate of a rectangle, polygon, path, circle, donut, and ellipse. All layers can be tapped.

To tap an object or automatically set the layer when digitizing the first coordinate of a wire on an existing wire or object, turn on *Auto Tap – Wire* in the *Layout Editor Options* form.

The level of hierarchy that you are allowed to tap is dependent on the *Display Levels* set in the *Display Options* form.

To tap specific layers purposes, create a prioritized list through the *Tap Purpose List* on the *Layout Options* form, or through the environment variables, `useTapPurposeList` and `tapPurposeList`. The default purpose is `drawing`, which can be overridden through *Tap Purpose List*.

For example, when *Tap Purpose List* is:

```
drawing    net
```

- The default purpose for `metal1` is `drawing`.
- When tapping an object on the LPP `metal1 net`, the active layer is changed to `metal1 net`.
- When tapping an object on an LPP which is not listed the *Tap Purpose List*, such as `metal1 pin`, the active layer does not change. The last layer chosen remains the active layer.

Once you have chosen a purpose for a layer, whether through the LSW or through tapping an object, the purpose will be used for subsequent create commands. If the layer has not yet been used in the current command, the current purpose will be used.

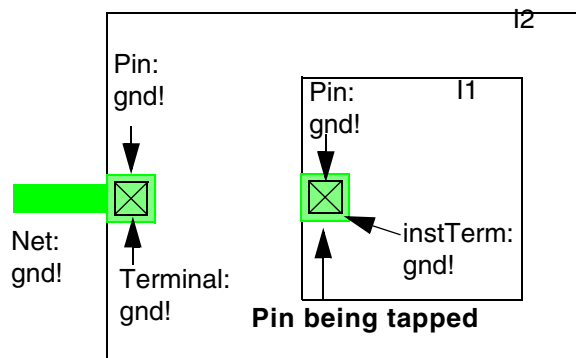
Tapping Internal Nets

When tapping on a pin that is at a lower level of the hierarchy, the connectivity must be traceable to the top level in order for the connectivity to be applied. This corresponds to the **MUST** connect relationship.

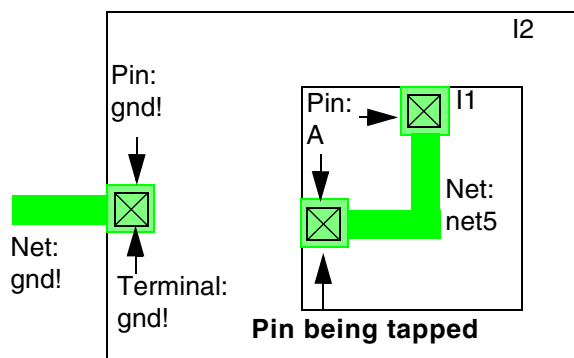
In Example 1 below, the pin `gnd!` at level 2 in the hierarchy is connected to net `gnd!`. At the top level there is a net `gnd!` and a pin `gnd!` to which the lower level `gnd!` is connected. In this case tapping on the pin at level 2 in the hierarchy will return correct net information.

In Example 2 below, when tapping on one of the `A` pins in `I1`, which is connected by internal net `net5`, the net information cannot be traced to a top level term/instTerm. In this example, no net name will be given to the new object being created.

Example 1



Example 2



If a net information can not be traced up to a top level net, then warning messages will be output to the CIW. If the wire does not have a net name, the software is free to merge the wire to another wire that does not have a net name at the top level. The result of such merge may not be the desired results. Therefore, it is advisable to manually cancel out the command when warning 105200 or 105201 messages appear.

When tapping is enabled, the net on the tapped shape is considered. Else, the net selected in the Navigator assistant, if any, is considered.

Tapping Vias

In Virtuoso L, when you click a via or a stacked via, you are prompted with the *Choose object to tap from* form to choose a layer from which to extend a wire from the via.

Note: When a via, which exists on the overlapped layers, is being tapped, the *Choose Object to Tap From* form is bypassed if the overlapped layer is already selected in Layers Assistant. This is only `TRUE` when smart snapping is disabled.

In Virtuoso XL and higher tiers, when you click a via or a stacked via, if there is an existing segment connected to one layer of the via, the resulting layer tap will be on the unoccupied layer of the via. However, in some situations, you may want to tap on a via and create a T-junction, rather than extend a wire from a dangling end of a wire.

To change the default behavior of Virtuoso XL and higher tiers and be prompted with the *Choose object to tap from* form when tapping a via, do the following.

1. Select *Options – Editor*.

The *Layout Editor Options* form opens.

2. Select the *Select from Overlaps* and *Include All (Wire) Via Layers* check boxes.

The associated environment variables are:

```
envSetVal("layoutlayerTapPickWireVia" 'boolean t)
envSetVal("layoutlayerTapPick" 'boolean t)
```

Tapping Results Based on validVias and validLayers

When tapping on an object, any via listed in the `validVias` constraint can be placed in the design and the resulting wire will be started on a layer that is listed in the `validLayers` constraint.

For example, if the `validLayers` are M2 and M3, and the `validVias` are M1M2 and M3M4, when tapping on a M1 or M4 shape, you must choose between M1 and M4 even though the valid layers are M2 and M3. The router can drop either a M1M2 or M3M4 via because they are both listed in the `validVias` constraint and the resulting wire would be started on a valid layer. Layers are only auto selected when there is only one available option.

Stacked vias are also allowed. In the example above, if an M4M5 via was also included in the `validVias`, then tapping on an M5 shape would be allowed.

In the same example of `validLayers` being M2 and M3, if two shapes were overlapping, one on M3 and one on M4, then the wire would start on M3 shape without choice. The reason is that it would require an M3M4 via to have the M4 shape on a legal layer, but doing so would short with the existing M3 shape, hence M4 would not be an option.

Creating Halos

You can now create a halo by layer for collinear wires. To do this:

1. *Choose Options – Editor – Auto Tap -- Wire.*
2. Select a scope for creating a halo from the drop-down list.
3. To create a dashed halo around a wire, bus, stranded wire. select the *Dashed* check box.
4. To create a halo in the color of that layer for each layer, select the *True Color* check box.
5. Click *OK*.

Adding the `lxStickyNet` Property to Wires

The `lxStickyNet` property allows a shape to retain the assigned connectivity. In the Virtuoso environment there are two types of nets, known and unknown nets.

- known nets: nets that are defined by pins and components.

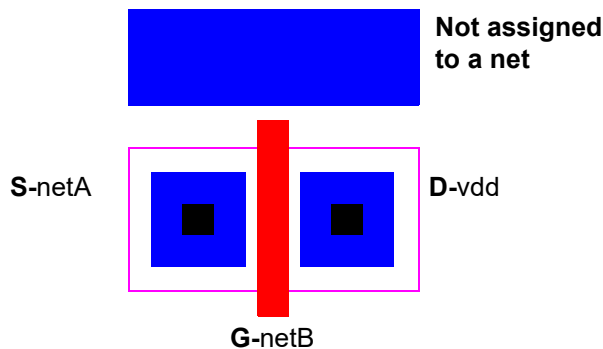
In the Virtuoso XL and higher tier editors, if a shape which is not assigned to a net, touches a shape on known net, the shape is assigned to that net. When the net assignment is defined by pins and components (`instTerms`), the assignment can not be re-assigned.

- unknown nets: shapes that are not assigned to a net

When a shape touches nothing, it is considered a floating shape.

- ☐ When specifying a net name for a floating shape in Virtuoso L, the shape is given the `lxStickyNet` property and the property is displayed on the property form.
- ☐ When specifying a net name for a floating pathSeg created in a route in Virtuoso XL and higher, the route is given the `lxStickyNet` property and the property is not displayed on the property form.
- ☐ The autorouter will assign the `lxStickyNet` property to routes created as pathSegs created in routes.

- ❑ The power router does not create pathSegs in routes, however the shapes will be assign the `lxStickyNet` property. When a shape is removed from connecting shapes the net name is retained.
- ❑ Only one element needs to be marked sticky on a floating island for all the elements to retain the assigned net.



In this example, known nets are those of the device's instTerms (component net assignments, netA, netB, and vdd).

The unknown net is the power rail. The power rail is not connected to the device, the layout editor extractor is free to assign and reassign the shape to a net.

In the Virtuoso XL editor and higher, to manually assign a net to the power rail shape and not allow the extractor to re-assign to a net, first assign the `lxStickyNet` property then assign a net name, The power rail will retain its connectivity.

Support for Voltage-Dependent Rules

Voltage-dependent rules are critical for design processes at 45 nm and less. The support for these rules is based on the ability to use user-defined layer purposes that represent voltage ranges and to define design rule constraint groups within these voltage ranges.

To support these rules, you can define purpose-aware constraints, such as those for width, spacing, and clearance, in the technology file. Purposes are used to propagate the voltage information, though without any connectivity; nets drawn on such purposes are, therefore, called pseudo nets.

The wire editor and the Wire Assistant support the voltage-dependent rules. You can use the voltage-aware, non-drawing layer purposes to create wires, buses, and vias. If you add a via and switch to a higher or a lower layer, the voltage-specific purpose is propagated to the higher or lower layer as you continue the wire or bus creation. The pseudo-net wires you create comply with the voltage-related spacing rules.

You can tap a wire on a voltage-specific layer-purpose pair and create new elements on the same voltage-specific layer-purpose pair.

Note: The purpose-aware constraint groups are not supported by the assisted routing commands, such as *Point to Point Routing*.

Virtuoso Interactive and Assisted Routing User Guide

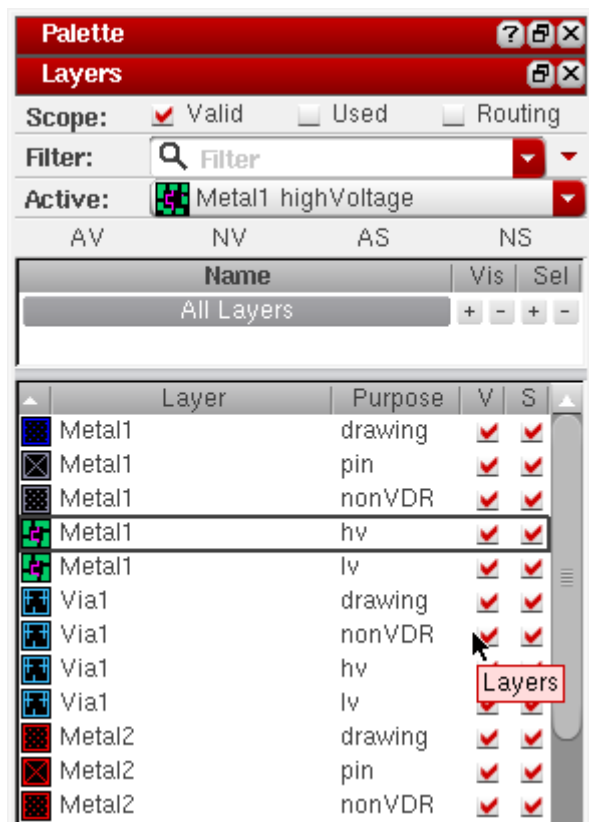
Setting up the Interactive Routing Environment

The following figure shows a section from a sample technology file where the voltage-related purposes are defined along with the parent purposes from which they are derived.

```
techPurposes (
; ( PurposeName Purpose# Abbreviation [Attributes] )
; ( ----- )
;User-Defined Purposes:
( lowVoltage 12 lv 'parent "drawing" 'voltageRange (0.0 1.8) )
( highVoltage 13 hv 'parent "drawing" 'voltageRange (1.8 18.0) )
) ;techPurposes
```

For more information about the `techPurposes` section of the technology file, see the [Virtuoso Technology Data ASCII Files Reference](#).

The figure below shows these voltage-specific purposes appearing in the *Layers* assistant:



Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

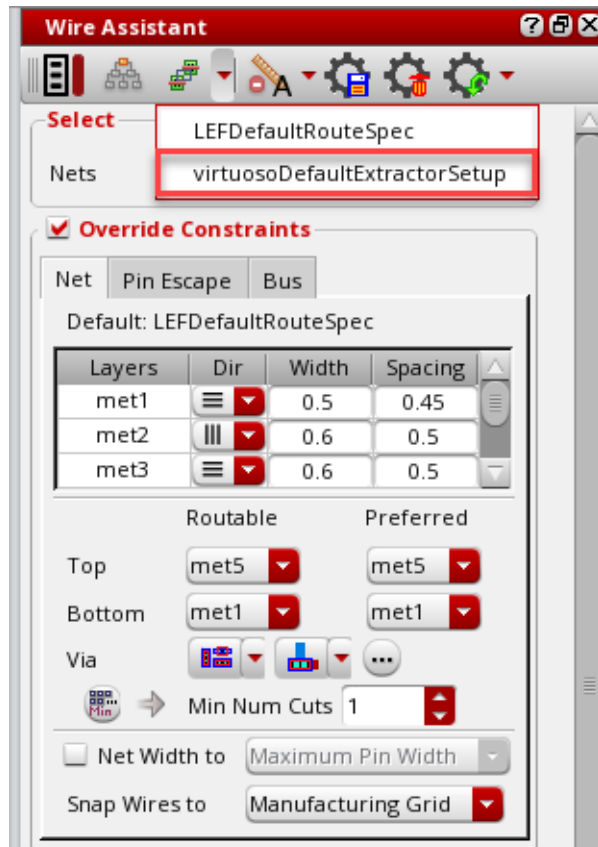
The section below from the sample technology file sets up a voltage-related constraint group.

```
; ( group          [override] )
; ( -----      ----- )
( "LPPtestSetup"      nil

    interconnect(
        ( validLayers ( ( Metal1 highVoltage) ( Metal2 highVoltage)
( Metal3 highVoltage) ( Metal1 lowVoltage) ( Metal2 lowVoltage)
( Metal3 lowVoltage) ( Via3 all) ( Metal4 all) ( Via4 all) ( Metal5 all)
( Via5 all) ( Metal6 all) ) )
        ( validVias      (M3_M2_new M2_M1_new M3_M2_std M2_M1_std) )
    ) ;interconnect
);LPPVolatgeSetup
```

For more information, see the purpose-aware `validLayers` constraint of the technology file.

From the *Seed Attributes From a Constraint Group* list on the Wire Assistant toolbar, you can select the voltage-related constraint group so that the voltage-related layer-purpose pairs from only that constraint group can be set in the *Override Constraints* section.



Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

The following sections from the sample technology file illustrate the voltage-related rule setups.

```
spacings (
  ( minWidth      "Metal1"      0.3 )
  ( minWidth      ("Metal1lowVoltage") 0.33 )
  ( minWidth      ("Metal1highVoltage") 0.34 )
  ( minWidth      "Metal2"      0.4 )
  ( minWidth      ("Metal2lowVoltage") 0.44 )
  ( minWidth      ("Metal2highVoltage") 0.45 )
  ( minWidth      "Metal3"      0.5 )
  ( minWidth      ("Metal3lowVoltage") 0.55 )
  ( minWidth      ("Metal3highVoltage") 0.56 )
); spacings
```

Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

```
spacingTables(  
  ;( constraint          layer1          [layer2]  
  ;(( index1Definitions  [index2Defintions]) [defaultValue] )  
  ;( table) )  
  ;( -----)  
  ( minVoltageSpacing    "Metal1"  
    (( "voltage"        nil      nil ))  
    (  
      0.0    0.38  
      1.8    0.39  
      3.3    0.4  
    )  
  )  
  ( minVoltageSpacing    "Metal2"  
    (( "voltage"        nil      nil ))  
    (  
      0.0    0.48  
      1.5    0.49  
      3.3    0.5  
    )  
  )  
  ( minVoltageSpacing    "Metal3"  
    (( "voltage"        nil      nil ))  
    (  
      0.0    0.58  
      1.5    0.59  
      3.3    0.6  
    )  
  )  
);spacingTables
```

For more information, see the [minVoltageSpacing](#) rule.

If there are voltage-dependent rules defined in the technology file, the wire editor applies them based on the following precedence, in the order specified:

Layer rule with voltage-dependent purpose > Voltage spacing rule with spacing dependent on the voltage values specified for one layer or two layers > Layer rule with parent purpose > Layer rule

Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

The override value specified in the *Create Wire* or the *Create Bus* form takes precedence over the value specified in the technology file if the override value complies with the rule.

For example, for `Metal1`, the technology file contains the following spacing rules:

- `minVoltageSpacing`
- `minSpacing ("Metal1" " 0.3")`
- `minSpacing [("Metal1" "drawing") "0.35"]`
- `minSpacing [("Metal1" "lv") "0.33"]` and `minSpacing [("Metal1" "hv") "0.35"]`

In addition, the *Bit Spacing* value (distance between the centerlines of bus wires) specified in the *Create Bus* form is 0.

In the absence of any override value, the wire editor derives the spacing between the bus wires as per the following lookup precedence:

- For lowVoltage (lv) layer purpose:
`minSpacing ("Metal1" "lv") > minVoltageSpacing > minSpacing ("Metal1" "drawing") > minSpacing ("Metal1")`
- For highVoltage (hv) layer purpose:
`minSpacing ("Metal1" "hv") > minVoltageSpacing > minSpacing ("Metal1" "drawing") > minSpacing ("Metal1")`

Consider a `cdsVia` device with a *Contact Layer* purpose, such as `open`, that does not have a parent purpose defined in the technology file and the purpose is not voltage-related. If you instantiate this `cdsVia` to change a wire from one layer to another, the purpose of the `cdsVia` master device is retained. You can verify this by checking the *Cut Purpose* field in the *Edit Via Properties* form for the `cdsVia`. For example, the following `cdsVia` device is installed in the technology file:

```
name: M1_M2_Via1
layer1 purpose1: Metal1 drawing
cutLayer cutPurpose: Via1 open
layer2 purpose2: Metal2 drawing
```

Create a wire on `Metal1 drawing` and create the `cdsVia M1_M2_Via1` to switch to `Metal2 drawing`. If you check the properties of the created `cdsVia`, you will notice the following:

```
Metal Purpose: drawing
Cut Purpose: open
Other Purpose: drawing
```

Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

For information about installing and creating cdsVias, see [Support for Voltage-Dependent Rules](#) in the Virtuoso Layout Suite documentation.

The interactive and assisted routing commands, such as, Create Bus, Point to Point routing, and Guided routing, use the voltage range information (`minVoltage`, `maxVoltage`) on a net to check or enforce Voltage Dependent constraints, such as voltage dependent spacing (`minVoltageSpacing`). For more information about editing *Min Voltage* and *Max Voltage* values, refer to [Editing Voltage Information](#) in the Virtuoso Layout Suite documentation.

The `voltageSyncNet` constraint is also supported for interactive routing commands. Lets see the voltage swing when `voltageSyncNet` is not defined.

```
Vswing = Max(Vmax_net1, Vmax_net2) - Min(Vmin_net1, Vmin_net2)
```

Voltage swing when `voltageSyncNet` is defined for net1 and net2.

```
Vswing = Max (abs(Vmax_net1 - Vmax_net2), abs(Vmin_net1 - Vmin_net2))
```

Here is an example with simple values to let you know the difference when `voltageSyncNet` is defined and when it is not defined.

Example:

Lets assume two nets, net1 and net2.

```
Vmin_net1 = 0, Vmax_net2 = 3.5
```

```
Vmin_net2 = 1, Vmax_net2 = 4.5
```

Voltage swing value when `voltageSyncNet` constraint is not defined for net1 and net2:

```
Max(4.5, 3.5) - Min(1.0, 0.0) = 4.5 - 0.0 = 4.5.
```

If you lookup the spacing from the table, the `minSpacing` is 0.3.

Voltage swing value when `voltageSyncNet` constraint is defined for net1 and net2:

```
Max(abs(4.5-3.5), abs(0.0-1.0)) = Max(1.0, 1.0) = 1
```

When you lookup the spacing from the table, the `minSpacing` is 0.065.

```
constraintGroups (
  ( "minVoltageSpacingCG"      nil      nil      'and
    spacingTables (
      ( minVoltageSpacing "Metall" "Vial"
        (( "voltage" nil nil ))
        (
          0.0  0.065
          1.5  0.15
          3.3  0.3
        )
      )
    )
  )
)
```

Support for Area-based Rules

You can define different rules for different regions. Once the rules are defined, area-based rules automatically adjust the edited shapes in the design. In interactive routing, area-based rules are disabled by default. To enable area-based rules by default, use the `weAreaBasedRulesEnabled` environment variable.

The following area-based rules are supported for interactive routing:

- `minWidth`
- `allowedWidthRange`
- `allowedLengthRange`
- `rectShapeDir`
- `minSpacing`
- `minArea`
- `Via Rules`

Note: The area-based rules are currently supported for *Create Wire* and *Create Bus* commands only. Also, by default, the area based rules follow the traverse level set to 1.

Virtuoso Interactive and Assisted Routing User Guide

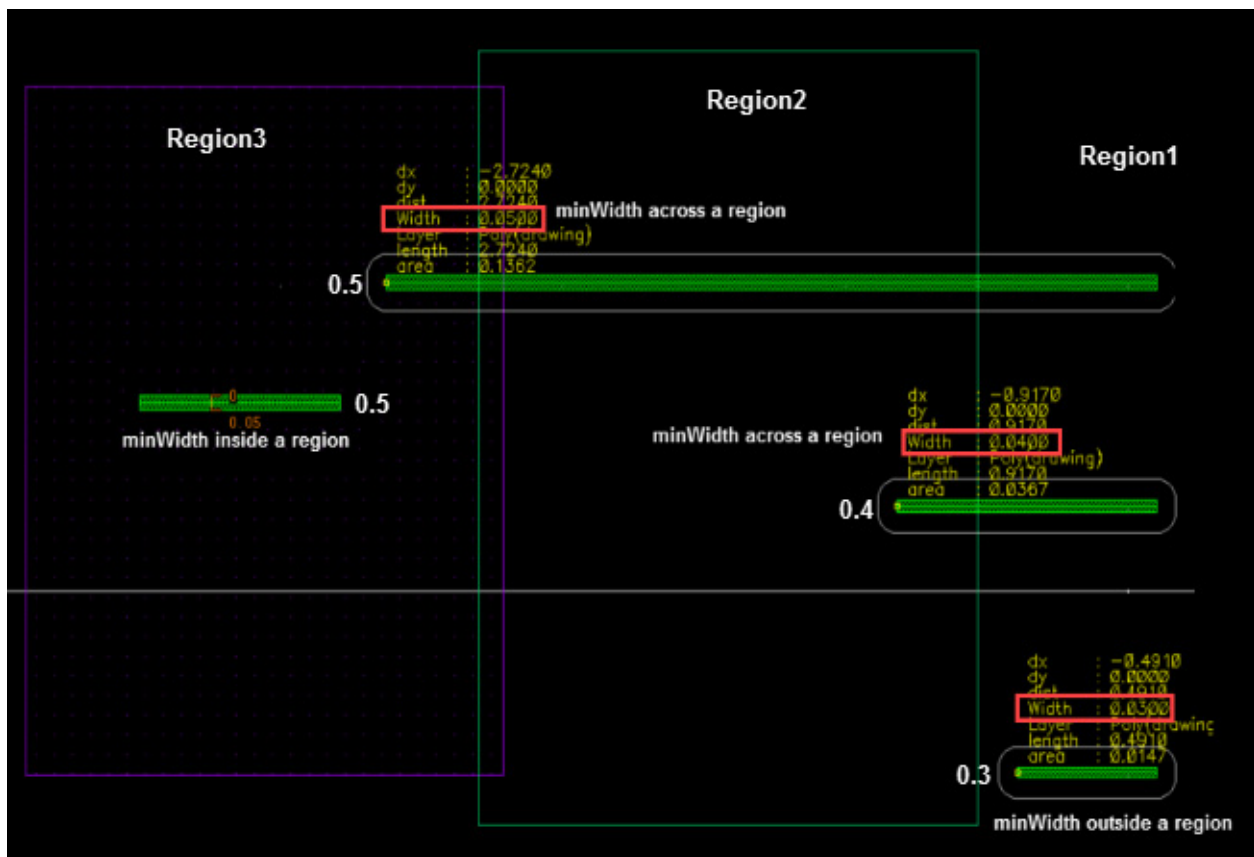
Setting up the Interactive Routing Environment

minWidth

When the pathSeg is in a region, it follows the minWidth rule of the region. The minWidth value of a pathSeg that spans regions is the maximum minWidth value across the spanned regions. For example:

```
spacings (
  ( minWidth "Poly" 'vertical 'insideLayers ("DMY1") 0.04 )
  ( minWidth "Poly" 'vertical 'insideLayers ("DMY2") 0.05 )
  ( minWidth "Poly" 'vertical 'outsideLayers ("DMY1" "DMY2") 0.03 )
  ( minWidth "Poly" 'horizontal 0.02 )
) ;spacings
```

The following figure shows an example of a pathSeg inside a region, outside a region, and across regions. It shows that when the pathSeg spans regions, the maximum minWidth value across the spanned regions is considered.



allowedWidthRange

When the pathSeg is in a region, it follows the `allowedWidthRange` rule of the region. The `allowedWidthRange` value of a pathSeg uses different widths in different regions when the pathSeg spans across regions. The width value is now controlled by the `weAreaBasedRulesWidthMode` environment variable.

When the width mode is specified as `commonWidth`, the width of a pathSeg that spans across regions is the width that is allowed for all the spanned regions. If multiple width values are allowed, the width value nearest to the user-specified width value is used. If there is no common width for the spanned regions, the pathSeg cannot be created across the regions. Here are some examples of the `allowedWidthRange` area-based rule when the width mode is `commonWidth`.

Example 1

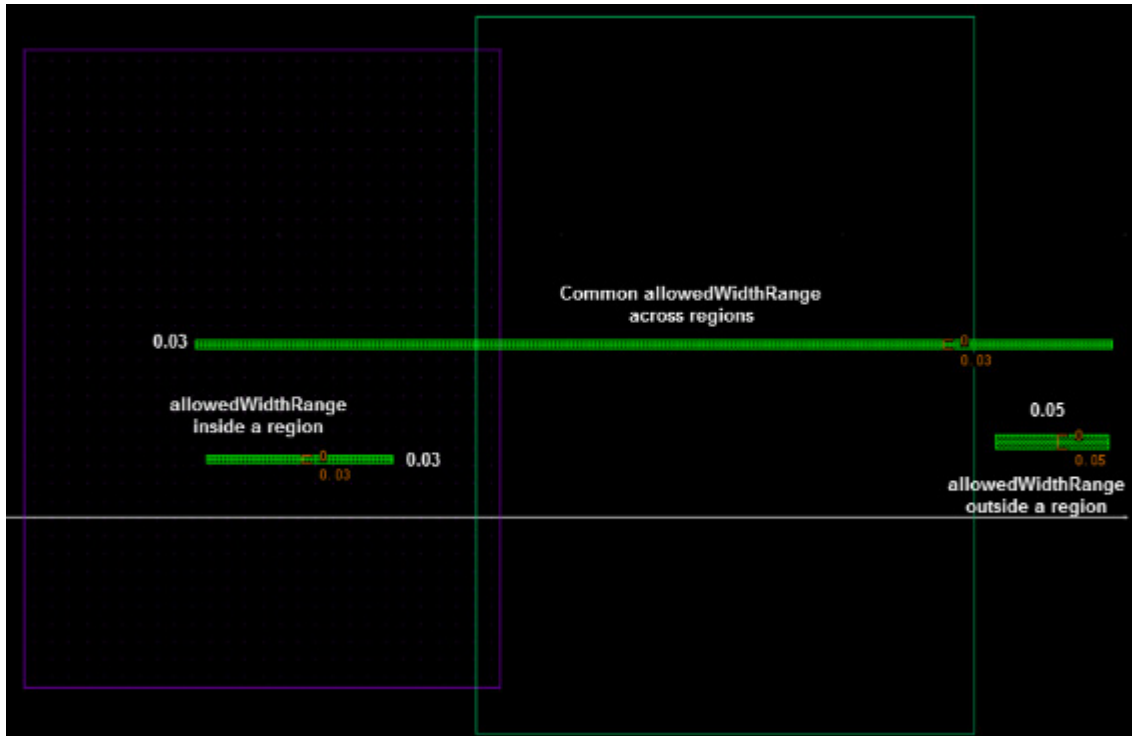
```
spacings (
  ( allowedWidthRanges "Poly" 'vertical    'insideLayers ("DMY1") (0.03 0.04) )
  ( allowedWidthRanges "Poly" 'vertical    'insideLayers ("DMY2") (0.03) )
  ( allowedWidthRanges "Poly" 'vertical    'outsideLayers ("DMY1" "DMY2") (0.03 0.04
0.05) )
  ( allowedWidthRanges "Poly" 'horizontal (0.03 0.05 ">= 0.08"))
) ;spacings
```

The following figure shows an example of `allowedWidthRange` that is specified for a pathSeg when inside a region, outside a region, and across regions. When the pathSeg

Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

spans across regions, the common `allowedWidthRange` of the spanned regions is considered.



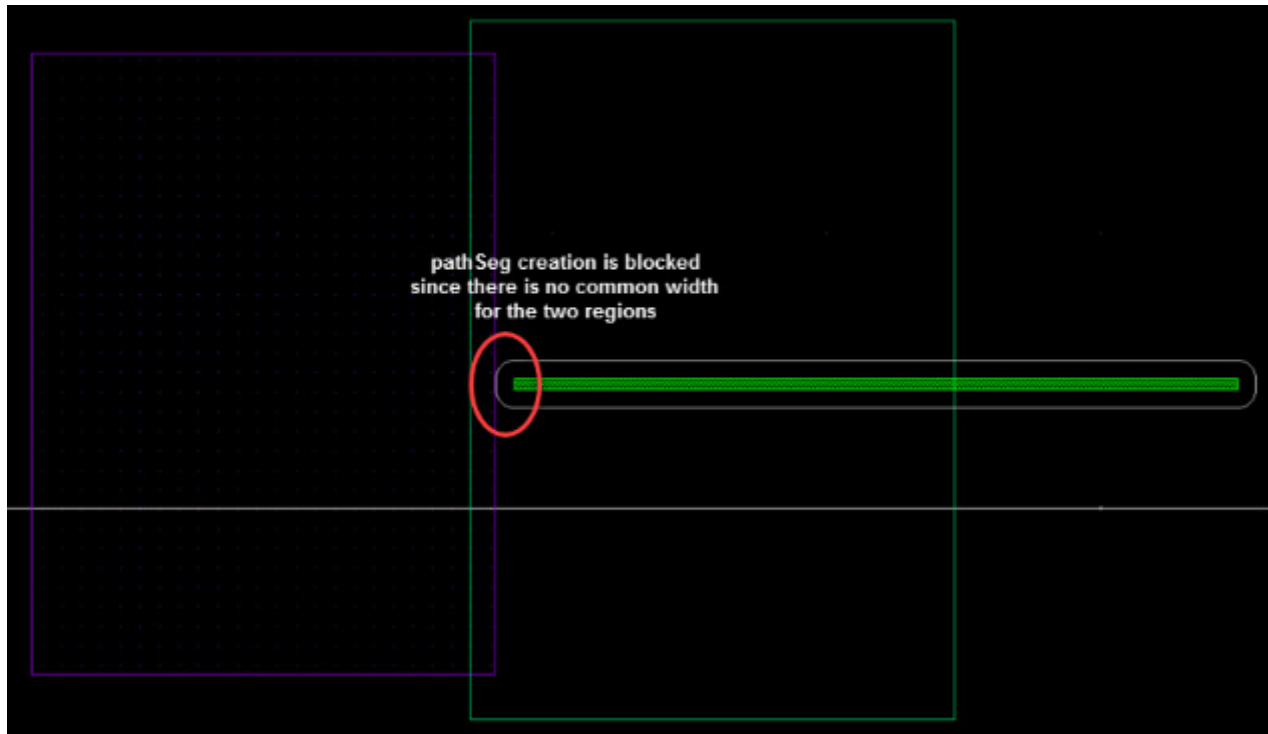
Example 2

```
spacings (  
  ( allowedWidthRanges "Poly" 'vertical 'insideLayers ("DMY1") (0.03 0.04) )  
  ( allowedWidthRanges "Poly" 'vertical 'insideLayers ("DMY2") (0.06) )  
  ( allowedWidthRanges "Poly" 'vertical 'outsideLayers ("DMY1" "DMY2") (0.03 0.04 0.05) )  
  ( allowedWidthRanges "Poly" 'horizontal (0.03 0.05 ">= 0.08"))  
) ;spacings
```

Virtuoso Interactive and Assisted Routing User Guide

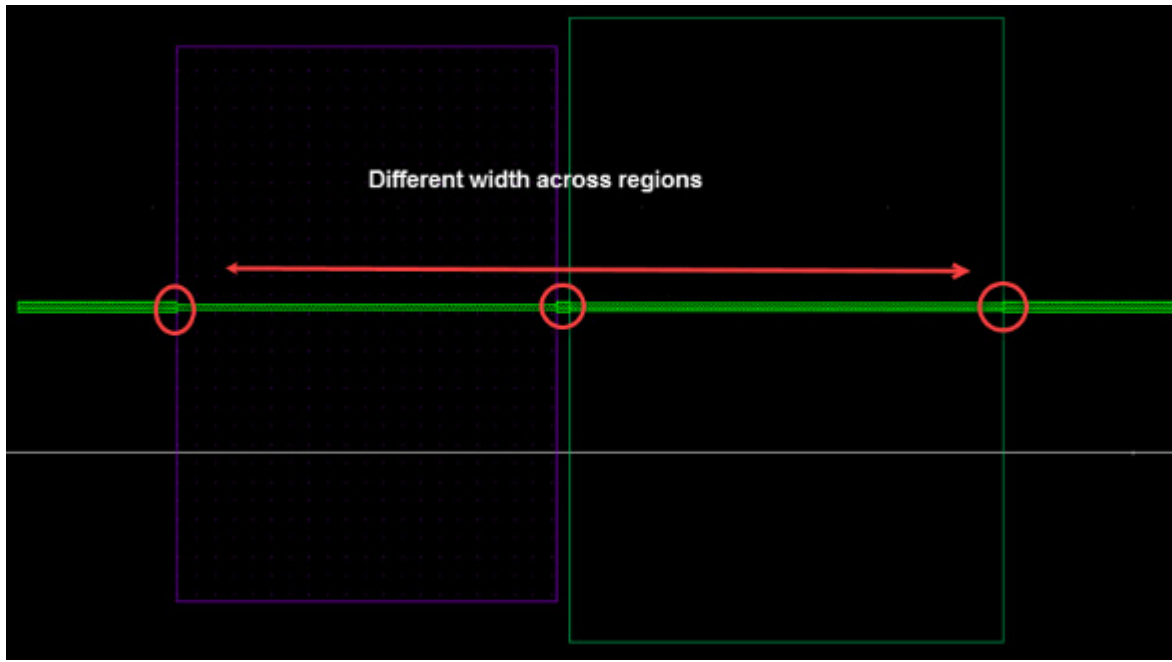
Setting up the Interactive Routing Environment

The following figure shows an example where the pathSeg cannot be created because there is no common width across regions.



However, when the width mode is specified as `nonCommonWidth`, the pathseg is split when it spans across regions. For each part of the pathseg, the width is adjusted according to the allowed width in each region. Therefore, the pathSeg has different widths for each part and

the width for each part is the allowed width that is nearest to the user-specified width, as shown in the following figure.



allowedLengthRange

When the pathSeg is in a region, it follows the `allowedLengthRange` rule of the region. However, when the pathSeg spans across regions, the common `allowedLengthRange` of the spanned regions is considered. If there is no common length for the spanned regions, the pathSeg cannot be created across the regions.

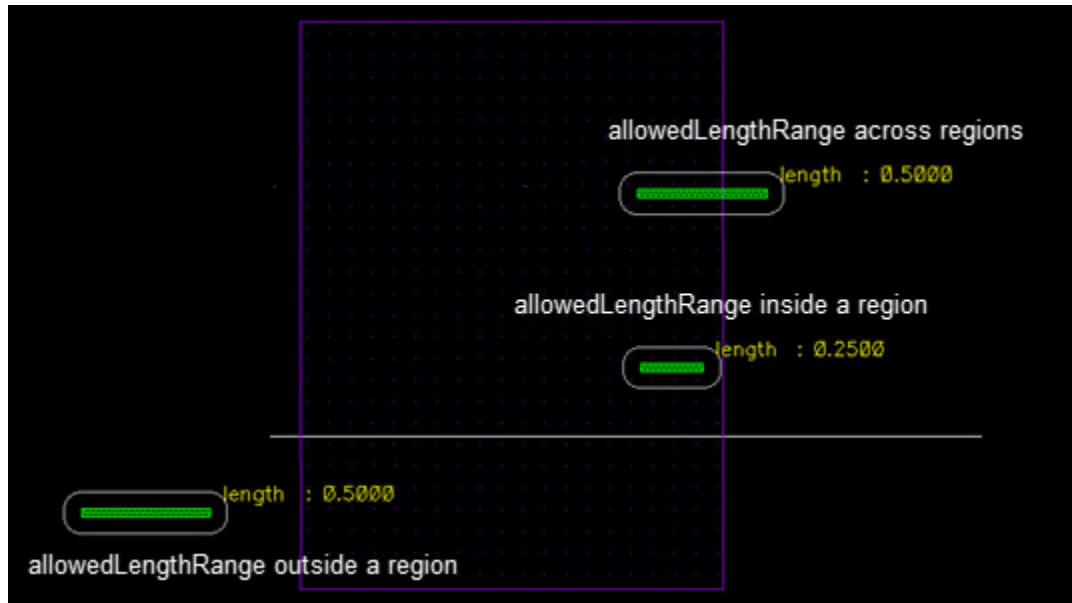
Example:

```
( allowedLengthRanges      "Poly"
  (( "width"   nil  nil  ) 'insideLayers ("NWell"))
  (
    0.01      (">= 0.25")
  )
)
( allowedLengthRanges      "Poly"
  (( "width"   nil  nil  ) 'outsideLayers ("NWell"))
  (
    0.01      (">= 0.5")
  )
)
```

Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

The following figure shows an example of a pathSeg inside a region, outside a region, and across regions. It shows that when the pathSeg spans regions, the common `allowedLengthRange` value across the spanned regions is considered.



rectShapeDir

When the pathSeg is in a region, it follows the `rectShapeDir` rule of the region. However, if the direction of the pathSeg in that region is not valid, the layer of the pathSeg is automatically changed to satisfy the `rectShapeDir` rule.

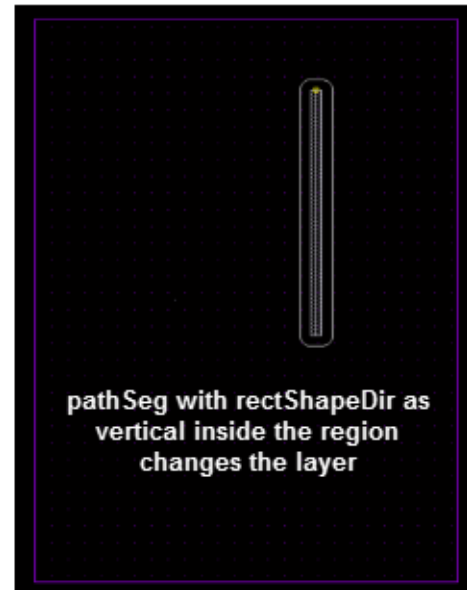
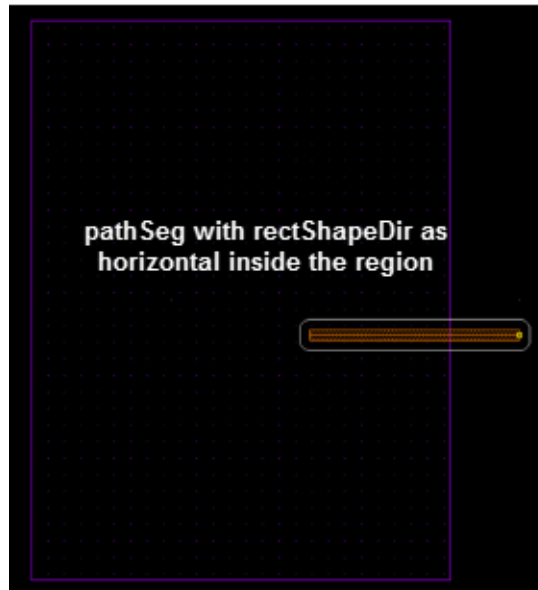
Example 1

```
spacings (
  ( rectShapeDir          "Poly"  'insideLayers ("DMY")
    {"horizontal"}
  )
) ;spacings
```

Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

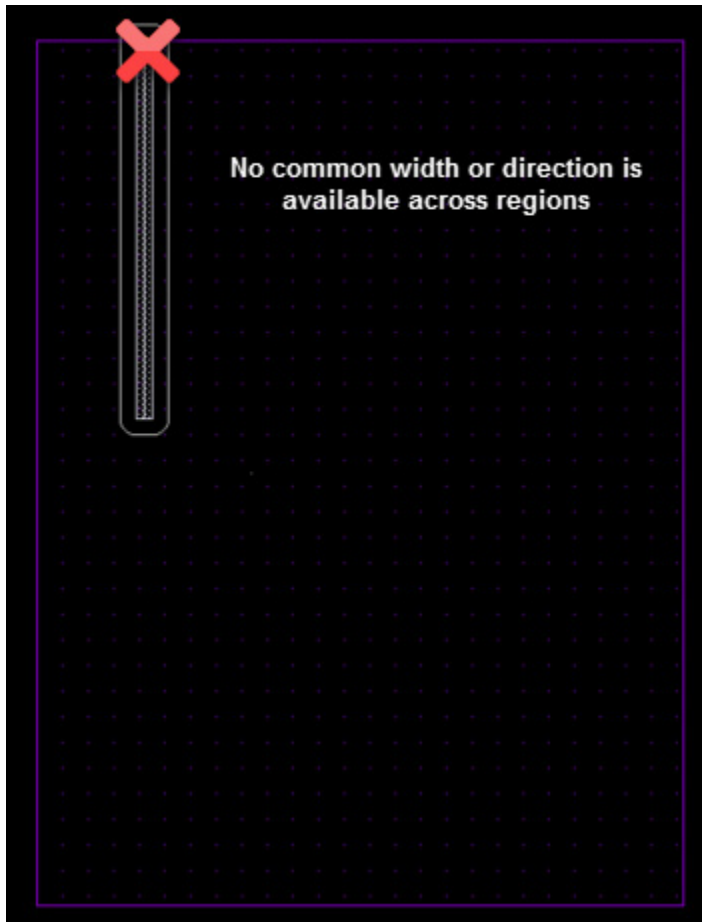
The following figure shows an example, where the `rectShapeDir` rule defined in the technology file supports the horizontal direction inside the region. As soon as the direction of the `pathSeg` is changed to vertical, the layer of the `pathSeg` is automatically changed.



Example 2

```
( rectShapeDir "Poly"
  (( "width" nil nil ) 'insideLayers ("DMY") )
  (
    0.03 "vertical"
    0.04 "vertical"
    0.05 "any"
  )
)
```

When the pathSeg spans across regions, it uses the common width or common direction of the spanned regions. However, if there is no common width or common direction for the spanned regions, the pathSeg cannot be created across regions.



minSpacing

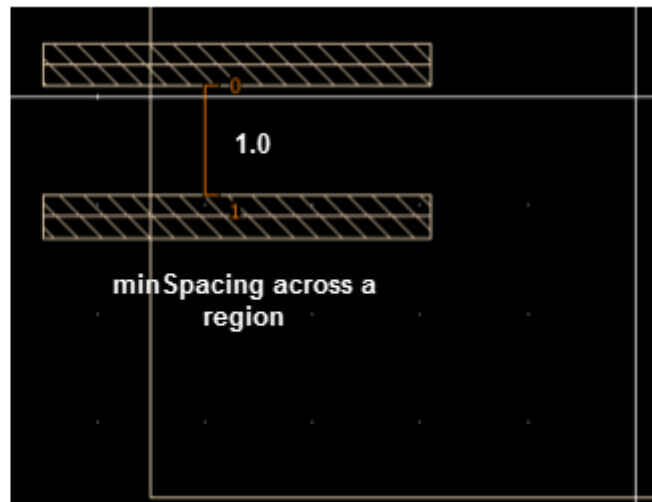
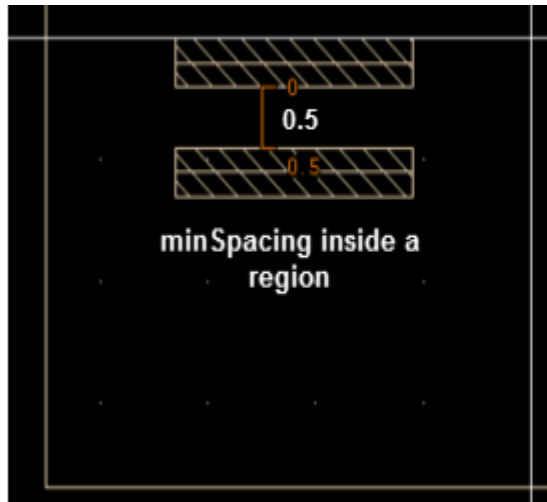
When the pathSeg is in a region, it follows the `minSpacing` rule of the region. The `minSpacing` value of a pathSeg that spans regions is the maximum `minSpacing` value across the spanned regions. For example:

```
spacings (
    ( minSpacing          "M1" 'insideLayers ("DMY") 0.05 )
    ( minSpacing          "M1" 'outsideLayers ("DMY") 0.1 )
) ;spacings
```


Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

The following figures show examples of a pathSeg inside a region and across regions. When the pathSeg spans regions, the maximum `minSpacing` value across the spanned regions is considered.



Note: If multiple `minSpacing` constraints are defined in the technology file of a design, during interactive routing, the constraints are merged and the resultant constraint is used to create a pathSeg.

minArea

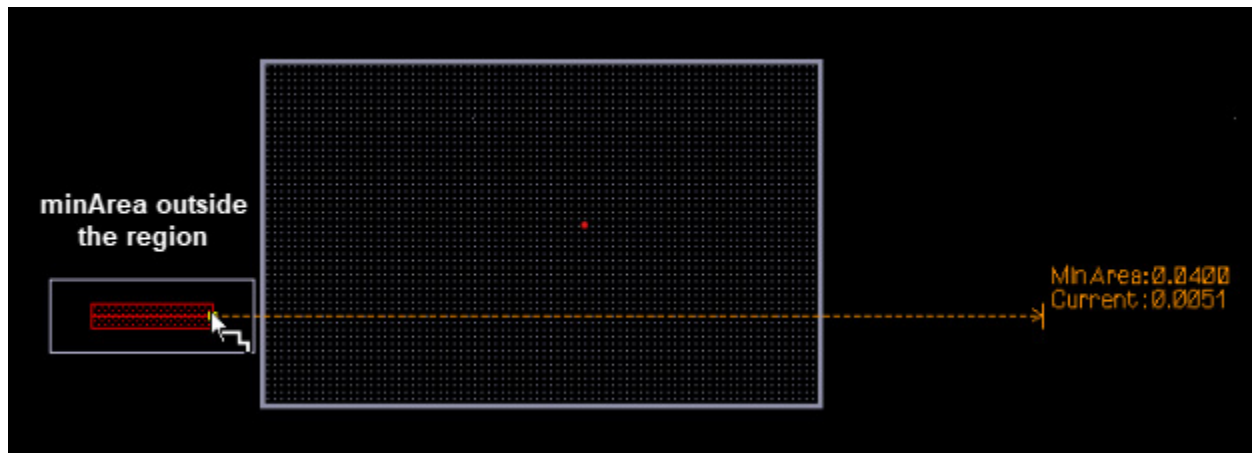
To support a `minArea` area-based rule, you must define two constraints in the technology file.

```
spacings (  
  ( minArea "Metal2" 0.04)  
  ( minArea "Metal2" 'insideLayers ("BuriedNWell") 0.06)  
) ;spacings
```

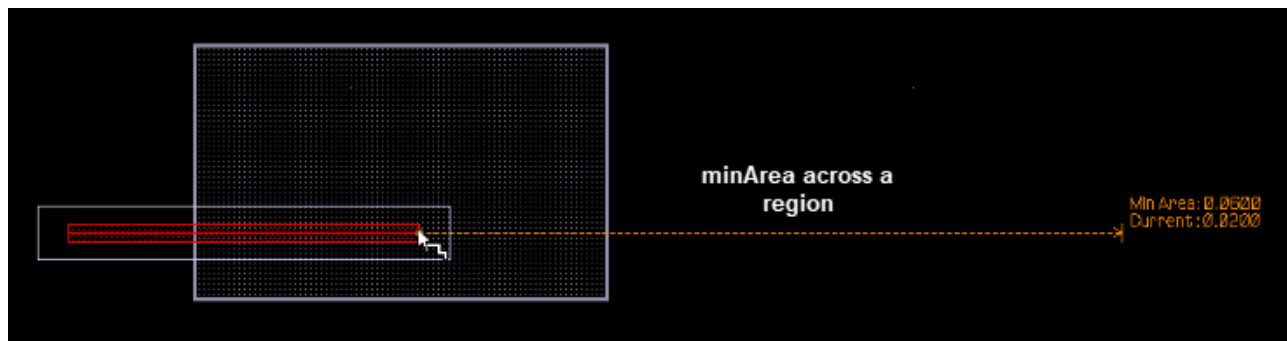
Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

When a pathSeg is outside a defined global region that does not overlap another region, the `minArea` constraint of the global region is supported, as shown in the following figure.



When a pathSeg is created inside a region, it follows the `minArea` rule of that region.



Note: When a pathSeg spans regions, the maximum `minArea` value across the spanned regions is considered.

The `minArea` area-based rule is also supported by *Create Bus*, *Create Stranded Wire*, and *Stretch* commands.

Important

For the Stretch command, it is necessary to have the following settings.
The *Keep Wires Connected to* option is set to *All wires and vias*.
The *Keep wires connected to shapes* option is selected.

Via Rules

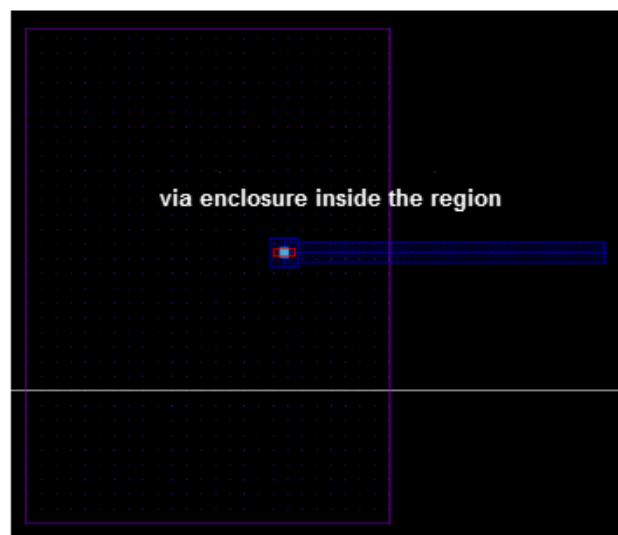
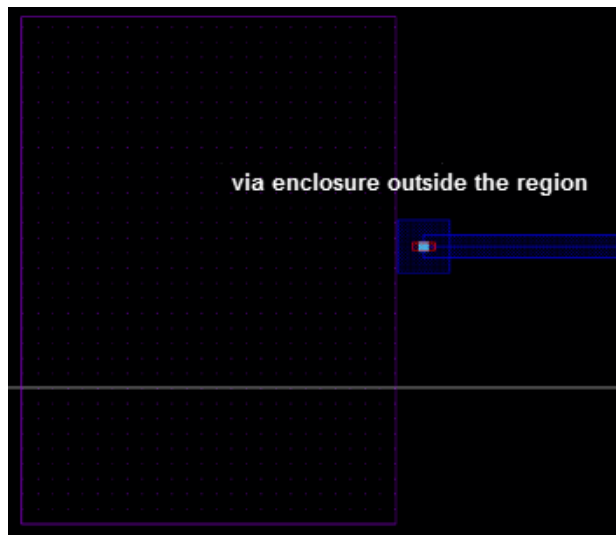
The following via extension area-based rules are supported for interactive routing.

Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

- `minExtensionDistance`
- `minOppExtension`
- `minQuadrupleExtension`
- `allowedCutClass`
- `viaKeepoutZone`

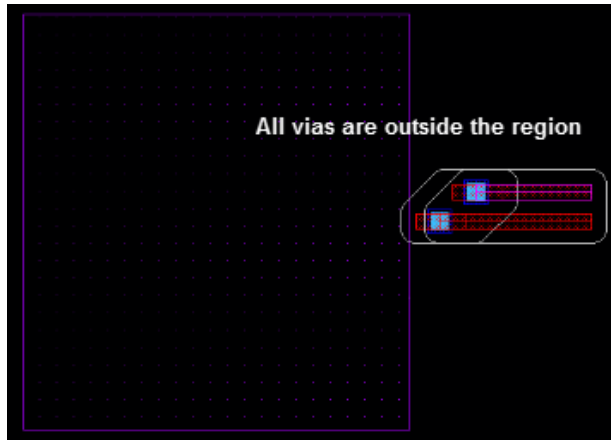
These via rules, when defined in a technology file, apply only to vias that are fully within a region. The enclosure of a via depends on area-based rules defined in the technology file. Also, it is not allowed to have a via partially inside and outside the region.



Virtuoso Interactive and Assisted Routing User Guide

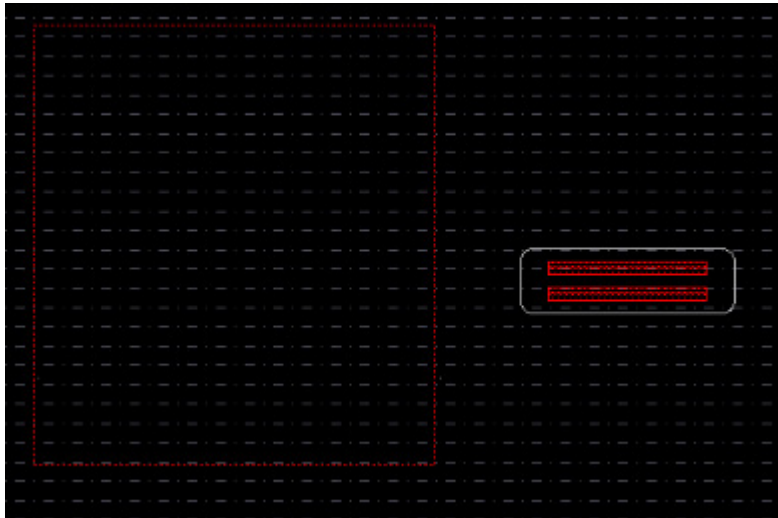
Setting up the Interactive Routing Environment

In addition, if there is more than one via, they must not be in different regions. The via extension is created according to the area-based rule of the region with which the via overlaps.

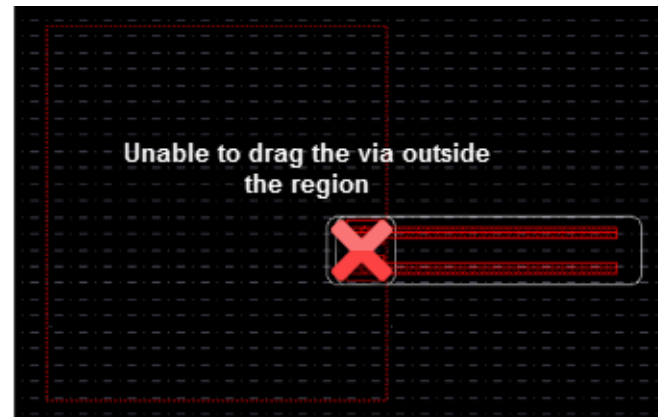


allowedCutClass

When the `allowedCutClass` rule is defined for the area inside a region, the area outside the region is considered invalid and vias cannot be added there. Conversely, when the `allowedCutClass` rule is defined for the area outside a region, the area inside the region is considered invalid and vias cannot be added there. The following figure shows an example when you are unable to add a via outside the region.

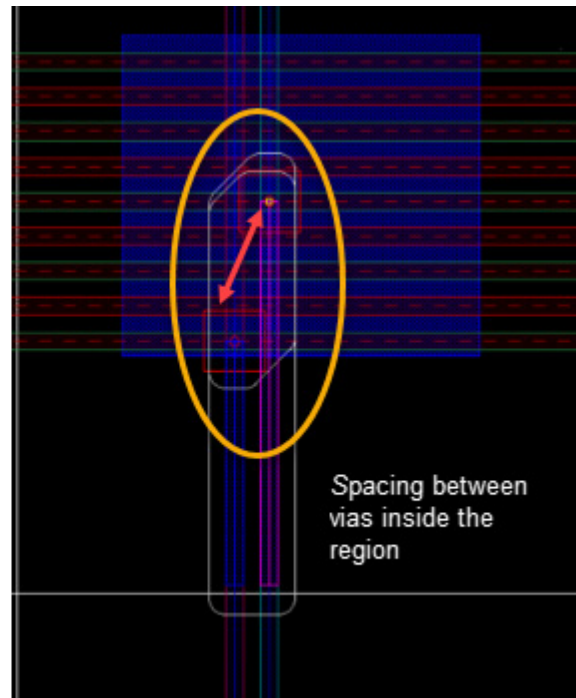
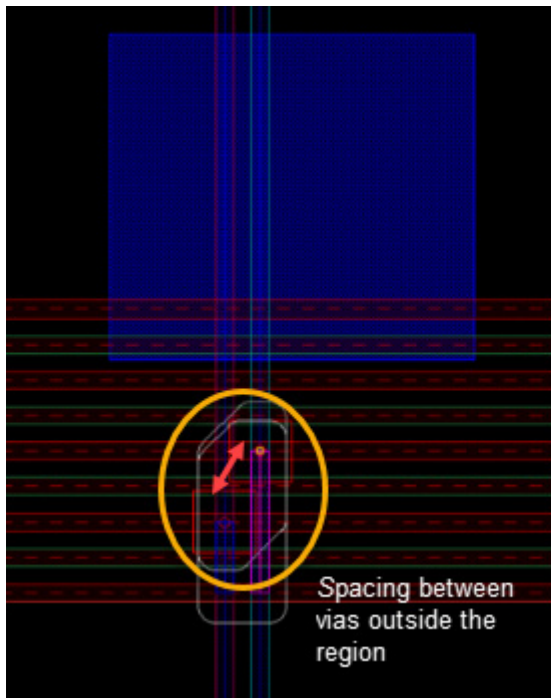


In addition, the following figure shows an example when a via is added inside the region and you are unable to move the via outside the region.



viaKeepoutZone

When the `viaKeepoutZone` rule is defined, the required spacing between vias is different in different regions. The following figure shows an example in the different spacing values when a via is outside and inside a region.



Using Mouse Button Controls

Clicking the right mouse button opens a context-sensitive menu at the point of click. The commands available on the context menu depend on the currently active command.

- When the *Create – Wiring – Wire* command is running, clicking the right mouse button opens the Create Stranded Wire Form. You can quickly select the required via, re-adjust the width for wire creation, or fine-tune the wire options.

For information about bindkey controls that appear on the right of various commands on the context menu, see Bindkeys and Access Keys.

- When the *Create – Wiring – Wire* command is running:
 - ❑ Pressing the arrow keys allows you to pan across the cellview in four directions.
 - ❑ Pressing the space bar key places a via or opens the Metal Density Options Form. If only one layer and one via are available for selection, a via is placed and the layer

is changed to the only available layer. If the current routing layer has more than one layers available for selection, the *Select Via* form opens so you can choose a layer or via type.

- When you are creating multiple wires, clicking the right button invokes the Edit Bus Form for bus routing-specific options.

Startup Warnings

Warning or error messages may be issued when invoking a command for the first time due to the loading of data into memory. These messages will appear only once and serve to indicate the issues that have been found with the current data, such as technology file and design constraint discrepancies.

Virtuoso Interactive and Assisted Routing User Guide

Setting up the Interactive Routing Environment

Technology Requirements

This section discusses the following:

- [Technology File Requirements](#)
- [Constraint Groups](#)
- [Supported Constraints and Parameters](#)
 - [Supported Constraints and Parameters \(Virtuoso Advanced Node for Layout Standard\)](#)
 - [Supported Constraints and Parameters \(Virtuoso Advanced Node for Layout Only\)](#)

Technology File Requirements

This section specifies minimum requirements for the technology file.

Specifying Layer Order

The `layerRules` section of the technology file specifies attributes for user-defined layers. You must specify the mask number for each interconnect layer and the preferred routing direction for each routing layer.

The mask number assigned to each layer in the technology file `functions` section is the layer sequence number, which helps determine if layers are adjacent to each other.

The mask number is used to:

- Order the layers in the *Select Via* form.
- Determine the next layer in sequence when using the *Via Up* or *Via Down* command.
- Determine automatically the “next” or “previous” layer when the *Create Wire* command is run with a non-routing layer as the current layer.

Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements

If mask numbers are not specified, layer-purpose priority is used. However, this is not as reliable as defining the mask numbers because mask numbers are independent of purposes.

Note: The router may not work if the mask numbers are not specified correctly.

Valid Values: cut, li, metal, ndiff, pdiff, nplus, pplus, nwell, pwell, poly, diff, recognition, other, unknown

Note that the following example is specific to space-based routing. The `layerRules` section in your techfile can have more layers and layer definitions.

```
layerRules (
    functions (
        ; ( layer function [maskNumber] )
        ; ( ----- )
        ( POLY      "poly"      0 )
        ( CO        "cut"       1 )
        ( Metall1   "metal"     2 )
        ( VIA       "cut"       3 )
        ( Metal2    "metal"     4 )
        ...
    )
)
```

Note:

- Include only one poly layer in the technology file because Virtuoso Space-based Router can route only one poly layer at a time.
- Define only one cut layer between each pair of metal layers.
- If the functions section is not defined or is defined without the mask numbers, vias are still available when the *Create – Wire* command is run. However, the following warning message is displayed.

```
\w *WARNING* geViaSet : Incomplete layer maskNumber, the order of the vias may not be correct.
```

For more information about functions, see [Technology File Layer Attributes](#) in *Virtuoso Technology Data ASCII Files Reference*.

Specifying Routing Layer Directions

Set a preferred routing direction for each routing layer.

```
layerRules (
    routingDirections (
        ; ( layer direction )
        ; ( ----- )
        ( POLY      "none" )
        ( Metall1   "horizontal" )
        ( Metal2    "vertical" )
    )
)
```

Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements

```
    ;routingDirections
) ;layerRules
```

For more information, see [Technology File Layer Attributes](#) in *Virtuoso Technology Data ASCII Files Reference*.

Specifying Via Definitions

At a minimum, define all standard vias for interconnect using routing layers in the `viaDefs` section of the technology file. Typically, this is a set of standard vias between the poly layer and the highest metal layer.

For example.

```
viaDefs(
  standardViaDefs(
    ;(
    viaDefName layer1 layer2 (cutLayer cutWidth cutHeight [resistancePerCut])
    ; (cutRows cutCol (cutSpace))
    ; (layer1Enc) (layer2Enc) (layer1Offset) (layer2Offset) (origOffset)
    ; [implant1 (implant1Enc) [implant2 (implant2Enc) [well/substrate]]])
    ;(
      ( mpoly Metall POLY ("CO" 0.09 0.09)
        (1 1 (0.11 0.11))
        (0.04 0.0) (0.04 0.01) (0.0 0.0) (0.0 0.0) (0.0 0.0)
      )
      ( m1m2 Metall Metal2 ("VIA" 0.1 0.1)
        (1 1 (0.1 0.1))
        (0.04 0.0) (0.04 0.0) (0.0 0.0) (0.0 0.0) (0.0 0.0)
      )
    )
  ...
)
```

By default, standard vias are automatically created before assisted routing. To disable the creation of standard vias and use only the existing vias in the `validRoutingVias` constraint, specify the following:

```
rdeSetVar "db.enable_create_derived_vias" "false"
```

In addition, define the custom vias required for routing.

```
customViaDefs(
  ;(
  viaDefName libName cellName viewName layer1 layer2 resistancePerCut)
  ;(-----)
  (VIAm1m2_2CUT_N tsmc65lp VIAm1m2 via Metall Metal2 0.95)
  (VIAm1m2_2CUT_S tsmc65lp VIAm1m2 via Metall Metal2 0.95)
  (VIAm1m2_2CUT_E tsmc65lp VIAm1m2 via Metall Metal2 0.95)
  (VIAm1m2_2CUT_W tsmc65lp VIAm1m2 via Metall Metal2 0.95)
  ...
)
```

Note: Via variants used by other tools are currently not recognized by Virtuoso.

For more information, see *Technology File Via Definitions and Via Specifications* in *Virtuoso Technology Data ASCII Files Reference*.

Specifying Valid Layers and Valid Vias

Define a list of valid layers and vias in the `virtuosoDefaultSetup` constraint group. Use the `foundry` constraint group to define only process rules. Do not use the `foundry` constraint group to define valid layers and valid vias.

```
;( group [override] )
;( ----- )
( "virtuosoDefaultSetup" nil
  interconnect(
    ( validLayers      (POLY      Metal1      Metal2 ...) )
    ( validVias        (mpoly     m1m2       ... ) )
  ) ;interconnect
) ;virtuosoDefaultSetup
```

Note: If no `validVias` statement exists, a full set of standard via is defined for the session based on the layers defined in the `validLayers` statement.

The automatic router uses vias that have both layers defined as valid routing layers. If a via has a metal layer that is not a valid routing layer, the router can still use the via for pin access provided it is in the valid via list. In particular, vias for which the bottom layer is oxide, well, or implant are not used during automatic routing. Virtuoso Space-based Router displays a warning message when initializing a design if the constraint group contains such vias.

Note: Pcell vias are not supported by Virtuoso Space-based Router.

For more information, see [validLayers](#) and [validVias](#) in the *Routing Constraints* chapter of the Virtuoso Technology Data Constraints Reference.

Limiting Routing on a layer

The current method for restricting the amount of wiring on any given layer is to apply a *taper* constraint group. The intent is to enable the router to connect to the pins on the restricted layer, but limit the amount of routing on the layer using a taper constraint group.

To restrict the amount of routing, define a taper constraint group similar to the one below. These examples illustrate how to restrict routing on the Poly layer, but this technique can be applied to any layer(s).

Note: The `validLayers` include only Poly and Metal1. This allows the router to connect to Poly and traverse to Metal1, but limits the amount of Poly routing to the `taperHalo` value. Routing on the restricted layer will occur only within the halo of the pins.

Example:

```
constraintGroups (
; ( group [override] )
; ( ----- )
( "virtuosoDefaultTaper" nil "taper"
  interconnect (
    ( validLayers (Poly Metall ) )
    ( validVias (M1_PO ) )
  ) ;interconnect
  spacings (
    ( taperHalo 1.4 )
  ) ;spacings
) ;virtuosoDefaultTaper
);constraintGroups
```

Note: All other applicable wiring constraint groups should *exclude* the layer(s) in the `validLayers` statement. Otherwise routing may occur beyond the taper value limit.

Example:

```
; ( group [override] )
; ( ----- )
( "virtuosoDefaultSetup" nil
  interconnect (
    ( validLayers (Metall Metal2 ...) )
    ( validVias (M1_Poly m1m2 ...) )
  ) ;interconnect
) ;virtuosoDefaultSetup
```

Via Usage Model

For detail routing, Virtuoso Space-based Router considers a limited number of vias and via variations for each layer pair. The standard via variations are chosen based on the constraints and the type of route (for example, taper).

The detail router chooses a limited number of custom vias and derived standard via variations based on the amount of routing resources required by each. The vias and the via variations chosen are those that require the fewest resources.

Specifying Track Patterns and Width Spacing Patterns

You can snap wires to track patterns and snap patterns. A track is a system reserved purpose and can be combined with any physical layer to create a layer-purpose pair. The tracks that make up the track pattern provide guidelines for laying interconnect routes.

For more information on how to create tracks and width spacing patterns, see [Creating Track Patterns](#) and [Using Width Spacing Patterns](#).

Supporting Colored Trim Metal Layers

Virtuoso Space-based router supports colored trim metal layers with the correct color assignment when creating trim shapes on a routing layer. A colored trim layer can have color information specified using the color attribute, which lets you add more than one color mask to a single trim layer.

For more information about how colored trim layers are defined in the technology file, see [functions](#).

Note: If only locked trim shapes cut the routing layer, then the created trim shape should be color locked.

Using Layer-Purpose Pairs

In the Virtuoso Studio design environment, you can define technology rules for layer-purpose pairs (LPP). However, OpenAccess rules are applicable only to layers, and not layer-purpose pairs. In the Virtuoso Studio design environment, any technology rule that is defined for a layer-purpose pair is stored as a private extension and is available only to Virtuoso applications.

Layers based on purposes can be selected from the Palette, provided the layer-purpose pair is valid and visible. However, when tapping an existing shape, the layer-purpose pair is not used directly. You can create a prioritized list of purposes that allow you to use a layer based on a purpose. See [Tapping Wires](#).

Poly Pins over an Implant Layer

If `minSpacing` is defined between poly and implant layers, then any poly pin shape that is fully overlapped by an implant layer is viewed as blocked and is not accessible to the automatic router and the Point-to-Point router, and results in routing failures.

Support For 45 nm Rules

In order to use the newer 45 nm technology rules, these rules must be coded in the technology file `techLayerProperties` section. The following example shows LEF57_AREA, LEF57_ENCLOSUREEDGE, and LEF57_SPACING for second EOL rule support.

```
techLayerProperties (
; ( propName                Layer1 [ Layer2 ]                PropValue )
; ( -----                ----- )
```

Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements

```
( LEF57_AREA    Metall1  "AREA 0.07 EXCEPTEDGELENGTH 0.41 EXCEPTMINSIZE 0.41 0.14 ;"
)
( LEF57_ENCLOSUREEDGE  Via2    "ENCLOSUREEDGE 0.08 WIDTH 0.26 PARALLEL 0.5 WITHIN
0.4 ;" )
( LEF57_SPACING  metall1    "SPACING 0.12 ENDOFLINE 0.10 WITHIN 0.035 MINLENGTH 0.07
PARALLELEDGE 0.12 WITHIN 0.10 ENCLOSECUT BELOW 0.05 CUTSPACING 0.15 ;")

) ;techLayerProperties
```

Supported Constraints with Supported Parameters

In interactive routing, constraints are considered only if the constraints are supported and do not have any unsupported parameters in the technology file. For Example, `minViaSpacing` is a supported constraint and therefore, the following statement is supported.

```
'(minViaSpacing "Vial" 0.5)'
```

However, the following statement is ignored because `orthogonalEnclosingMetal` is an unsupported parameter in the `minViaSpacing` constraint.

```
'(minViaSpacing "Vial" 'orthogonalEnclosingMetal 0.5)'
```

Constraint Groups

A constraint group contains a set of constraints or rules to be applied to a design under different circumstances, which allows the flexibility to experiment with less or more stringent process rules at different stages of the design process. Constraint groups are classified as follows.

1. Foundry constraint group

Example: `foundry`

2. Application-specific constraint group(s)

Examples: `virtuosoDefaultExtractorSetup`, `virtuosoDefaultSetup`,
`LEFDefaultRouteSpec`

3. User-defined constraint group(s)

Examples are unlimited: `shieldTheseNets`, `wideWires`

Foundry, Application-specific and User-defined constraint groups may be created automatically during a translation process such as `lef2oa`, and are based on the information in the LEF/DEF file. 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.

Constraint groups are stored in the technology or design database and can be created by your CAD group or by the layout designer. Constraint groups can also be created automatically during the translation process, such as `lef2oa`.

In Virtuoso XL and higher tiers, you typically have the following three constraint groups as a required minimum: `foundry`, `virtuosoDefaultExtractorSetup`, and `virtuosoDefaultSetup`.

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.

Define the complete set of process rules in the `foundry` constraint group. Do not depend on a technology file translated from elsewhere to contain all needed rules. For example, a 65nm tech file translated from CDB will not include all required rules because not all 65nm rules are code able in CDB.

Creation and usage

■ Creating the Foundry Constraint Group

The `foundry` constraint group is created and populated with information during the `lef2oa` translation. The translator get the information from section within the LEF file.

Alternatively, you can create the `foundry` constraint group by loading an ASCII technology file containing the foundry constraint group information through the Technology File Manager Toolbox, which you can invoke from the CIW.

■ Information in the foundry constraint group

The `foundry` constraint group contains all of 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.

Most all functionality uses the `foundry` constraint group at some point. As the last stop in constraint precedence lookup, the `foundry` constraint group is the final place and many times the only place the value(s) for a constraint are defined.

The `foundry` constraint group is not typically writable for most users, but can be viewed in the Process Rule Editor. 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. Within this dialog you can set the scope to Technology and view the foundry constraint group.

Application Constraint Groups

An application constraint group is used to target a basic set of rules within a given application. Although you can define numerous application specific constraint groups and switch between them at any time, you can reference *only* one as the default by setting the *Wire* cyclic field in the *Layout Editor Options* form (*Options – Editor...*) at any given time. It is important to note, however, that at any given time during routing or wire editing, the rules to apply to specific objects may be derived from a number of different constraint groups, based on specific constraint group lookup precedence.

Application-specific constraint groups can be stored within the technology database or directly on the design itself (i.e. `lib/cell/cellview`).

Examples: `virtuosoDefaultExtractorSetup`, `virtuosoDefaultSetup`, `LEFDefaultRouteSpec`

Specifying the Default Constraint Group for Routing

You can define numerous application-specific constraint groups and switch between them, but you can reference only one constraint group at any given time.

Note: You must set the required constraint group at the beginning of a session and use the Process Rule Editor to vary the constraints applied on different nets while working on the design. If you update the constraint group while working on a design, delete all routing that would be affected by the changes made to the constraint group and re-route.

`virtuosoDefaultExtractorSetup`

The `virtuosoDefaultExtractorSetup` constraint group is used by default by the extractor for connectivity extraction. This constraint group must contain a list of valid interconnect layers and can optionally include a list of valid vias. The layers can be physical, such as metal and poly, or derived, such as those used to define stop layers and bulk layers. In the technology file, physical layers are defined in the `layerRules` section and derived layer are defined in the `techDerivedLayers` section.

Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements

The `setupConstraintGroup` environment variable in your `.cdsenv` file is used to specify `virtuosoDefaultExtractorSetup` as the default constraint group for connectivity extraction. If required, you can change this default value by specifying the name of another constraint group. If the `setupConstraintGroup` environment variable is not specified or is set to an empty string, the extractor prompts you for a constraint group to use for extraction.

The `virtuosoDefaultExtractorSetup` constraint group can be stored in the technology database or in the design database along with the library/cell/view information.

Note: Though it is possible to specify `virtuosoDefaultExtractorSetup` as the default constraint group for routing, it is not recommended for use for routing. This is because the constraint group is designated for connectivity extraction and the layers and vias that it defines as valid may not be suitable for routing.

virtuosoDefaultSetup

The `virtuosoDefaultSetup` constraint group is by default used for interactive and automatic routing. For example, if a signal selected for routing does not have a constraint group assigned to it, the constraints in the `virtuosoDefaultSetup` constraint group apply. Moreover, the set of valid layers and vias defined in the `virtuosoDefaultSetup` constraint group is used exclusively for wiring and is usually a subset of that defined in the `virtuosoDefaultExtractorSetup` constraint group. You can also specify in this constraint group any process rules that should override those defined in the foundry constraint group.

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 `wireConstraintGroup` environment variable in your `.cdsenv` file is used to specify `virtuosoDefaultSetup` as the default constraint group for interactive and automatic routing. If the `wireConstraintGroup` environment variable is not set, interactive and automatic routing use the value of the `setupConstraintGroup` environment variable as the default constraint group. Because the constraint group specified with `setupConstraintGroup` is for extraction, you do not want it to be used for routing. Therefore, ensure that you set a default constraint group for routing.

You can also specify `virtuosoDefaultSetup` as the default constraint group for interactive and automatic routing by using the *Default Wire Constraint Group* list in the Layout Editor Options form.

LEFDefaultRouteSpec

The `LEFDefaultRouteSpec` constraint group contains rules typically used in digital standard cell applications or custom and mixed signal applications that require only metal layers.

The `LEFDefaultRouteSpec` constraint group is specific to a LEF/DEF design flow and is created during `lef2oa` translation. As a result, the valid layers, the valid via definitions, and the valid rules become part of the `LEFDefaultRouteSpec` constraint group.

Note: The `lef2oa` translator translates any `NONDEFAULTRULES` in LEF as constraint groups in the OpenAccess technology database.

The `LEFDefaultRouteSpec` constraint group contains valid layer and via definitions. The via definitions typically include standard vias and several custom vias used by other routers such as NanoRoute in the Innovus environment. The `LEFDefaultRouteSpec` constraint group also contains process rules for wide metal shapes.

You can specify `LEFDefaultRouteSpec` as the default constraint group for interactive or automatic routing by selecting it from the *Default Wire Constraint Group* list in the *Wire Editing* section of the *Layout Editor Options* form.

virtuosoDefaultTaper

If routing requires tapering, for best results define a `virtuosoDefaultTaper` constraint group with type `taper`. Explicitly defining a taper constraint group ensures that the correct layers and widths are used for tapered wires.

The primary purpose of a constraint group of this type is to taper the width of a wire to match the width of the pin to which it connects. You can also adjust the widths of taper routing. Many different constraints can be included in the taper constraint group.

You can also use a constraint group of type `taper` to restrict routing on a layer to control the taper length. For more information, see [Limiting routing on a layer](#).

Note: After you customize a constraint group of type `taper`, such as to restrict routing on poly, you cannot use the constraint group to also taper the widths of wires on other layers.

The `wireTaperConstraintGroup` environment variable in your `.cdsenv` file is used to specify `virtuosoDefaultTaper` as the default constraint group for tapering wires. The `virtuosoDefaultTaper` constraint group is only used when the router cannot connect to a target pin by using the regular net, design, or default constraints.

How the Router Uses the Taper Constraint Group

When you invoke the design, there are three constraint groups the application searches for by default. These constraint groups, whether they are defined in your technology library or not, are specified in the default `.cdsenv` with the following environment variables.

- `layout setupConstraintGroup string "virtuosoDefaultExtractorSetup"`
- `layout wireConstraintGroup string "virtuosoDefaultSetup"`
- `layout wireTaperConstraintGroup string "virtuosoDefaultTaper"`

It is important to note, however, that at any given time during routing or wire editing, the rules to apply to specific objects may be derived from a number of different constraint groups, based on specific constraint group lookup precedence.

First, the router attempts to route a net using constraints from the net constraint group, design constraint group, and application default constraint group. The precedence order is

- a. net constraint group
- b. application default constraint group
- c. design constraint group
- d. foundry constraint group

If the router cannot connect to a pin for some reason, it attempts to connect using constraints from the input/output taper constraint group, application default taper constraint group. The precedence order is

- a. input/output taper constraint groups
- b. application taper constraint group

Defining the Taper Constraint Group

When routing requires tapering, for best results define a `virtuosoDefaultTaper` constraint group. Explicitly defining a taper constraint group ensures that the correct layers and widths are being used for the tapered wires. You can control the global taper routing mode using the environment variable `setTaperMode`.

Also, in order to load the taper constraint, the constraint must be specifically marked with `taper` tag. The following is an example of a taper constraint group.

```
constraintGroups (
;( group [override] )
;( ----- )
```

Virtuoso Interactive and Assisted Routing User Guide

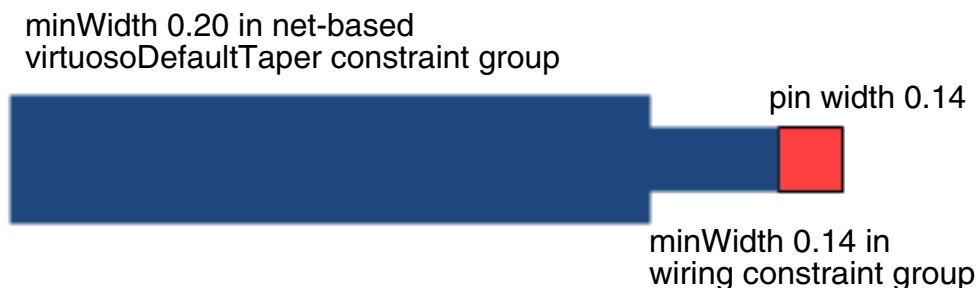
Technology Requirements

```
( "virtuosoDefaultTaper" nil "taper"
interconnect(
( validLayers (Poly Metall ) )
( validVias (M1_PO ) )
) ;interconnect
spacings(
( taperHalo 1.4 )
) ;spacings
) ;virtuosoDefaultTaper
;constraintGroups
```

You can then use the Technology File Manager to load the constraint group.

Note: If you include all the layers in the `validLayers` constraint of the taper constraint group, but some of your nets require a more restricted layer set, use the input and output taper constraint groups to reduce the `validLayers` list for those specific nets. You can use the Process Rule Editor to assign the `validLayers` constraint to the input/output taper constraint groups.

Note: If you have a taper constraint group attached to the net and the `minWidth` defined in that constraint group is greater than the width of the pins, the router may taper down using the `minWidth` defined in the wire constraint group when connecting to those pins. This optimizes routing for pin connections. The router does not consider the behavior as a violation of the net based constraint group. See the illustration below.



User-Defined Constraint Groups

Non-default constraint groups defined in the technology LEF will be translated as user-defined constraint groups and stored in the technology database.

A common example of a user-defined constraint group would be for wide wire rules for a particular set of signals. User-defined constraint groups can be used whenever particular routing considerations need to be applied to signal(s).

Note: If you attach a constraint group to a net and the `minWidth` defined in that constraint group is greater than the width of the pins, the router may taper down using the `minWidth` defined in the wire constraint group when connecting to those pins. This optimizes routing for

pin connections. The router does not consider the behavior as a violation of the net based constraint group.

customRoutingConstraintGroup Use Model

The `customRouting` limits the use of constraints that can be used. This limitation is based on the override limitation. The Virtuoso Space-based router can override only the following constraints: `minWidth`, `minSpacing (One Layer)`, `minNumCuts`, `validLayers`, `validVias`, `minSpacing (Two Layers)`, and `minOppExtension`.

Note: Virtuoso Space-based router does not recognize derived layers. For more information, see [Derived Layer Support](#) in the Virtuoso Design Rule Driven Editing User Guide.

Storing Constraint Groups

The foundry constraint group is stored in the OpenAccess technology database. Application-specific and user-defined constraint groups are also stored in the OpenAccess technology database. Constraint groups stored in the technology database apply to all designs associated with the design specific constraints.

Design-specific constraints that apply to specific objects in a design, instead of the entire design, are stored in the OpenAccess design database.

You can also store any user-defined constraint groups within the technology database, provided you have write permission to the technology database. This has the advantage of being a centrally accessible location and provides added control over the definitions in the constraint groups.

Note: Constraint groups stored in a technology database apply to any design derived from it.

Constraint Group Lookup Precedence

In Virtuoso, the `foundry` constraint group provides process rules for all objects. However, you can override the rules in the `foundry` constraint group for specific objects or for the design.

A technology database can contain multiple constraint groups that define the same constraints. When this is the case, the constraint groups are applied according to the following precedence:

1. The constraint group, being referenced by a command that is currently active in the design window.

Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements

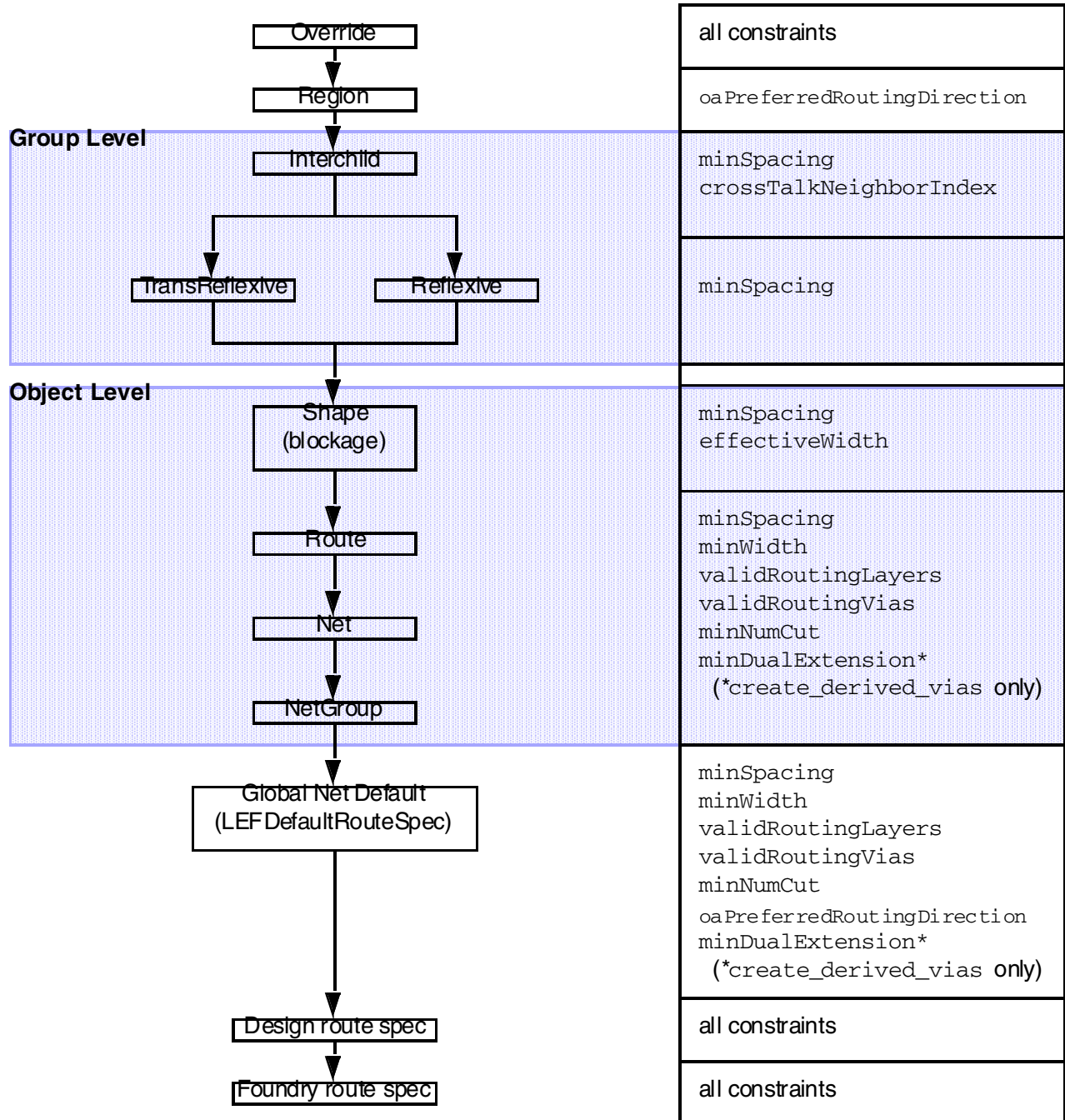
2. A setup or a constraint group recognized as the constraint group of first precedence by Virtuoso.
3. The `foundry` constraint group.

Virtuoso searches for the constraint groups it recognizes in the order of precedence. It applies first the hard constraint it finds. If a hard constraint is not found, it applies the first soft constraint that it finds.

Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements

The following diagram shows the simplified constraint search hierarchy.



Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements

The following table shows constraint group lookup precedence in the Virtuoso Space-based Router. Precedence starts with constraints attached to objects and ends with constraints in the Foundry constraint group.

Entities	Examples
Net	
Net group	reflexive, transreflexive, default
Application-specific constraint group	virtuosoDefaultSetup constraint group
Design	constraints specific to this design
Foundry	process technology rules

For more information, see [*Getting Started with Virtuoso Unified Custom Constraints*](#) and [*Virtuoso Technology Data Constraints Reference*](#).

Important

When routing a design, it is important to maintain the constraint group definition used for specific nets throughout the routing process. Changing constraint groups on a net can result in unpredictable layers and constraints being applied during routing.

Scalar Versus Table Spacing Rule Precedence

When both scalar and table spacing rules exist in the technology database, the rules are applied as follows.

- When both are present in a single constraint group, the first spacing rule found is the one applied. The second one is ignored. This precedence holds true regardless of whether the value is a scalar spacing rule or a table spacing rule.
- When each exists alone in its own constraint group, the scalar value is used as a default regardless of the precedence order. The table values are also used, but only for the larger widths, not the default.

Supported Constraints and Parameters

Interactive and Assisted routing commands use technology library constraints and constraint groups to edit the design and to know about the potential rules violations during interactive editing. The following table shows the constraints supported by the Interactive and assisted routing commands.

ConstraintName	Interactive Routing Commands	Assisted Routing Commands
<u>allowedWidthRanges</u>	yes	yes
<u>cutClasses</u>	yes	yes
<u>horizontalOffset</u>	yes	yes
<u>horizontalPitch</u>	yes	yes
<u>largeRectViaArrayAllowed</u>	no	yes
<u>maxDiffDensity</u>	no	yes
<u>maxFloatingArea</u>	no	no
<u>maxLength</u>	yes	no
<u>maxNumMinEdges</u>	no	yes
<u>maxWidth</u>	yes	no
<u>minArea</u>	yes	yes
<u>minAreaEdgeLength</u>	no	yes
<u>minCutClassSpacing(Two layers)</u>	Yes	yes
<u>minDiagonalSpacing</u>	yes	no
<u>minDiagonalWidth</u>	yes	no
<u>minEdgeAdjacentDistance</u>	no	yes
<u>minEdgeAdjacentLength</u>	yes	yes
<u>minEndOfLinePerpSpacing</u>	yes	yes
<u>minEndOfLineSpacing</u>	no	yes
<u>minEndOfLineExtensionSpacing</u>	yes	yes
<u>minExtensionEdge</u>	no	yes
<u>minFillToFillSpacing</u>	yes	yes

Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements

ConstraintName	Interactive Routing Commands	Assisted Routing Commands
<u>minHoleArea</u>	no	yes
<u>minLargeViaArrayCutSpacing</u>	no	yes
<u>minLargeViaArraySpacing</u>	yes	yes
<u>minLargeViaArrayWidth</u>	yes	yes
<u>minLength</u>	no	yes
<u>minNeighborViaSpacing</u>	yes	no
<u>minNumCut</u>	no	yes
<u>minOppExtension</u>	yes	yes
<u>minOppSpanSpacing</u>	yes	yes
<u>minOutsideCornerEdgeLength</u>	no	yes
<u>minParallelSpanSpacing</u>	no	yes
<u>minParallelViaSpacing</u>	no	yes
<u>minParallelWithinViaSpacing</u>	yes	yes
<u>minProtrusionNumCut</u>	no	yes
<u>minRectArea</u>	no	yes
<u>minSameMetalSharedEdgeViaSpacing</u>	yes	yes
<u>minSameNetSpacing</u>	no	no
<u>minSpacing</u>	yes	yes
<u>minWidth</u>	yes	yes
<u>minViaExtension</u>	yes	yes
<u>minViaSpacing</u>	no	yes
<u>minVoltageSpacing(two layer)</u>	yes	yes
<u>redundantViaSetback</u>	yes	yes
<u>stackable</u>	no	yes
<u>taperHalo</u>	no	yes
<u>verticalOffset</u>	no	yes
<u>verticalPitch</u>	yes	yes

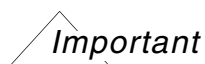
Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements

ConstraintName	Interactive Routing Commands	Assisted Routing Commands
<u>viaSpacing</u>	yes	yes
<u>viaStackingLimits</u>	yes	yes
<u>allowedLengthRanges</u>	yes	yes
<u>minSideExtension</u>	yes	no
<u>minExtensionOnLongSide</u>	yes	yes
<u>minWireOverlap</u>	yes	yes
<u>rectShapeDir</u>	yes	yes
<u>forbiddenCutClassSpacingRange</u>	yes	yes
<u>minQuadrupleExtension</u>	yes	no
<u>minEndOfLineEdgeExtension</u>	yes	yes

Constraint Manager Constraints	Interactive Routing Commands	Assisted Routing Commands
<u>shielding</u>	yes	yes
net priority	no	yes
process rule overrides	yes	yes

Supported Constraints and Parameters (Virtuoso Advanced Node for Layout Standard)



Requires Virtuoso Advanced Node Option for Layout Standard (95512) license.

Constraint Name	Interactive Routing Commands	Assisted Routing Commands
<u>allowedLengthRanges</u>	yes	no
Parameters: viaLayers, exceptViaLayers, cutClassSizes		
<u>minSideExtension</u>	yes	yes
<u>minExtensionOnLongSide</u>	yes	yes
<u>minWireOverlap</u>	yes	yes
<u>minDirectionalOverlap</u>	yes	yes

Note: The constraints listed in the table above and their corresponding ASCII techfile syntax are fully supported by OpenAccess. However, only the *Create Wire* command in Wire Editor is fully compliant with these Local Interconnect (Li) constraints.

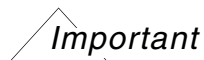
The constraints listed in the table are also supported by Stretch and Reshape commands.

Supported Constraints and Parameters (Virtuoso Advanced Node for Layout Only)

The Interactive and assisted routing commands have been enhanced to provide support for new rules. The commands have been enhanced to support fully interactive and automatic multi-patterned routing to avoid complex coloring conflicts. This simplifies layout creation and minimizes coloring errors that can occur when designing at 10nm. In addition, Virtuoso Space-based Router has added support for pre- and post-optimization of routes to account for the complex EM rules at 10nm and below.

Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements



Requires Virtuoso Advanced Node for Layout (95511) license.

Constraint Name	Interactive Routing Commands	Assisted Routing Commands
<u>allowedLengthRanges</u> Parameters: exceptEndToEndGapRange	yes	no
<u>allowedSpacingRanges</u> Parameters: exceptWidthEdge	yes, Create Bus and Create Stranded Wire	yes
<u>allowedWidthRanges</u> Parameters: measureHorizontal, measureVertical	yes	yes
<u>allowedWidthSpacingPatterns</u>	yes	no
<u>allowedWireTypes</u>	yes	no
<u>minViaSpacing</u> Parameters: centerToCenter, exactLength, manhattan, sameNet, sameMetal, exceptExactAligned, horizontal, vertical The parallelRunLengthRange, viaConnectivityType, exceptViaConnectivityType parameters are supported by the Create Bus and the Create Stranded Wire commands.	yes	yes
<u>minSpacing</u> Parameter: any	yes	yes
<u>orthogonalSnappingLayer</u> Note: This constraint is supported for the Stretch command as well.	yes	yes
<u>sameMaskOnLayer</u>	yes	no

Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements

Constraint Name	Interactive Routing Commands	Assisted Routing Commands
<u>minEndOfLineSpacing</u> Parameters: width, distance, maxLength minLength twoSides, sameMask diffMask, endToEndSpace otherEndWidth extension. Note: Skip otherEndWidth parameter when endToEndSpace does not apply.	Supported only by Create Bus	yes
<u>minSpanLengthSpacing</u> Parameters: horizontal vertical any, eolWidth, minSpanSpacingRanges, minSpanForExceptEolWidth, sameMask.	Supported only by Create Bus	no
<u>endOfLineKeepout</u> Parameters: width, cornerOnly, exceptFromFrontEdge, exceptFromBackEdge, widthRanges, mask1 mask2 mask3, diffMask.	Supported only by Create Bus	no
<u>minParallelSpanSpacing</u>	Supported only by Create Bus	yes
<u>minEndOfLineExtensionSpacing</u>	Supported only by Create Bus	yes
<u>orthogonalWSPGrid</u> Parameters: 'mask1 'mask2 'mask3, 'upperLineEndWSSPDefName Note: The orthogonalWSPGrid constraint is used in the interoperability flow with Innovus. It is used to initiate the trim creation at specific locations.	yes	Supported by the Stretch command.
<u>viaSpacing</u> Parameters: edgeExtension, layer, extensionDirection	Supported only by Create Bus	no

Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements

Constraint Name	Interactive Routing Commands	Assisted Routing Commands
<u>minSideSpacing</u> Parameters: prl, sameMask, widthRanges, lengthRanges, horizontal vertical, exceptSpacingRanges, cutClass, otherLayer, cornerEuclidian, orthogonalPair, 'exactAlignedSpacingRange, insideLayers outsideLayers. Note: The horizontal vertical parameter specifies the direction in which spacing is measured. If the direction is not specified, spacing is measured in any direction.	Supported by <i>Create Bus and Create Stranded Wire</i>	no
<u>minCutEdgeSpacing</u>	Supported by <i>Create Bus</i>	no
<u>minSpacing(Two layers)</u>	Supported by <i>Create Bus</i>	no
<u>minVoltageExtension</u>	yes	no
<u>minVoltageSpacing (One layer)</u>	Supported by <i>Create Bus</i>	no
<u>minCornerSpacing</u>	Supported by <i>Create Bus</i>	no
<u>minHoleArea</u>	Supported by <i>Create Stranded Wire</i>	yes

Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements

Constraint Name	Interactive Routing Commands	Assisted Routing Commands
<u>minCornertoCornerDistance</u> Parameters: extensionLength, oppositeDirection Note: Create Wire automatically fixes the DRD error. If it is not able to fix the violation, the following warning message appears. A minCornerToCornerDistance violation occurs while creating the u-shaped pathSeg. This is because the distance between the two corners is not large enough. To avoid such a violation, ensure that the middle pathSeg of the u-shaped pathSeg is longer.	Supported by <i>Create Wire</i>	no
<u>minCutRoutingSpacing</u> Parameters: cutClass, exceptSameNet, enclosingLayer, anyOppositeExtension, insideLayers, insidePurposes, outsideLayers, outsidePurposes, overlappingLayer	Supported by <i>Create Stranded Wire</i> and <i>Create Bus</i>	no
<u>mergedViaCornerToCornerSpacing</u>	Supported by <i>Create Bus</i>	no
<u>rectShapeDir</u> Parameters: exceptWidth {widthHorizontal widthVertical}	yes	no
<u>minViaSpacing</u> Parameters: fixedCutClassList, otherFixedCutClassList, horizontalSpacingRange, verticalSpacingRange, farEnds	yes	no
<u>minArea</u>	Supported by <i>Create Wire</i>	no

Virtuoso Interactive and Assisted Routing User Guide

Technology Requirements

Constraint Name	Interactive Routing Commands	Assisted Routing Commands
<u>minOppExtension</u>	yes	no
Parameter: 'widthDirection		
<u>allowedSpanLengthRanges</u>	yes	no
<u>minLength</u>	yes	no
'width		
<u>minNeighborExtension</u>	Supported by <i>Create Bus</i>	no
<u>allowedJogWidths</u>	yes	no
This constraint is enforced if and only if the layout snapWireGrid environment variable is set to Snap Pattern.		

Working with Wires

This chapter describes how to set options for the Virtuoso® wire editing capability and create wire interconnect.

- [Introduction to Wires](#)
- [Creating Wires](#)
- [Editing Wires](#)
- [Splitting Wires and Vias](#)
- [Support for Local Interconnect \(Li/MOL\) Layers](#)
- [Smart Snapping in Interactive and Assisted Routing Commands](#)
- [Support for Mosaics in Create Wire](#)
- [Support for cdsGenVia for Different Purposes](#)
- [Setting Dynamic Measurement for the Area Calculation](#)
- [Support for Pixel Based Gravity](#)
- [Displaying Wire Summary](#)

Introduction to Wires

This section covers the following concepts.

- [Types of Wires](#)
- [Wire Elements](#)
- [Wire Objects](#)
- [Wire Parameters](#)

Types of Wires

You can create a wire using *Paths* or *Path Segments* (pathSegs). The different types of wires you can create include:

- Geometric Wires
- Symbolic Wires

For information about how to create these wires, see [Creating Wires](#). For information about how to distinguish between the two types of wires, see [Differentiating Between Geometric and Symbolic Wires](#).

Geometric Wires

Geometric wires are created using paths, pathSegs, and vias that are not contained within a route. Paths cannot be placed in routes, therefore, paths are always geometric data. Geometric wires can be created in Virtuoso Layout Suite. Geometric wires are usually used for special routing (power, ground, clock, and so on), which is not the default for creating wires in Virtuoso. Geometric wires can be used to create custom interconnect that auto signal routers do not modify when re-routing, ripping-up, and pushing wires. The autorouter does not create, delete, or re-route geometric data but it may connect to geometric data. You can convert Geometric Wires to Symbolic Wires by using *Edit – Wiring – Decompose Trunks*.

Symbolic Wires

Symbolic wires are created using pathSegs and vias that are contained within routes. Routes are internal containers used by the system to group pathSegs and vias. They are usually attached to nets and can have default constraint groups assigned. In Virtuoso, symbolic wires can be created by using the *Create – Wiring – Wire* or the *Create – Wiring – Bus* command in all tiers of Layout Suite. Symbolic wires are typically used for signal routing, which is the default for creating wires in Virtuoso. The Virtuoso Space-based router only creates well-formed symbolic wires with matching end points.

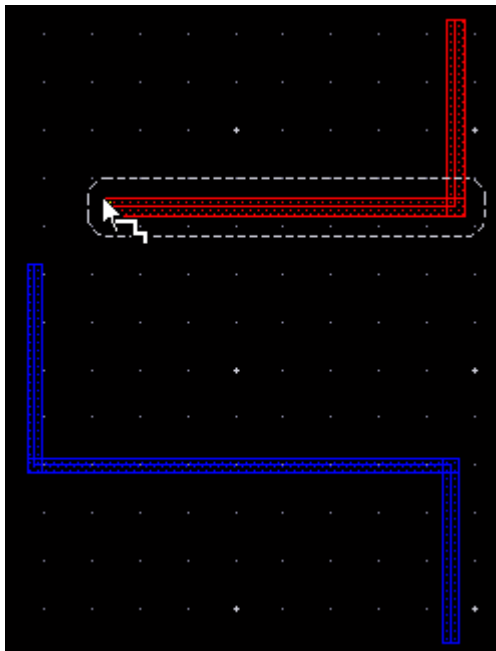
Symbolic data can be modified by pushing wires by using DRD and automatic routing if the routing status is different than `Locked` or `Fixed`. The *Point to Point* and automatic router create symbolic wires. Symbolic wires correspond to the DEF NETS statement definition.

The advanced features of the wire editor works only with symbolic wires.

Differentiating Between Geometric and Symbolic Wires

In XL, if you enable the *Notify* and *Enforce* modes in the DRD Options form (*Options – DRD Edit*), any geometric wires and shapes and fixed/locked wires (even if symbolic) are dimmed if you start the *Create – Wiring – Wire*, *Create – Wiring – Bus*, or the *Edit – Stretch* command.

This is illustrated in the figure below. In the DRD Options form, the *Enforce* and *Notify* DRD modes are selected. The blue wire is a geometric wire created on `net_a`. If you try to create a symbolic wire on `net_b`, represented by the red wire in the figure below, the blue geometric wire is dimmed.

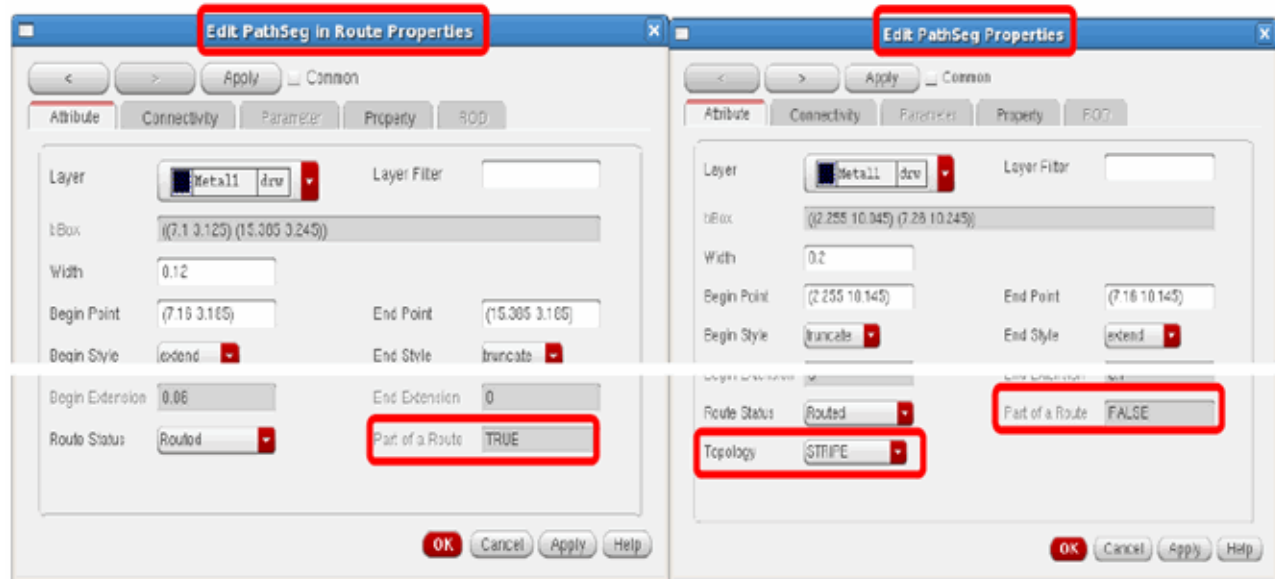


DRD Enforce mode works on pre-routed paths and segments.

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

The fastest and easiest way to identify if a wire is symbolic or geometric is using the Property Editor.



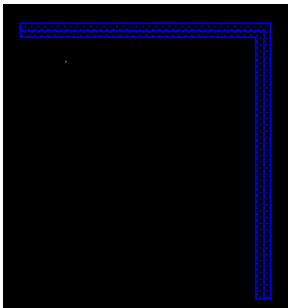
Wire Elements

Wire elements include paths, pathSegs, and vias. The description of each element as it is used when creating wires is as follows:

■ Paths (*Create – Shape – Path*)

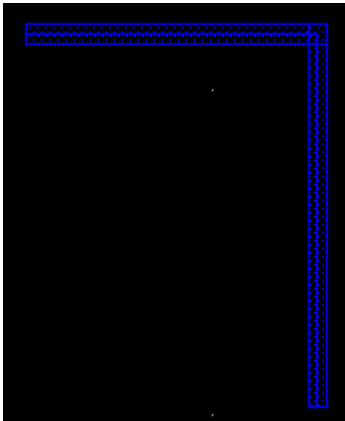
Using paths for routing creates geometric data. Paths support anyAngle mode. When used to create non-orthogonal routing, paths can lead to off-grid vertices depending on the width and angle being used. Paths are not supported by Innovus (Cadence digital place and route platform) as a route element, they are considered as a special nets. You must use pathSegs instead of paths when saving data to Innovus.

digital.



■ PathSegs (*Create – Wiring – Wire*)

When pathSegs are created in routes, the data is symbolic. When pathSegs are not created in routes, the data is geometric. PathSegs can be used to create on-grid 45 degree wires. The width of a diagonal pathSeg is manipulated to force vertices of the perimeter on the manufacturing grid. PathSegs cannot be used for any-angle routing and are limited to orthogonal and diagonal routing. A pathSeg has a start point and an end point.



■ Vias

Two types of vias are supported: standard vias and custom vias and via variants.

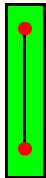
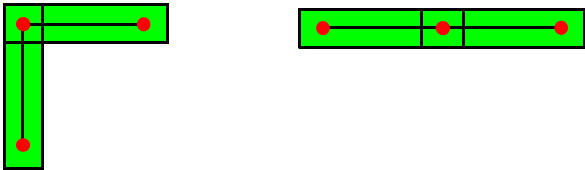
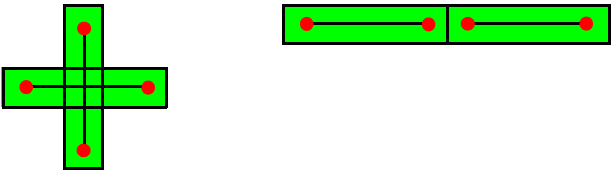
The `standardViaDefs` has a unique name and is associated with two layers and a list of via parameters with default values. It is a predefined device, generated based upon the value of its parameters. When creating wide wires, standard vias support via arrays. Rows and columns of vias are placed based on the area of metal overlap. Standard vias also allow you to edit the cut pattern of via arrays.

The `customViaDefs` has a unique name and is associated with a master cellview and two layers. A `customViaDefs` is a special purpose via. The `lib/cell/view` is declared in the technology file for the custom via. It could be a fixed via or a Pcell.

You can have all standard vias, all custom vias, or a mixture of standard and custom vias.

Wire Objects

Wire objects include segment (pathSeg), continuous pathSegs, and T-junction. The description of each object as it is used when creating wires is as follows:

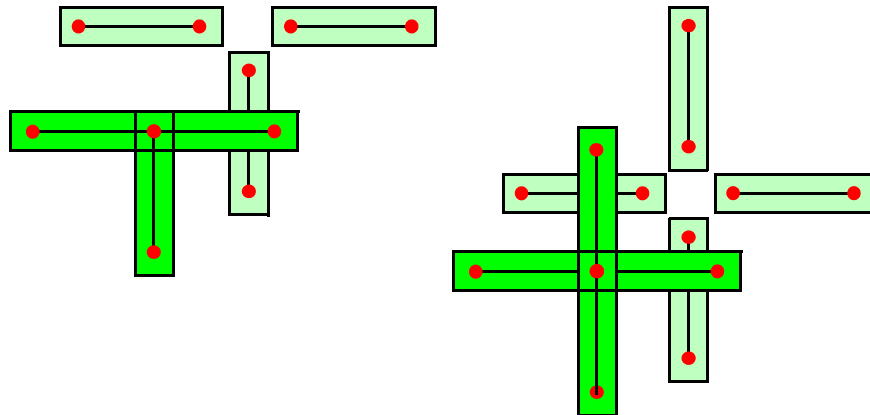
Segment (pathSeg)	A pathSeg is a shape used to realize physical routing that can either be orthogonal or diagonal. A pathSeg has a start and end point.  Starting point Ending point
Contiguous pathSegs	Two pathSegs are contiguous when they are on the same layer and when they have an overlapping point (start or end). Contiguous pathSegs  Non-continuous pathSegs 

Virtuoso Interactive and Assisted Routing User Guide

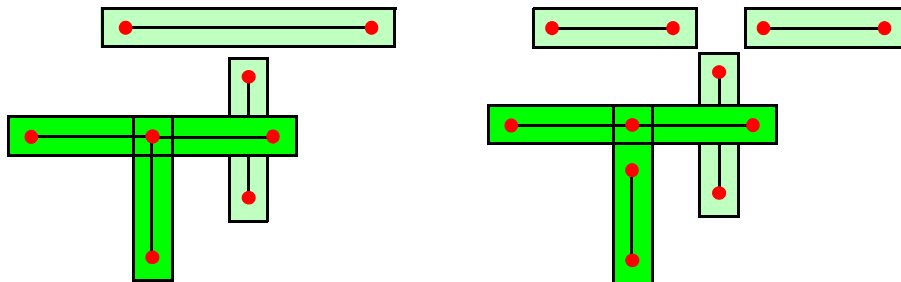
Working with Wires

T-junction A T-junction is a point where there are more than two contiguous pathSegs.

T-junction

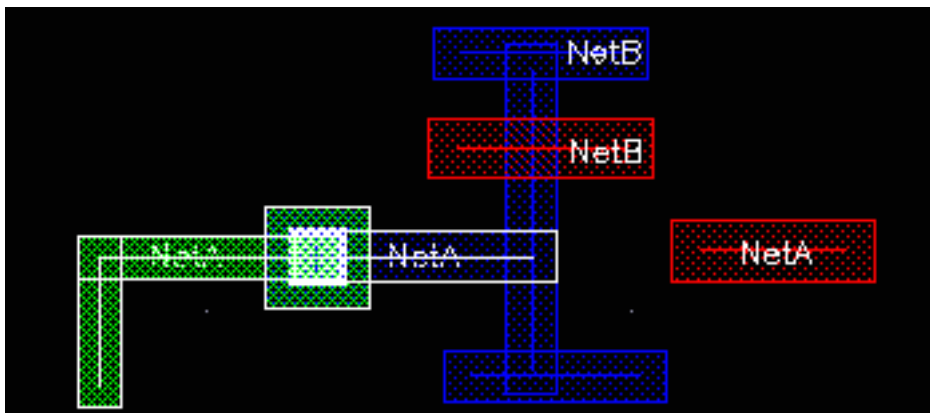


Not a T-junction

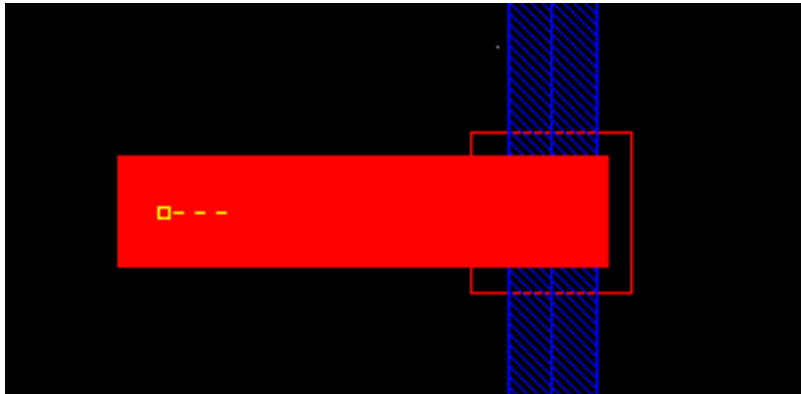


Selecting a Single Object or Via

Full or partial selection (press F4) selects a single wire element or via.



Partial selection (press F4) selects a single vertex of a wire element.



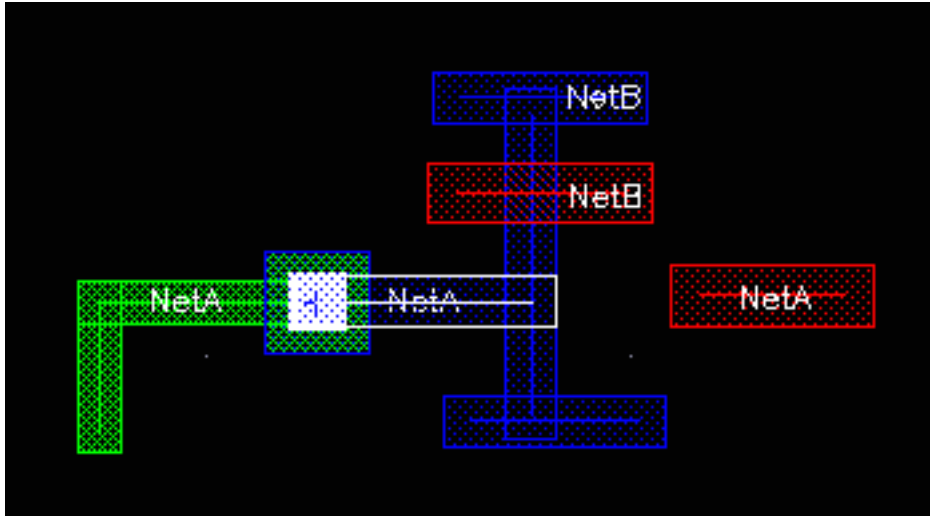
Routing Granularity Options

For selecting routing objects, you can specify *Routing Object Granularity* in the *Selection Options (Options – Selection)* form. The granularity options are supported only in the current hierarchy and in the Layout XL and higher tiers. The granularity options include:

- Shapes or Vias
- Entire Wire
- Connected Shapes
- All Shapes on a Net
- Net

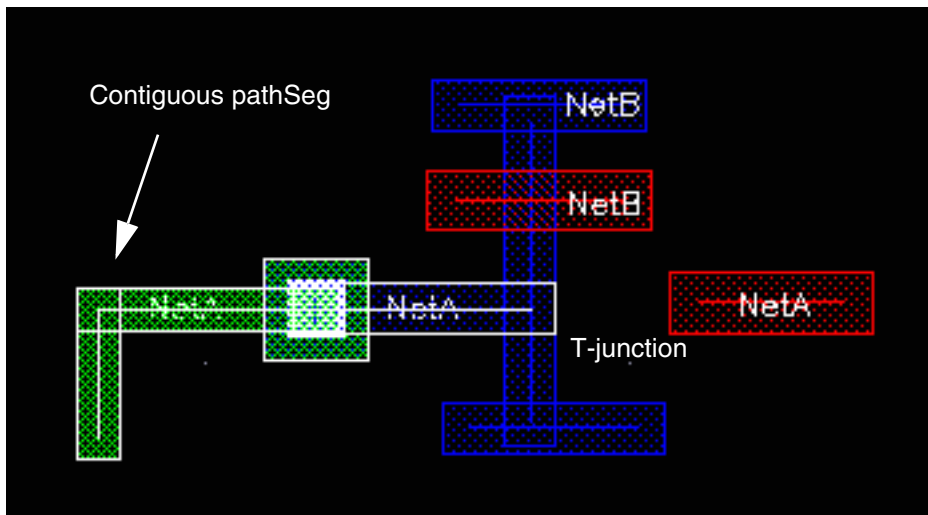
Shapes or Vias

In the *Shapes and Vias* mode, when a via or shape that is used to make a wire is selected, only the single object is highlighted.



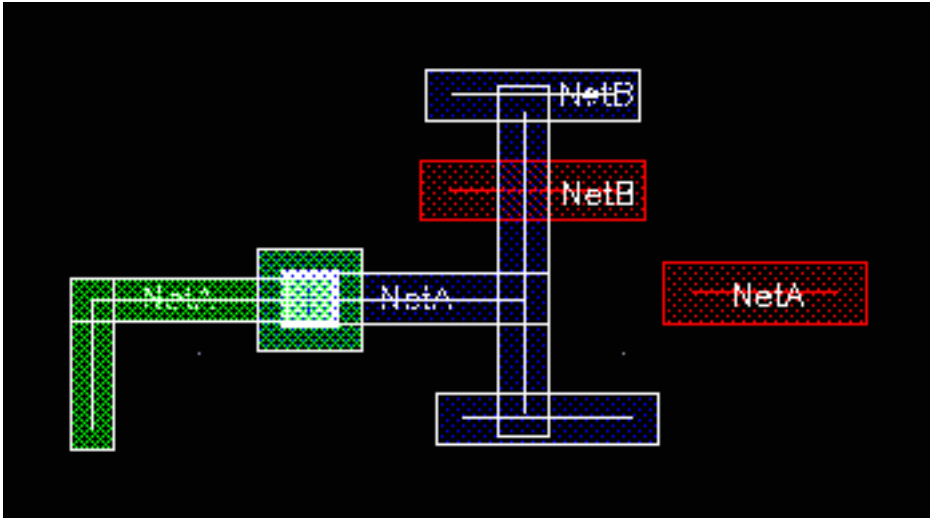
Entire Wire

In the *Entire Wire* mode, when a shape or a via is selected, the selection is extended to select all the objects forming the wire.



Connected Shapes

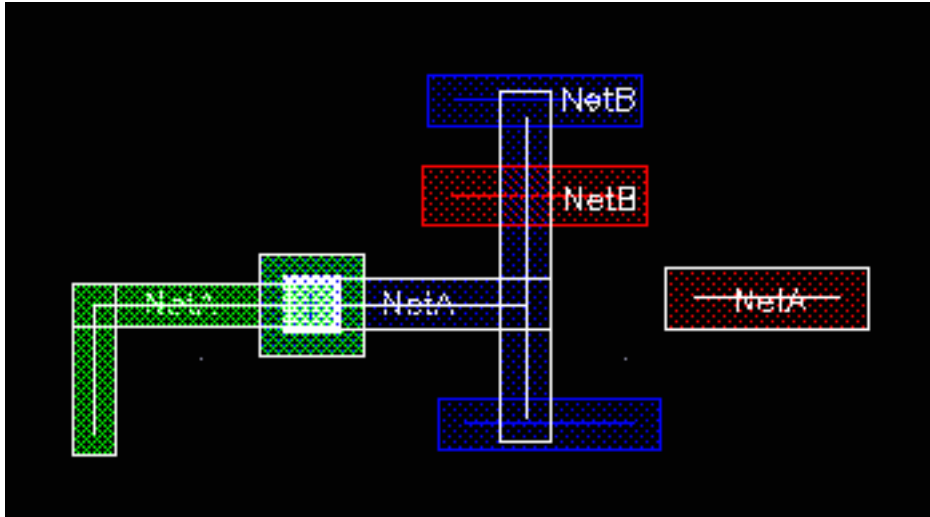
In the *Connected Shapes* mode, when a shape or a via is selected, the selection is expanded to select all the objects that form a physical connection. This mode does not take into account the net assignment on shapes and vias. The selection expansion stops if more than 1000 objects are found. In the case of more than 1000 objects, only the object on which the cursor rests is selected.



All Shapes on a Net

In the *All Shapes on a Net* mode, when a shape or a via is selected, the selection is expanded to select all the objects assigned to the same net. The selection expansion stops

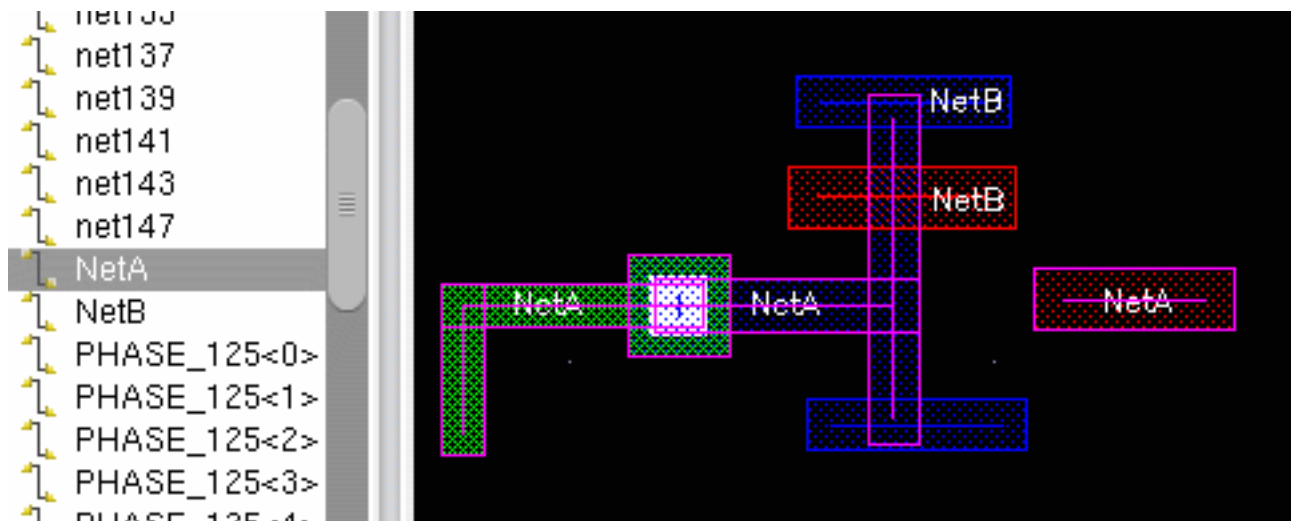
if more than 20,000 objects are found. In the case of more than 20,000 objects, only the object on which the cursor rests is selected.



Net

In the *Net* mode, the net object is selected. If a shape has no assigned net, it will not be highlighted.

The net is selected in the Navigator assistant and is probed in the layout window. The method in which the net is displayed (probed) in the layout window is determined by the *Highlight Options* form. The shapes assigned to the net can be highlighted or flightlines can be displayed. There is no limit on the number of shapes that can be highlighted.



Virtuoso Interactive and Assisted Routing User Guide

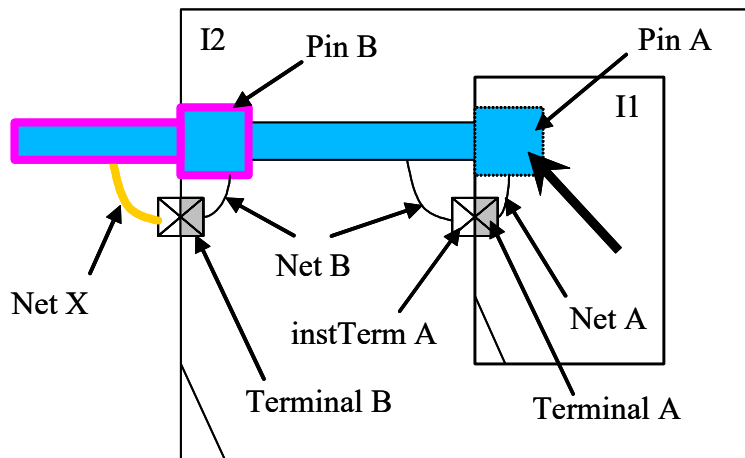
Working with Wires

In the *Net* mode, a net in the current edited cellview can be selected by clicking on any pin of an instance that is hierarchically connected to it.

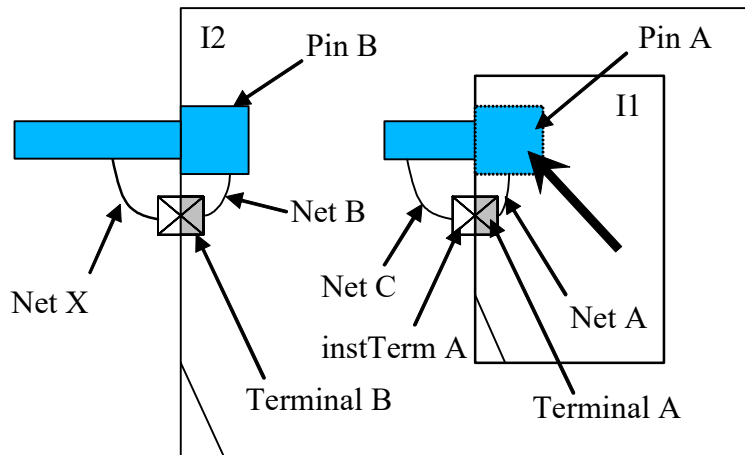
In Example 1: clicking on Pin A or Pin B selects the net X. Observe that when net X is selected, only the pins on level-1 are highlighted

In Example 2: clicking on Pin A selects nothing because the pin does not have a direct connection to a top level net.

Example 1



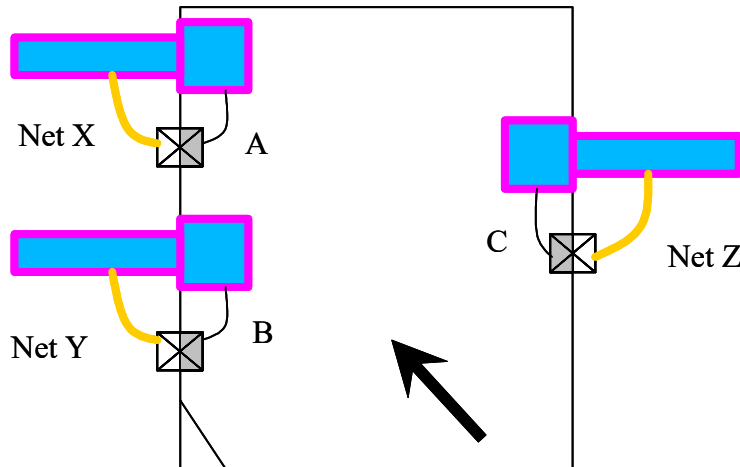
Example 2



Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

In the *Net* mode, clicking on an instance selects all the nets connected to it, if there is no pin at the cursor. In the example below, clicking on the instance selects the nets X, Y and Z.



The interactive routing commands support dynamic display of net name when the *Drag net name* option in the Net Name Display Options form is selected.

Wire Parameters

Width is one of the important parameters of a wire. The default wire width is determined from the `minWidth` constraint that can be set in different types of constraint groups. For information on constraint group precedence, see [Constraint Groups](#).

To set the wire width from the current constraint automatically, the current density must be specified in the technology file. Also, ensure that for `layerRules`, the `avgDCCurrentDensity` value is specified under `currentDensity` in the technology file.

Then, set the `maxDCCurrent` constraint on the object whose width is to be set. For example, to set the max DC current to 0.2mA on the net `myNet`, specify the following:

```
create_constraint_group -name myGroup -net myNet
set_constraint -group myGroup -constraint maxDCCurrent -FltValue 0.2
```

The router calculates the current-based width and snaps it up to the closest manufacturing grid. Then the existing `minWidth` constraints are also taken into account and the larger of the two is considered. In addition, if discrete wire widths are specified, the nearest greater wire width is selected. For example, the following five discrete wire widths have been specified: 1, 2, 4, 6, 8. Assume that based on the current constraint, the calculated width needs to be at least 4.3333. In this case, the nearest greater width value, which is 6, is considered.

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

When tapping a wire, if the *Tap Attributes* option in the Layout Editor Options form is on, the width of the new wire is determined from the width of the wire being tapped, provided it is compliant with the `minWidth` and `maxWidth` constraints.

The three environment variables `weTapWidthOnPins`, `weTapWidthOnVias`, and `weTapWidthOnWires` also determine the width of the wire being tapped.

For information about wire width when starting from pins, see [Setting the Wire Width](#).

This section covers the following topics:

- [Overriding Default Width Values By Using a Constraint Group](#)
- [Setting the Wire Width](#)

Overriding Default Width Values By Using a Constraint Group

You can create a constraint group to override the default `minWidth` and `minSpacing` values and attach the constraint group to a specific net or nets.

Note: The net overrides for `minwidth` and `minSpacing` do not apply to pin escape wires.

1. Create and load a constraint group.

The constraint group in the following example is called `wide`. The width and spacing values in the `wide` constraint group are greater than the `minWidth` and `minSpacing` values in the `foundry` constraint group.

```
;( group      [override] )
;( ----- )
( "wide      "nil

spacings (
  ( minWidth      "Metal1"0.3 )
  ( minSpacing    "Metal1"0.17 )
  ( minWidth      "Metal2"0.48 )
  ( minSpacing    "Metal2"0.3 )
  ( minWidth      "Metal3"0.48 )
  ( minSpacing    "Metal3"0.3 )
  ( minWidth      "Metal4"0.48 )
  ( minSpacing    "Metal4"0.3 )
  ( minWidth      "Metal5"0.48 )
  ( minSpacing    "Metal5"0.3 )
  ( minWidth      "Metal6"0.48 )
  ( minSpacing    "Metal6"0.3 )
) ;spacings
```

2. In XL, choose *Window – Assistants – Navigator*.

This opens the *Navigator* assistant.

3. Select the net for which you want to override the default values.

4. Choose *Window – Assistants – Constraint Manager*.

This opens the *Constraint Manager*.

5. Click the *Process Rules Editor* icon  .

The *Process Rules Editor* window opens.

6. Ensure that the required net is selected in the *Process Rule Editor*.

7. From the *Apply Constraint Group* list, select the constraint group that you created and loaded earlier.

8. Click the blue arrow button to attach the constraint group to the net.

The window updates to show the User-Defined constraint group.

9. Click the plus sign to display the values in the constraint group.

10. Exit the *Process Rule Editor*.

The width and spacing values set on the selected net will be used when routing the net interactively or automatically.

Setting the Wire Width

Right-click the mouse button when the *Create Wire* command is active. Select the *Use Width* command on the right-click context menu to display the options. This command also appears on the [Edit Bus Form](#).

The default width of the wire is determined from the constraint group selected in the [Layout Editor Options form](#).

When starting a wire, the width of the wire can be tapped from a pin, a via, or another wire. This is determined from the *Tap Attributes* in the [Layout Editor Options form](#) as well as the *Tap Wires Width*, *Tap Vias Width*, and *Tap Shapes and Pins Width* options in the *Use Width* context-sensitive menu. The *Tap Wires Width* option is used when starting from a wire, the *Tap Vias Width* option is used when starting from a via, and the *Tap Shapes and Pins Width* option is used when starting from a shape or a pin.

If the width is not tapped, it is determined by the *Constraint Overrides* options and *Default Constraints* option available in the *Use Width* context-sensitive menu.

During the wire creation, the value of the *Width* field determines the width for the current segment and the value of the width mode in the context-sensitive menu describes how it has been computed. Modifying the width in the *Width* field of the Create Wire form automatically changes the mode to *Constraint Override - by layer*. Changing the mode accordingly changes the width of the existing wire.

Note: If you select the *Use Width – Default Constraints* option from the right-click context menu, it will overrule the width specified in the *Width* field of the Create Wire form. The Width field is updated to reflect the used width when changing the mode.

While digitizing points to create the wire, add vias to change layers. The method to control the selection of vias is described in detail in the [Changing Layers and Selecting Vias](#) section. The width after a change of layer is again computed using the same mode. In order not to change the width, select the *Constraint Override - Same for all layers* mode.

For information about changing layers, see [Changing Layers and Selecting Vias](#). For information about creating via arrays, see [Creating an Array of Vias](#).

Creating Wires

The *Create Geometric Wire* enables you to create paths, pathSegs, and vias that are not in routes. The *Create Wire* command enables you to create pathSegs and vias that are in routes. You can define the wire characteristics by using the [Point to Point Form](#) and the *Create Wire* form. You can also define wire characteristics by using the [Wire Assistant](#) while creating the wire. All these settings override the values in the application constraint group that you specify in the *Wire Editing* section of the *Layout Editor Options* form. The override values are persistent in the current session.

Before creating wires, ensure that the following requirements are satisfied:

- The required layer and via definitions exist in the technology file. For more information, see [Technology Requirements](#).
- The required settings for wires are configured. For more information, see [Setting up the Interactive Routing Environment](#).

This section covers the following topics:

- [Creating a Wire](#)
- [Creating a Diagonal Wire](#)
- [Creating a Floating Wire](#)
- [Creating a Wire with 45 Degree Segments](#)
- [Creating an Odd Width Wire](#)
- [Creating Wires Snapped to Track Patterns](#)

Creating a Wire

You can create two [Types of Wires](#): geometric and symbolic.

1. To create geometric wires, open the [Create Wire Form](#) using the `leHiCreateGeometricWire` command.
To create symbolic wires or pathsegs, select *Create – Wiring – Wire*. Press F3 to open the [Create Stranded Wire Form](#).

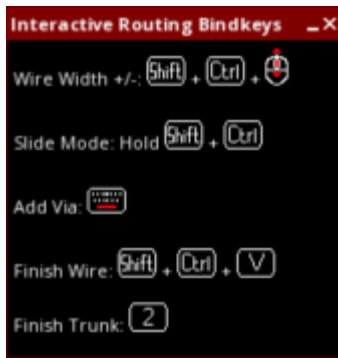
You can use one of the following alternative methods to invoke the command and then press F3 to open the respective form:

- Click the *Create Wire* icon  on the *Create* toolbar.

- ❑ Right-click at any free space in the design canvas to display the *Layout* context menu and select *Create – Wire*.

Note: The above alternative methods open the [Create Stranded Wire Form](#).

When you start the *Create Wire* command in Layout EXL, the *Interactive Routing Bindkeys* hint box is displayed at the top-left corner of the layout canvas. The hint box describes the most useful bindkeys and shortcuts for the *Create Wire* command.



2. (Geometric only) Select *Paths* or *Segments* from the *Create Geometric Wire* form to specify the type of wire element to be used for creating wires. For more information, see [Wire Elements](#).

(Symbolic) The *Create Wire* form, by default, uses pathSegs in route for creating single wires.

Note: While creating a geometric wire the 20nm rules such as `minWireOverlap`, `allowedWidthRange`, `allowedLengthRange`, and `rectShapeDir` are not supported for paths.

3. (Optional) In the *Net Name* field, specify a net name for the wire.

The wire being created is assigned to the specified net. If the specified net does not exist in the design, the net is created.

4. To change the width of the current wire, type the desired width in the *Width* field.

The default width displayed in the *Width* field is determined from the `minWidth` constraint. For information about overriding the default value, see [Overriding Default Width Values By Using a Constraint Group](#).

If the command is active but the form is hidden, you can use *Use Width* option on the [Create Stranded Wire Form](#) to change the width of the wire or open the form to specify a value. See [Setting the Wire Width](#).

The *Create Wire* command follows a center-line model, which requires that half the wire width is a multiple of the manufacturing grid. In addition, when you create a wire, the center line of the wire snaps to the manufacturing grid. If the width you specify is a multiple of the manufacturing grid, the width is adjusted for snapping.

- ❑ During *Create Wire* and *Create Geometric Wire* commands (leHiCreateGeomtericWire), the length of a path(Seg) segment is checked with respect to its width. The option to check the length of a path(Seg) segment and enforce the same is the *Half Width Check option* in the Layout Editor Options form.

Note: The Create Geometric Wire command generates paths or pathSeg segments, while the Create Wire command only create pathSegs.

- When *Half Width Check* option is set to *off*, which is the default, no warning is issued when you create a path(Seg) segment that has a length less than half the width.
- When *Half Width Check* option is set to *notify*, a dialog box is displayed informing you that an error condition has been detected, but allows you to continue with the operation.
- When *Half Width Check* option is set to *enforce*, you will not be allowed to create a path or pathSeg with length less than half the path width and a dialog box is displayed informing you that the path or pathSeg length is less than half the width.

5. (Optional) Turn on *Justification* to control which direction to offset the path or pathSeg from the digitized points of the path or pathSeg.
6. (Optional) To offset the wire from the pointer by a specified distance, type the offset amount in the *Offset* field. This is a numeric field and does not allow a blank entry. When an entry other than a numeric value is given, a warning is issued and the offset is set to the previous value.

Note: When you change the justification or the offset in the Create Wire form, the already generated shapes are not changed and are retained as is.

7. Select the beginning and ending extension style.

- ❑ If you use paths to create a wire, choose *truncate*, *round*, *extend*, or *variable*.
- ❑ If you use pathSegs to create a wire, set the *Begin Style* and *End Style* to one of *truncate*, *extend*, *variable*, or *custom*.
- ❑ The default extension is *truncate* for path or pathSeg endpoints, and *variable* for paths or pathSegs ending on a via.

Note: The begin and end style that are finally applied to a pathSeg depend on the begin and end extension values you specify for them. For example, if you choose *custom* as the *Begin Style* and *End Style* but the extension value corresponds to that of *extend*, the pathSeg created will have *extend* begin and end styles. You can view the begin and end style of the pathSeg created in the *Property Editor* assistant.

8. Specify a ROD name if you want to create the wire as a ROD object. This is applicable only for *Paths*.
9. Choose a value from the *Snap Mode* cyclic field. If set to *orthogonal* or *diagonal*, you can choose to create orthogonal or diagonal wires.
10. Specify if you want the wire to snap to the center of rectangular pins.
11. (In Layout XL and higher tiers) Select *Auto Terminate* while creating a wire(s) for terminating the wire(s) with a single click.
12. (In Layout XL and higher tiers) You can choose to *Probe Nets*.
13. (In Layout XL and higher tiers) Select *Pin Covers* to automatically cover the source and target pins.
14. To start creating a wire, do one of the following:
 - ☐ In the *Net Name* field, specify a net name to start creating a wire in empty space in the design display area. See [Creating a Floating Wire](#).
 - ☐ Pre-select an I/O pin before starting the *Create Wire* command and then digitize subsequent points.
 - ☐ Area select a pin (I/O pin or Soft Block pin), via, path, or wire to tap the layer and attributes to define the starting point of the wire. Digitize the wire. For more information about tapping objects, see [Tapping Wires](#).
 - ☐ (In Layout XL and higher tiers) To use the smart snapping feature of wire, switch on the *Smart Snapping* command on the [Create Stranded Wire Form](#).

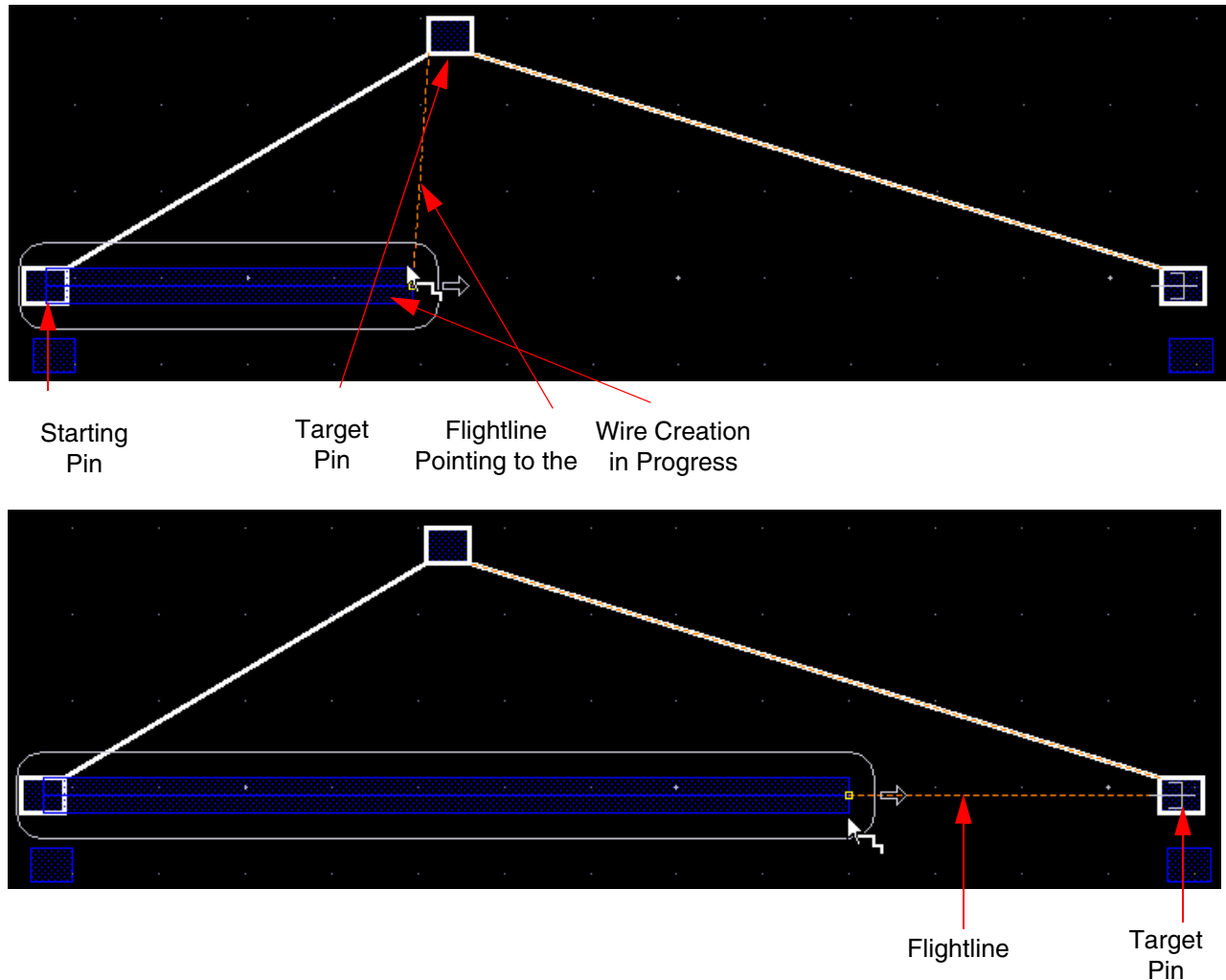
For more information, see [Smart Snapping in Interactive and Assisted Routing Commands](#).

Note: While creating a wire from a top-level pin, the smart snapping feature ignores the access direction of the pin. When starting a wire from an instance pin, the smart snapping feature respects the access direction of the instance pin. If the access direction of the instance pin is `nil`, all directions (top, bottom, left, right) are available for starting the wire.

Virtuoso Interactive and Assisted Routing User Guide

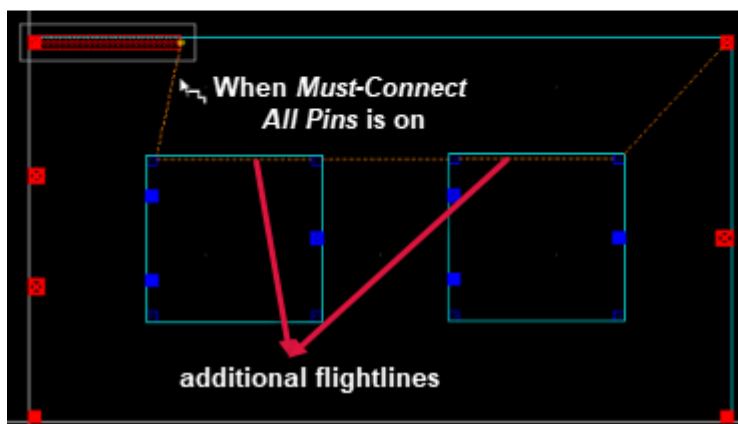
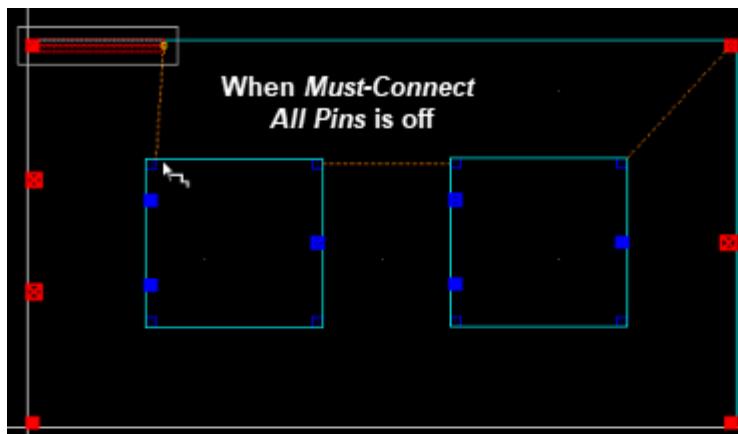
Working with Wires

When using the *Create Wire* command, flightlines are displayed between current net targets. The flightline from the current pointer position to the closest target is dynamically updated as the pointer moves.



The *Create Wire* command identifies terminals with the must-connect pin setting and displays the additional flightlines for all the pins that need to be connected. The

flightlines are displayed between the currently selected pinFig of the source terminal and the end points for all pins of the destination terminal.



For more information, see [Setting a Pin Connectivity Model](#).

- ❑ For floating wires, select a layer from the Layers Assistant. For more information, see [Creating a Floating Wire](#).

In Virtuoso Layout Suite XL and more advanced versions, when starting a path or pathSeg from an existing path or pathSeg, the starting point is snapped to the centerline of the tapped path or pathSeg. In Virtuoso Layout Suite L, the starting point remains as digitized.

15. Change the wire layers. For information about changing layers, see [Changing Layers and Selecting Vias](#).
16. To remove the last segment of a wire, use **Backspace**.

Multiple wire editing supports *Undo*, *Redo*, and **Backspace**.

17. To finish creating the wire, press **Enter**. For more information, see [Finishing a Trunk](#).

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

You can use *Smart Snapping* on the Create Stranded Wire Form to snap the wire to the target pin. The wire snaps only to pins that are on the current layer and on the same or an unassigned net. If the pin is on the same net as the wire, smart snapping auto-terminates the wire by adding a segment to the snapping point.

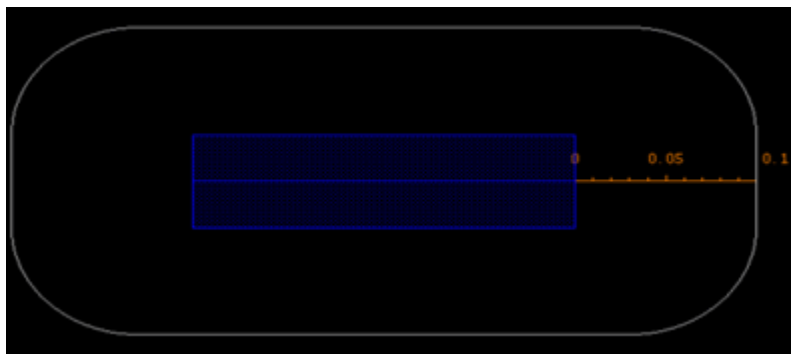
The *Create Wire* command allows starting or finishing wires in open space, at locations in the design where there are no objects. The *Create Wire* command also allows starting or terminating wires on existing pathSegs to create "T" connections.

Note: In the *DRD Options* form (*Options – DRD Edit*), if DRD is enabled and *Interactive Display* is set to *Halos*, a halo is created around a wire that is being created or edited if the wire violates a spacing constraint. The halo created around the wire is not uniform. If there are 2D spacing tables in the technology file, the halo spacing on the edges of the wire is based on the length of the wire, whereas the spacing on the ends of the wire is based on the width of the wire.

In addition, the *Create Wire* command displays halo taking into consideration the `minEndOfLineSpacing` constraint defined in the technology file. For example:

```
( minEndOfLineSpacing      "Metal1"      'width 0.09 'distance 0.07
'minLength 0.2 'twoSides 0.1 'ref  "METAL1.SP.3"  'description
"Minimum Metal1 End of Line Spacing" )
```

In the above example, the `minEndOfLineSpacing` is 0.1. Therefore, the halo spacing that is considered by the *Create Wire* command is 0.1.



Note: The halo created around a wire is not accurate. To get accurate results and for better performance, enable the DRD *Notify* or *Enforce* mode.

Creating a Diagonal Wire

When creating wires, you have the option of using paths or pathSegs as elements of the wire. Using *Segments* (pathSegs) allows you to create a string of joined path-like objects with the

assurance that the mask outline of the wire is on the manufacturing grid, including points where adjacent pathSegs join. Using paths to create wires does not assure that the paths are on grid, since paths can be created in *anyAngle* mode.

PathSeg end styles are automatically adjusted so that they are in accordance with the manufacturing grid. End styles are adjusted to extend style. When reshaping a pathSeg the same adjustments occurs; the resulting pathSegs will be adjusted to adhere to the manufacturing grid.

Creating a Floating Wire

To create a wire in the empty space in the design display area:

1. Choose *Create – Wiring – Wire*.
2. Press F3 to open the form.

The Create Stranded Wire Form appears.

3. In the *Net Name* field, type in a net name for the wire.
4. In the *Layers* assistant, select a layer for creating the wire.
5. Digitize points where you want to route the wire.

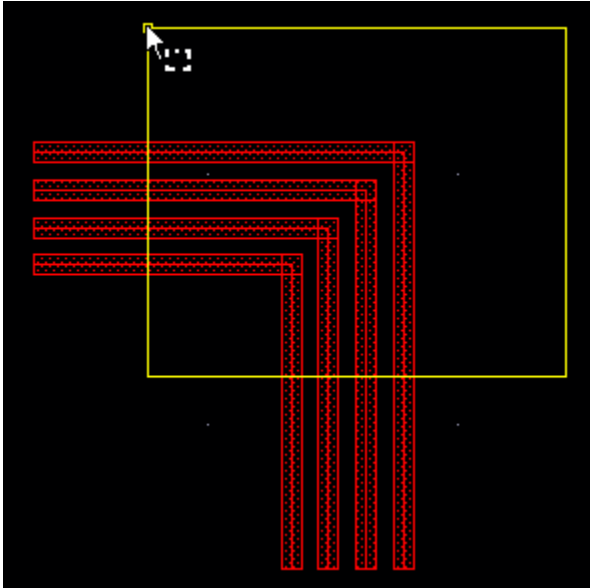
Creating a Wire with 45 Degree Segments

The *Wire to 45* command is available only for path segments, both with single and multiple wires (bus). This command is available in Layout XL and higher tiers.

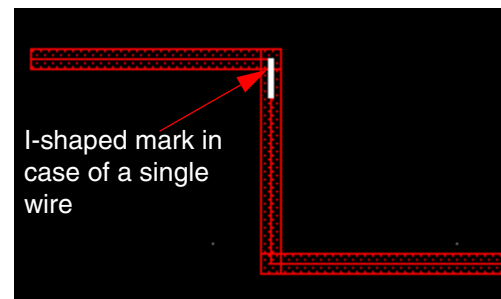
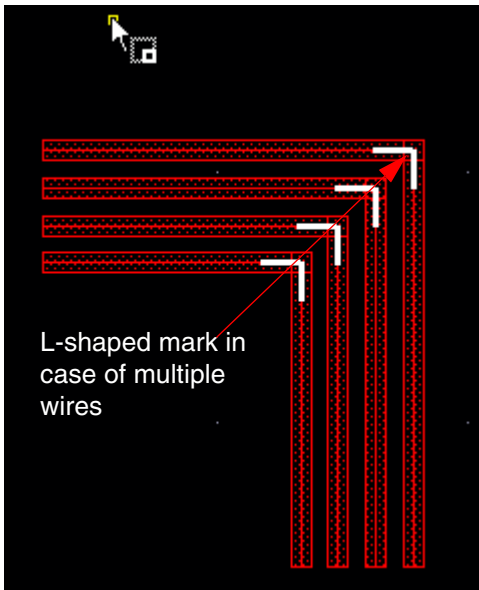
With the *Wire to 45* command you can adjust the right-angled corners to create 45-degree wire segments that are design rule correct. The *Wire to 45* command can be applied to a bus only if all wires are on the same layer. To start the *Wire to 45* command:

1. Choose *Edit – Wiring – Wire to 45*.

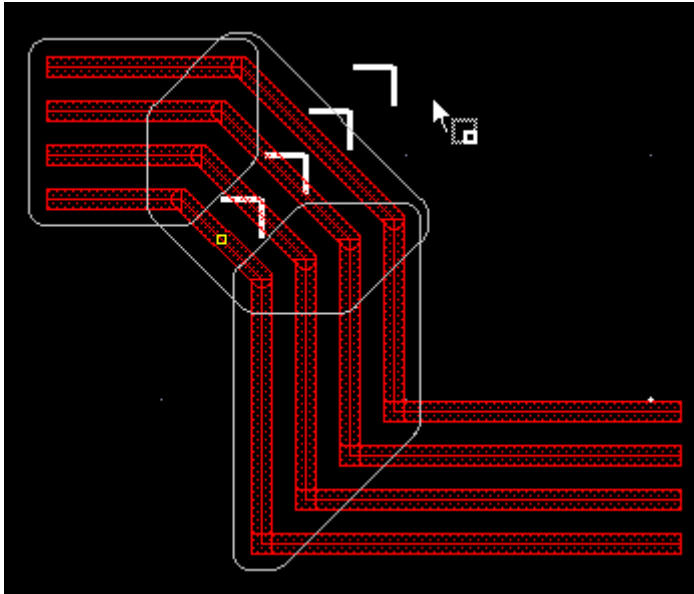
2. Click and drag to select the right-angled corners of a bus, as shown in the following figure.



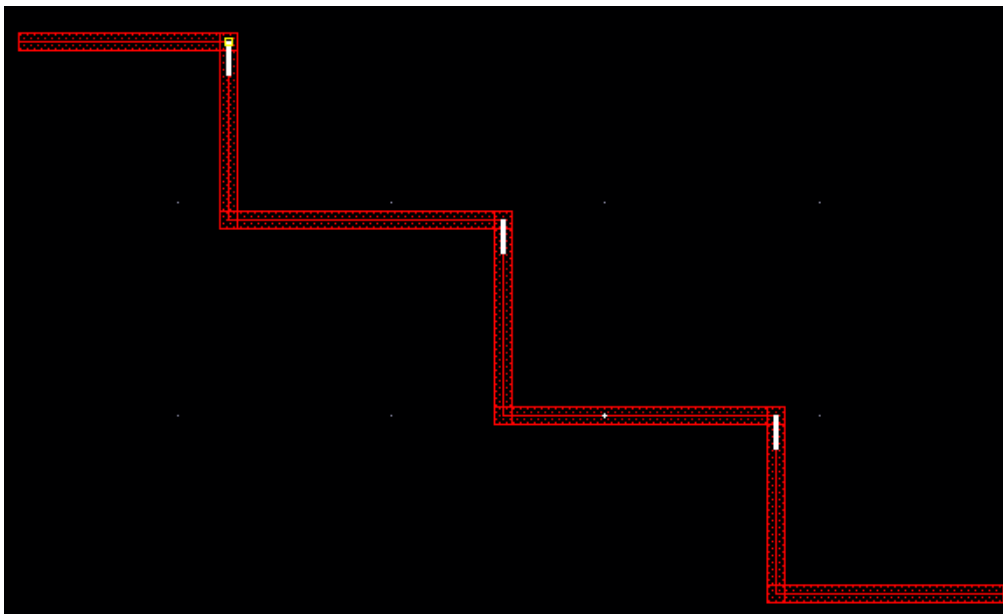
A white, L-shaped mark appears on the inside line of the wires, as shown in the following figure. In case of a single wire, a white, I-shaped mark appears on the inside line of the wire.



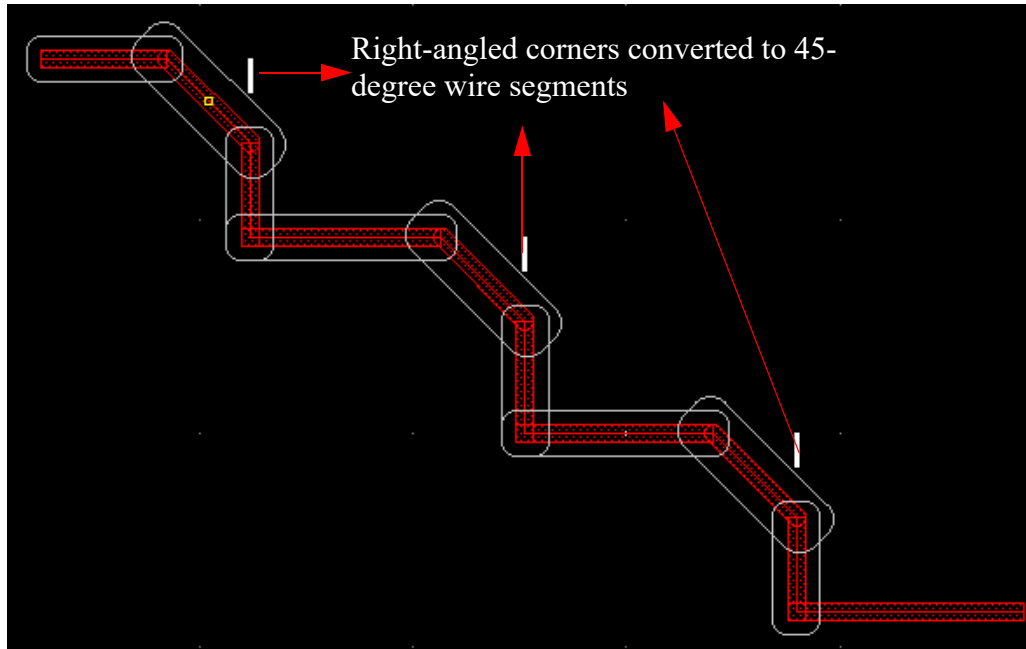
3. Click a wire with the mark and drag it to the point where you want to create the 45 degree wire segment. The bus with right-angled corners changes to the 45-degree segments, as shown in the following figure.



You can select and adjust multiple right-angled corners for a single-wire path segment. To select multiple right-angled corners, hold down the `Shift` key and then select the required corners using the mouse, as shown in the following figure.



Now when you click the wire with the mark and drag it to the point, multiple right-angled corners are converted to 45-degree wire segments, as shown in the following figure.



Note: In the bus mode, you can select and adjust only one right-angled corner at a time.

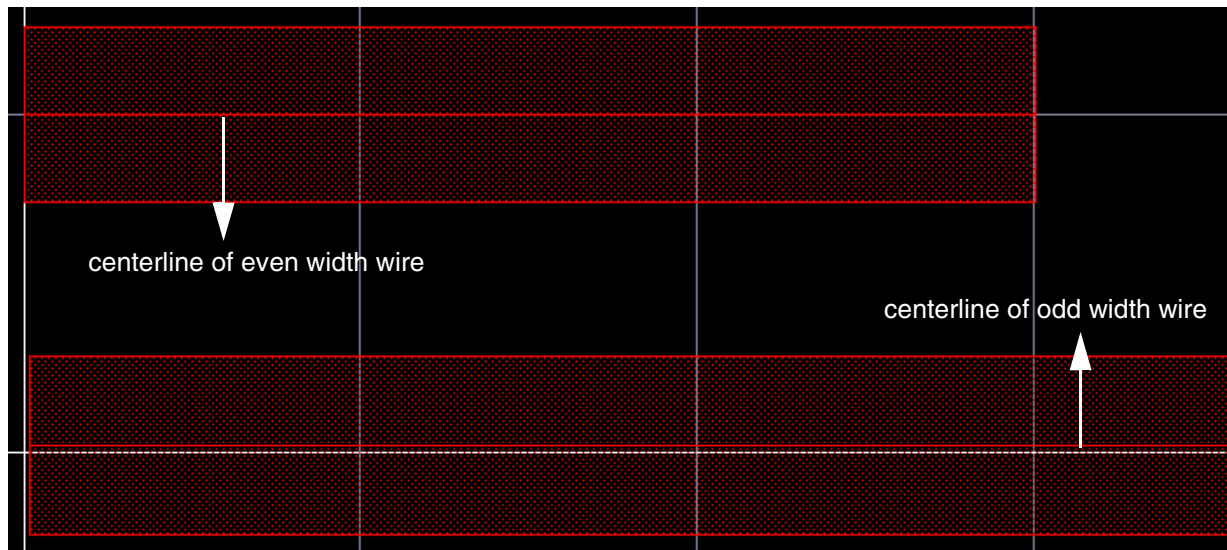
Creating an Odd Width Wire

Before IC6.1.6 ISR4, when the width of the wire was specified as an odd value in the Create Wire form, it was automatically adjusted to the smallest even width greater than the input. For example, if the manufacturing grid (mfgGrid) is $0.001\mu\text{m}$ and the width specified by you is $0.015\mu\text{m}$, the width of the wire is automatically adjusted to $0.016\mu\text{m}$. However, now you have the flexibility of creating a wire with an odd width. An odd width is a width which is an odd multiple of the mfg. For example if $\text{mfgGrid}=0.001\mu\text{m}$ and the width of the wire is $=0.015\mu\text{m}$, this is an odd width of the wire.

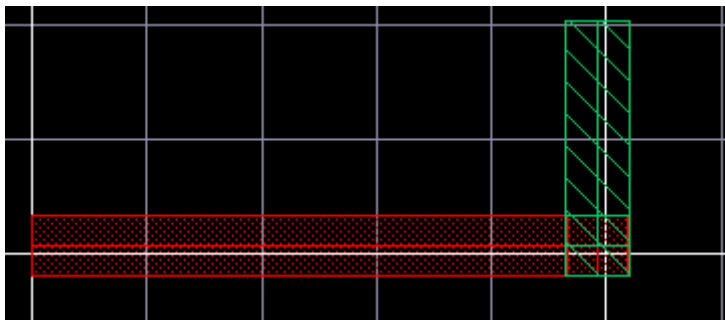
When you create a wire with an even width, the centerline of the wire segment is snapped to the mfgGrid. However, when creating a wire with an odd width, the centerline of the wire segment is off the mfgGrid. The steps to create a wire with an odd width are as follows:

1. Choose *Create – Wiring – Wire* to start the Create Wire command.
2. Press **F3**. The Create Wire options form displays.
3. Specify an odd value in the *Width* field so that the wire with an odd width is created.
When the wire is created with an odd width, the centerline of the wire segment is not on

the mfgGrid, whereas the edges of the wire segment are on the mfgGrid, as shown in the following figure.



4. When creating a wire with an odd width, you can insert a via by selecting the *Via Up <metal layer>* option from the *Create Wire* context-sensitive menu. The following figure displays a via created between Metal2 and Metal3 wire segments.



Creating Wires Snapped to Track Patterns

Snapping wires to track patterns is supported in all tiers of Virtuoso Layout Suite. You can select the snapping mode as *Manufacturing Grid*, *Routing Grid*, *Off Grid*, *Track Pattern*, and *Snap Pattern*. When the *Snap To Track* option is selected in the Layout Editor Options form, you are allowed to create wires only in the area where there are intersecting horizontal and vertical tracks. When stretching wires, if more than one linear section is selected, snapping to track is disabled.

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

The objects that are snapped to grids include wires, paths, and pathSegs. The following commands support track pattern snapping:

- *Create – Wiring – Wire*
- *Edit – Stretch*
- *Edit – Move*
- *Edit – Copy*
- *Create – Via*

For displaying and creating track patterns, see [Creating Track Patterns](#) in the Virtuoso Layout Suite documentation.

For more information on colored tracks, see [Track-Based Coloring](#).

To enable snapping of wires to track patterns, perform the following steps.

1. Choose *Options – Editor*.
2. Select the *Track Pattern* option from the *Wires* drop-down list in the *Snapping* group box. When the *Track Pattern* option is selected, the *Enforce directions* check box can be selected.



When the *Enforce directions* check box is selected, the wire is snapped to the center line of the track pattern. Also, the wire inherits the width of the track pattern and the direction in which the wire is created is the same as the direction of the track pattern. The *Enforce* check box is selected by default.

However, if the *Enforce directions* check box is deselected, the width of the wire is the default width and there is no restriction on the direction in which the wire is snapped.

Note: The *Enforce* check box is only enabled when the *Wires* option is either set to *Track Pattern* or *Snap Pattern*.

If existing wires are snapped to tracks, the wires can be edited and snapped to tracks using the *Move*, *Stretch*, and *Copy* commands. When the *Track Pattern* option is selected from

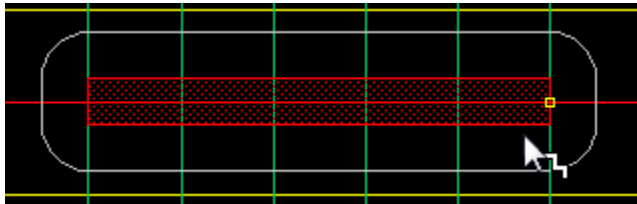
the *Wires* drop-down list and the *Move* and *Copy* commands are selected, the pin center of rectangular pins is snapped to tracks.

Snapping of Wire on Tracks

During wire creation, the valid track patterns of a layer are displayed. Also, only the tracks corresponding to the current layer and a layer above or below it are shown.

Before wire creation, the valid track patterns of the current layer are dynamically displayed. When a wire is being created, only the tracks to which wire is snapped is displayed. The track patterns of the adjacent layer (upper and lower) are also displayed.

In order to have the wires snapped to the track patterns and WSPs, ensure that the snapWireGrid environment variable is set before wires are created. Thus when a wire is created on a metal layer, the magnet automatically pulls the wire towards track patterns intersections and the wire is snapped to the nearest track intersections with its adjacent upper and lower layer. Finally, the created wire inherits the width and direction of the track to which it is snapped, as shown in the following figure.



Important

The *Create Wire* command follows the tracks and is unable to create vertical wires if there are only horizontal tracks on a metal layer and vice versa.

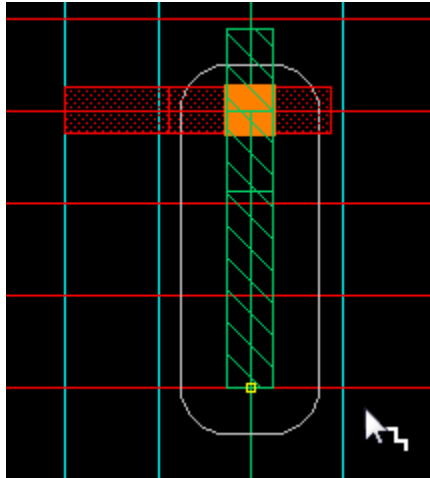
Automatic Via Insertion on Tracks During Wire Snapping

During wire creation, the track magnet snaps to the intersection of the upper or lower layer tracks. When you click on the track intersection, a via is automatically placed and the wire

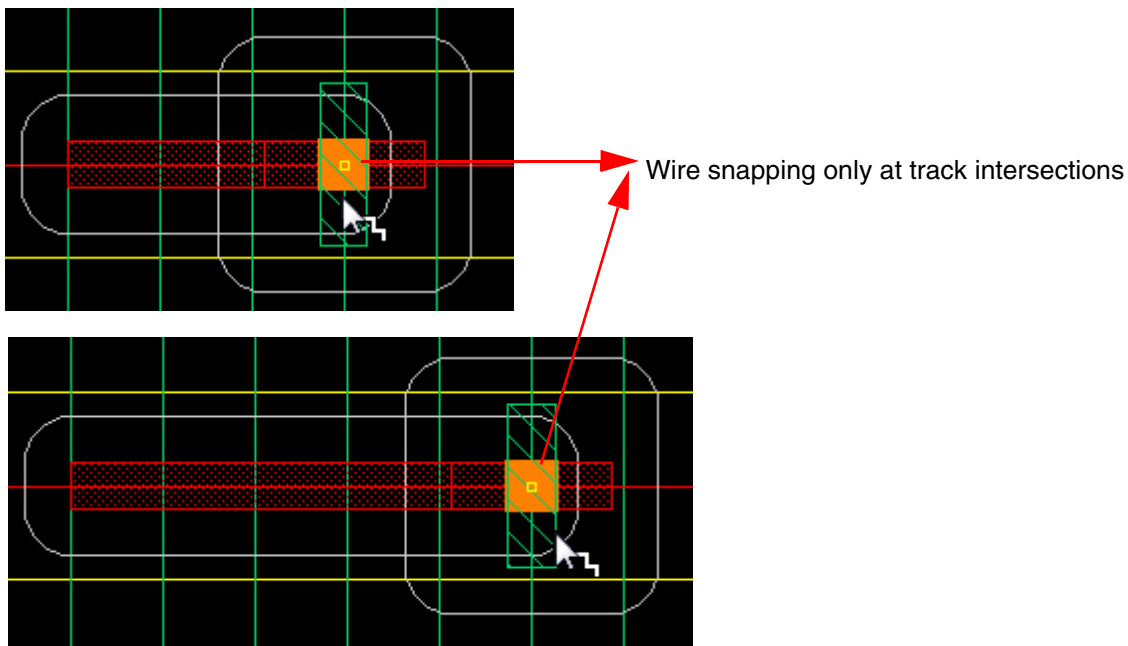
Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

then changes the direction and moves in the perpendicular direction, as shown in the following figure.



You can also place a via before clicking on a track intersection by using the *Via up to <layer_name>* and *Via Down to <layer_name>* options in the *Create Single Wire* context-sensitive menu. In this case, the wire always snaps at the track intersection, as shown in the following figure.



After the via is automatically placed, one of the metal layers is selected by default if the tracks of the lower and upper layers coincide. However, you can switch between the valid tracks of

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

the upper and lower layers by selecting the *Cycle Snapping Layer* option in the *Use Width Spacing Pattern* submenu. In the following set of figures, Figure A shows that the wire is created on `Metal4`, and when the via is placed on the track intersection, `Metal5` is selected

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

as the default layer. Figure B displays the switch between the selected layers when the *Cycle Snapping Layer* option is clicked and `Metal3` layer is now selected.

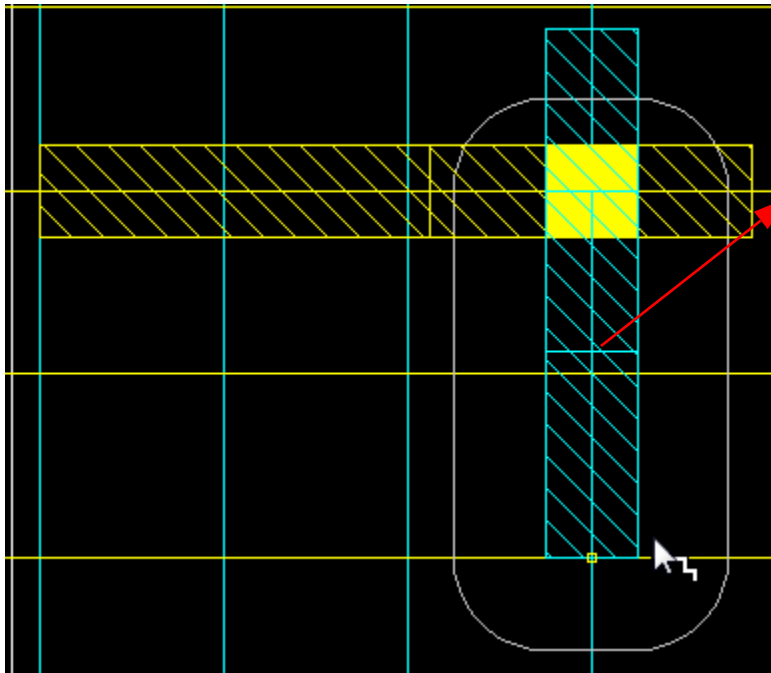
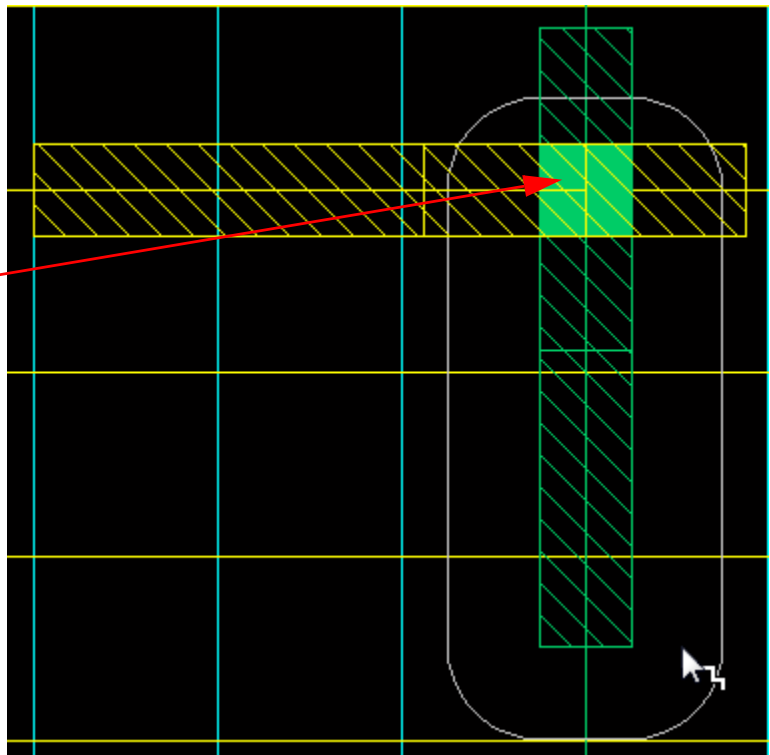


Figure A: Default layer selection (Metal5) for via insertion

Figure B: Layer selection (Metal3) using the *Cycle Snapping Layer*



Assigning a trackPattern Constraint to a Net

You can assign a `trackPattern` constraint to a net by attaching a constraint group to a net default constraint group and that constraint group in turn has a `trackPattern` constraint. The constraint describes a list of allowed `trackPattern` names per routing layer to which wires are snapped to. The `trackPattern` names can be specified in the *Create – Track Pattern* GUI.

Editing Wires

This topic covers the following sections:

- [Editing Wire Properties](#)
- [Editing Via Properties](#)
- [Editing Cut Patterns](#)
- [Multiple Wire Editing](#)
- [Editing Wire Width](#)
- [Editing Layer-Purpose Pair](#)
- [Adjusting Vias While Editing Layer-Purpose Pair](#)

Editing Wire Properties

To display or modify a wire element, follow these steps:

1. Select wire element you want to edit and choose *Edit - Properties* [q].

The attributes of the selected wire element display in the form. If all elements of the wire are selected, they can be accessed using the *Next* and *Previous* buttons on the *Edit Properties* form.

2. Click the *Next* or *Previous* buttons to choose a specific element of a wire.
3. Type or select new values for the wire element and click *OK*.

For information about editing properties, see [Editing and Defining Properties](#), in the Virtuoso Layout Suite documentation.

The following tables shows the attributes of wire elements, which can be modified.

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

PathSeg Element Attributes

Attribute	Modifiable
Layer	Yes
bBox	No
Width	Yes
Begin Point	Yes
End Point	Yes
Begin Style	Yes
End Style	Yes
Begin Extension	No
End Extension	No
Route Status	Yes
Part of a Route	No
Color	No
State	No
Connectivity	
Route Net Name	Yes
Net Criticality	Yes
Property	
<i>propertyName</i>	Yes

Path Element Attributes

Attribute	Modifiable
Layer	Yes
bBox	No
Points	Yes
Width	Yes
Begin Extension	No
End Type	Yes
End Extension	No
Route Status	Yes
Part of a Route	No
Color	No
State	No
Connectivity	
Net Name	No
Net Criticality	Yes
Property	
<i>propertyName</i>	Yes

Editing Via Properties

For information about editing properties of vias, see [Editing and Defining Properties](#) in the Virtuoso Layout Suite documentation.

Editing Cut Patterns

Only standard vias are supported when editing the cut pattern of a via array.

To modify a cut pattern, follow these steps:

1. Select standard via array you want to edit and choose *Edit – Properties* [q].

The *Edit Via Properties* form appears.

2. Click on *Edit Cut Pattern*.

The Edit Cut Pattern form appears and the current pattern for the via array is displayed.

3. Select one of the following:

- ☐ Click *Quick Edit* and select a predefined pattern from the *Patterns* cyclic field.
- ☐ Click *Wall Paper* and select a predefined pattern from the *Patterns* cyclic field. Further edits can be made by turning on or off the buttons of the pattern to edit the array.
- ☐ Click *Manual Edit* and turn on or off the buttons of the entire array pattern.

4. Click *OK* in the *Edit Cut Patterns* form to update the data of the via array.

Multiple Wire Editing

Multiple wire editing is allowed in Virtuoso Layout Suite XL and higher tiers.

When interactively routing multiple wires, if you enter points and then want to back up, use the backspace to undo the points you entered, then enter the new points. Do not draw segments back over previously drawn segments.

Each net involved in multiple wire editing can have its own constraints. If a net is part of a bus or bundle, constraints can be specified on the bus or bundle to share the same constraints. Net-based constraints are supported, providing the ability to have different widths in the wires in a bus, or different spacings between different pairs of adjacent wires in the bus.

Wires created follow the applicable constraints for the nets to which each individual wire belongs.

- If two adjacent wires in the bus have different spacing requirements, the larger value is used.
- If there is a taper rule constraint specified on a terminal, a wider or narrower wire width will be routed within the specified taper window.
- If there is a minimum extension constraint, wire ends connecting to vias are adjusted appropriately.

For information about finishing multiple wire connections automatically, see Finishing a Trunk.

Editing Wire Width

You can use the Property Editor assistant or the Edit Properties form in order to change the width of a single segment or the width of several segments simultaneously. However, this essentially considers segments independently. Changing the width of a segment would create notches if its begin and end extensions' values and styles are set improperly. The connected segments at its begin and end points must be considered in order to compute the extension style and values appropriately, thus avoiding notches.

Therefore, a new capability has been introduced wherein; when changing the width of a wire segment, its begin and end extensions' values and styles are adjusted automatically in order to avoid notches with other connected segments. This enhancement only works for pathSegs and the behavior depends on the set of selected segments. Consider a specific segment whose width is being changed: if all the segments of the spine to which this segment belongs to are selected, then changing the width of this specific segment actually changes the width of all the segments of the spine to the same value. Otherwise, the width of only this segment is changed.

- [Using the Property Editor Assistant](#)
- [Using the Edit Properties Form](#)

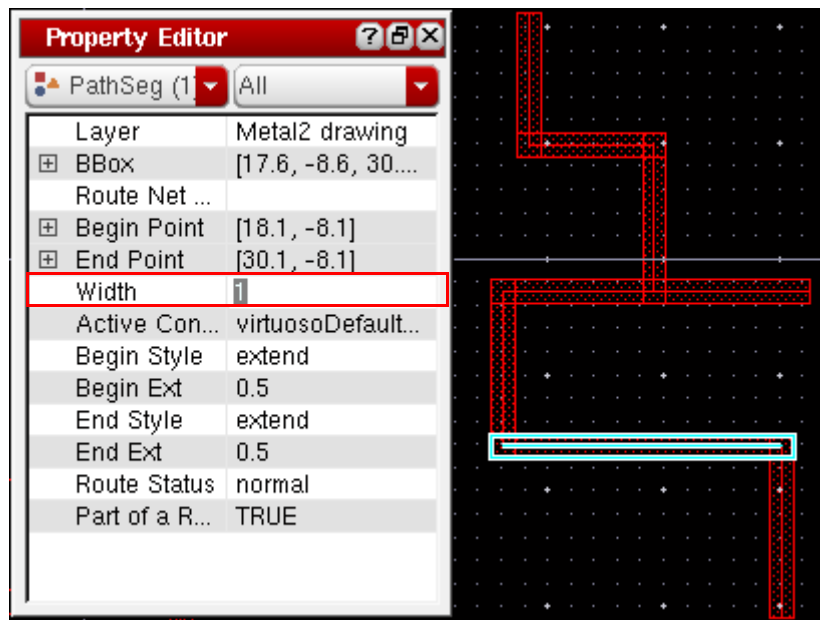
Using the Property Editor Assistant

Using the Property Editor assistant, you can change the width of a single wire, multiple wires, or a spine. To change the width using the Property Editor assistant, from the layout window, select the wires whose width you want to change and then choose *Window – Assistants – Property Editor*. Alternatively, right-click anywhere in the menu bar or the toolbar area and

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

choose *Property Editor*. The *Property Editor* assistant is displayed. The following figure shows the Property Editor assistant along with the selected wire in the layout window.



You can now edit the width of a single wire, multiple selected wires, a single spine, or multiple spines using the *Width* field of the Property Editor assistant.

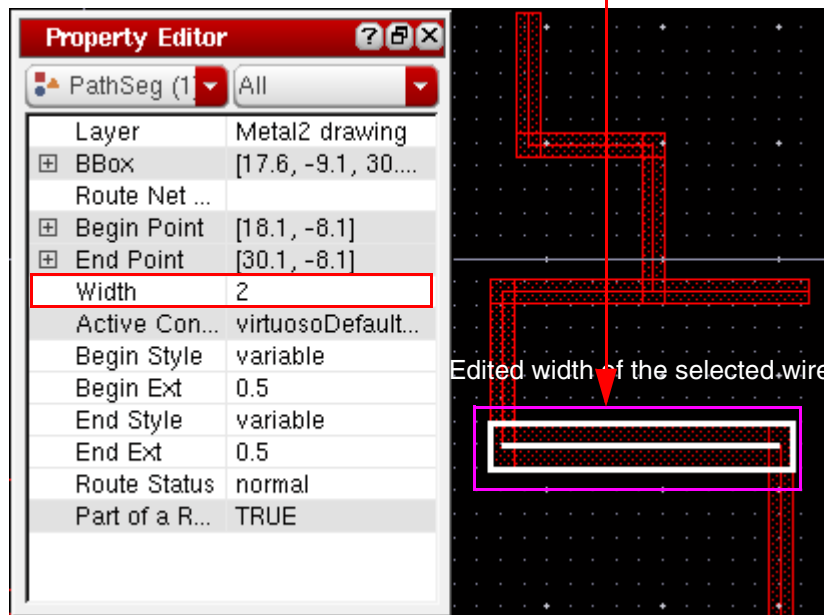
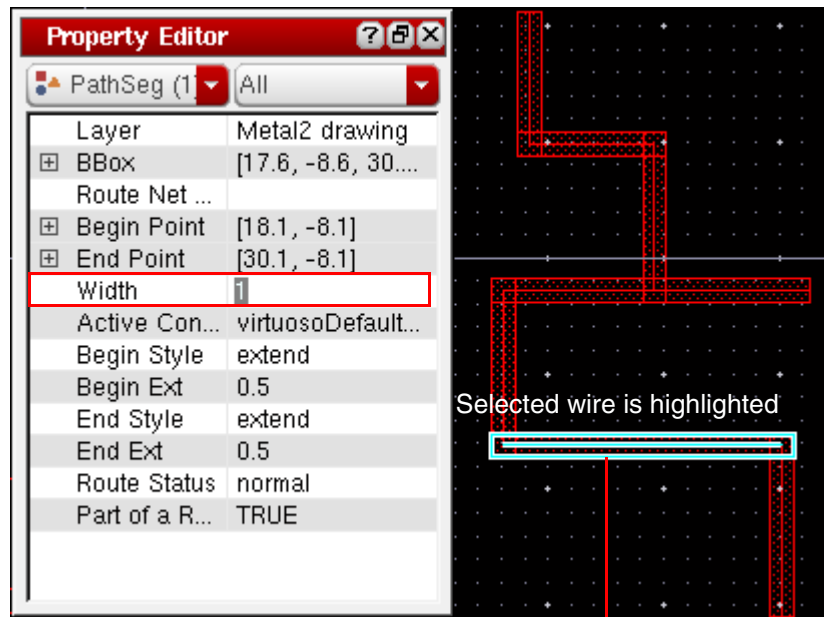
Selecting a Single Wire

You can select a single wire only when the *Path Spine* mode is OFF. When a single wire is selected, you can edit the width of a wire by specifying a new width in the *Width* field of the Property Editor assistant. The following figure shows how the selected wire is highlighted and

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

the width of the selected wire is changed in the layout canvas, when a new width is specified in the *Width* field of the Property Editor assistant.

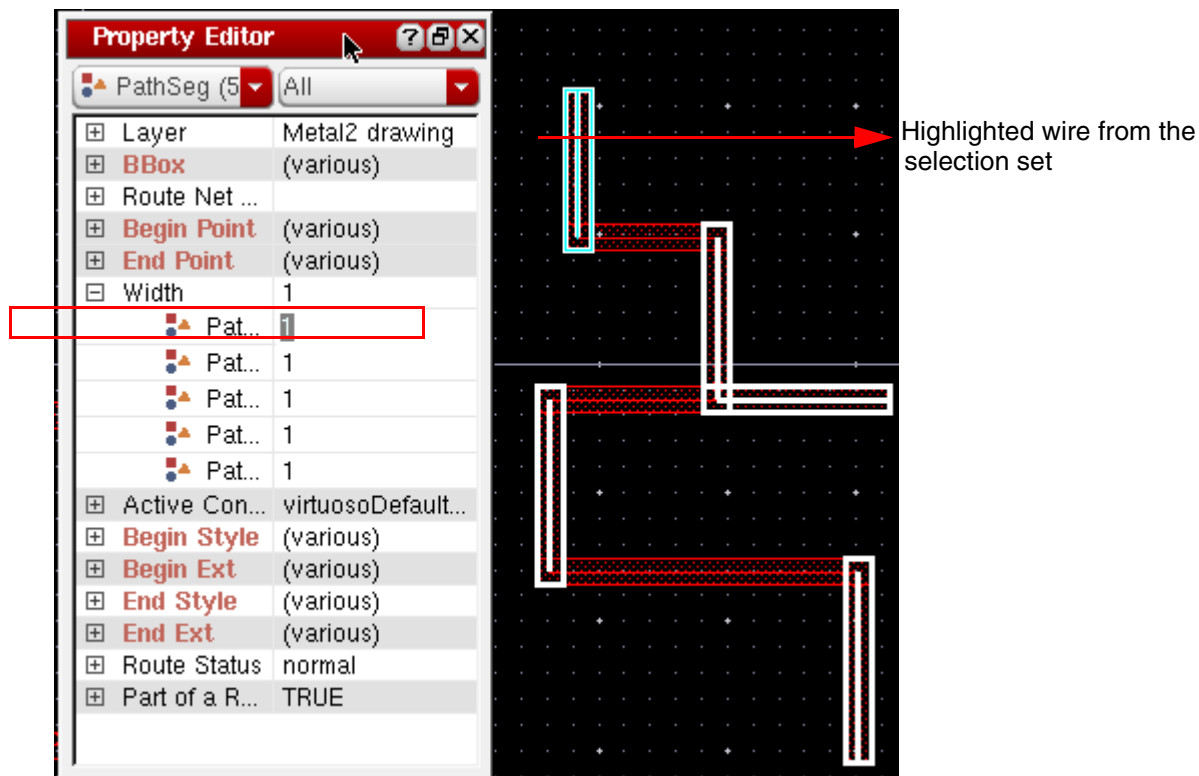


Selecting Multiple Wires

When the *Path Spine* mode is OFF, you can select multiple wires from the layout design. Once the wires are selected, you can either change the width of multiple wires simultaneously or change the width of a single wire from the selection set.

■ Editing the width of a single wire from a selection set

Using the Property Editor assistant, you can edit the width of a single wire from the selection set. Place the pointer on the wire, which you want to edit, in the Property Editor assistant. The following figure displays how the wire on which the pointer is placed is highlighted in the layout widow.

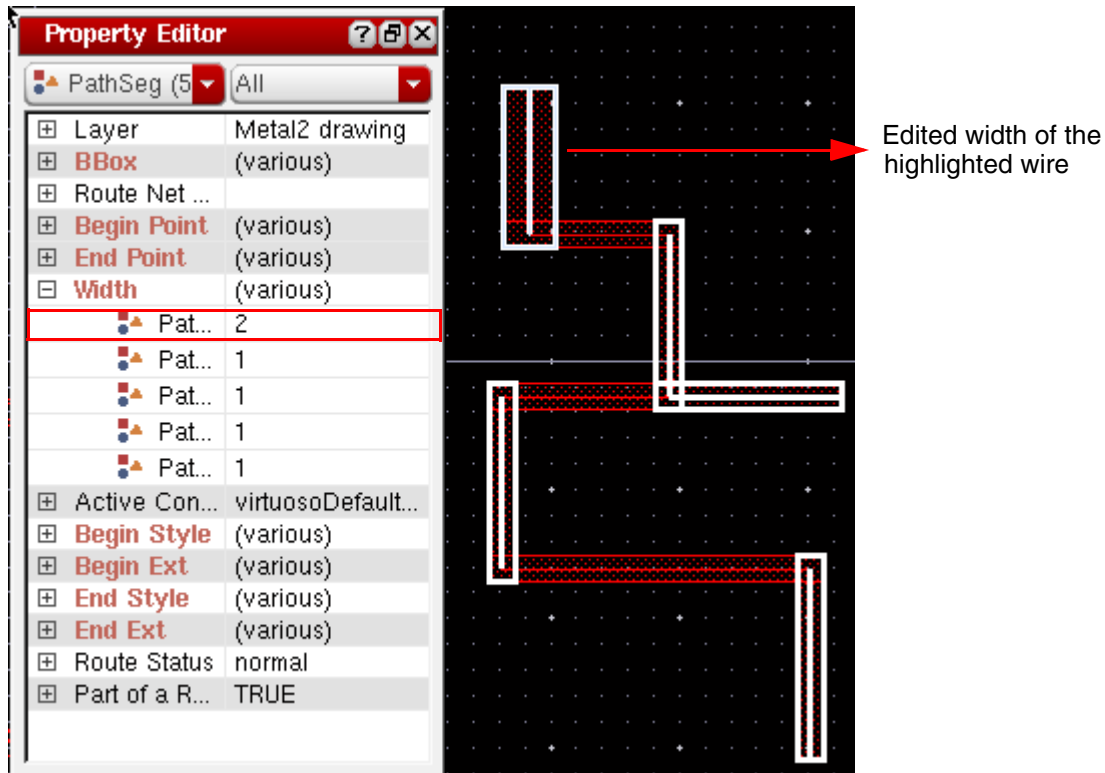


By default, the first selected wire is highlighted in the Property Editor assistant.

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

To change the width of the highlighted wire, specify the new width value in the column next to the selected wire in the Property Editor assistant and press the **Enter** key. The following figure displays the changed width of the highlighted wire from the selection set.

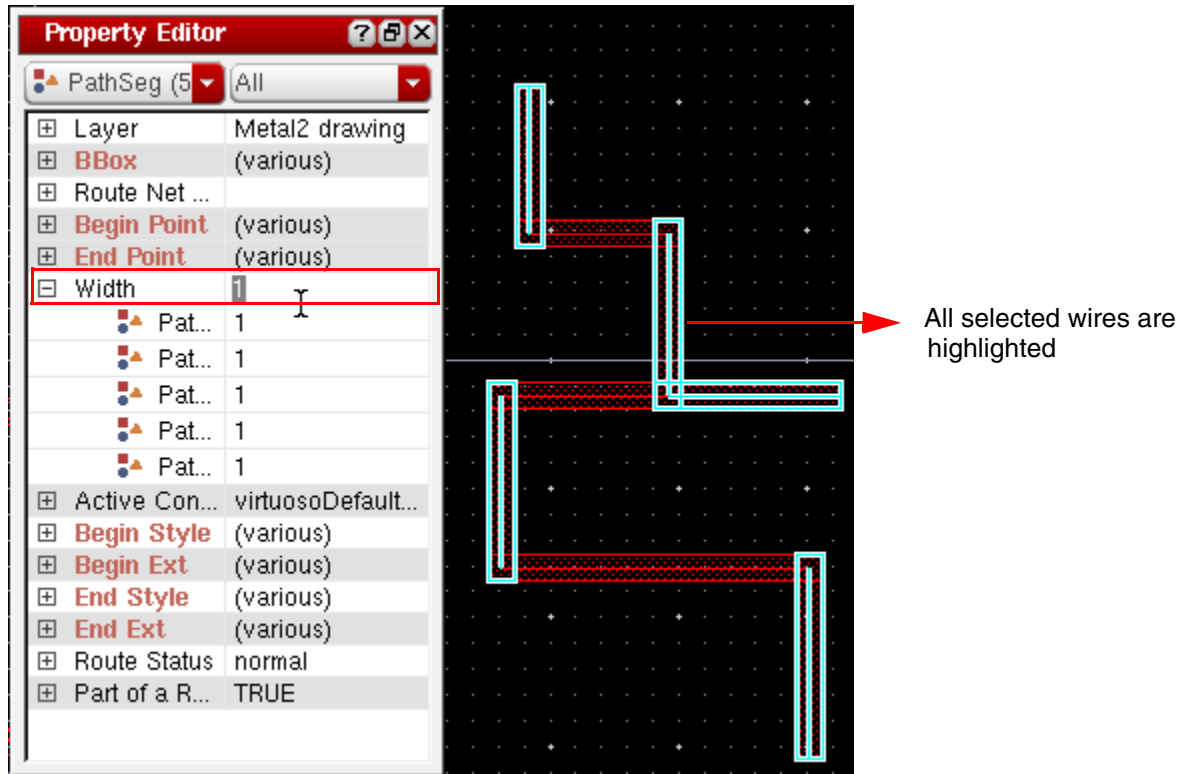


- Editing width of multiple wires simultaneously

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

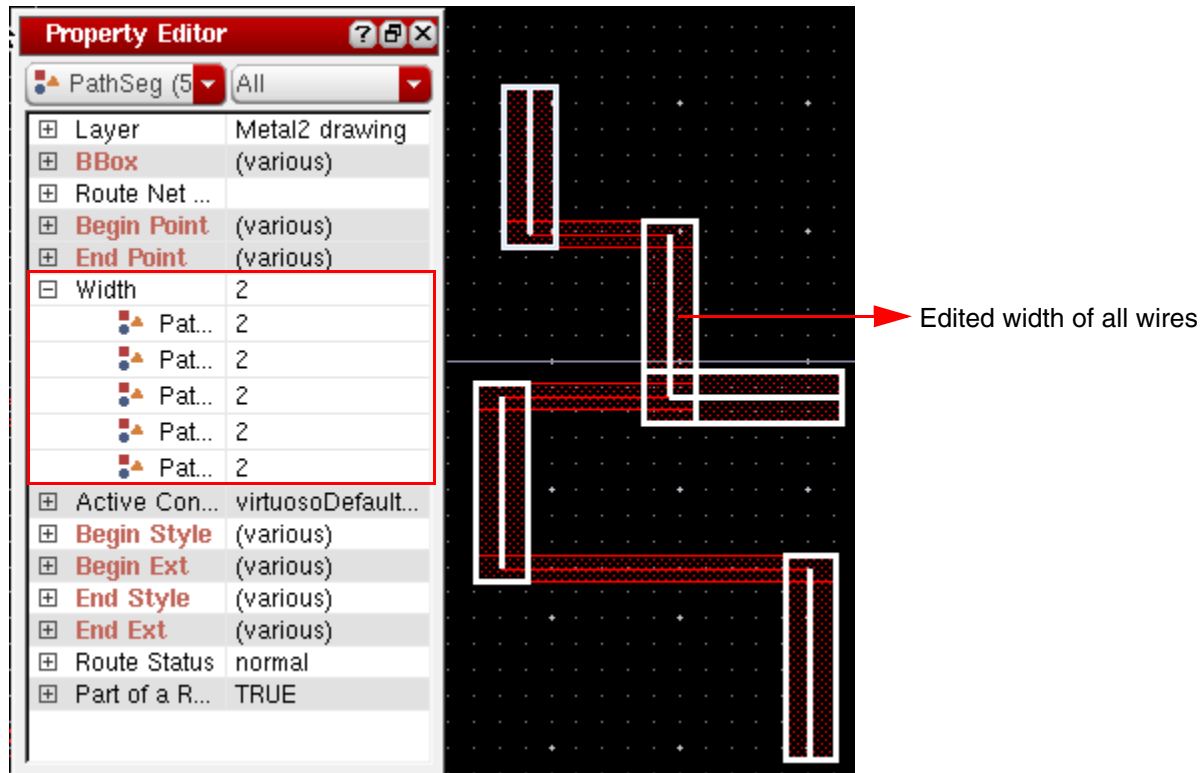
To edit the width of multiple wires simultaneously, place the pointer on the *Width* field in the Property Editor assistant. All the wires in the selection set are highlighted, as shown in the following figure.



Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

Specify the new width value in the column next to the *Width* field of the Property Editor assistant and press **Enter**. The following figure displays how the width of all the selected wires has changed simultaneously.



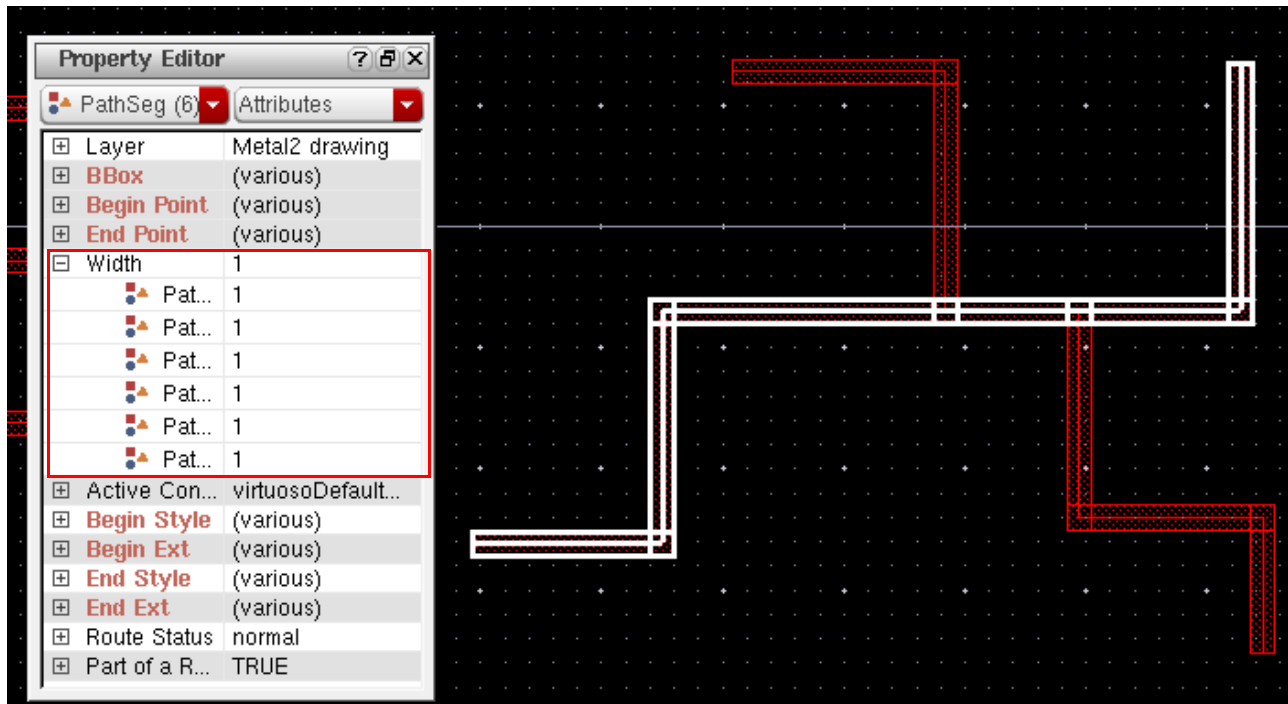
Selecting a Spine

Using the Property Editor assistant, you can also change the width of a set of selected wires, which forms a spine. To change the width of a spine, select a spine from the layout window.

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

The following figure displays the selected spine along with each of its elements in the Property Editor assistant.

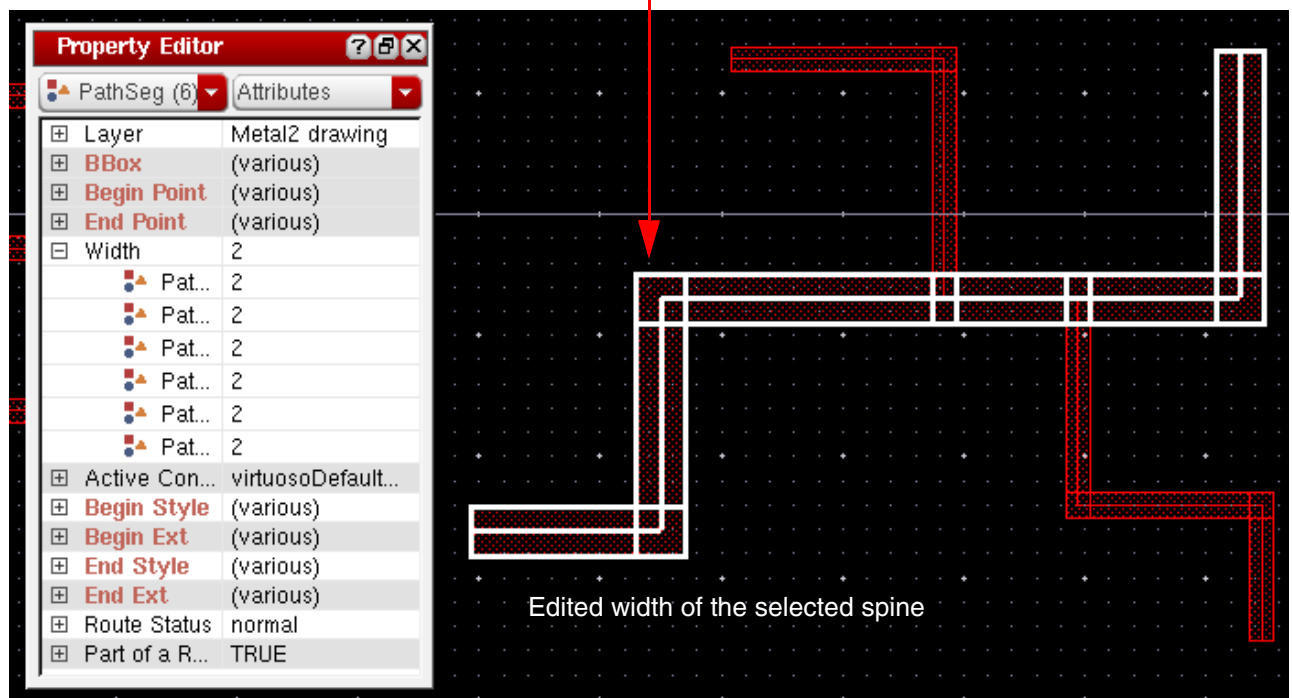
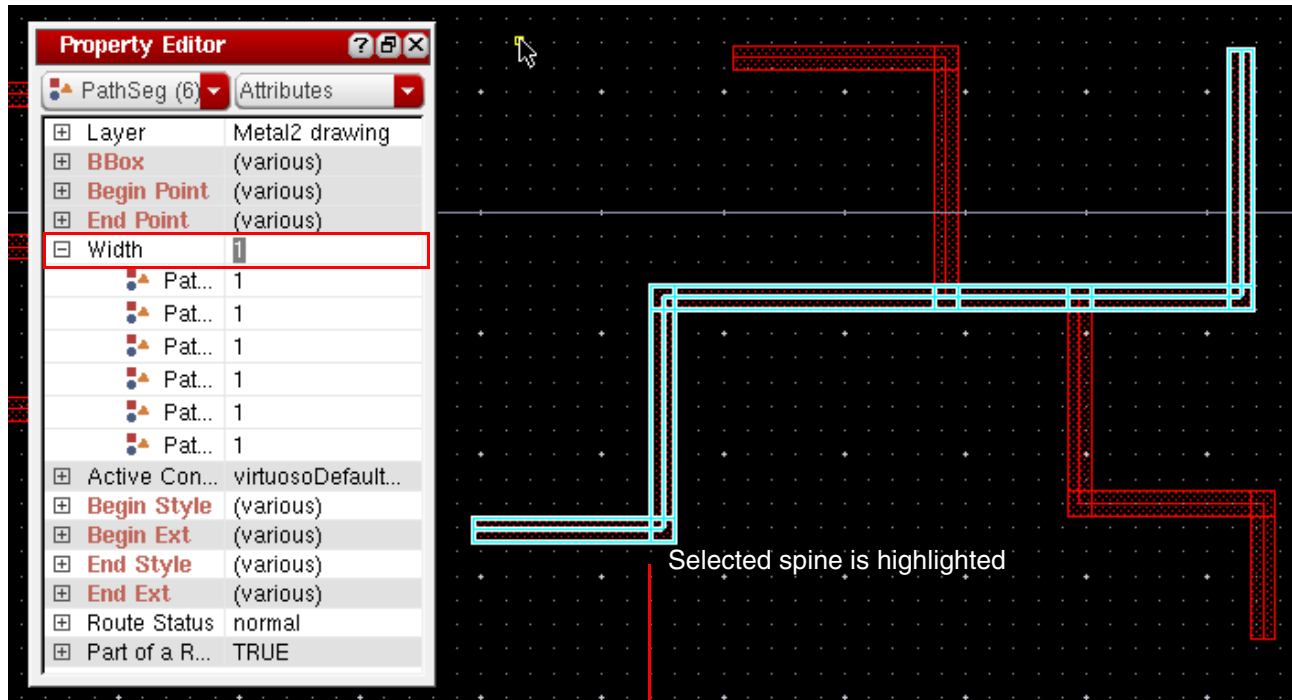


You can edit the width of entire spine by placing the pointer on the *Width* field of the Property Editor assistant and then specifying the new width value in the *Width* field. The following figure displays how the entire spine is highlighted when the pointer is placed on the *Width*

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

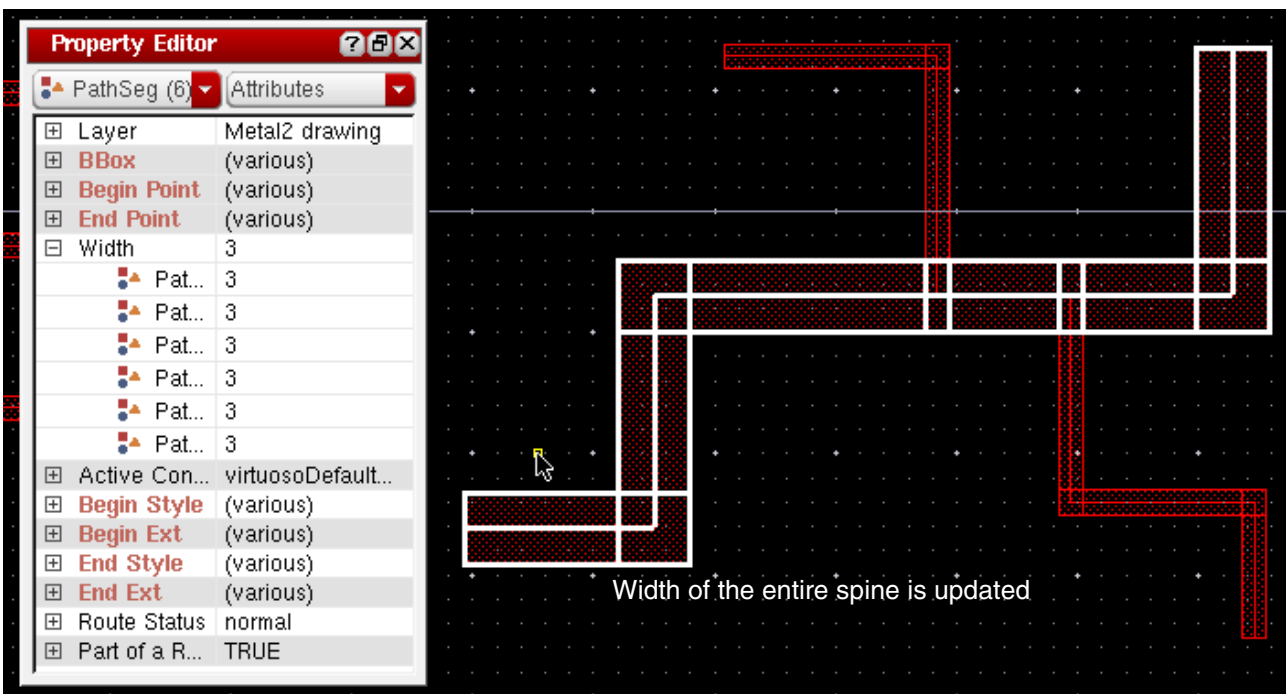
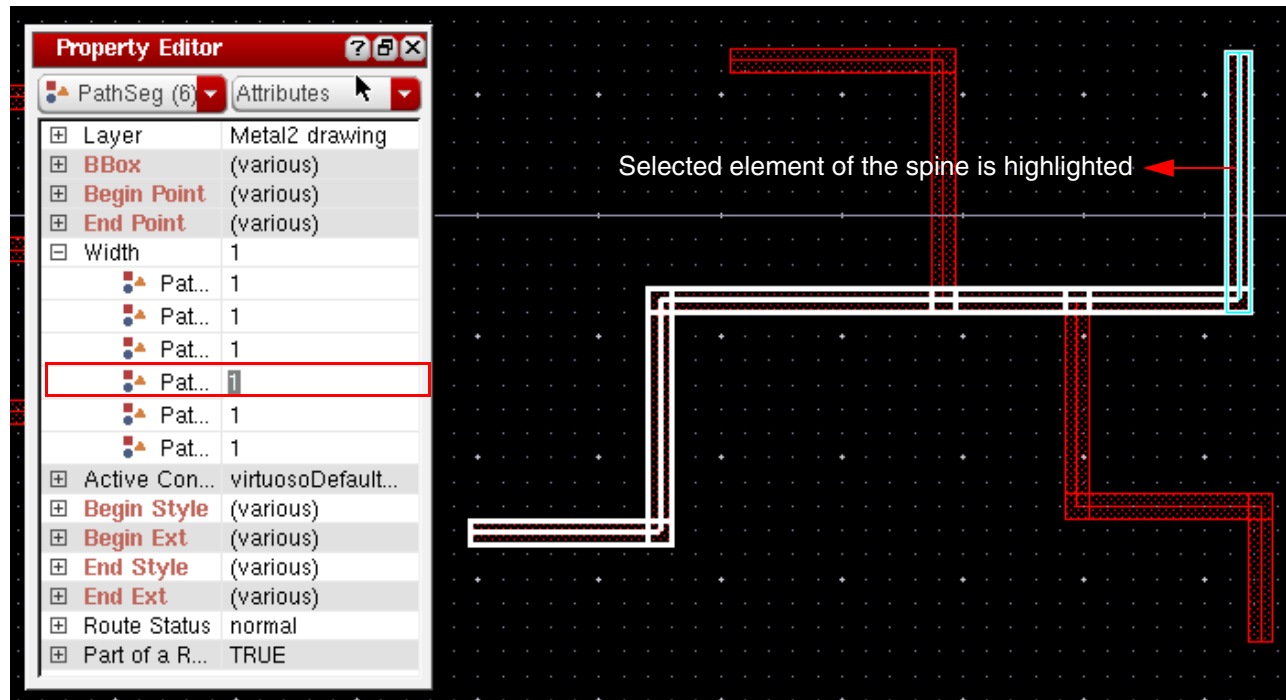
field. The figure also shows how the width of the entire spine is changed when a new value is specified in the *Width* field of the Property Editor assistant



Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

Note that if an entire spine is selected and if the width of one element of the selected spine is updated, the entire spine is updated with the same width value. The following figure shows that by editing the width of one element of the selected spine, the entire spine is updated with the same width value.

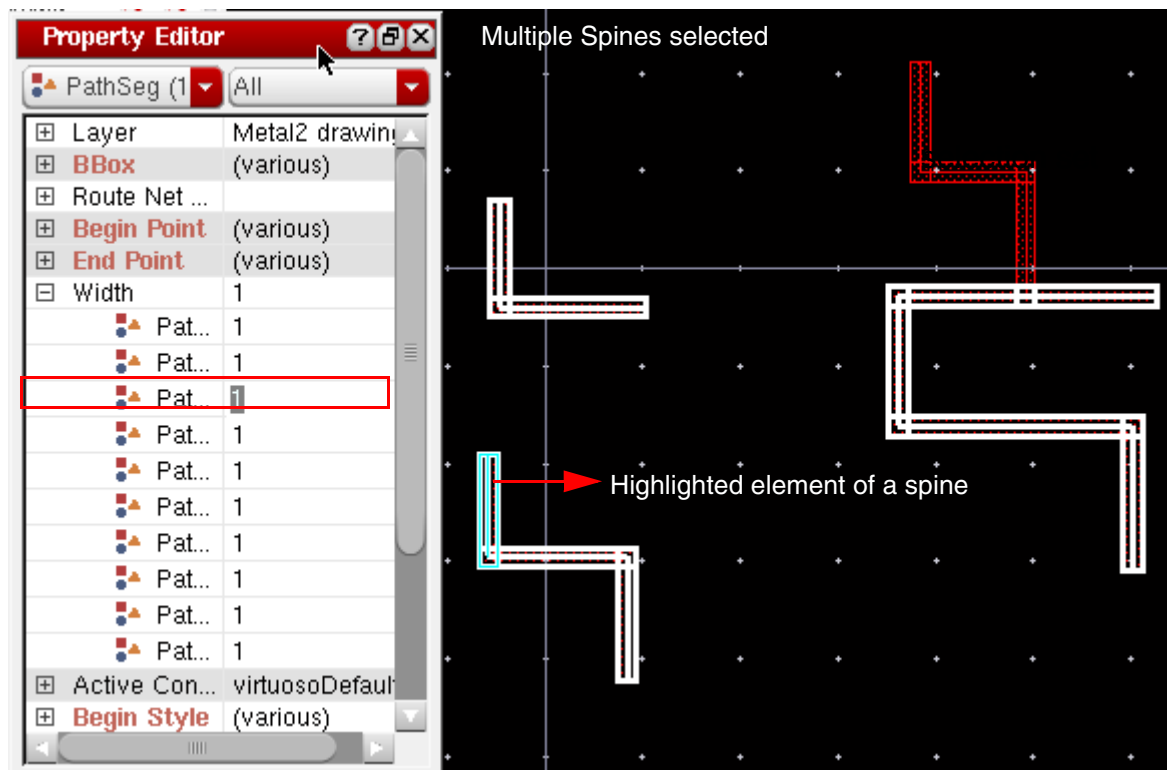


Selecting Multiple Spines

You can edit the width of multiple spines simultaneously or edit the width of a single spine from the selection set.

■ Editing width of a single spine from a selection set

Using the Property Editor assistant, you can edit the width of a single spine from the selection set. To edit the width of a single spine from the selection set, place the pointer on the element of the spine whose width you want to change. The following figure displays how the element of the spine on which the pointer is placed is highlighted in the layout window.

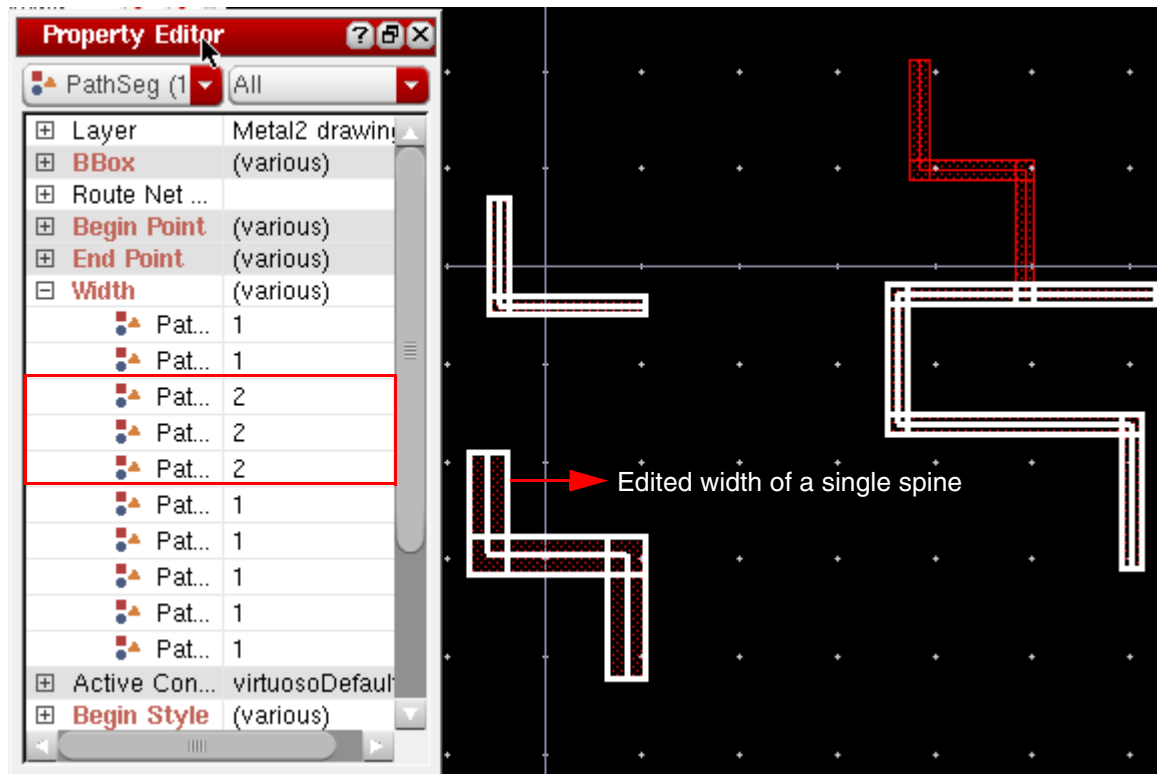


To change the width of the highlighted spine, specify the new width value in the column next to the selected element of the spine in the Property Editor assistant, and press

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

Enter. The following figure displays how the width of the spine, whose element was selected in the Property Editor assistant, has changed.

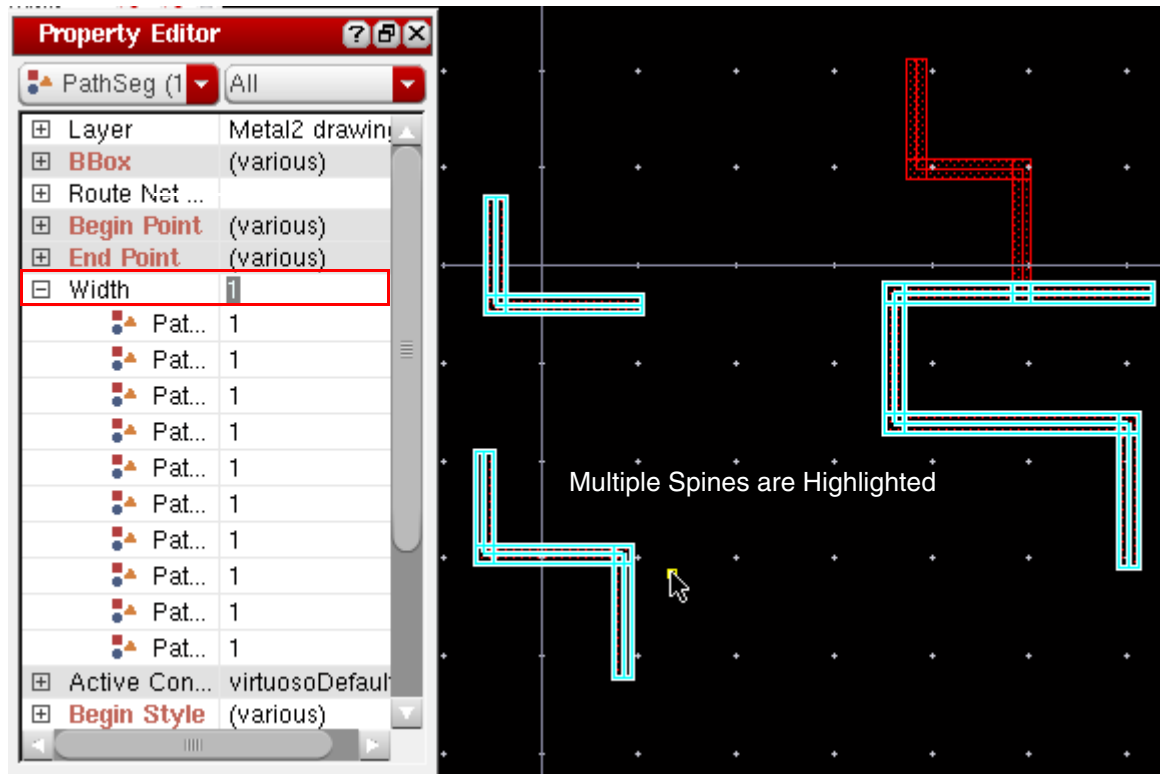


- Editing width of multiple spines simultaneously

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

To edit the width of multiple spines simultaneously, place the pointer on the *Width* field in the Property Editor assistant. All the spines in the selection set are highlighted, as shown in the following figure.

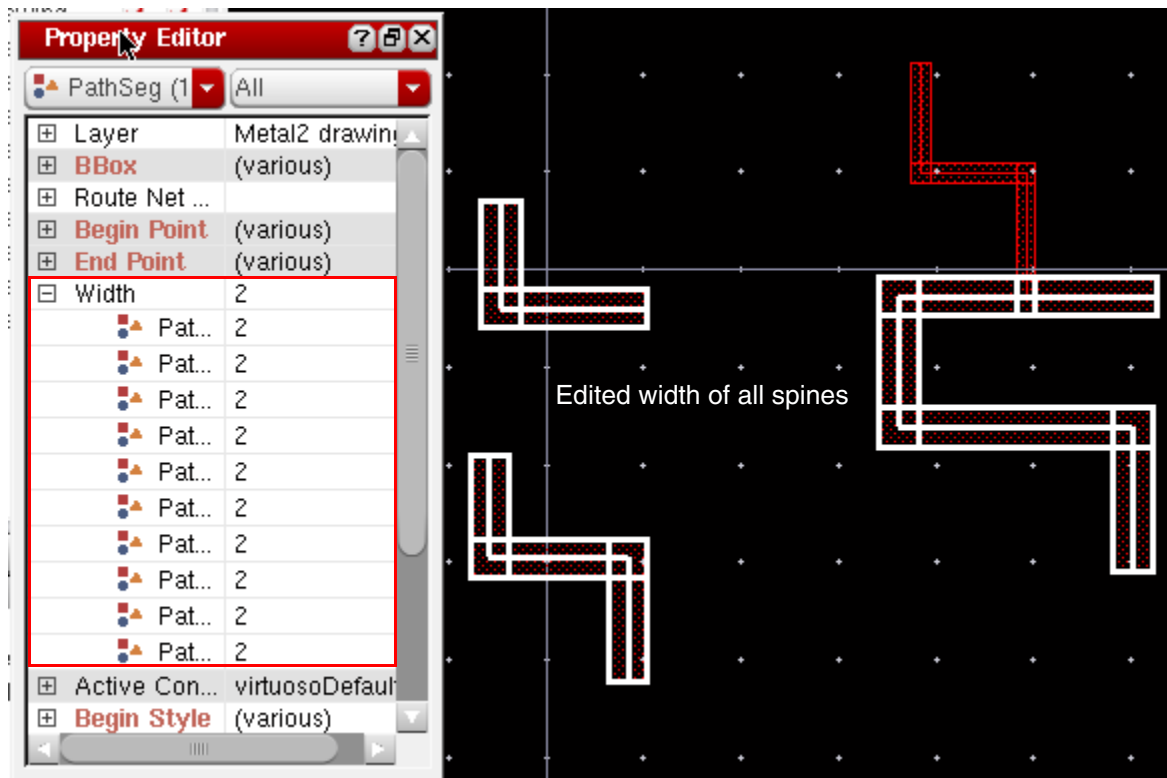


To change the width of all the spines simultaneously, specify the new width value in the column next to the *Width* field of the Property Editor assistant and press **Enter**. The

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

following figure displays how the width of all the selected spines has changed simultaneously.



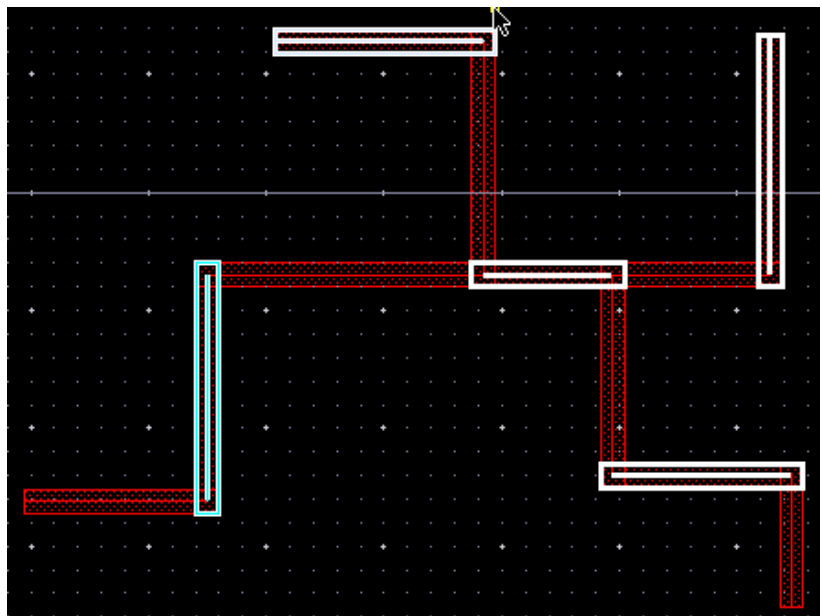
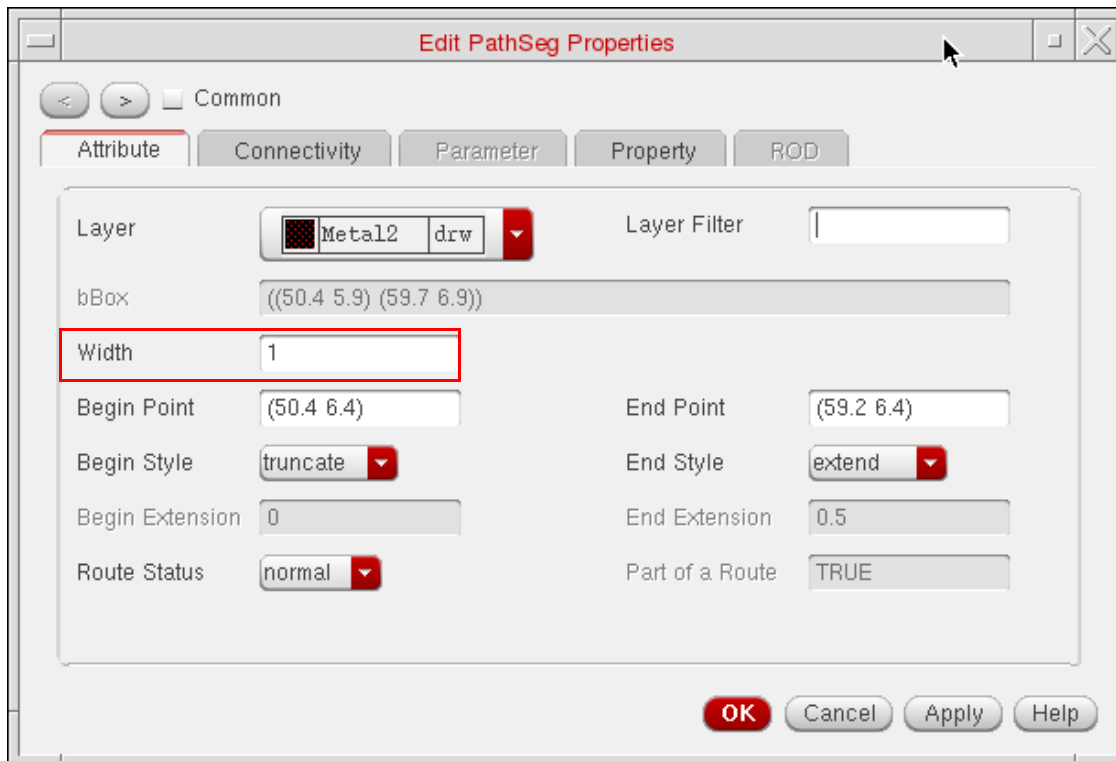
Using the Edit Properties Form

Using the Edit Properties form, you can change the width of a single element, multiple elements, or a spine. To change the width, select the wires or the spine whose width you want to change and then choose *Edit – Basic – Properties*. Alternatively, click the *Edit Properties* icon on the toolbar. The Edit Properties form is displayed. The following figure

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

shows the Edit Properties form along with the multiple wires selected in the layout window when the *Path Spine* mode is OFF.



You can edit the width of a single wire, multiple wires, or a spine by specifying a new width value in the *Width* field of the Edit Properties form.

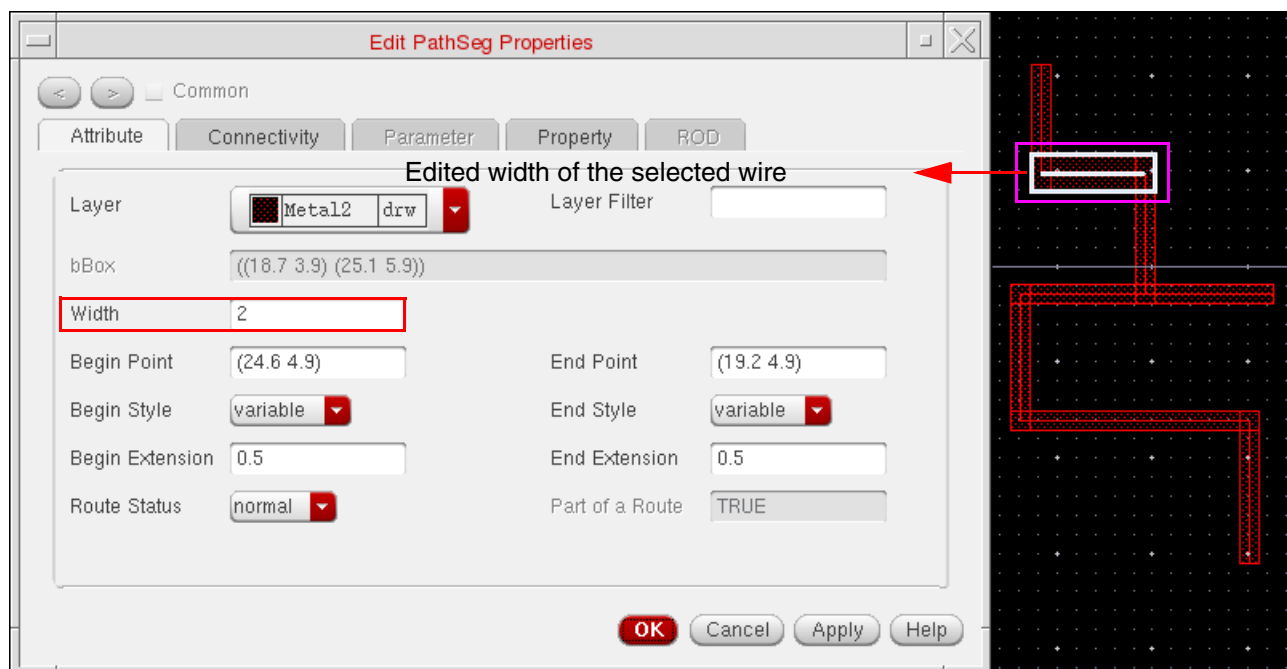
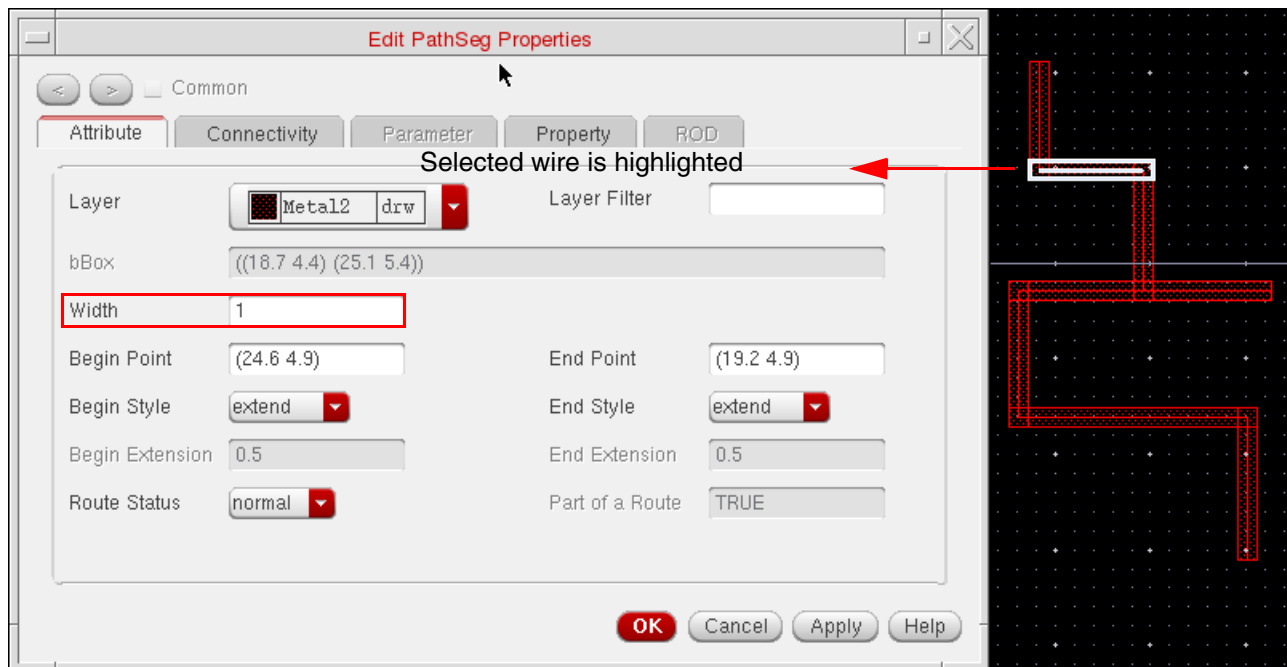
Selecting a Single Wire

When only a single wire is selected, you can edit the width of a wire by specifying a new width in the *Width* field. The following figure shows that the selected wire is highlighted and how

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

the width of the selected wire is changed in the layout canvas when a new width is specified in the *Width* field within the Edit Properties form.



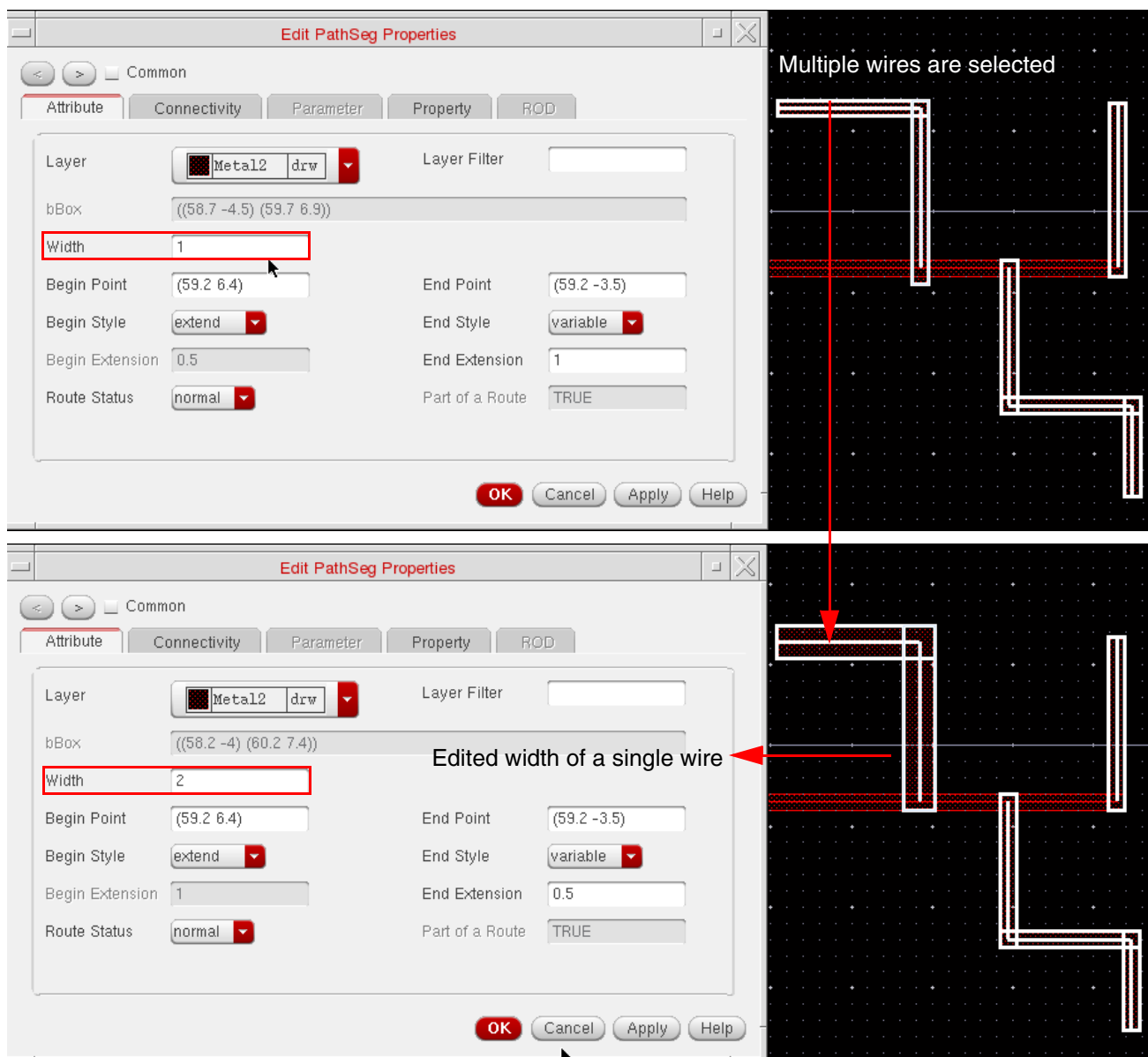
Note: The *Common* option is not applicable when you are editing the width of a single wire.

Selecting Multiple Wires

Using the Edit Properties form, you can either change the width of multiple wires simultaneously or change the width of a single wire from the selection set.

■ Editing Width of a Single Wire From a Selection Set

While editing the width of a single wire from the selection set, the wire that has been first selected in the selection set is highlighted. Specify a new width value in the Width field. The following figure displays the changed width of a wire, which was first selected in the selection set.



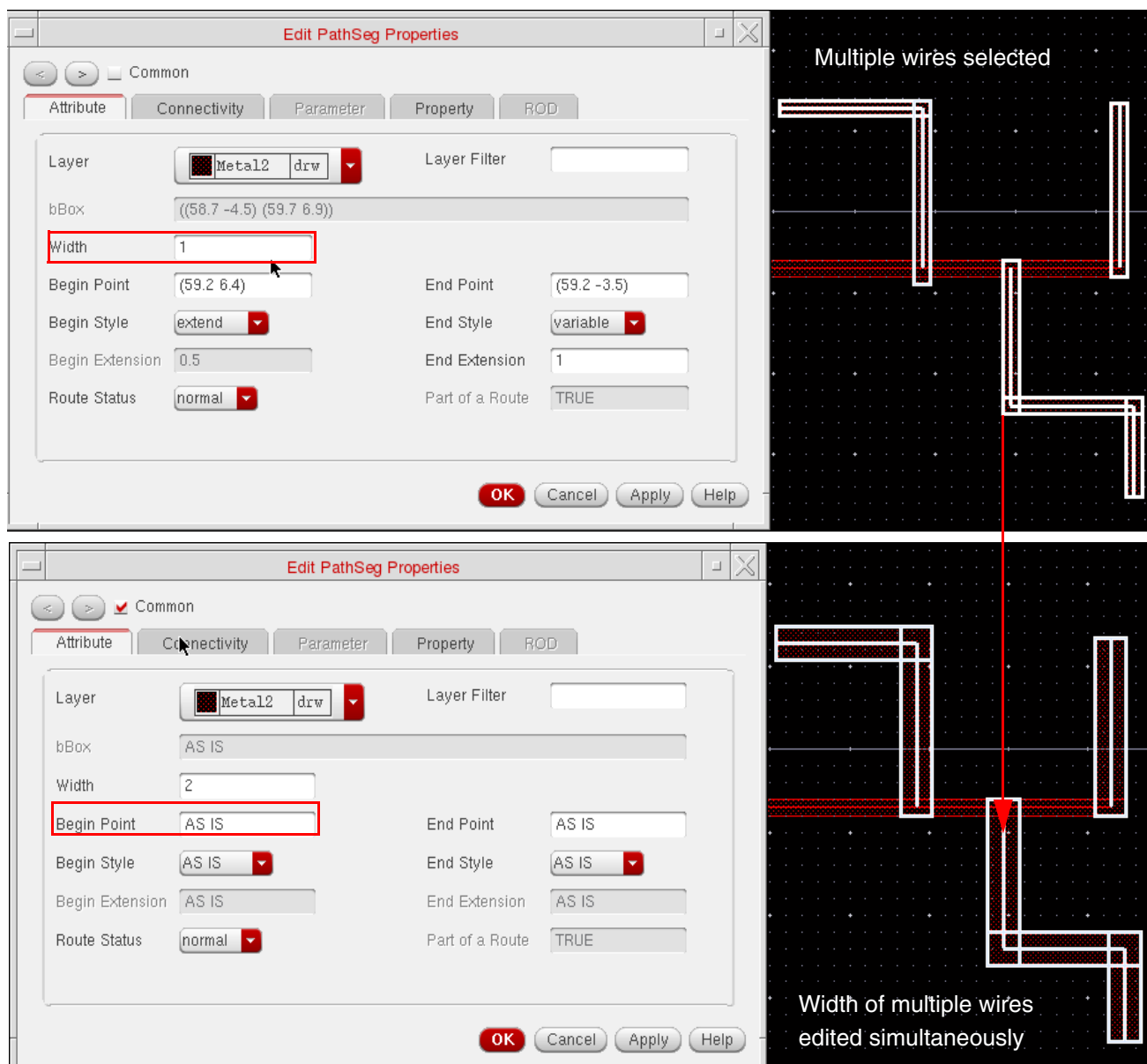
Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

While editing the width of a single wire from the selection set, if you want to change the width of the next or the previous wire in the selection set, you can navigate to the next or the previous wire in the selection set by using the *Next* and *Prev* buttons in the Edit Properties form.

■ Editing the width of multiple wires simultaneously

To edit the width of multiple wires simultaneously using the Edit Properties form, select the *Common* check box at the top of the Edit Properties form. The following figure displays how the width of multiple wires has been changed simultaneously when the *Common* check box is selected.



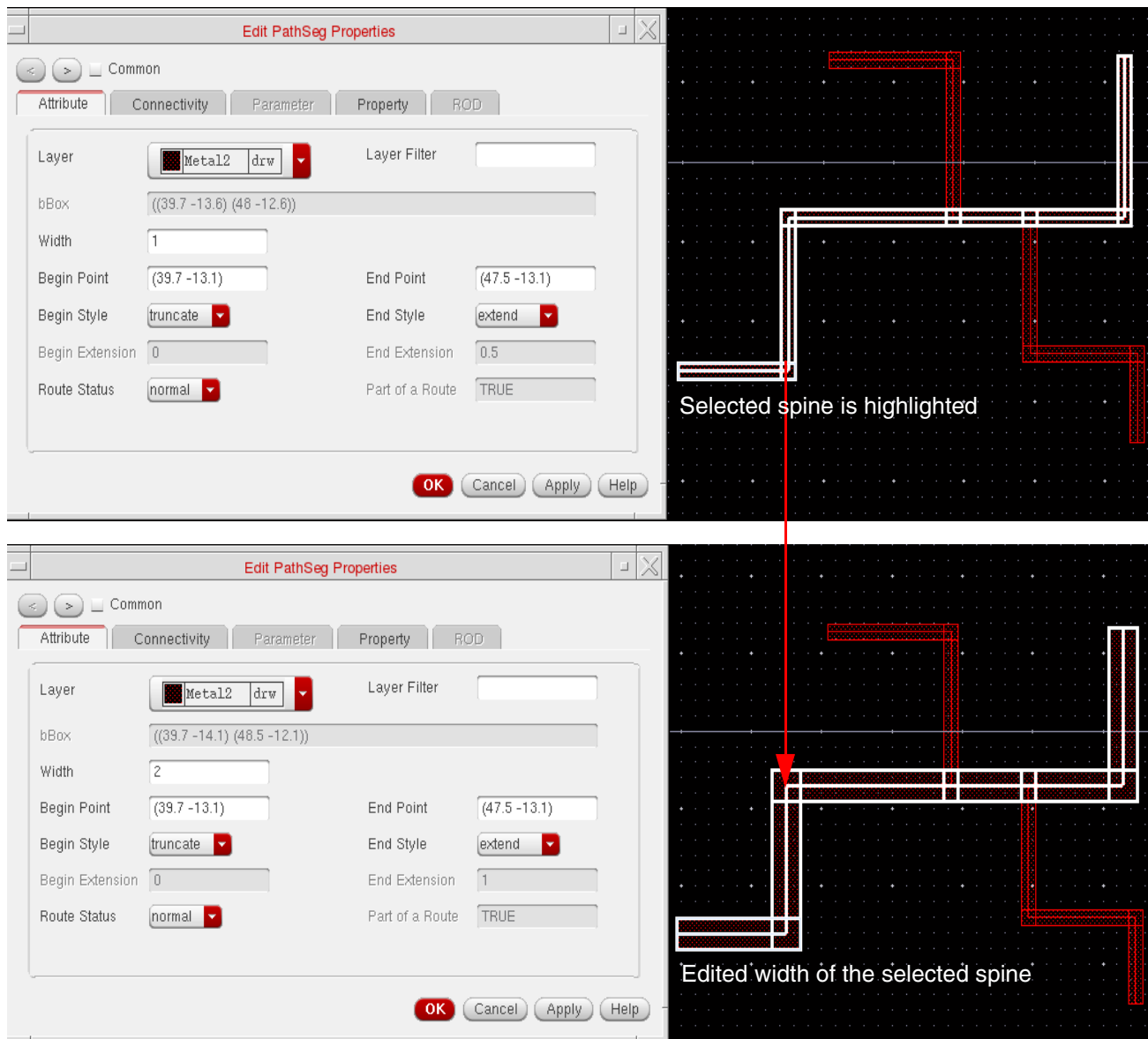
Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

Selecting a Spine

You can also change the width of a spine using the Edit Property form. To change the width of a spine, select a spine from the layout window. The spine can only be selected when the *Path Spine* mode is ON.

You can edit the width of the entire spine by specifying the new width value in the *Width* field within the Edit Property form. The following figure displays how the width of the entire spine has been edited when you specified the new value in the *Width* field.



Note: The *Common* option in the Edit Properties form has no effect when the width of a single spine is edited.

Selecting Multiple Spines

Using the Edit Properties form, you can either change the width of multiple spines simultaneously or change the width of a single spine from the selection set.

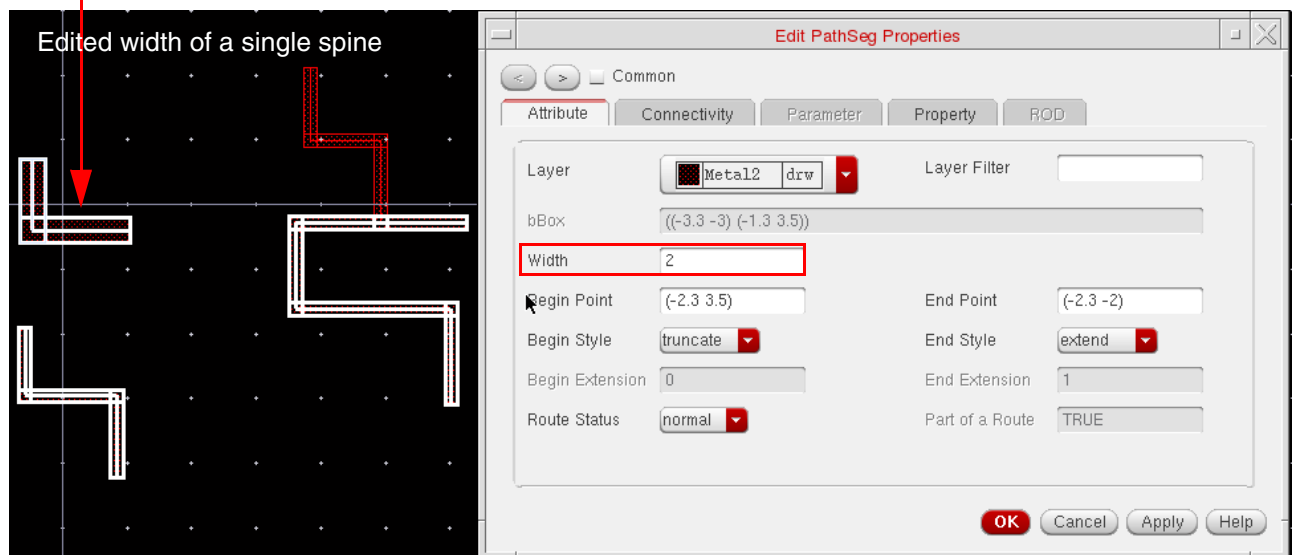
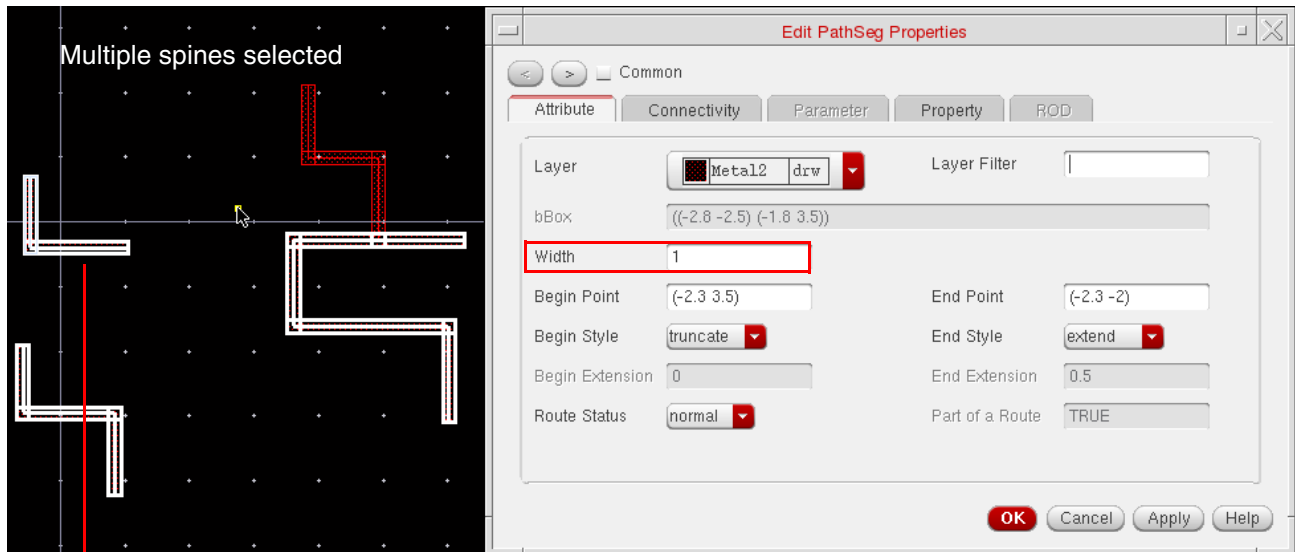
- Editing the width of a single spine from a selection set

While editing the width of a single spine from the selection set, the spine that has been first selected in the selection set is highlighted. Specify a new width value in the *Width*

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

field. The following figure displays the changed width of a spine, which was first selected in the selection set.



Note: While editing the width of a single spine from the selection set, the *Common* check box in the Edit Properties form is deselected.

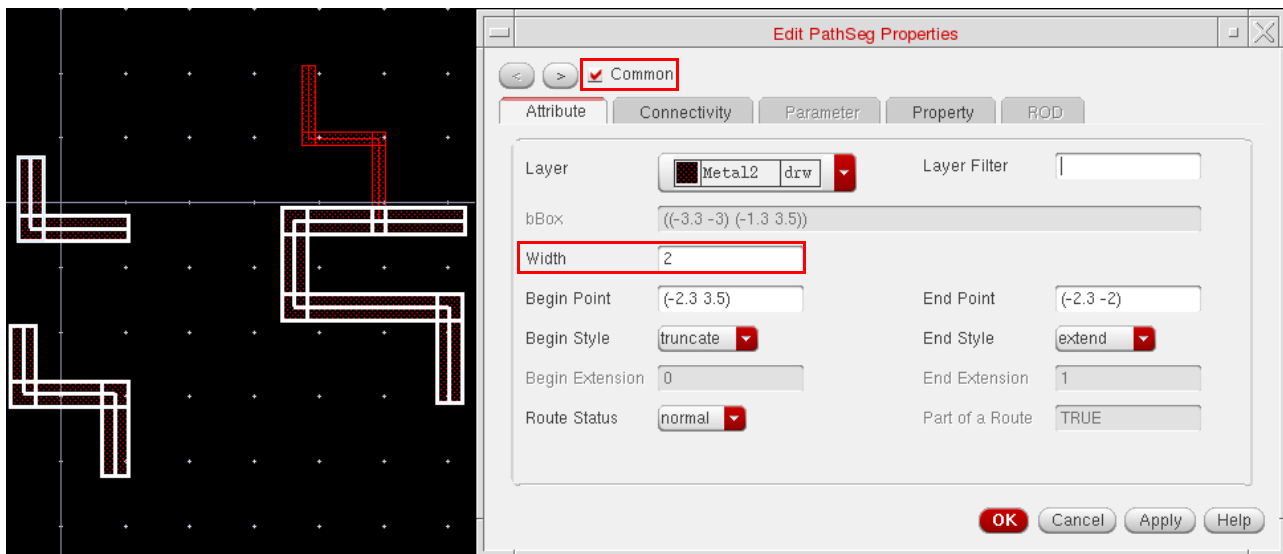
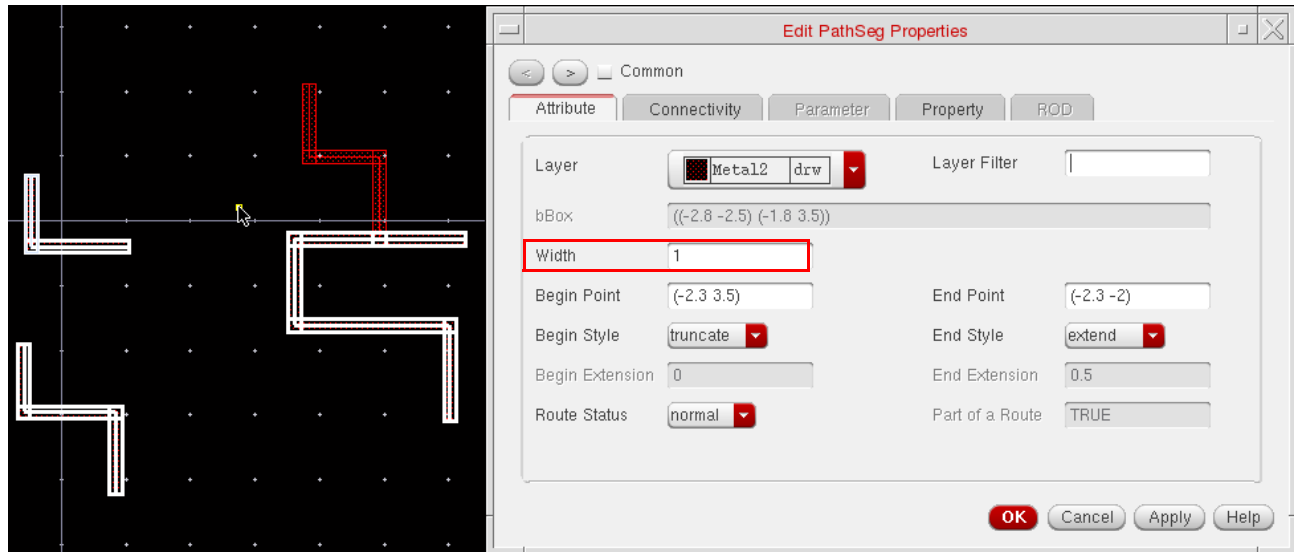
If you want to change the width of the next or the previous spine in the selection set, you can navigate to the next or the previous spine by using the *Next* and *Prev* buttons in the Edit Properties form.

■ Editing the width of multiple spines simultaneously

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

To edit the width of multiple spines simultaneously using the Edit Properties form, specify a new value in the Width field and select the *Common* check box at the top. The following figure displays how the width of multiple spines has been changed simultaneously when the *Common* check box is selected.



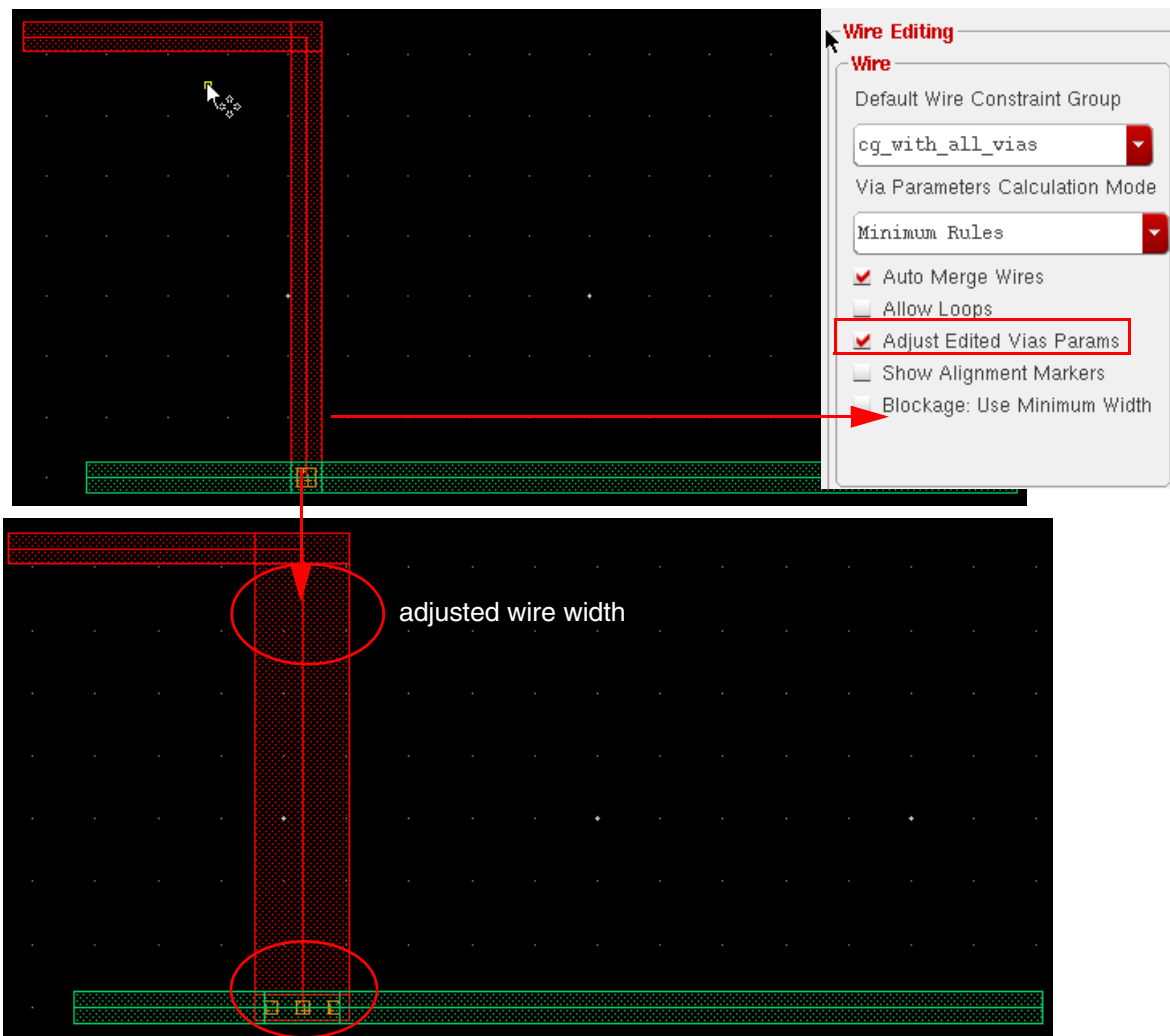
Adjusting Via While Editing Wire Width

The parameters of the vias that are connected to a segment's begin and end points can be adjusted when the width of a segment is changed. The parameters are updated only if it is a standard via. The steps to adjust the via parameters connected to a segment while editing its width are as follows.

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

1. Select a wire with via parameters.
2. Select the *Adjust Edited Vias Params* check box in the *Wire Editing* group box in the Layout Editor Options form.
3. Specify the new width value for the selected wire in the *Width* field.
4. Click *OK*. The following figure displays the via adjustment when the width of a wire is edited.



Editing Layer-Purpose Pair

You can change the layer-purpose pair of a single segment or of several segments simultaneously. Consider a specific segment whose layer-purpose pair is being changed: if

all the segments of the spine to which this segment belongs to are selected, then changing the layer-purpose pair of this specific segment changes the layer-purpose pair of all the segments to the same layer-purpose pair. Else, the layer-purpose pair of this segment is only changed.

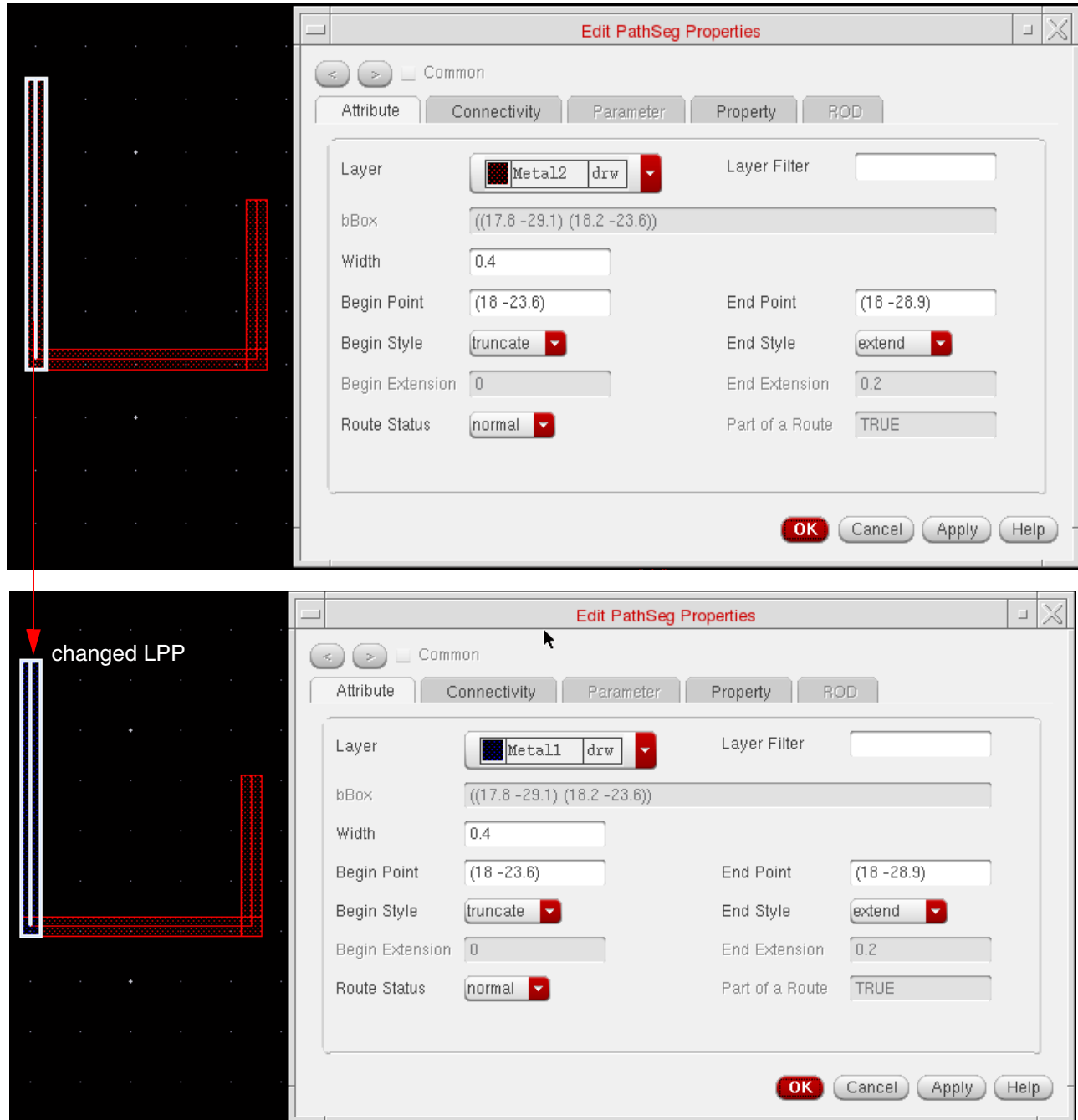
Changing the Layer-purpose Pair of a Single Wire

You can change the layer-purpose pair of a wire using the Edit Properties form or the Object Property assistant. To change the layer-purpose pair, select the layer-purpose pair for the

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

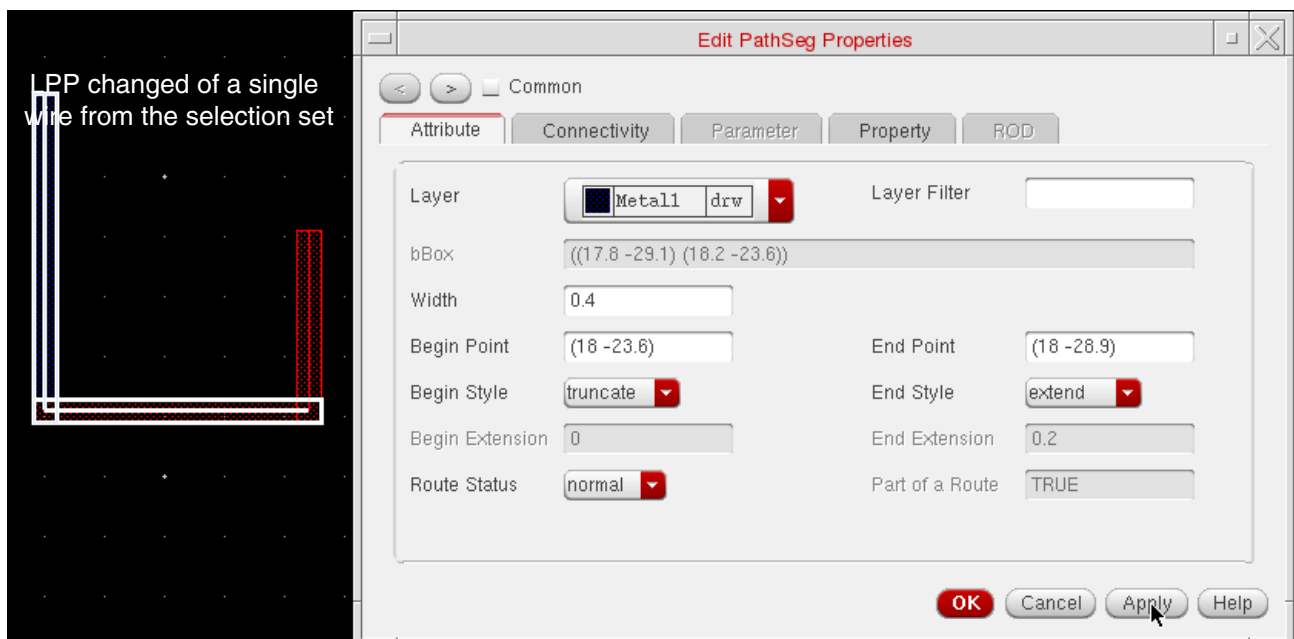
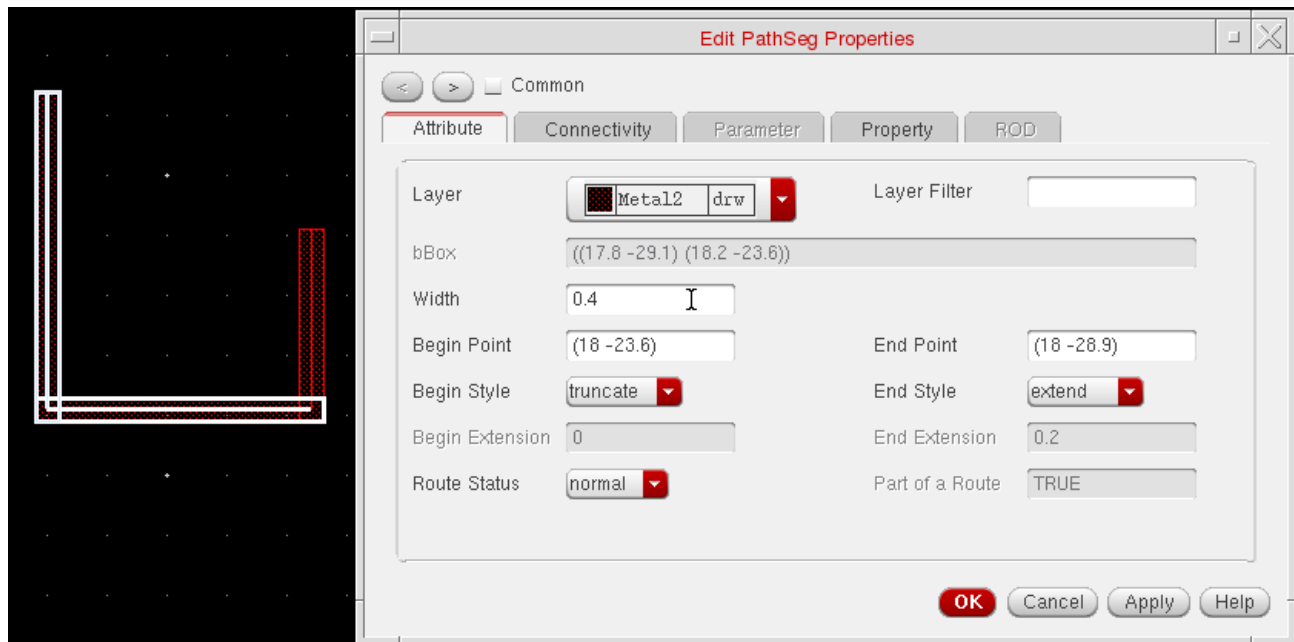
selected wire from the *Layer* drop-down box in the Edit Properties form. The following figure displays the changed layer-purpose pair for the selected wire.



Changing the Layer-Purpose Pair of Multiple Wires

Editing Layer-Purpose Pair of a Single Wire

You can also change the layer-purpose pair of a single wire from the selection set. The following figure displays the changed layer-purpose pair of a wire, which was first selected in the selection set.

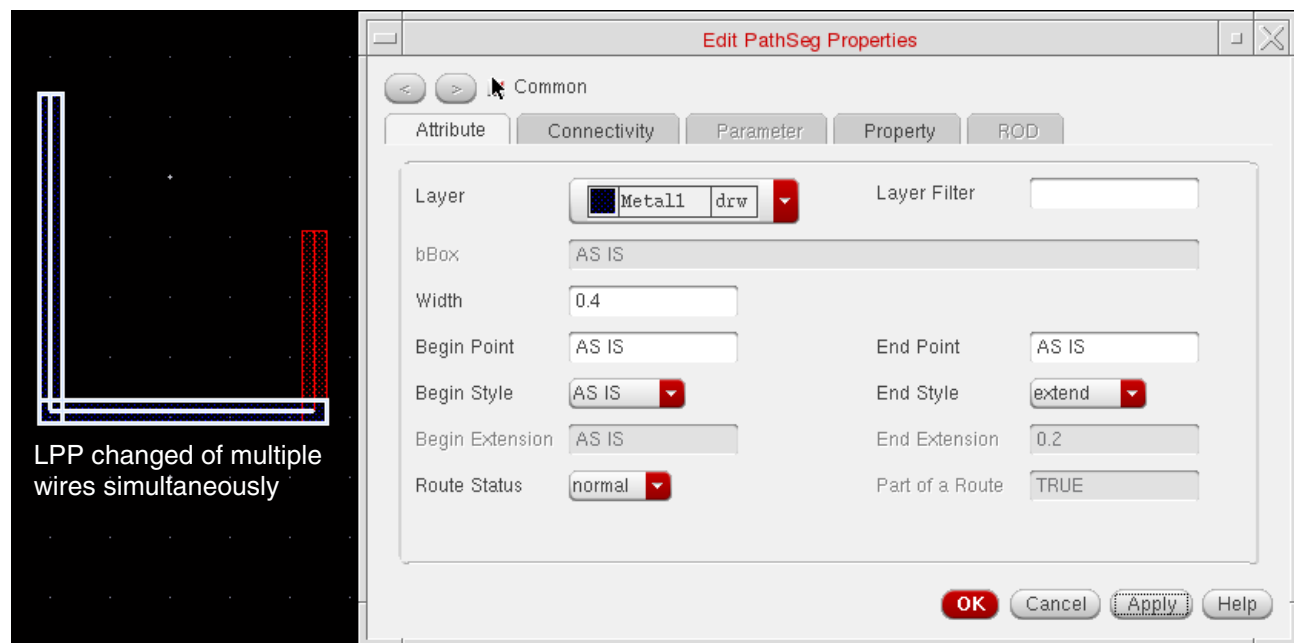
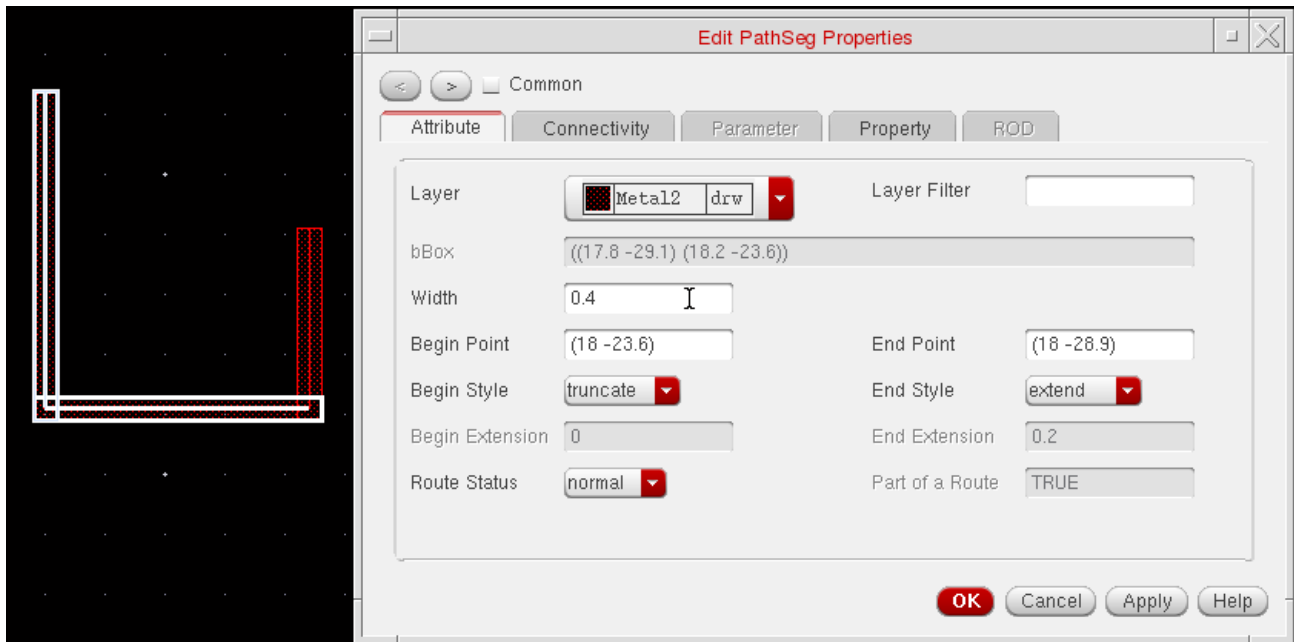


Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

Editing Layer-Purpose Pair of Multiple Wires Simultaneously

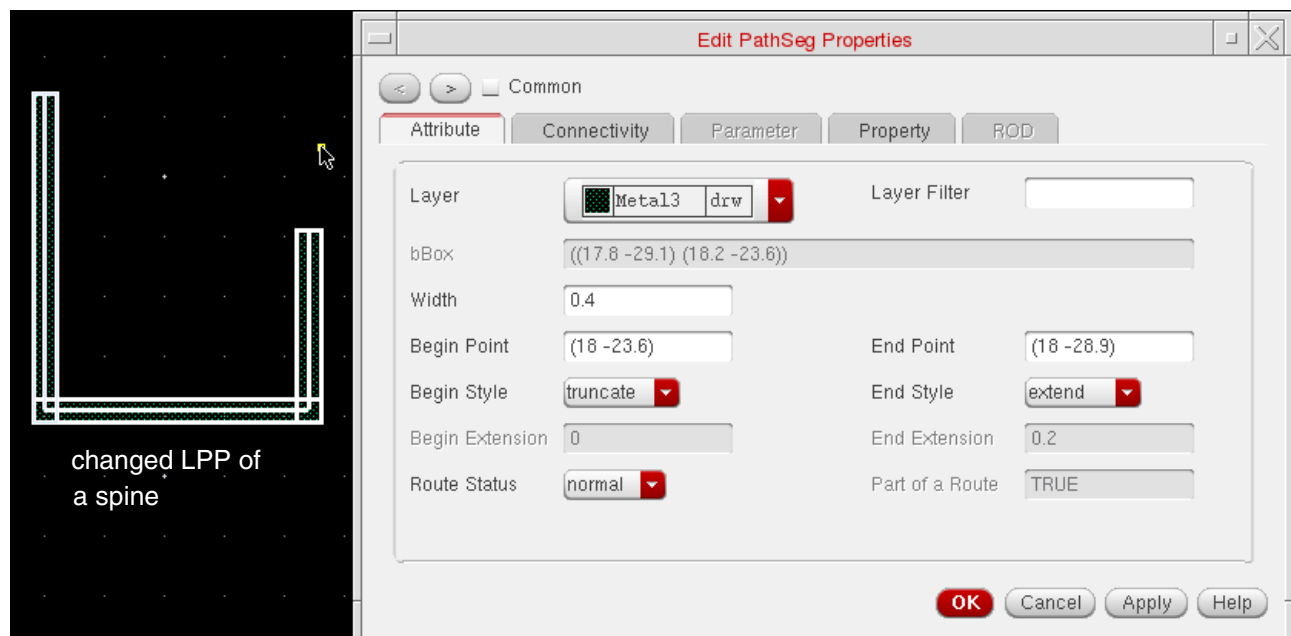
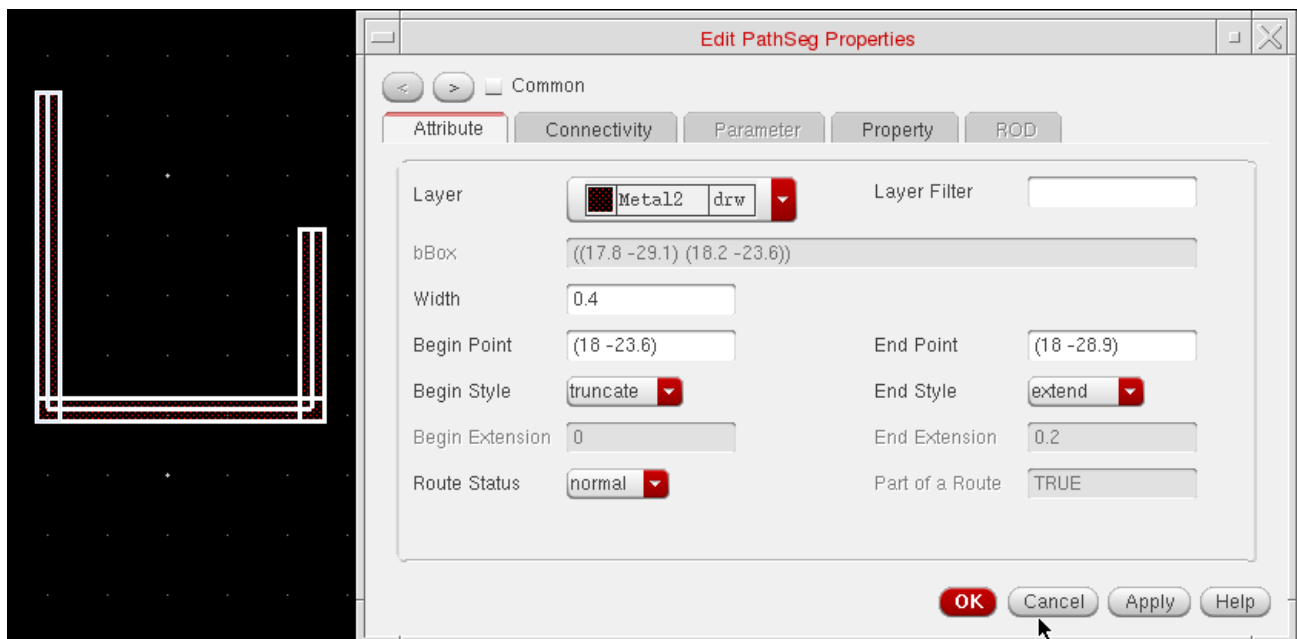
To edit the layer-purpose pair of multiple wires simultaneously using the Edit Properties form, select the *Common* check box on top of the Edit Properties form. The following figure displays how the layer-purpose pair of multiple wires is changed simultaneously when the *Common* check box is selected.



Changing the Layer-Purpose Pair of a Spine

To change the layer-purpose pair of a spine, select a spine and then select the layer-purpose pair for the selected wire from the *Layer* drop-down box in the Edit Properties form. The following figure displays the changed layer-purpose pair for the selected wire.

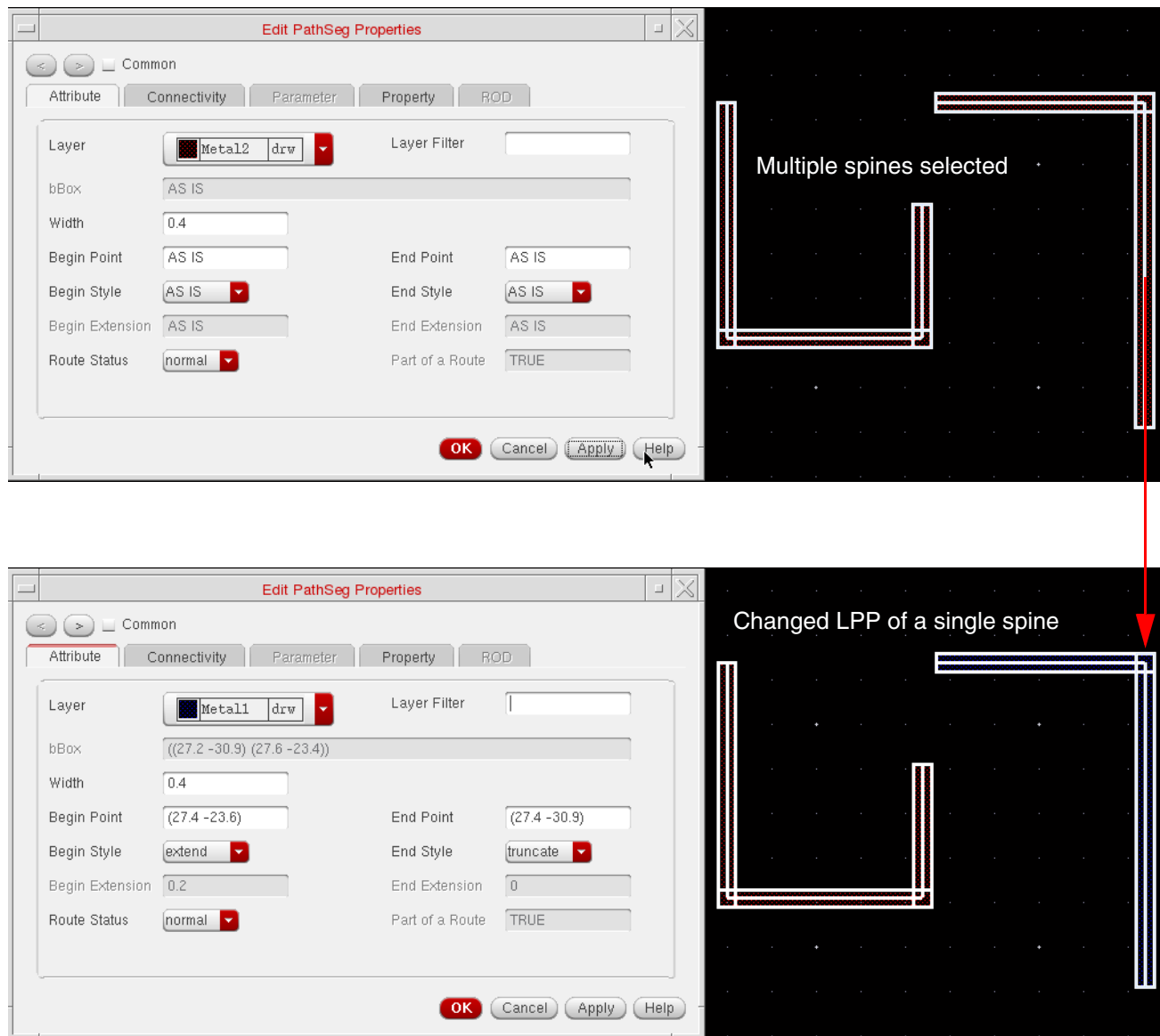
Note: For changing the layer-purpose pair of a spine, you do not need to have the *Common* check box selected.



Changing the Layer-Purpose Pair of Multiple Spines

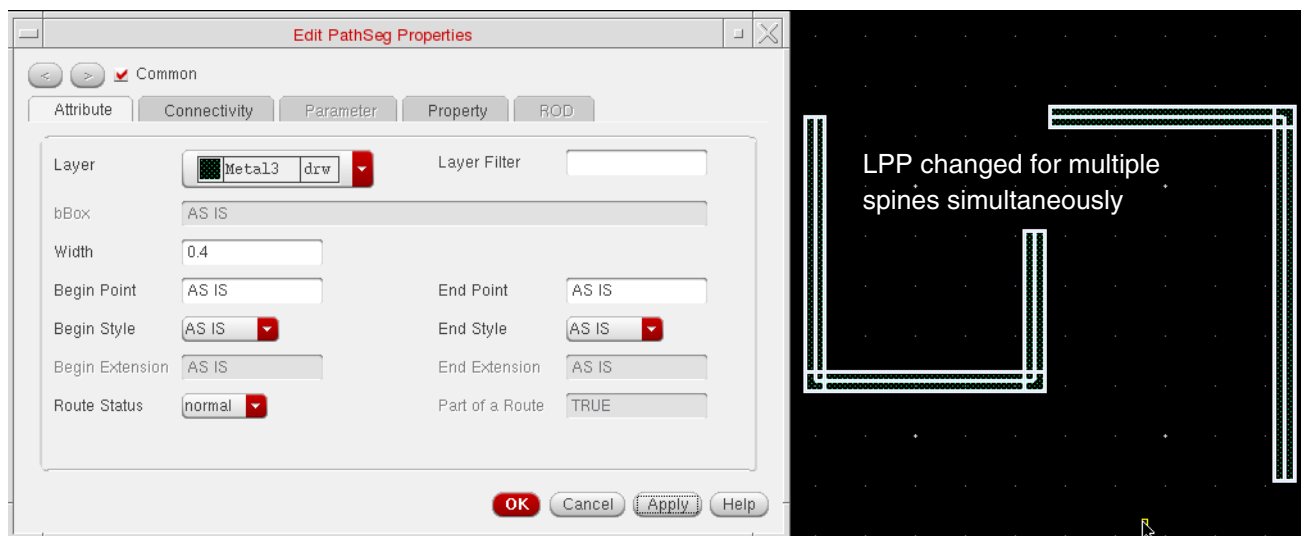
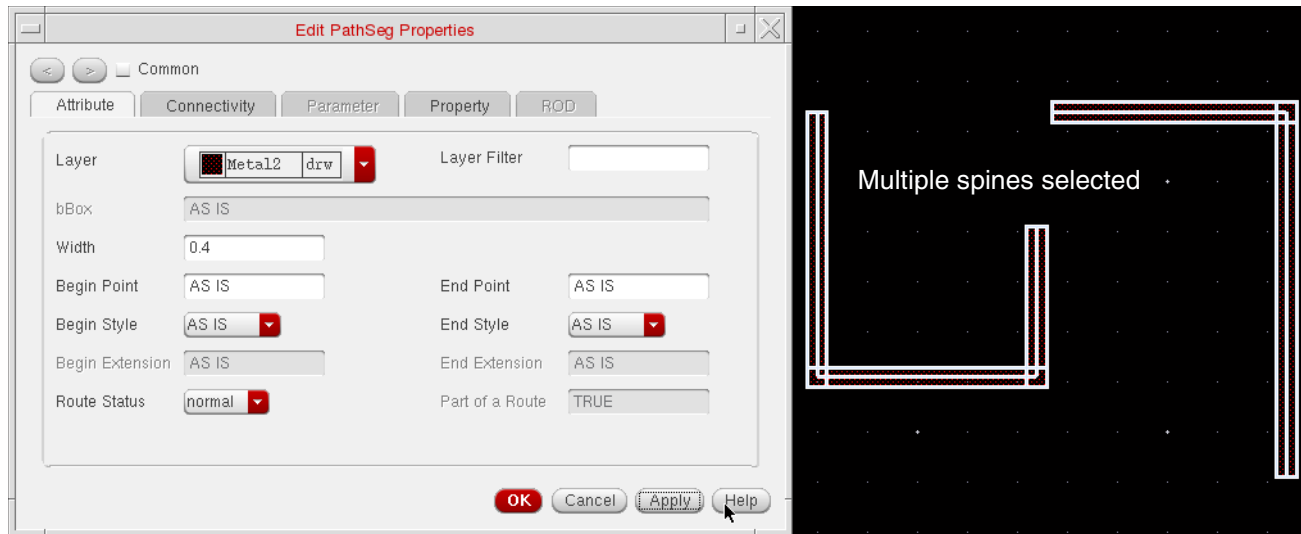
Changing Layer-Purpose Pair of a Single Spine

You can also change the layer-purpose pair of a single spine from the selection set. The following figure displays the changed layer-purpose pair of a spine, which was first selected in the selection set.



Changing LPP of Multiple Spines Simultaneously

To edit the layer-purpose pair of multiple spines simultaneously using the Edit Properties form, select the *Common* check box at the top of the Edit Properties form. The following figure displays how the layer-purpose pair of multiple spines is changed simultaneously when the *Common* check box is selected.



Adjusting Vias While Editing Layer-Purpose Pair

You can change the layer-purpose pair of a pathSeg. While changing the layer-purpose pair of a pathSeg, the via on the pathSeg automatically gets adjusted. Depending upon the

change in layer-purpose pair of a pathSeg, the vias are either added or deleted such that the resulting geometries are still physically connected to each other.

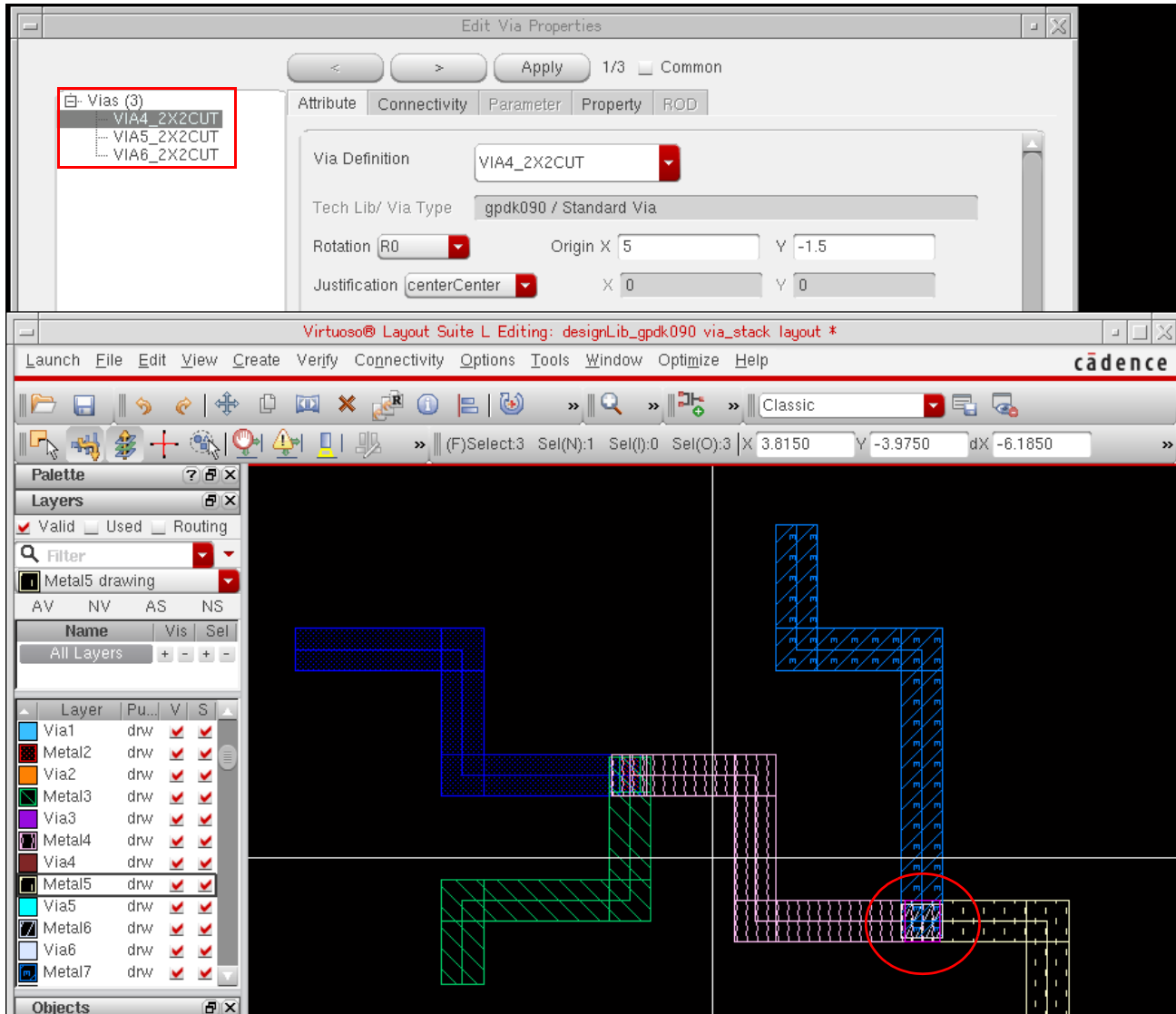
The steps to adjust vias while changing the layer-purpose pair are as follows.

1. Select a wire with via.
2. Right-click the selected wire. The context-sensitive menu is displayed.
3. From the context-sensitive menu, select *Properties*. The Edit PathSeg in Route Properties form is displayed.
4. In the Edit PathSeg in Route Properties form, click the *Layer* drop-down list and select a layer-purpose pair.
5. Click *OK*.

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

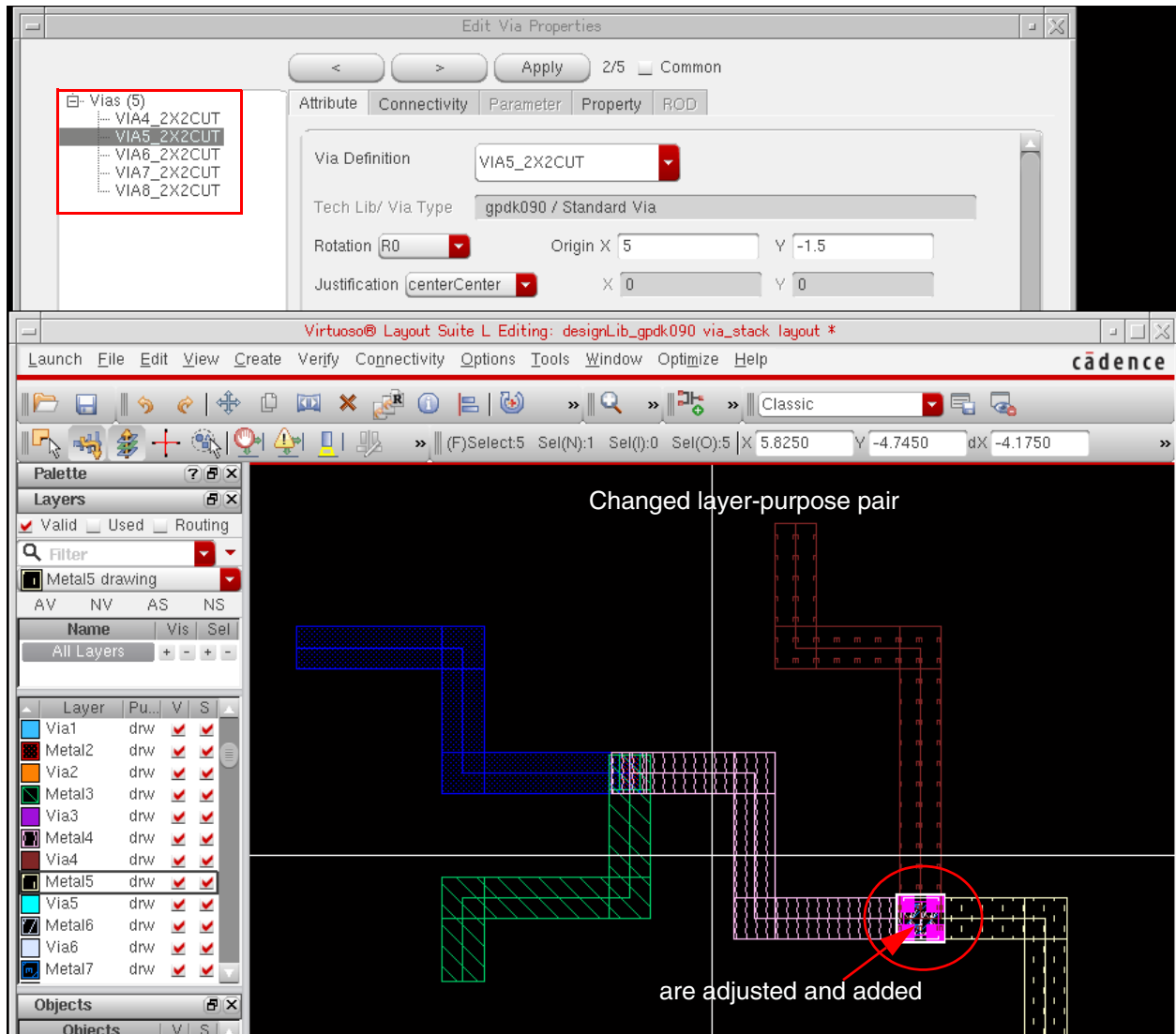
Let us consider an example, where the vias exists between Metal 4 drw and Metal17 drw, as shown in the following figure.



Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

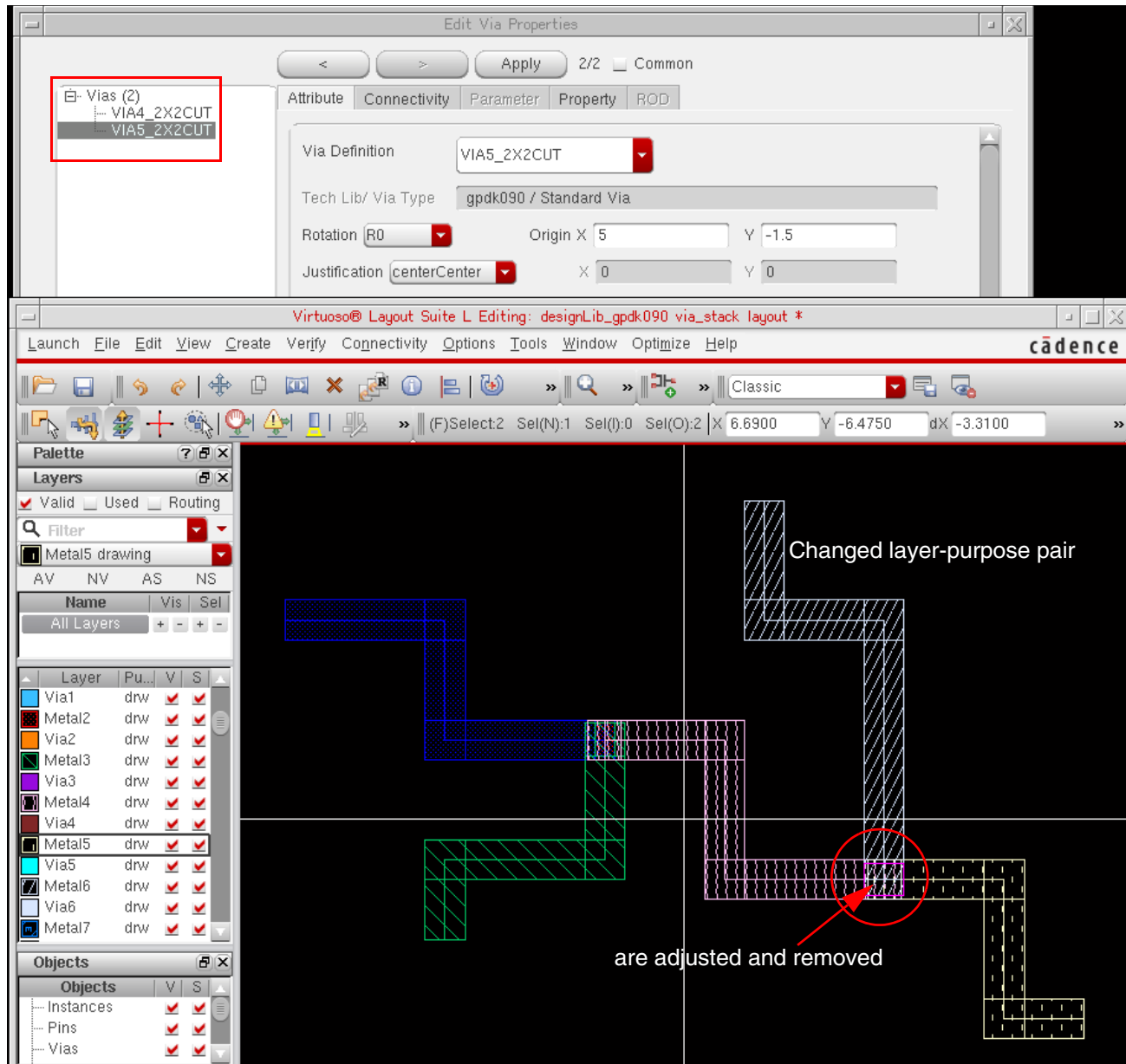
When the layer-purpose pair of a wire is changed from Metal17 drw to Metal19 drw, the vias are automatically adjusted, as shown in the following figure. In this scenario, the vias are added when the layer-purpose pair is changed.



Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

Now, when we again change the layer-purpose pair of a wire from Metal19 drw to Metal16 drw, the vias are again adjusted automatically. In this scenario, vias are automatically removed when the layer-purpose pair is changed.



Splitting Wires and Vias

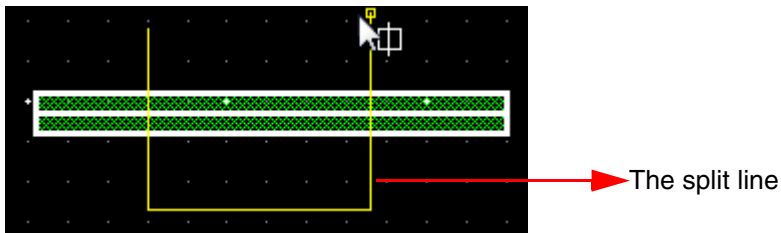
When a wire is selected, the *Split* command lets you split and stretch a portion of a wire.

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

To split and stretch a wire:

1. Select the wire you want to split.
2. Choose *Edit – Advanced – Split* [Control+S]. Alternatively, you can open the Split Form by pressing F3. The Split form opens.
3. Click on the points of the split line.

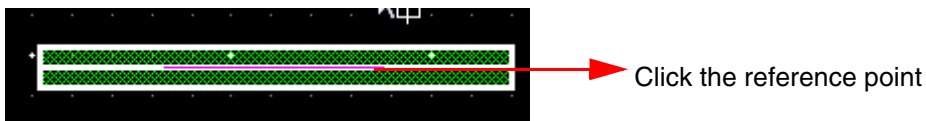


Note: Here the mode is set to *No wires and vias*.

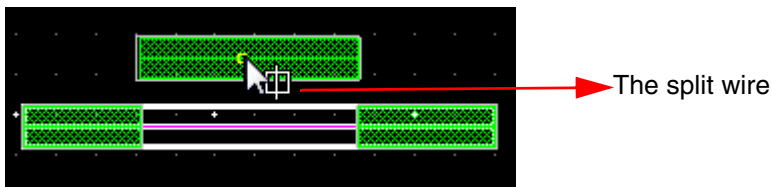
Use the middle mouse button to toggle between the split segments. In the above example, the split alternately highlights the wire section interior to the split lines, and the two segments of the wire exterior to the split lines.

When the split line is drawn clockwise, the split command selects the parts of the wire outside the split lines first. When the split shape is drawn counterclockwise, the split command selects the parts of the wire interior to the split lines first. In the case of a single split line, toggling with the right mouse button alternates between the sides of the object divided by the split line.

4. Click on the reference point for the stretch.



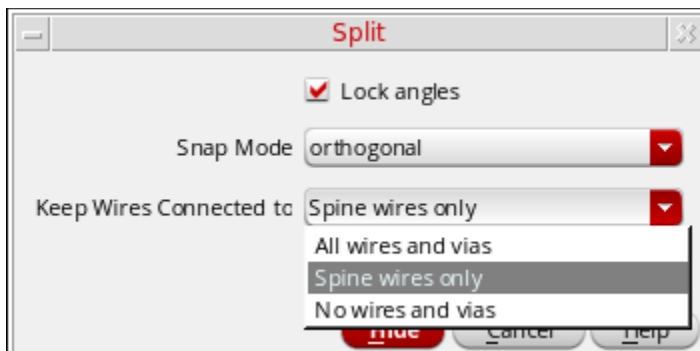
5. Click the new location to split the wire.



The above figure shows the split wire in no connection mode. You can maintain the connection when the wires and vias are split.

Using Keep Wires Connected to Split Command

When a set of wires and vias are connected to each other, and when a wire and via are selected and split, they are not connected any longer. To maintain the connection when the wires and vias are split, use the *Keep Wires Connected To* option of the Split form. The three options that you can use to maintain the connection even when wires and vias split are shown in the following figure.



- **Spine wires only** maintains connections to spine wires but not to connecting vias when splitting a spine. This option is selected by default. For information about selecting a spine, see [Selecting a Path Vertex](#).
- **All wires and vias** maintains pathSeg-to-pathSeg and pathSeg-to-via connections during a split. When this option is selected, the *Lock Unselected Vias* option is enabled. Selecting the *Lock Unselected Vias* option, locks the unselected vias and does not allow you to change the position of the via.
- **No wires and vias** does not maintain pathSeg-to-pathSeg and pathSeg-to-via connections during a split.

You can split wires and vias when the spine selection mode is ON, which is the default mode, and when it is OFF. The behavior of splitting wires and vias is different in both the situations.

Using the Spine wires only Option

To maintain connections to spine wires but not to connecting vias when splitting a spine, select the *Spine wires only* option from the *Keep Wires Connected to* drop-down list in the Split form.

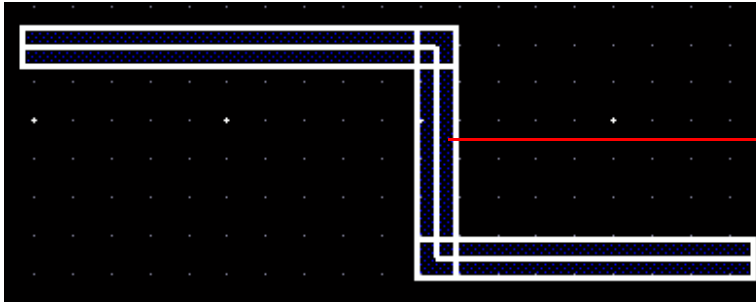
When Spine Selection Mode is ON

- When vertical and horizontal spine wires are selected as a part of a split line

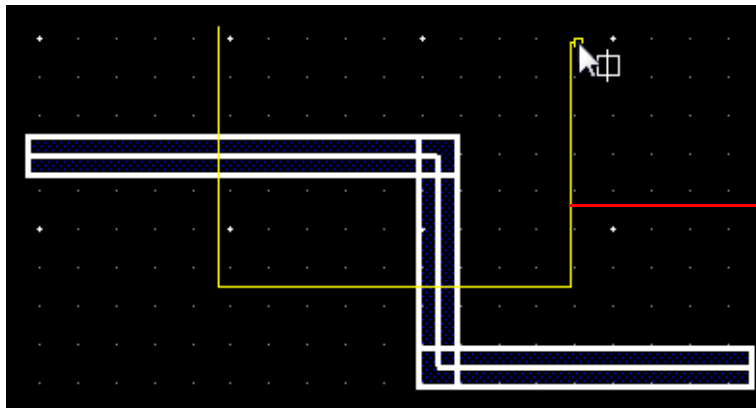
Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

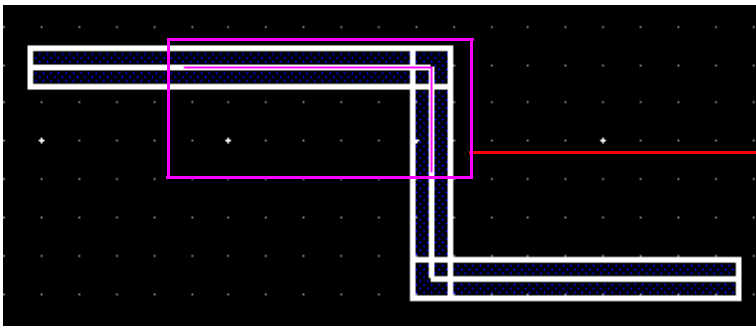
The following figures illustrate splitting and stretching of spine with Spine selection mode ON.



The selected spine when spine selection mode is ON



The split line

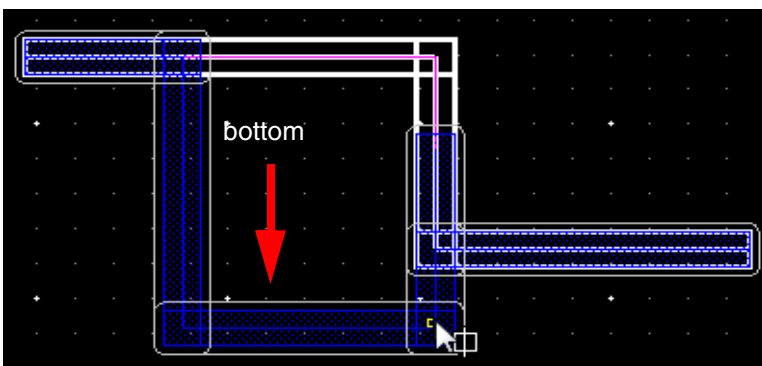
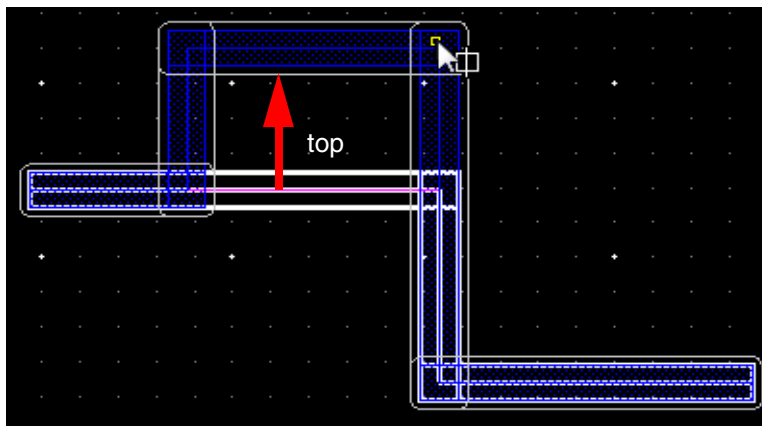
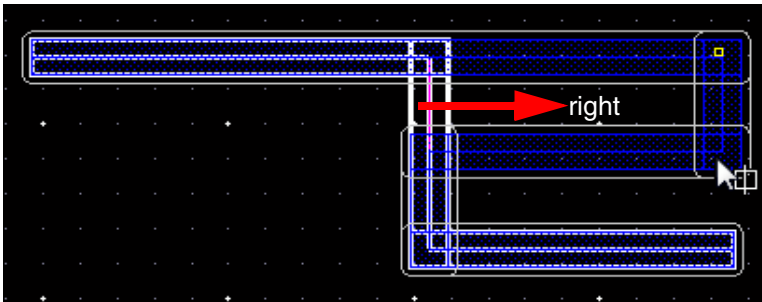
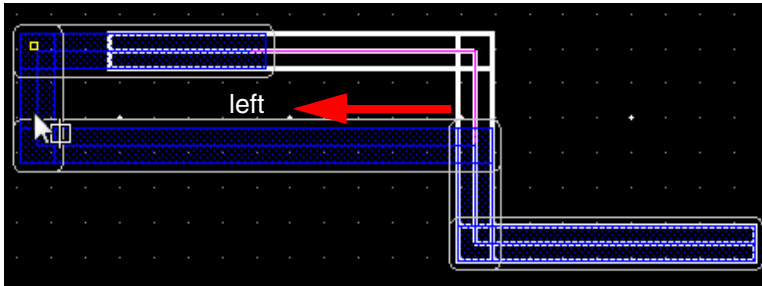


The highlighted portion will be split

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

The following figures illustrate how the spine is split when it is stretched along the left, right, top, and bottom directions.

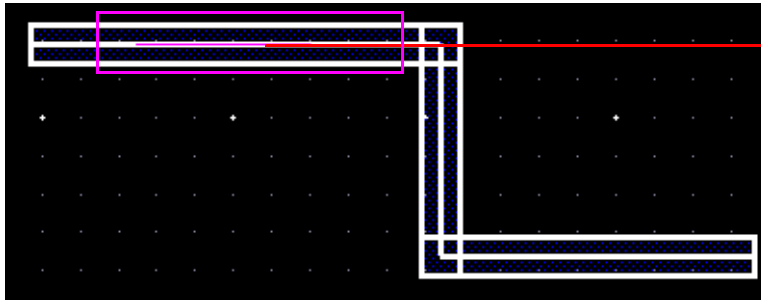


Virtuoso Interactive and Assisted Routing User Guide

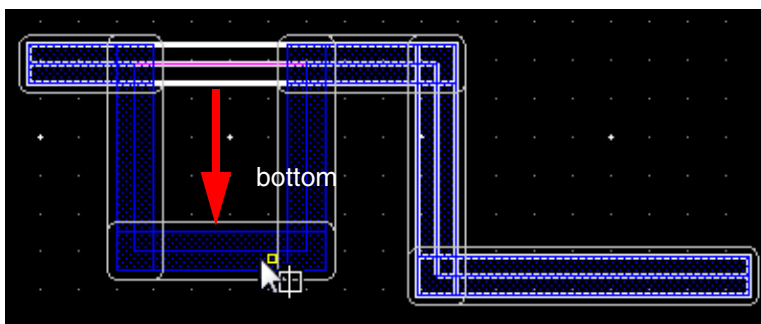
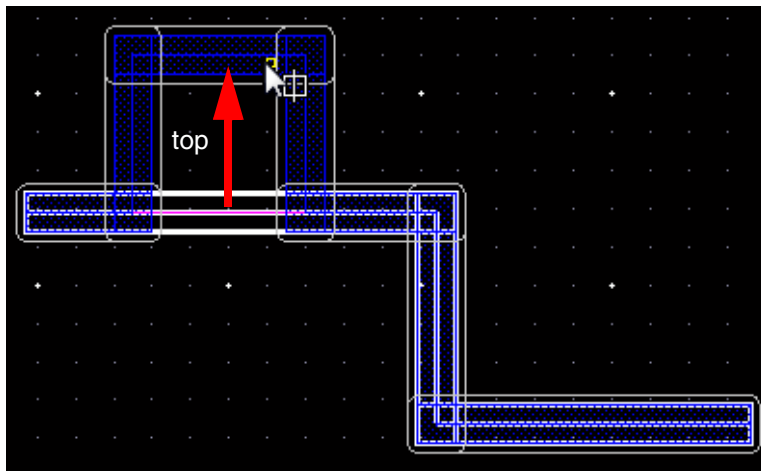
Working with Wires

- When the vertical spine wire is not selected as a part of a split line

Consider the following scenario where the vertical spine wire is not selected as a part of a split line. In this case, you are not able to split the spine in the left and the right directions. The split can only happen along the top and the bottom directions. The following figures illustrate the spine split only in the top and bottom directions.



The highlighted portion is split only along the top and bottom directions

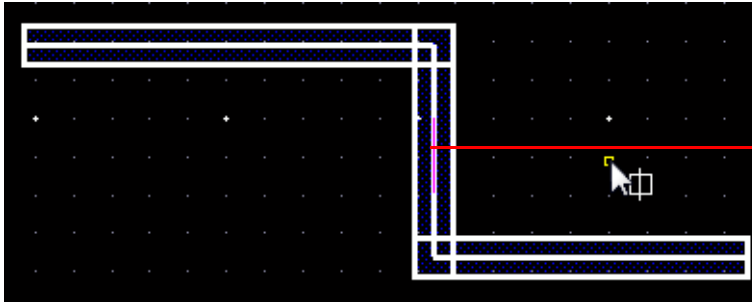


Virtuoso Interactive and Assisted Routing User Guide

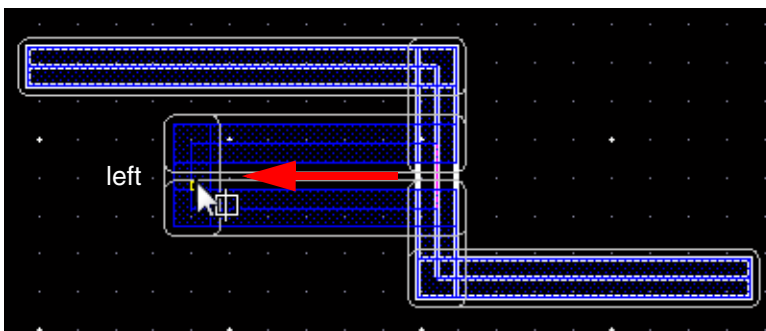
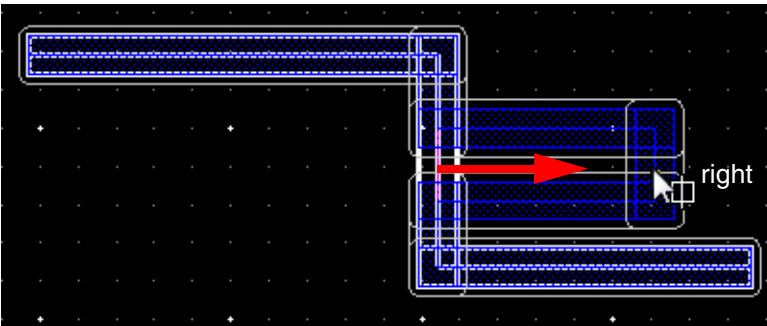
Working with Wires

- When the horizontal spine wire is not selected as a part of a split line

Consider the following scenario where the horizontal spine wire is not selected as a part of a split line. In this case, you are not able to split the spine along the top and bottom directions. The split can only happen along the left and right directions. The following figures illustrate the spine split only in the left and right directions.



The highlighted portion is split only along the left and right directions



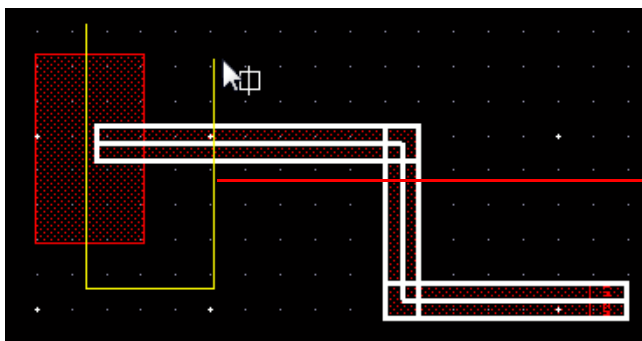
Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

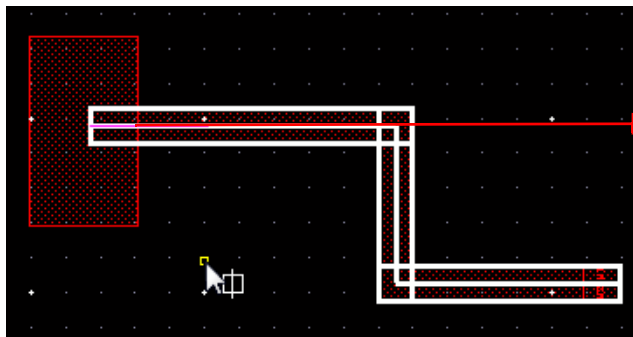
■ Spine connected to a shape and is part of a split line

You can also split a spine that is connected to a shape and still maintain the connection of the spine with the shape. To maintain the connection of a spine with a shape, select the *Keep wires connected to shapes* check box below the *Keep Wires Connected* to list box.

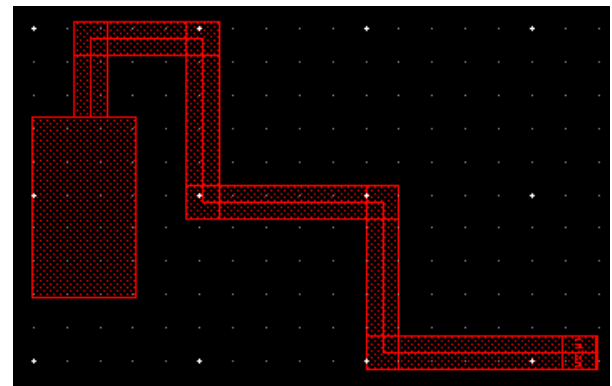
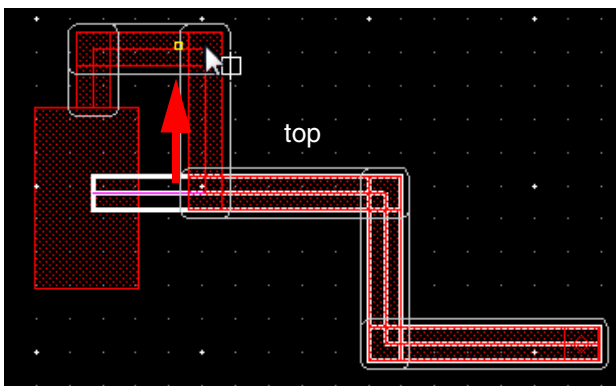
The following figures illustrate how the connection is kept when the spine is split and stretched along the top direction with the *Keep wires connected to shapes* checkbox selected and Spine selection mode ON.



The split line

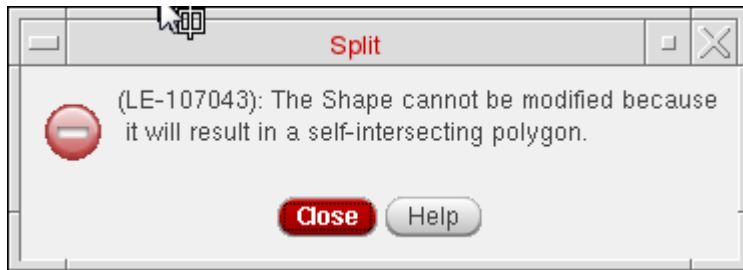


The highlighted portion is split



- Spine connected to a shape where the shape is a part of the split line

You cannot split and stretch a shape when the selected shape is connected to a spine and is a part of the split line. A message is displayed, as shown in the following figure.



When Spine Selection Mode is OFF

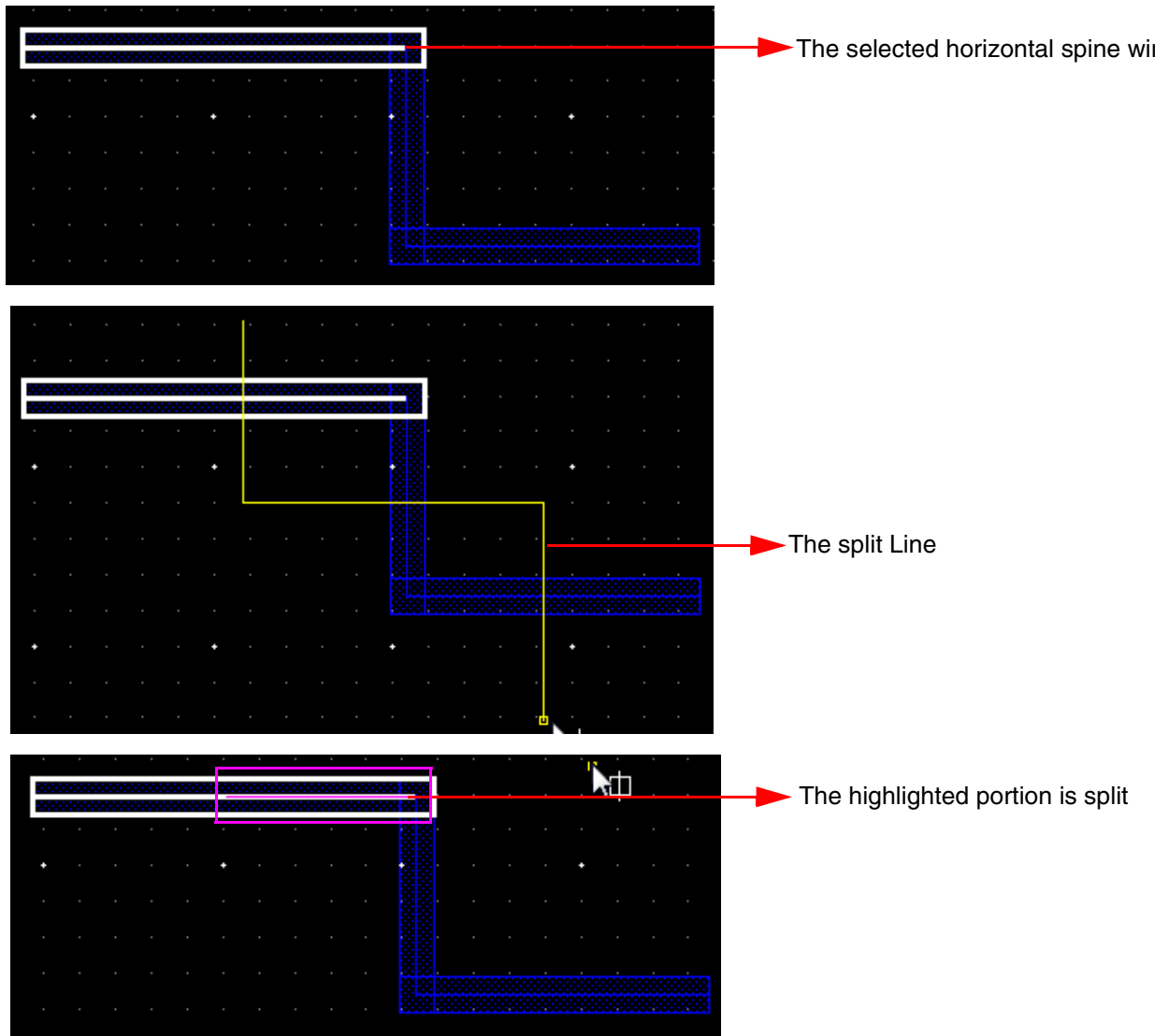
When the Spine selection mode is OFF, either a vertical or a horizontal spine wire is selected.

- When horizontal spine wire is selected

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

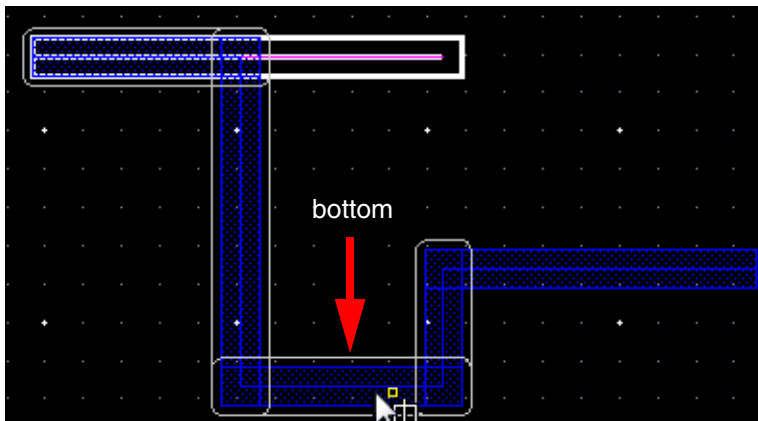
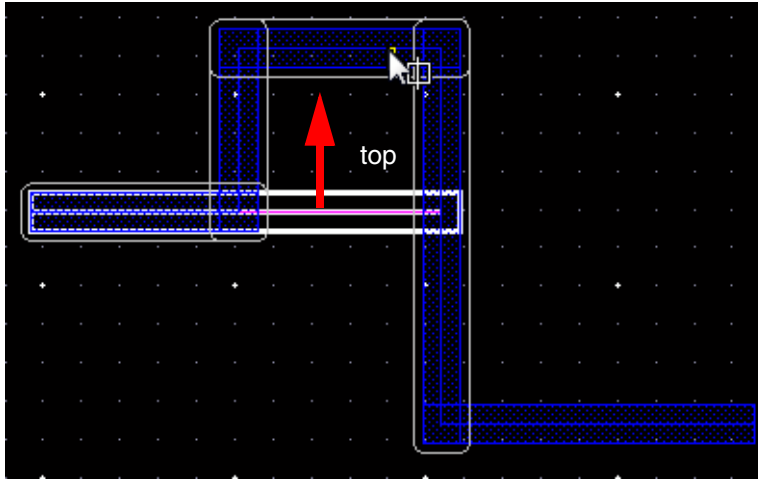
The following figures illustrate splitting and stretching when a horizontal spine wire is selected.



Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

In this case, the wire can split only along the top and the bottom directions and not in the left and right directions. The following figures illustrate the spine split only along the top and bottom directions.

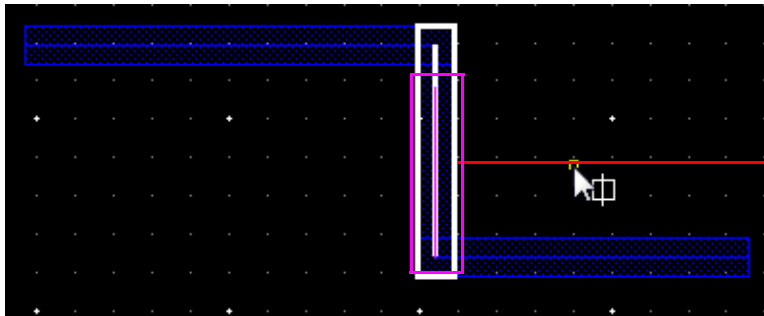


Virtuoso Interactive and Assisted Routing User Guide

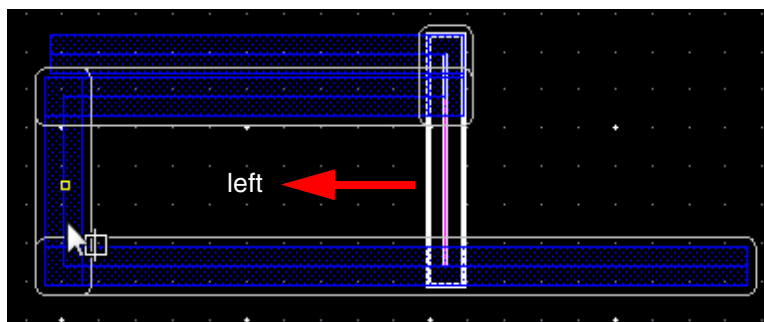
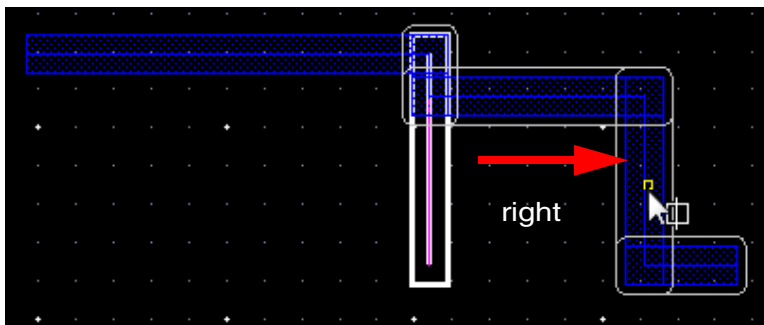
Working with Wires

■ When the vertical spine wire is selected

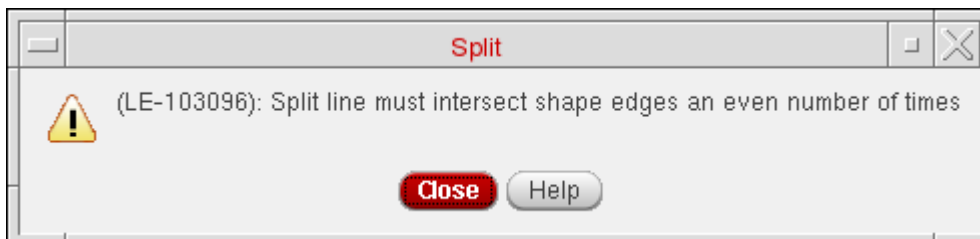
When a vertical spine wire is selected, the wire can split only along the right and left directions and not in the top and bottom directions. The following figures illustrate the spine split only along the left and right directions.



The highlighted portion is split only along the left and right directions



The following error message is displayed when the split line does not enclose the selected spine wire.

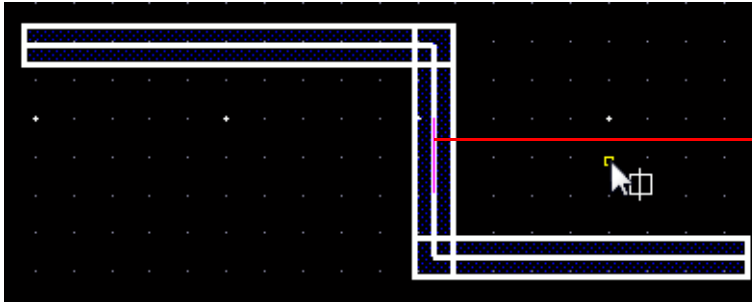


Virtuoso Interactive and Assisted Routing User Guide

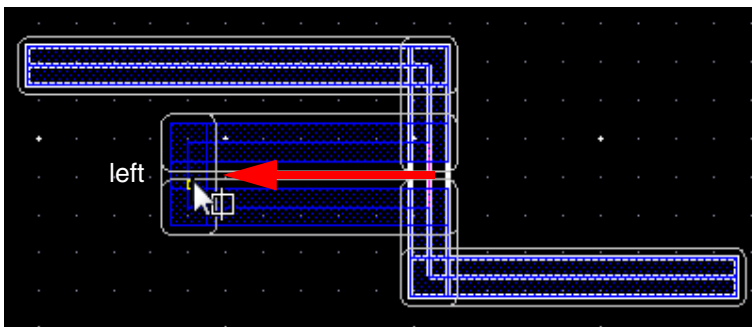
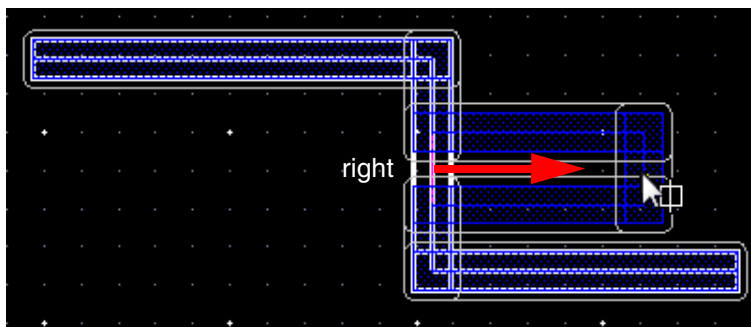
Working with Wires

- When the horizontal spine wire is not selected as a part of a split line

Consider a following scenario where the horizontal spine wire is not selected as a part of a split line. In this case, you are not able to split the spine along the top and bottom directions. The split can only happen along the left and right directions. The following figures illustrate the spine split only along the left and right directions.

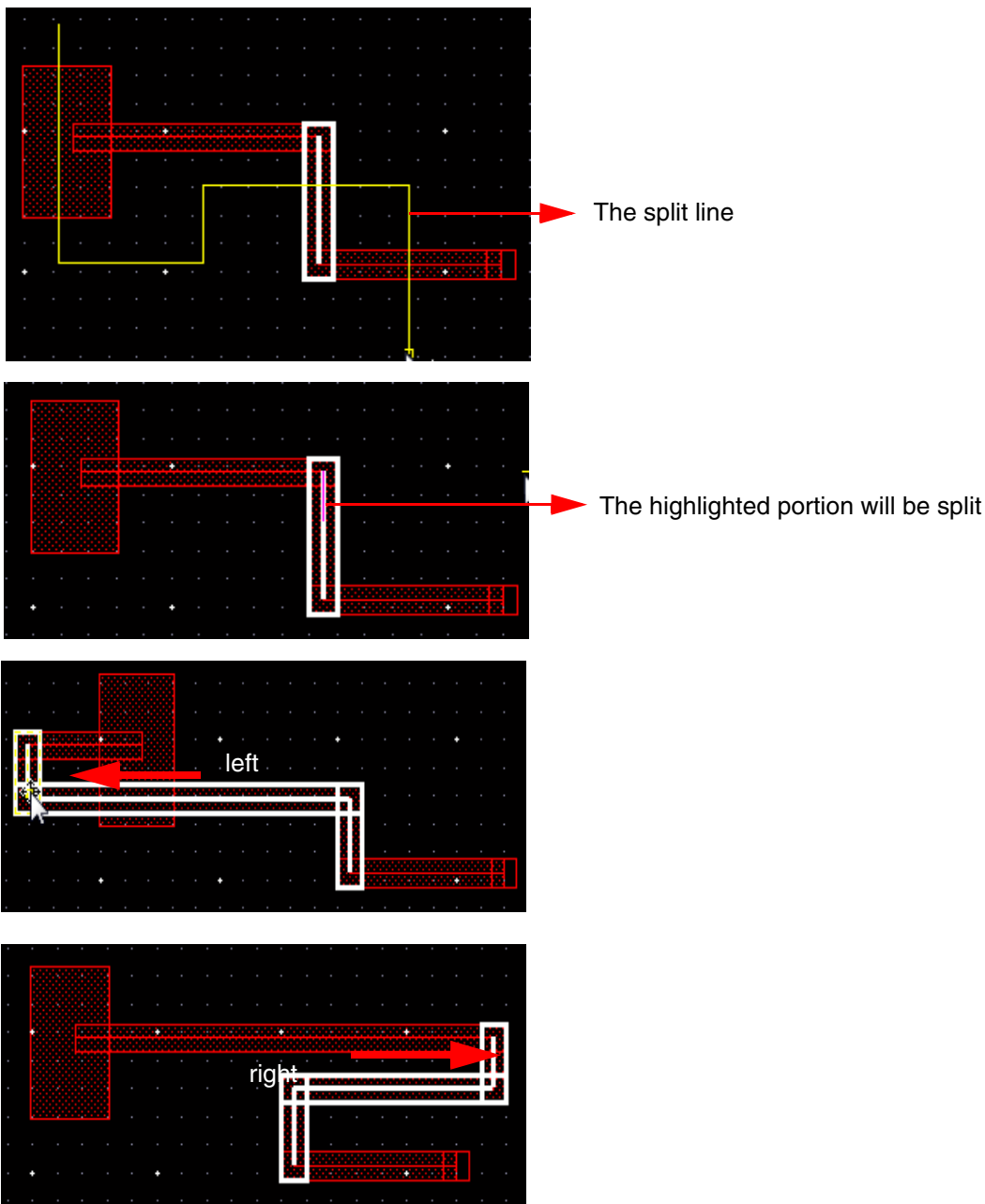


The highlighted portion is split only along the left and right directions



- Vertical spine wire connected to a shape and is a part of a split line

You can also split a vertical spine wire that is connected to a shape and still maintain the connection of the wire with the shape. To maintain the connection of a wire with a shape, select the *Shape* check box below the *Keep Wires Connected to* list box. The following figures illustrate how the connection is maintained when the wire is split and stretched along the left and right directions with the *Keep wires connected to shapes* checkbox selected and Spine selection mode OFF.

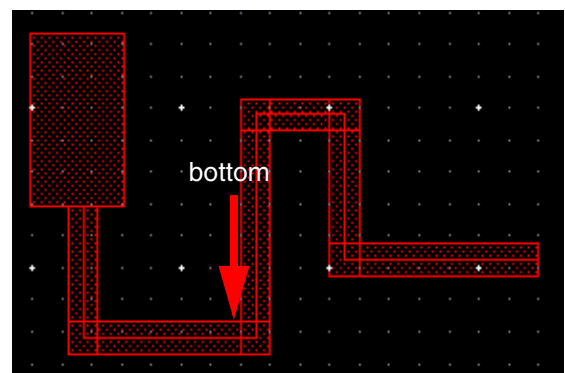
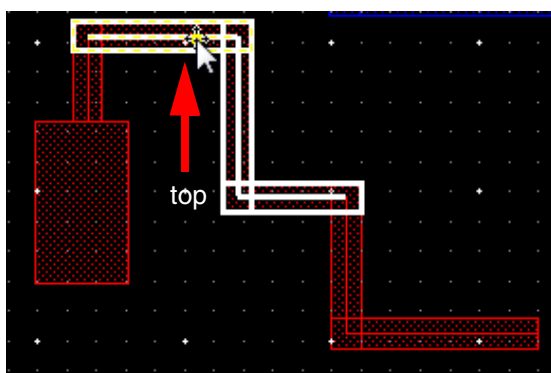
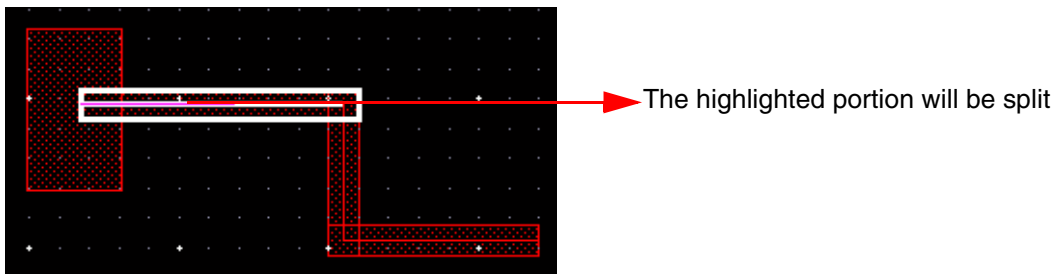
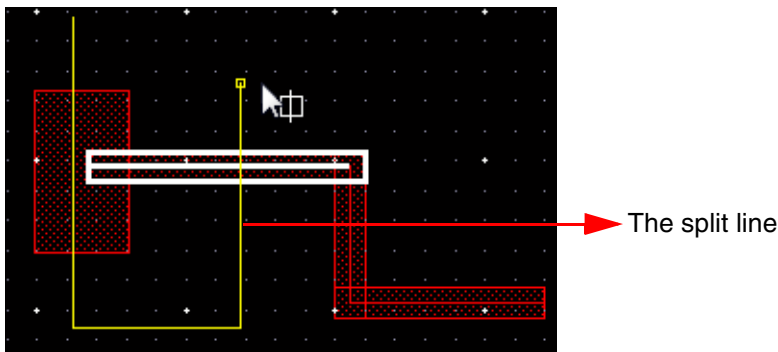


Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

- Horizontal spine wire connected to a shape and is a part of a split line

You can also split a horizontal spine wire that is connected to a shape and still maintain the connection of the wire with the shape. To maintain the connection of a wire with a shape, select the *Keep wires connected to shapes* check box below the *Keep Wires Connected to* list box. The following figures illustrate how the connection is maintained when the horizontal wire is split and stretched along the top and bottom directions with the *Shapes* checkbox selected and Spine selection mode OFF.



Using the All Wires and Vias Option

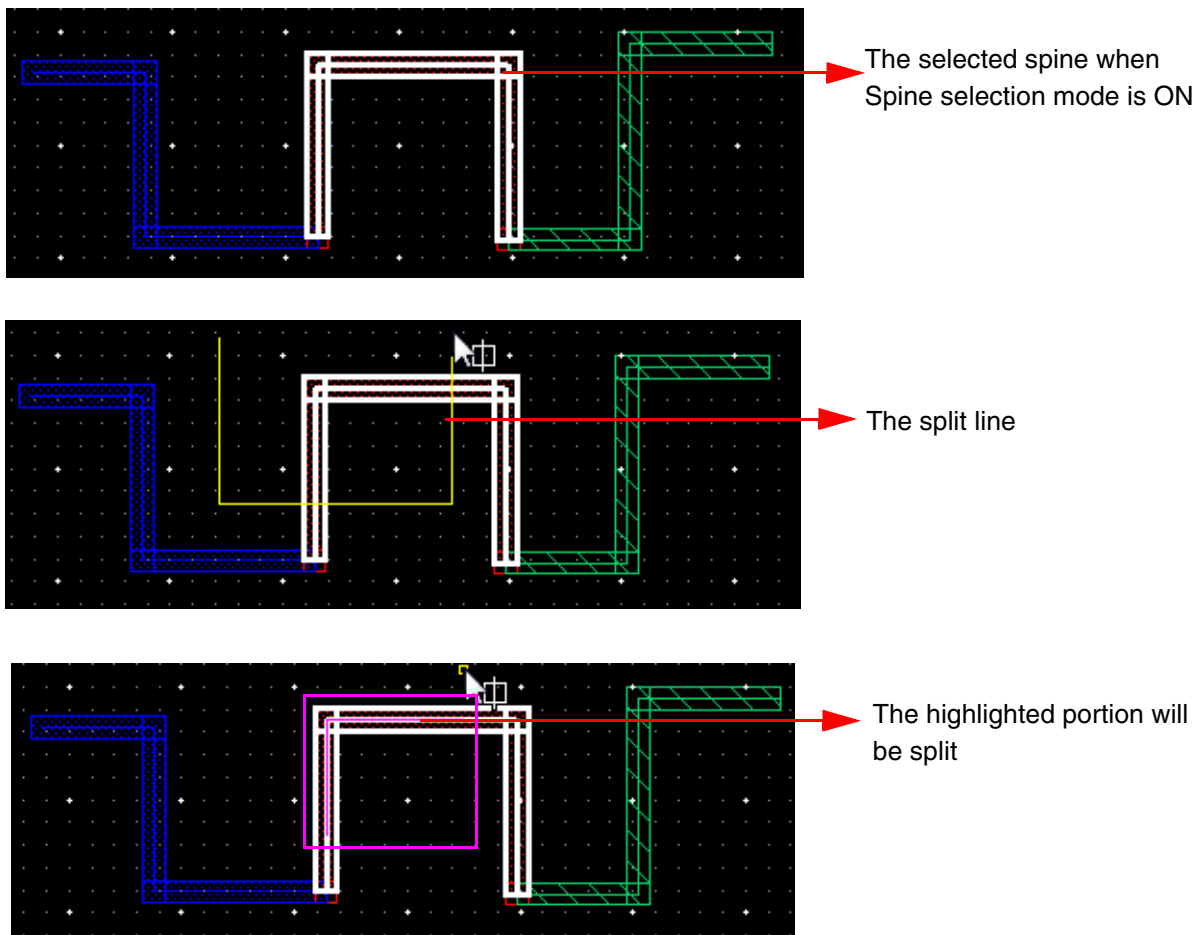
To maintain the pathSeg-to-pathSeg and pathSeg-to-via connections during a split, select the *All Wires and Vias* option from the *Keep Wires Connected to* drop-down list in the Split form. When the *All Wires and Vias* option is selected, the *Lock Unselected Vias* option is enabled. Selecting the *Lock Unselected Vias* option, locks the vias that are not selected while splitting a wire.

The behavior of how the spine is split when the Spine selection mode is ON is different from when it is OFF.

When Spine Selection Mode is ON

- When spine wires are selected as a part of a split line

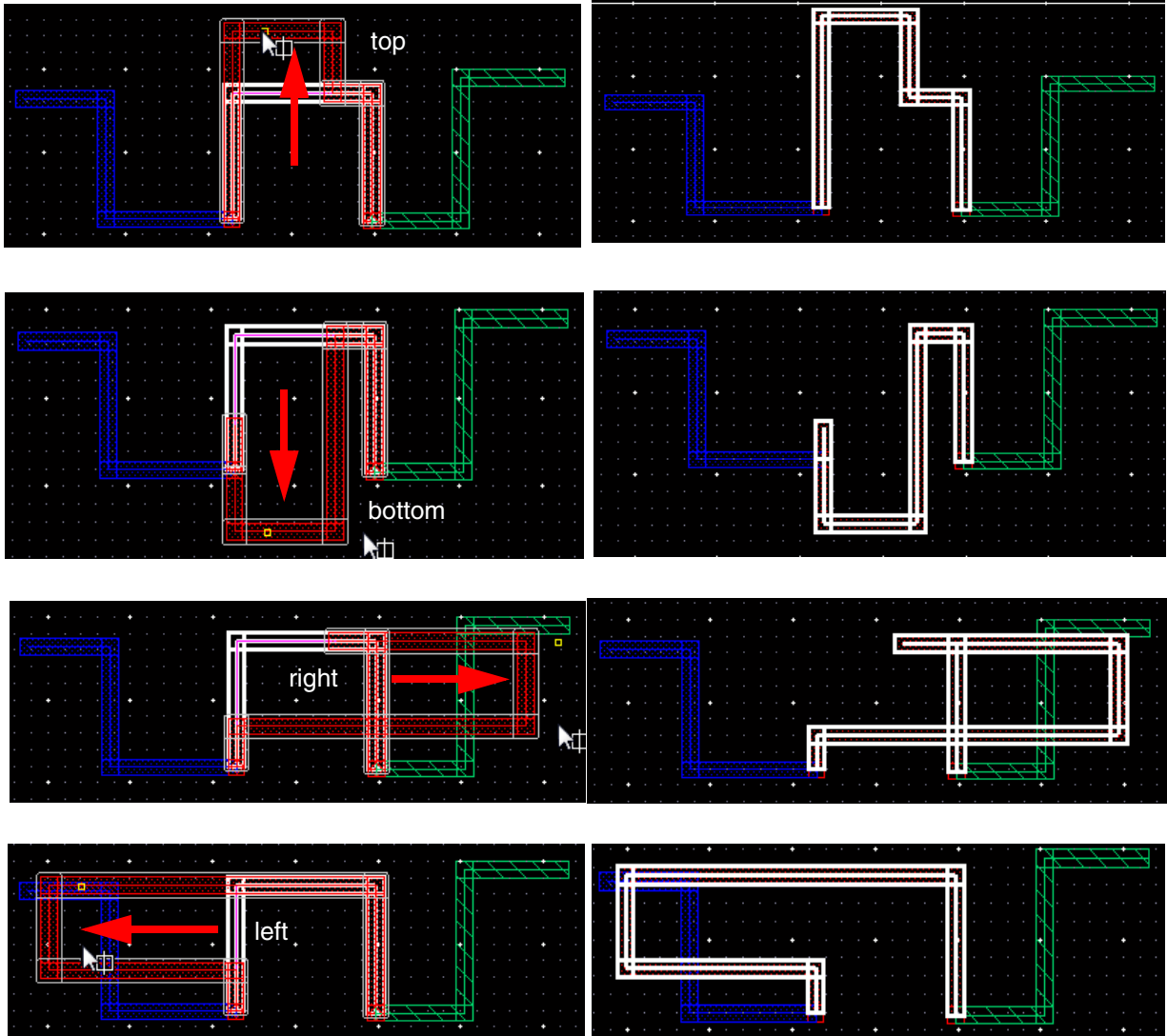
The following figures illustrate the splitting and stretching of spine with Spine selection mode ON.



Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

The following figures illustrate how the selected spine is split when it is stretched along the left, right, top, and bottom directions.

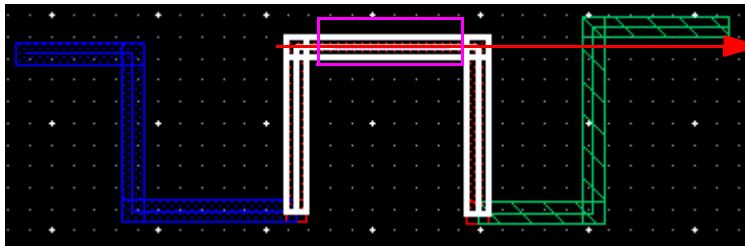


Virtuoso Interactive and Assisted Routing User Guide

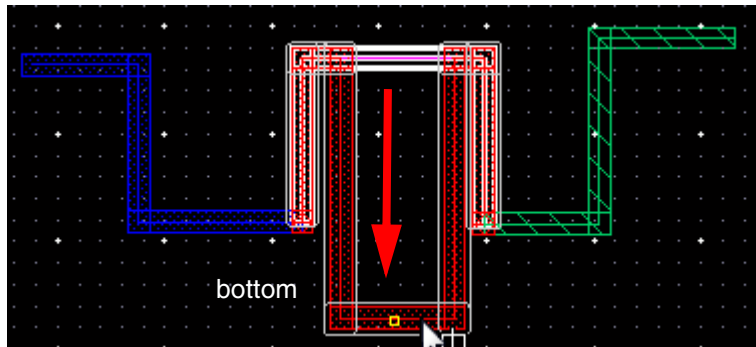
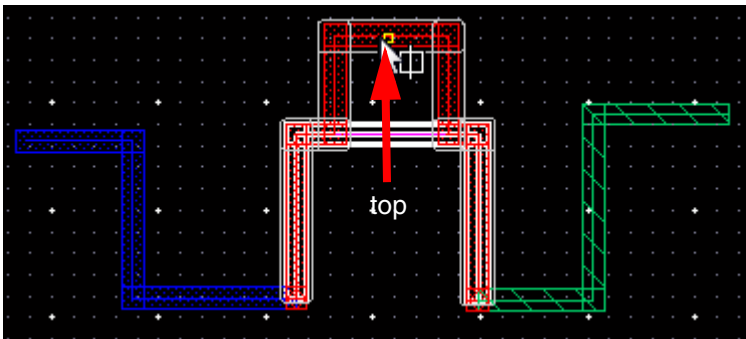
Working with Wires

- When only the horizontal spine wire is selected as a part of a split line

Consider the following scenario where only the horizontal spine wire is selected as a part of a split line. In this case, you are not able to split the spine along the left and the right directions. The split can only happen along the top and the bottom directions. The following figures illustrate the spine split along the top and bottom directions.



The highlighted portion is split along the top and bottom directions

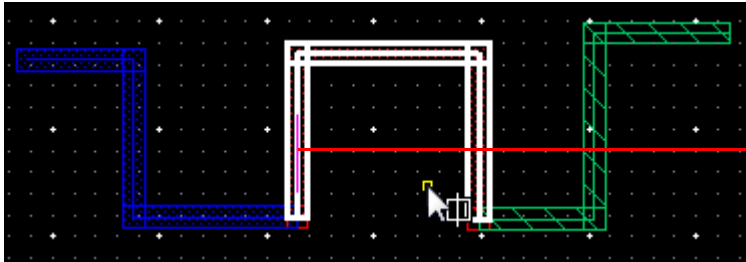


Virtuoso Interactive and Assisted Routing User Guide

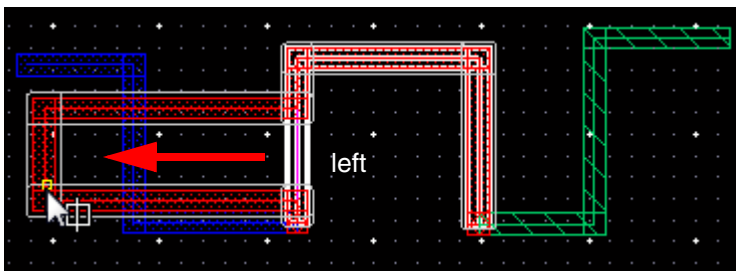
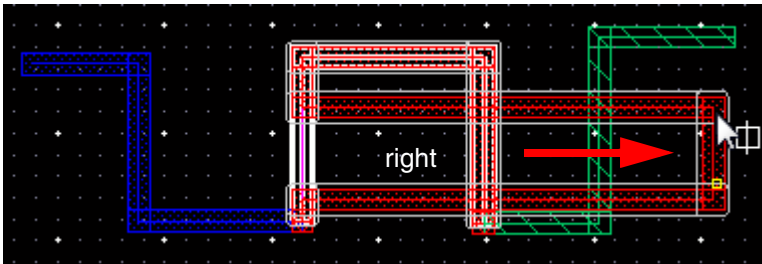
Working with Wires

- When only the vertical spine wire is selected as a part of a split line

Consider the following scenario where only the vertical spine wire is selected as a part of a split line. In this case, you are not able to split the spine along the top and bottom directions. The split can only happen only along the left and right directions. The following figures illustrate the spine split along the left and right directions.



The highlighted portion is split only along the left and right directions

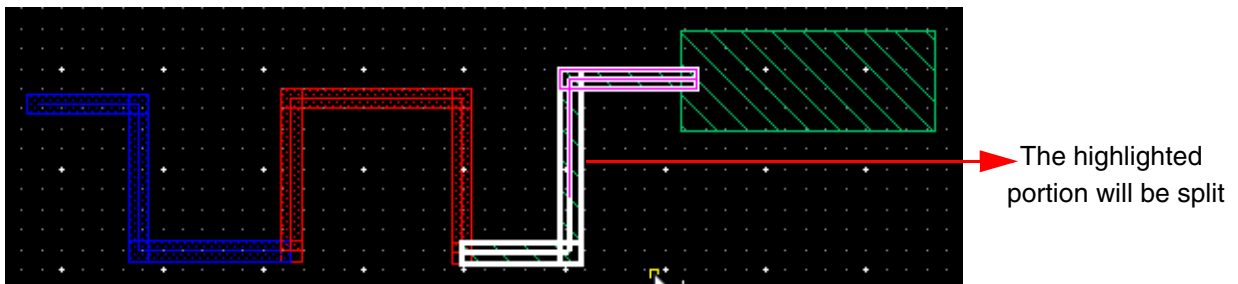
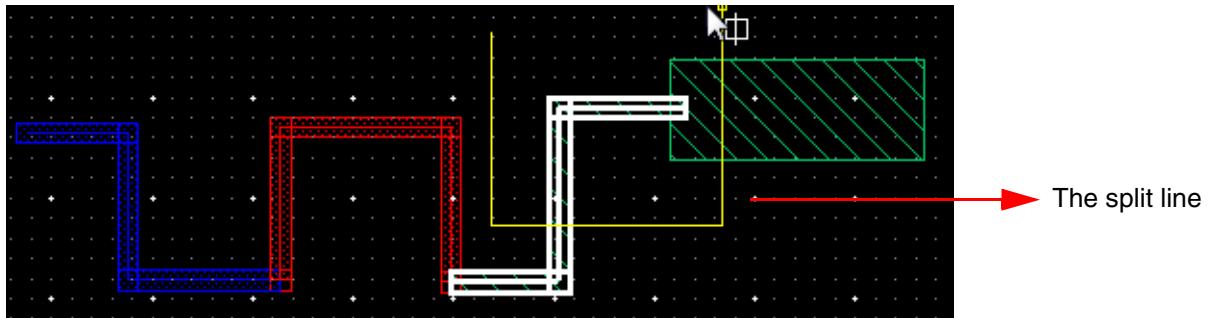


Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

■ Spine connected to a shape and is a part of a split line

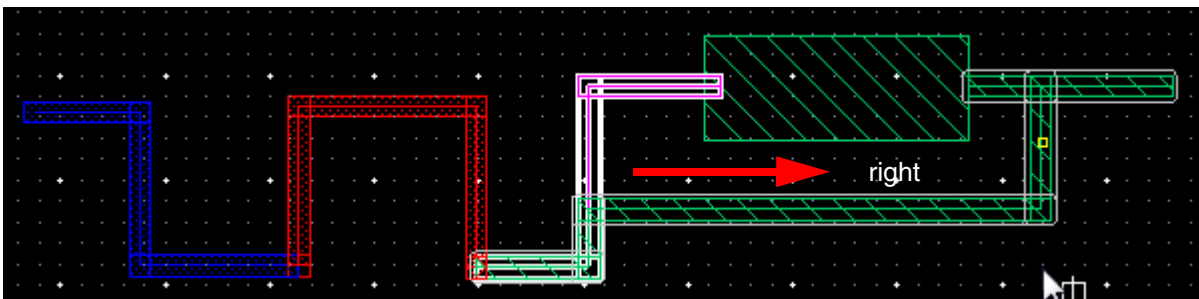
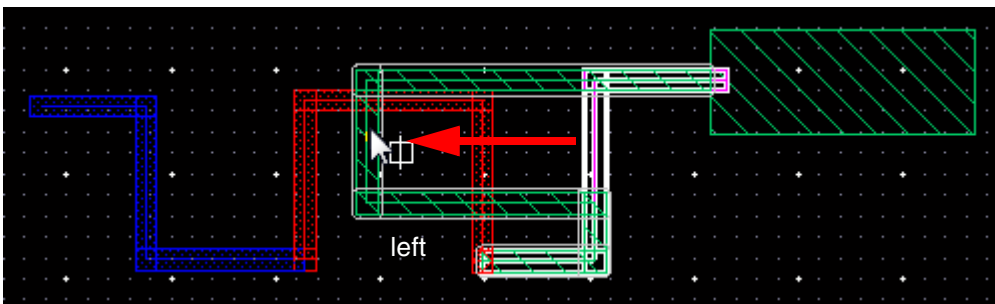
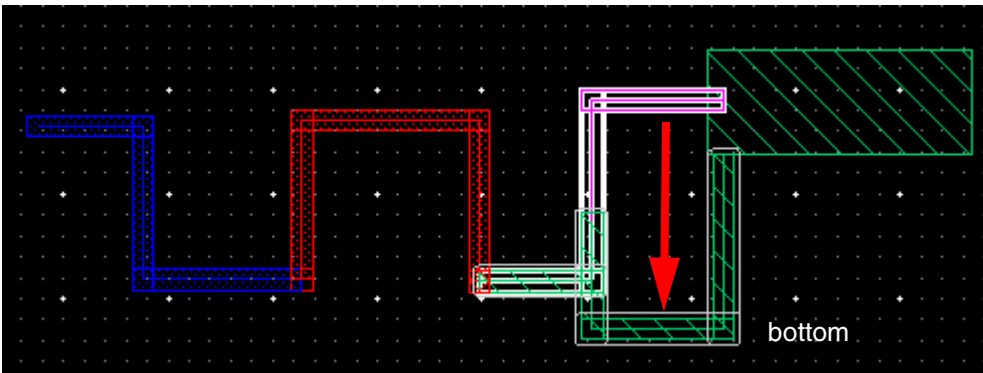
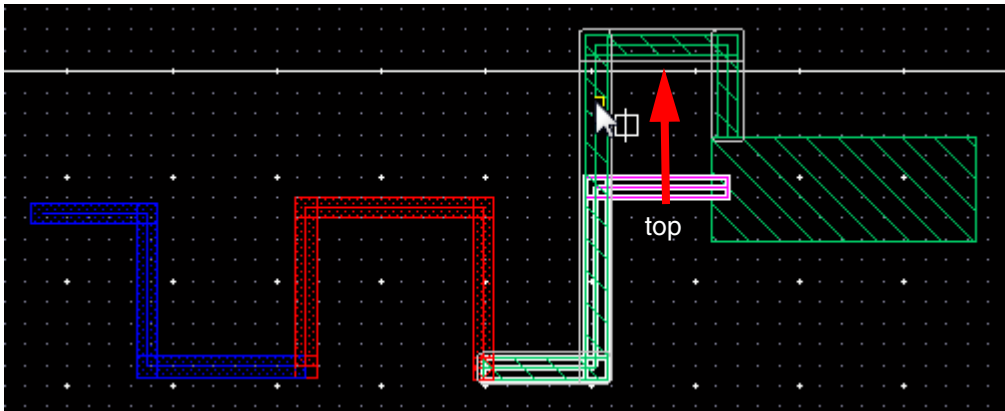
You can also split a spine that is connected to a shape and still maintain the connection of the spine with the shape. To maintain the connection of a spine with a shape, select the *Keep wires connected to shapes* check box below the *Keep Wires Connected* to list box. The following figures show the split line and the highlighted portion of the spine wire that will be split.



Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

The following figures illustrate how the spine wire is split along the top, bottom, left, and right directions and maintains the connection with the shape and other wires and vias.

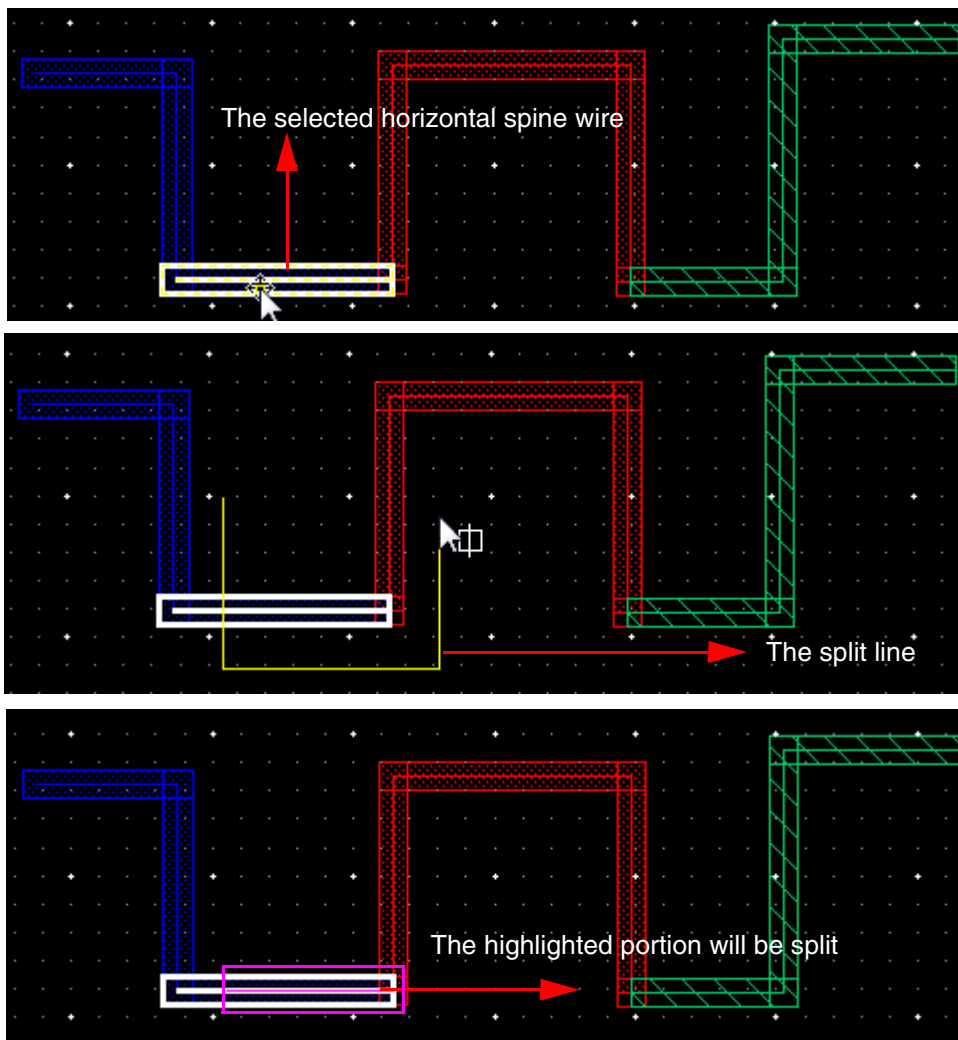


When Spine Selection Mode is OFF

When the Spine selection mode is OFF, either a vertical or a horizontal spine wire is selected.

■ When horizontal spine wire is selected

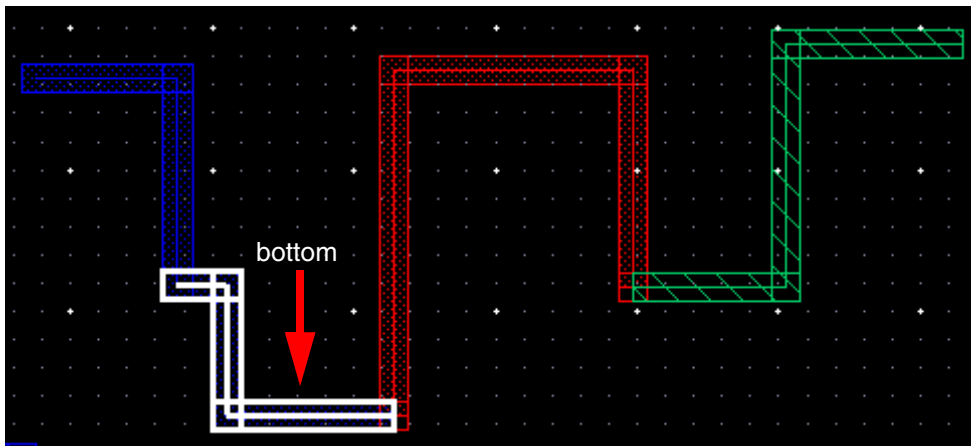
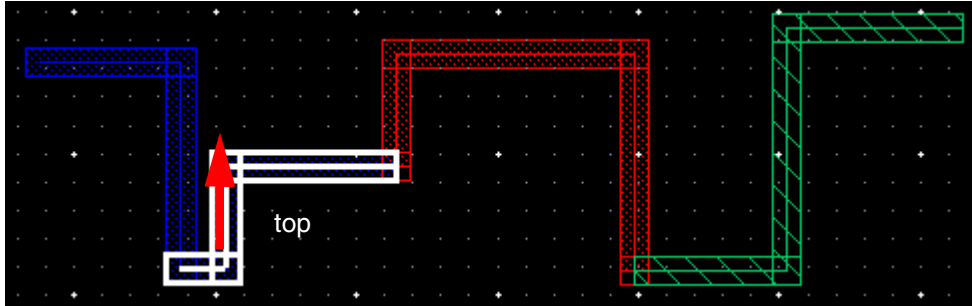
The following figures illustrate splitting and stretching when a horizontal spine wire is selected.



Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

In this case, the wire can split only along the top and the bottom directions and not along the left and right directions. The following figures illustrate the spine split along the top and bottom directions.

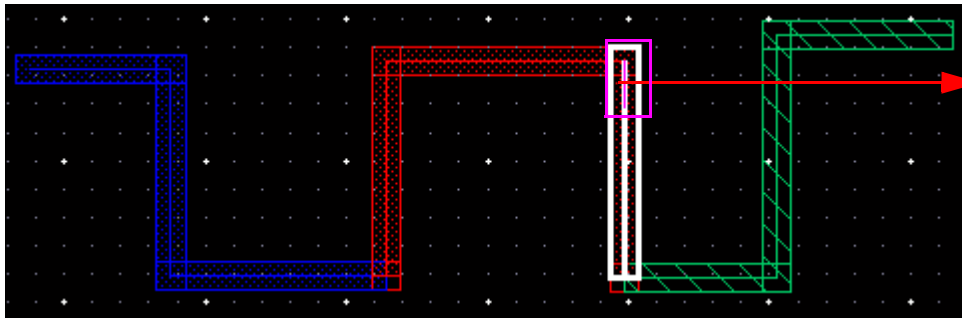


Virtuoso Interactive and Assisted Routing User Guide

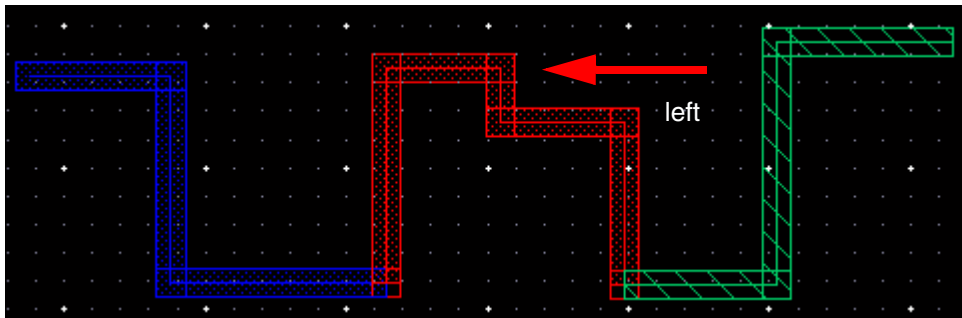
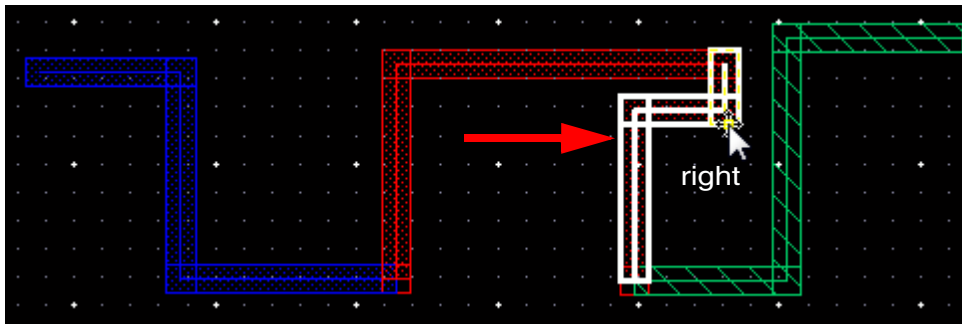
Working with Wires

■ When the vertical spine wire is selected

When a vertical spine wire is selected, the wire can split only along the right and left directions and not along the top and bottom directions. The following figures illustrate the spine split along the left and right direction.



The highlighted portion is split only along the left and right direction

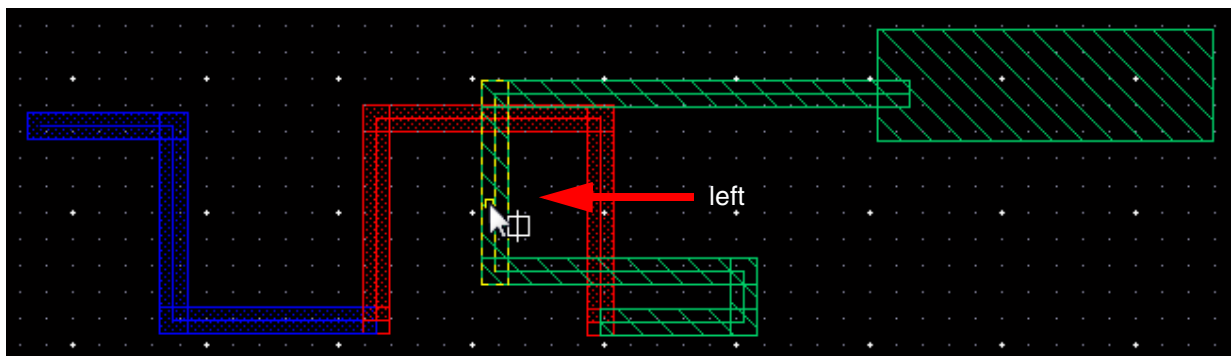
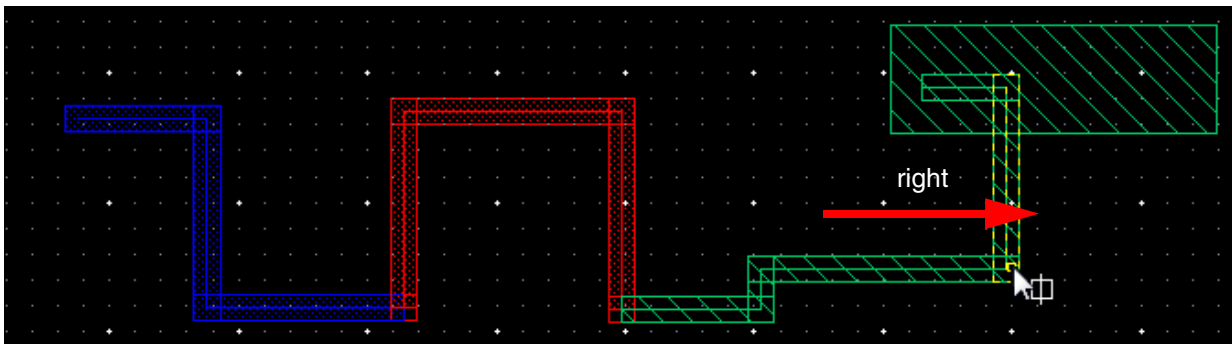
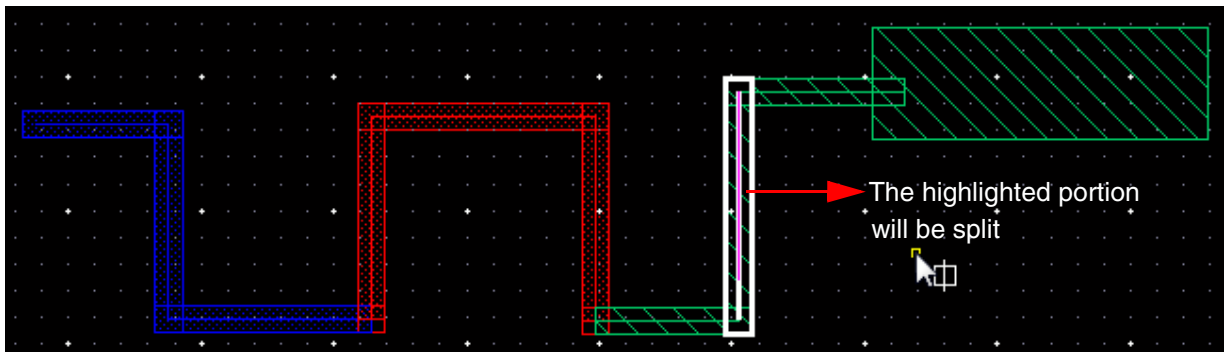


Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

- Vertical spine wire connected to a shape and is a part of a split line

You can also split a vertical spine wire that is connected to a shape and other spine wires, and still maintain the connection of the wire with the shape. To maintain the connection of a wire with a shape, select the *Keep wires connected to shapes* check box below the *Keep Wires Connected to* list box. The following figures illustrate how the connection is maintained when the wire is split and stretched along the left and right directions with the *Shapes* checkbox selected and Spine selection mode OFF.

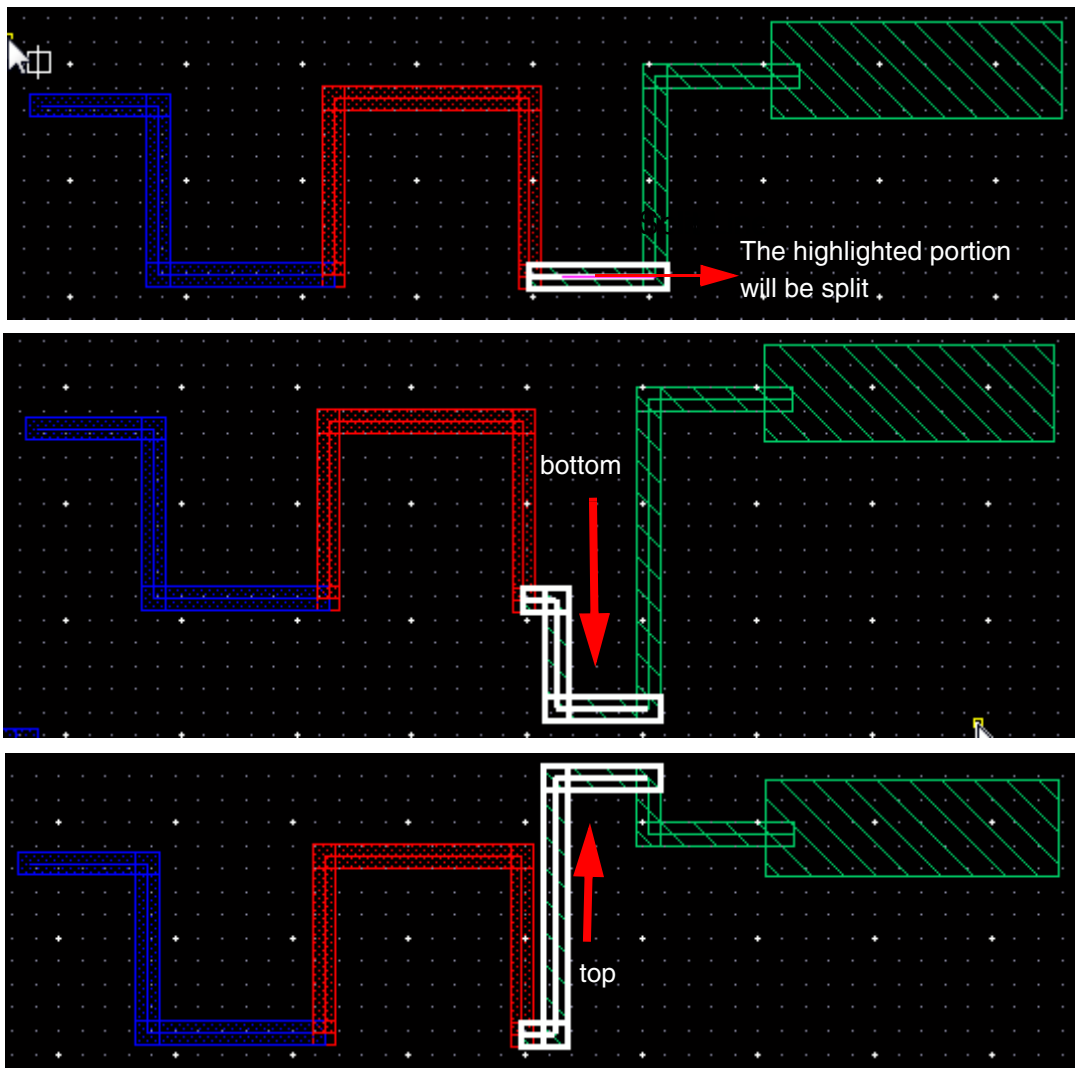


Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

■ Horizontal spine wire connected to a shape and is a part of a split line

You can also split a horizontal spine wire that is connected to a shape and other spine wires and still maintain the connection of the wire with the shape. To maintain the connection of a wire with a shape, select the *Shape* check box below the *Keep Wires Connected to* list box. The following figures illustrate how the connection is maintained when the horizontal wire is split and stretched along the top and bottom directions with the *Keep wires connected to shapes* checkbox selected and Spine selection mode OFF.



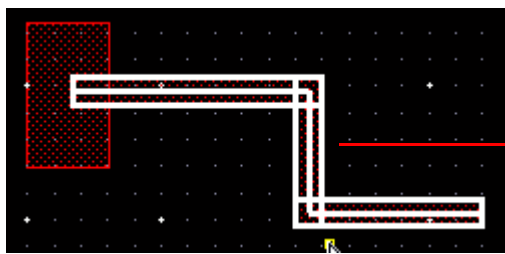
Using the No Wires and Vias Option

If you do not want to maintain connections to spine wires, and vias, select the *No wires and vias* option from the *Keep Wires Connected to* drop-down list in the Split form. The behavior of how the spine is split when the Spine selection mode is ON is different from when it is OFF.

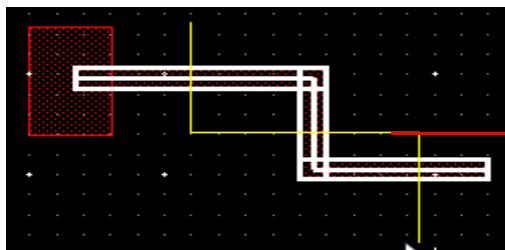
When Spine Selection Mode is ON

- When vertical and horizontal spine wires are part of a split line

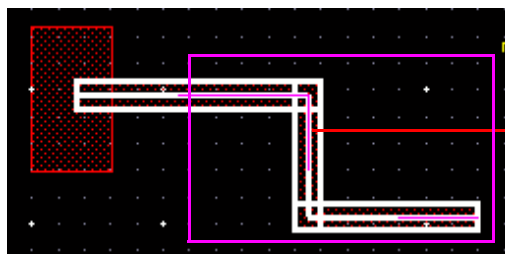
The following figures illustrate how the spine is split when the *No wires and vias* option is selected with Spine selection mode ON.



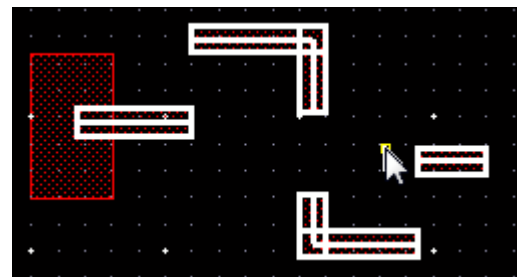
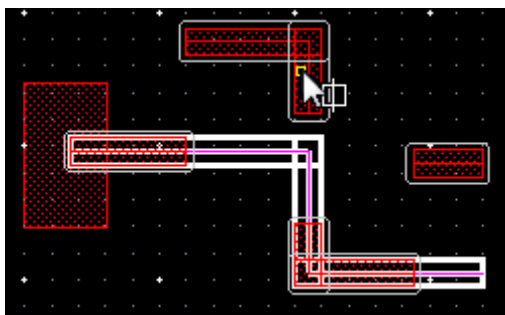
The selected spine when Spine selection mode is ON



The split line



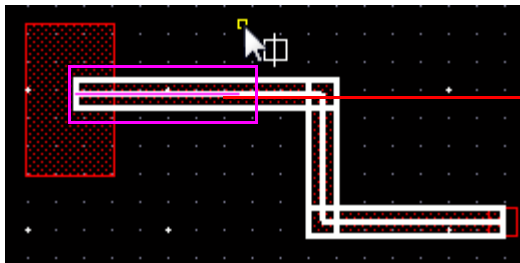
The highlighted portion will be split



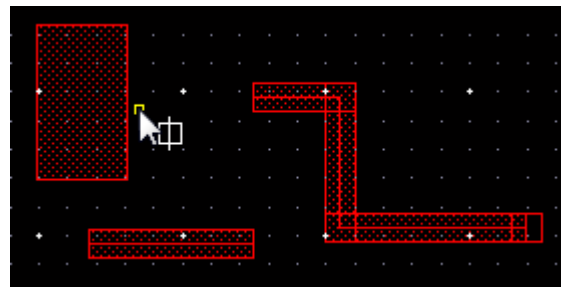
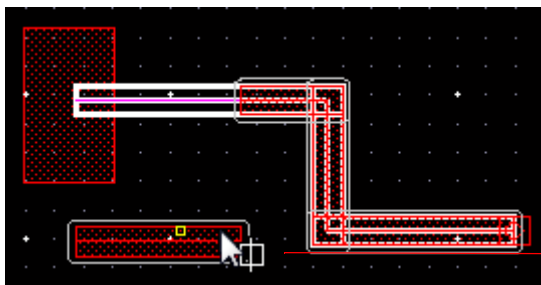
Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

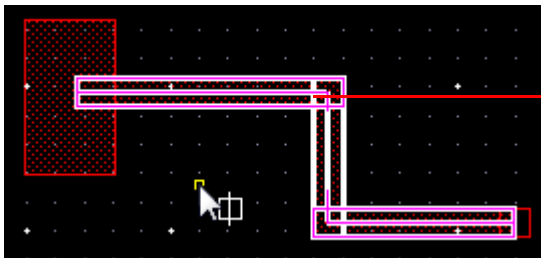
- The following figures illustrate how a spine is split when a vertical spine wire is not selected as a part of a split line.



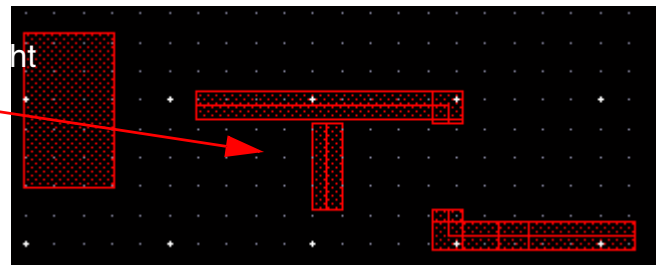
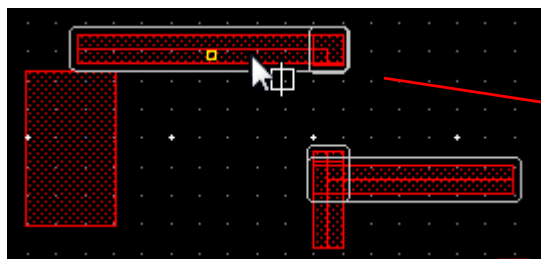
The highlighted portion is split along the top and bottom directions



- The following figures illustrate how a spine is split when the horizontal spine wire is not selected as a part of a split line.



The highlighted portion is split along the left and right directions

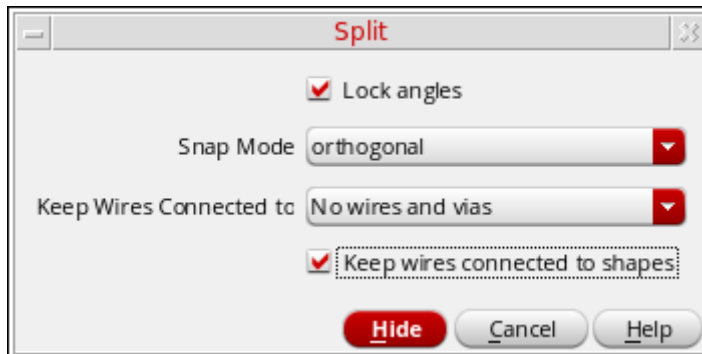


Virtuoso Interactive and Assisted Routing User Guide

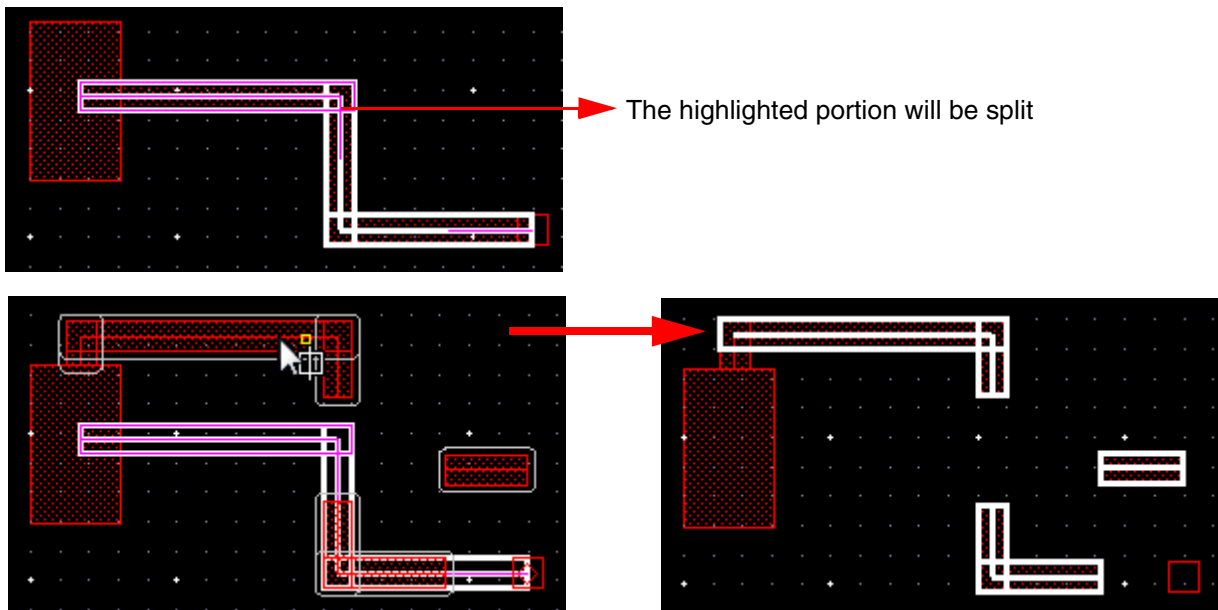
Working with Wires

■ Spine connected to a shape and is part of a split line

You can also split a spine that is connected to a shape and still maintain the connection of the spine with the shape. To maintain the connection of a spine with a shape, select the *Keep wires connected to shapes* check box below the *Keep Wires Connected* to list box.



The following figures illustrate how the connection is maintained when the spine is split and stretched with the *Shapes* checkbox selected and Spine selection mode ON.

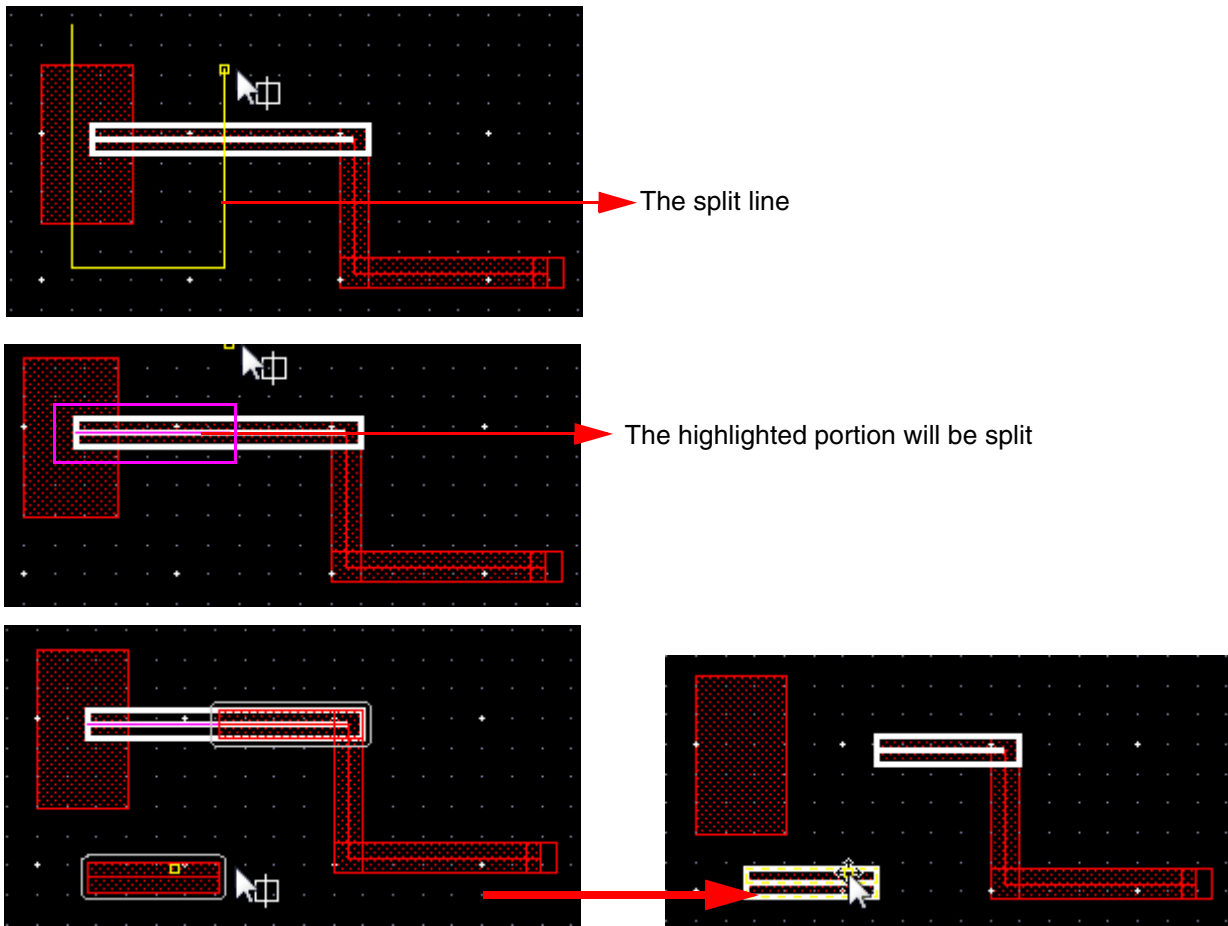


When Spine Selection Mode is OFF

When the Spine selection mode is OFF, either a vertical or a horizontal spine wire is selected.

- When horizontal spine wire is selected

The following figures illustrate the splitting and stretching when a horizontal spine wire is selected.

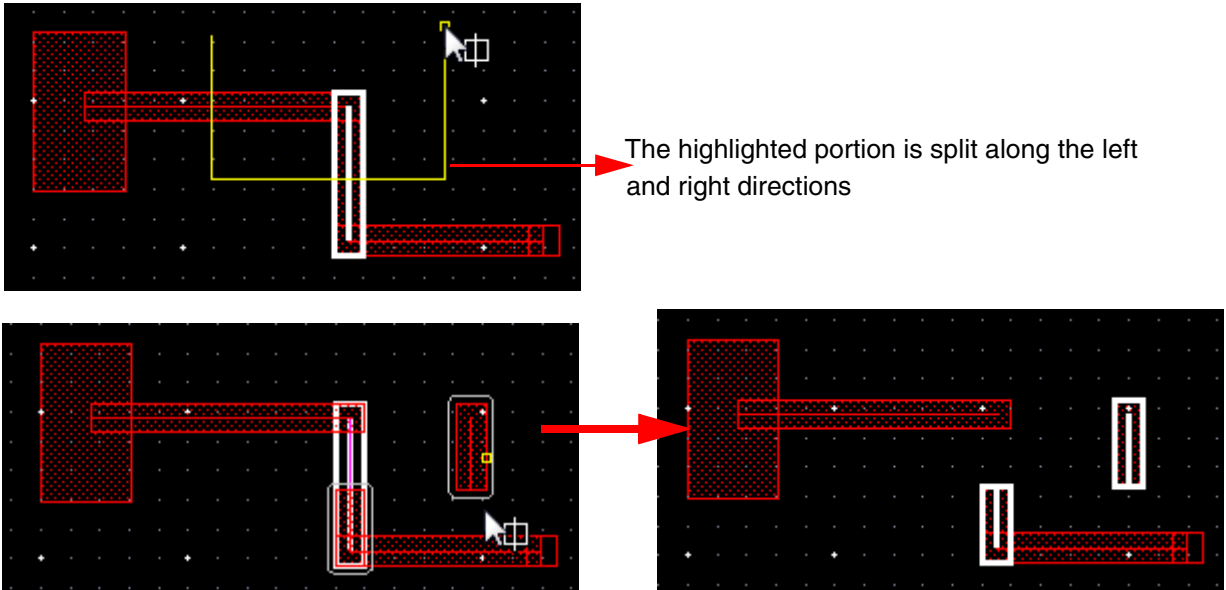


Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

■ When the vertical spine wire is selected

The following figures illustrate the splitting and stretching when a vertical spine wire is selected.

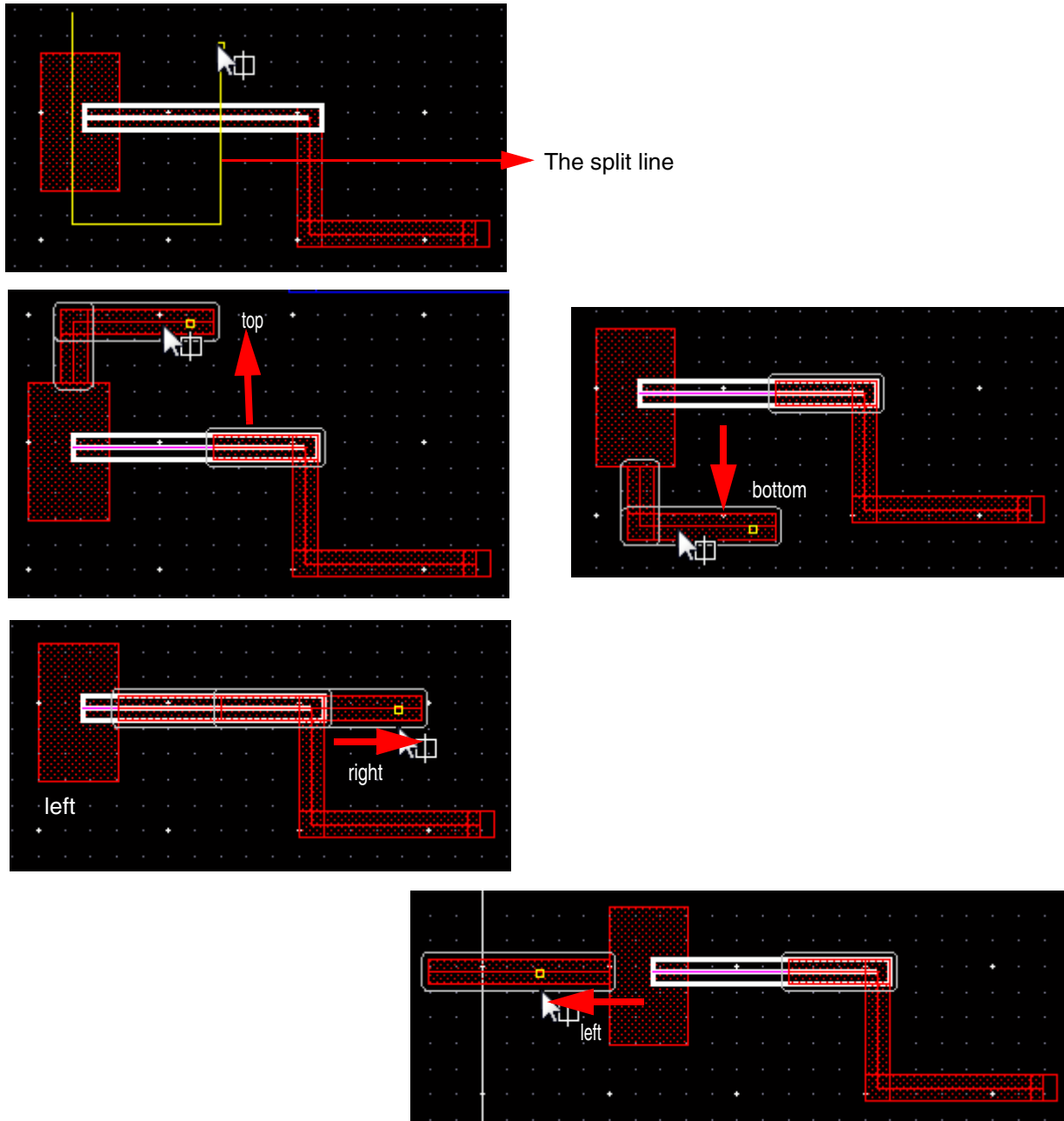


Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

- Spine wire connected to a shape and is part of a split line

You can also split a spine wire that is connected to a shape and still maintain the connection of the wire with the shape. The following figures illustrate how the connection is maintained when the wire is split and stretched with the *Shapes* checkbox selected and Spine selection mode OFF.



Support for Local Interconnect (Li/MOL) Layers

Traditionally, creating a simple contact aperture helps establish an electrical connection between the first metal layer and polysilicon/diffusion. However, in the Advanced Node technology, LI conductive layers are overlapped to establish such an electrical connection. To help establish such connections, Wire Editor has been enhanced to support:

- Digitizing wires on layers with function type `Li`. For example, `Li1`, `Li2`, and `LiPO`.
Note: The Local interconnect layers `Li1`, `Li2`, and `LiPO` can be tapped in interactive wire editing and assisted routing commands such as Point-to-Point and Guided Routing.
- Switching between `Li` layers using `Li Vias`
- Changing layers up and down the connectivity stack
 - Local interconnect layers through `metal1`
 - Poly through `metal1`
- Using `CutPoly` layer, which is supported by VLS - XL extractor.
Note: Shorts between poly pins can be resolved by adding shapes in `CutPoly` layer.

Wire Editing with Li/MOL Layers

The wire editor follows a centerline connectivity model, defined in OpenAccess, which requires a physical via to be used when transitioning between layers, so even though the technology does not require a via, the tools connectivity model does. This is a cumbersome use model and unnecessarily adds extra steps as far as local interconnect layers are concerned. In order to make the Li use model more intuitive for the layout designer, the Wire Editor use model has been enhanced to automatically generate and place the required vias when transitioning to and from the `Li` layers. This would help you to automatically add vias when connecting `Li` layers. You only need to know the layer on which you want to draw the wire, the layer from which the wires are coming from, and the layer to which they are going. The various Li wiring use models are described below.

Inserting an MOL Via Between Two Interconnect Layers

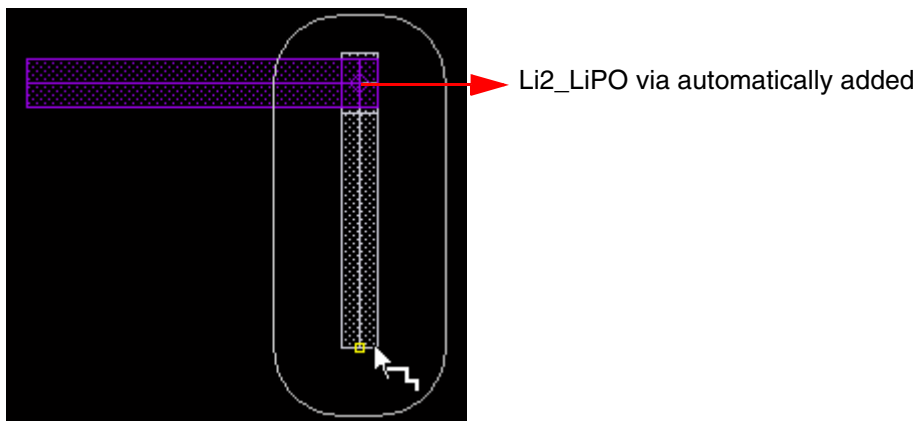
When you create a wire on a `Li` interconnect layer and click to make an orthogonal turn, a MOL via is automatically added to transition to another `Li` layer. However, this is possible only if the Wire Editor can determine a single destination `Li` layer based on the `rectShapeDir` and `minWireOverlap` rules defined in the technology file for these interconnect layers.

Virtuoso Interactive and Assisted Routing User Guide

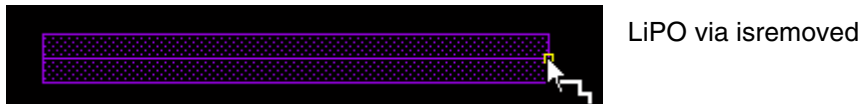
Working with Wires

For example, while creating an `Li1` wire and clicking to make a turn does not drop a MOL via; whereas the `minWireOverlap` rule between `Li1` and `Li2` determines that both wires must have same direction. Conversely, creating a `Li2` wire and clicking to make a turn does drop a `Li2_LiPO` via automatically and continues creating a `LiPO` wire, as shown in the following figures.

1. When you create a wire on the `LiPO` layer and click to make a turn, a `Li2_LiPO` via is automatically added, as shown in the following figure.



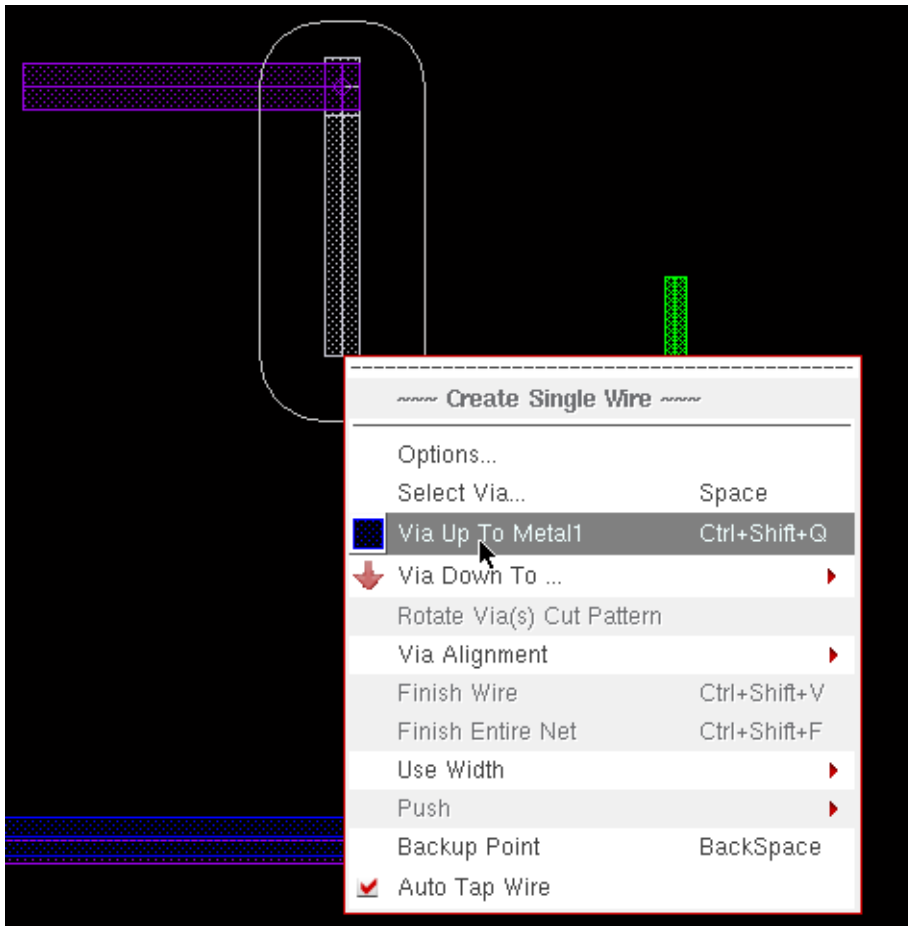
2. Dragging back the wire in the same direction, the `Li2_LiPO` via is removed, as shown in the following figure.



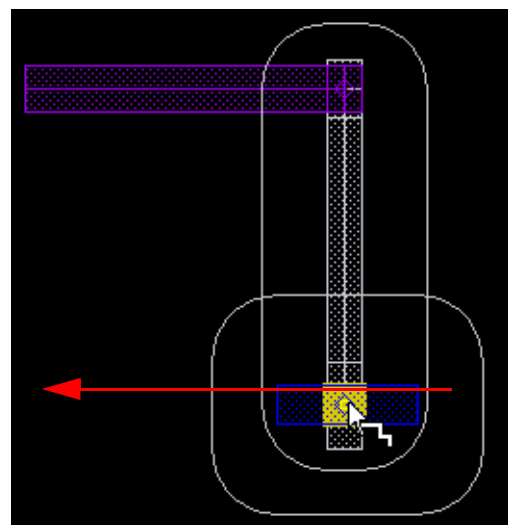
Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

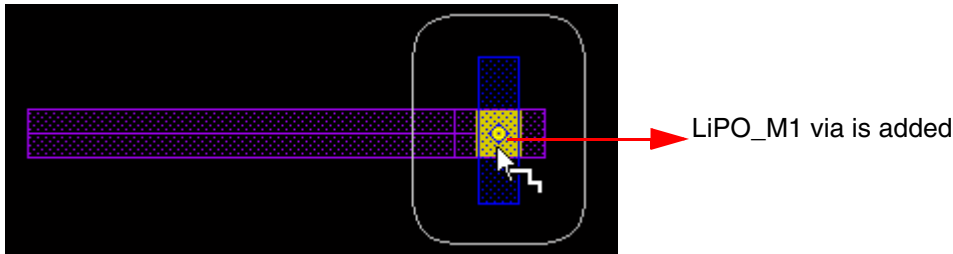
3. When you start a wire on `LiPO` and click to make a turn, right-click and select *Via Up To Metal1*, as shown in the first figure below. The `Li2_M1` via is added, as shown in the second figure below.



Li2_M1 via is added



4. Now, when you drag back to the original `LiPO` direction, the `Li2` wire and the auto dropped via `LiPO_Li2` disappears. The `LiPO` wire is dragged and the terminating via is added as `LiPO_M1`, that is, the same exit layer.



Starting a Wire from an Existing Lithium Path Segment

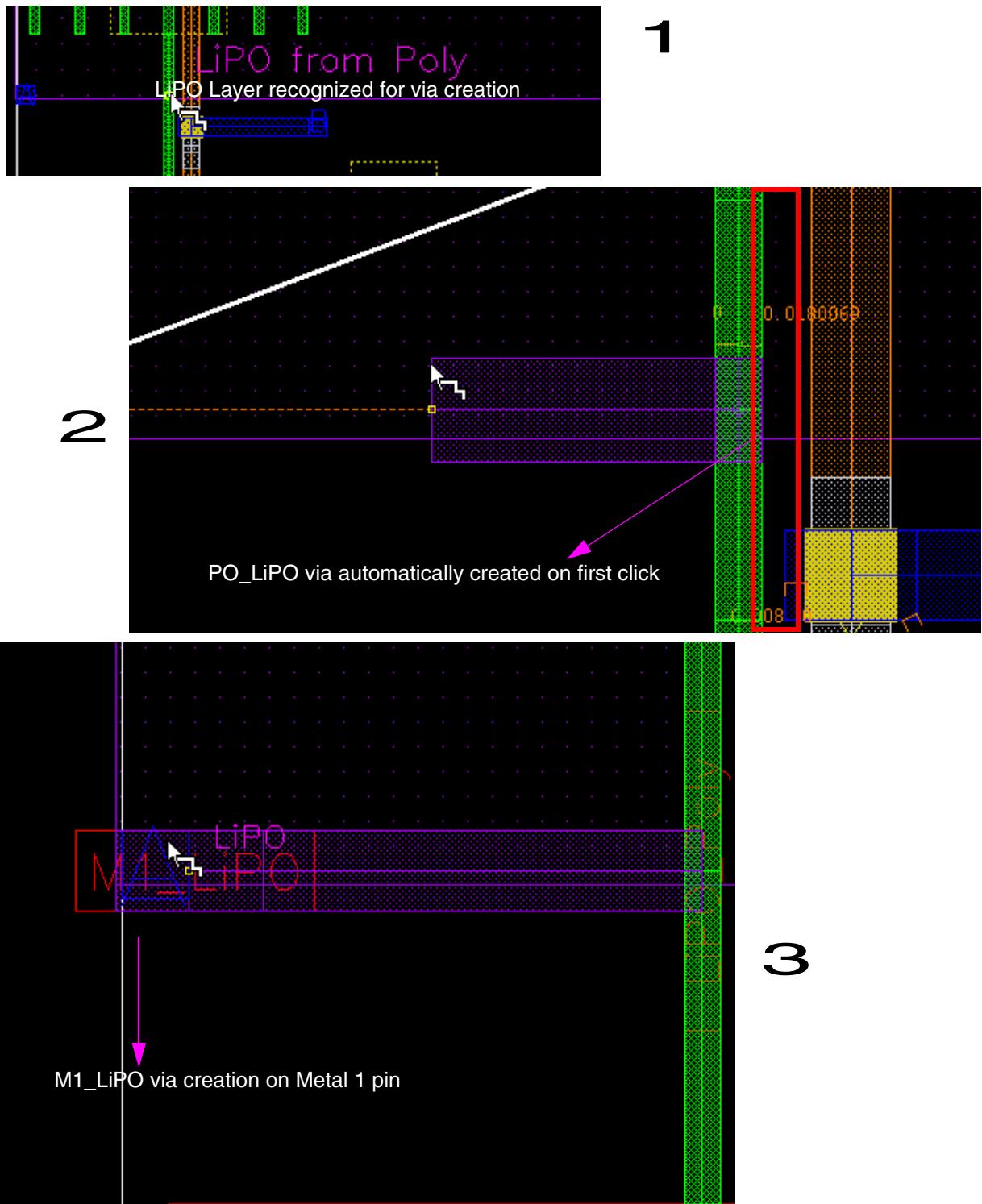
The Wire Editor gives preference to the layer selected in the Layers palette for local interconnect layers. If the selected layer in the Layers palette is a local interconnect layer and if a MOL via exists between the selected layer and the target object, the wire is going to be smart snapped. The Wire Editor automatically drops the MOL via so that you do not have to perform the extra step by using the *Via Up* or *Via Down* command. The added via is then automatically dropped and snapped to the centerline of the adjacent `Li` layer. The following example displays that when you click a `Poly` wire, a `PO_LiPO` via is automatically created and the via is centered on the `Poly` wire. The steps for this are as follows:

1. If the `LiPO` layer is selected in the Layers assistant then, when you place the cursor on `Poly` layer, `LiPO` layer is automatically recognized for via construction as shown in Figure 1 below. The cursor on `Poly` recognizes `LiPO`.
2. When `LiPO` layer is accepted, `PO_LiPO` via is created, as shown in Figure 2 below.

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

3. As you continue to drag the `LiPO` wire and reach close to the `Metal1` pin, `M1_LiPO` via is generated, as shown in Figure 3 below.



Starting a Wire from a Rectangle Pin

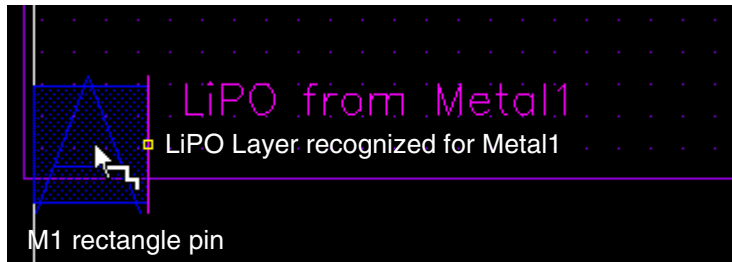
When creating a wire starting from a `Metal1` pin and the layer selected in the Layers assistant is `LiPO`, the `Create Wire` command automatically creates a via that is between the pin and the wire. The following steps describe how a via is created.

1. When you place the cursor on `Metal1` pin, `LiPO` layer is automatically recognized for via construction, as shown in Figure 1 below. The cursor on `Metal1` pin recognizes `LiPO`.
2. When `LiPO` layer is accepted, `M1_LiPO` via is created, as shown in Figure 2 below.

Virtuoso Interactive and Assisted Routing User Guide

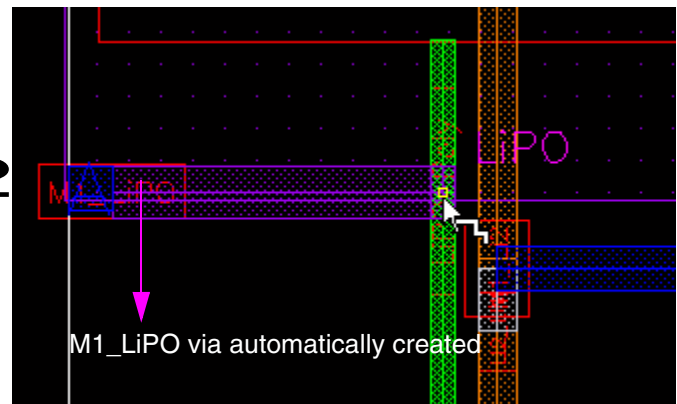
Working with Wires

3. When you drag the `LiPO` wire and you find another wire on the `Poly` layer, `PO_LiPO` is created, as shown in Figure 3 below.

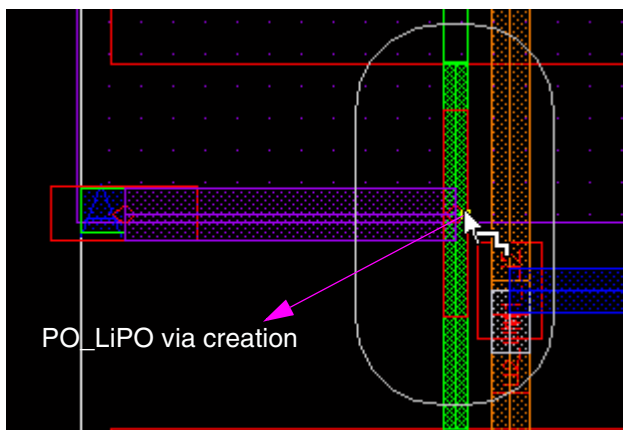


1

2

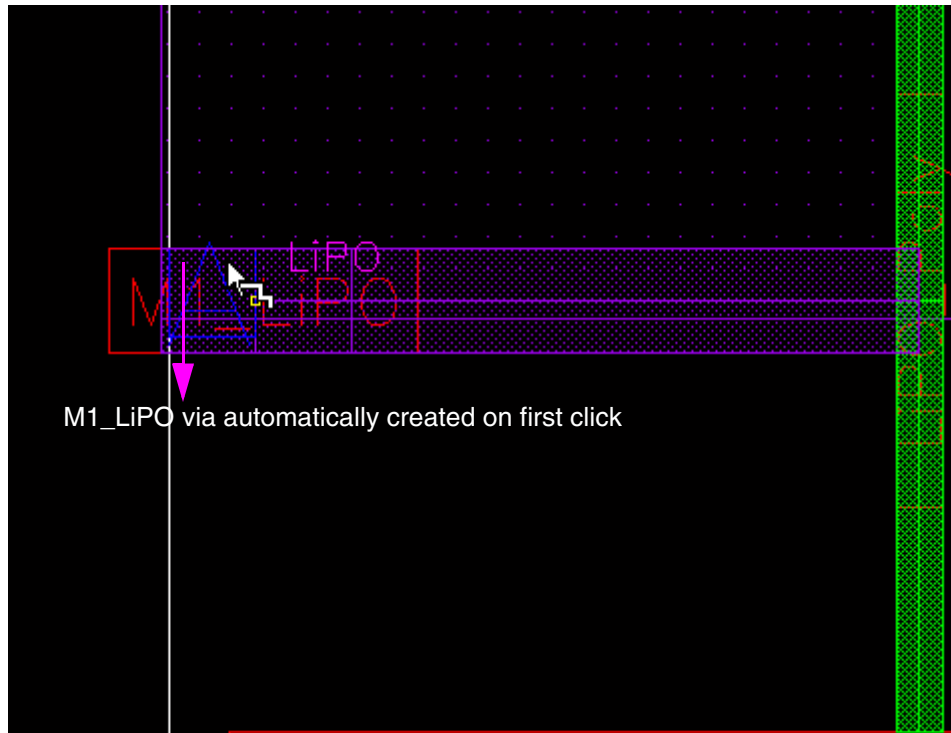


3



Terminating an Li Wire on an Existing Li PathSeg or Rectangle Pin on the Same Net

When the cursor crosses an adjacent `Meta11` layer on the same net, the via is automatically generated and displayed on the end of the cursor, as shown in the following figure.

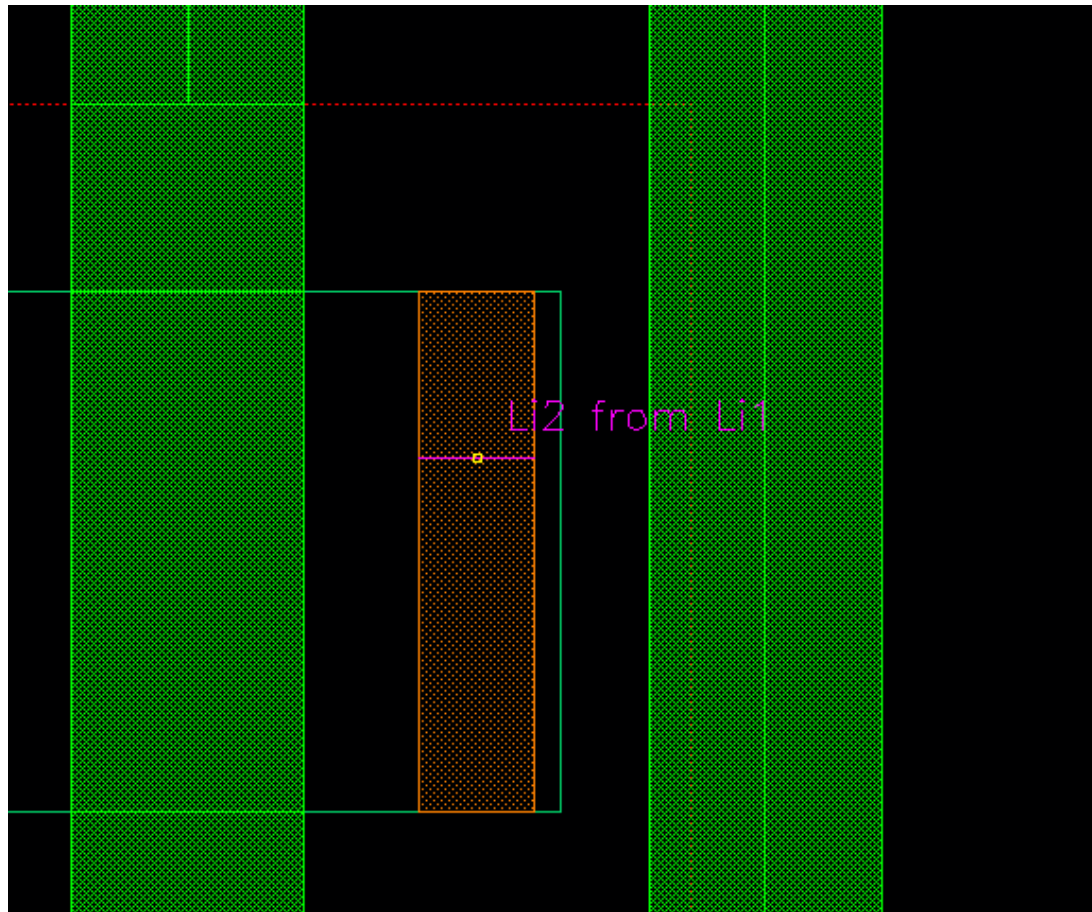


The via origin of the created via is snapped to the center of the wire or pin on the `Meta11` layer. When the wire created on the `Li` layer is dragged across the adjacent layer boundary, the created via disappears and wire creation continues.

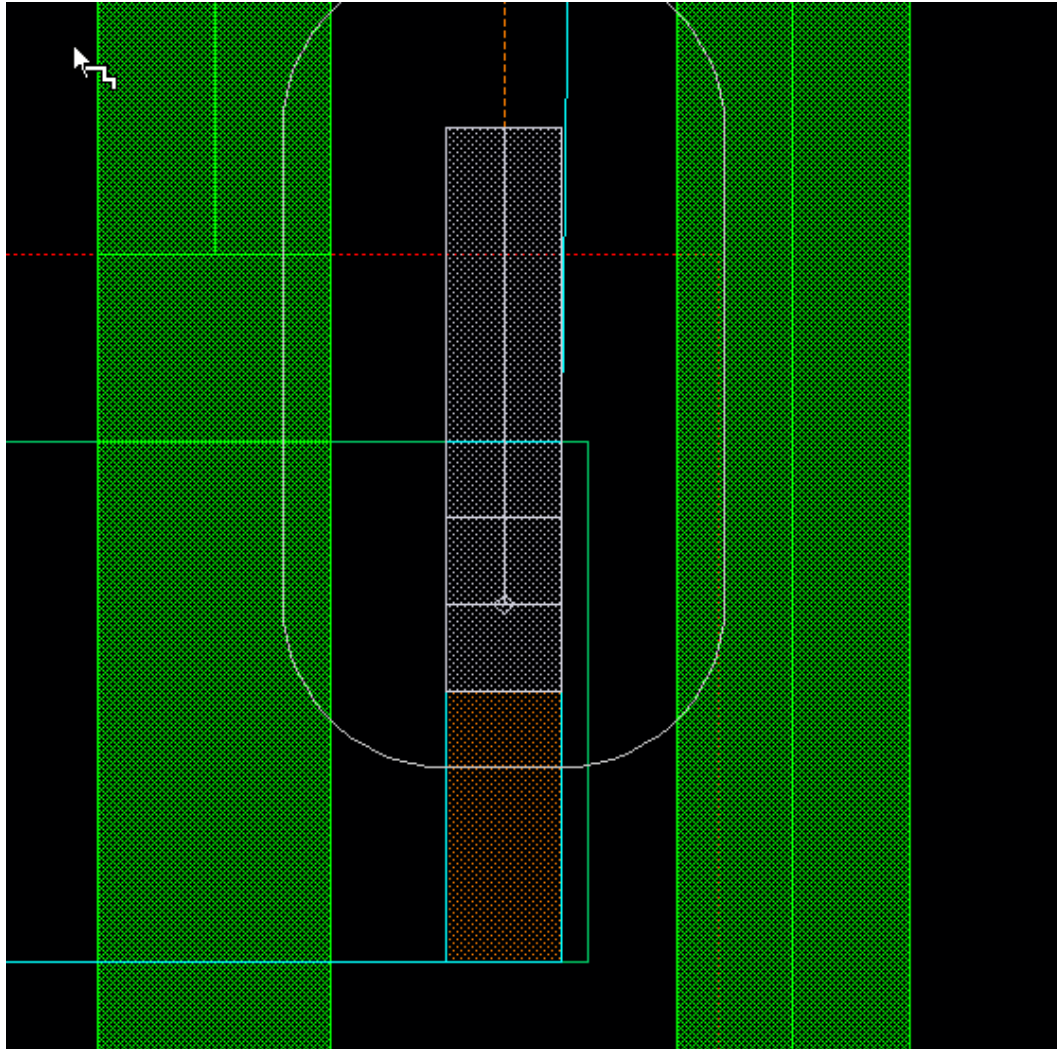


Snapping Via to Centerline of a Pin

While placing a via on an existing pin, the via is snapped to the centerline of the pin, thus avoiding the `rectShapeDir` and `minWireOverlap` violations, as shown in the following figure.



The following figure displays smart snapping to the centerline of the pin and a via automatically being dropped on a Li layer.



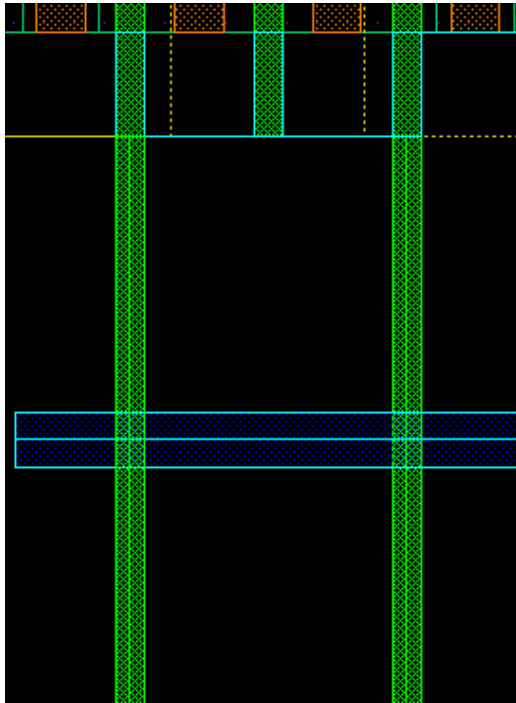
Enhanced Snapping in Wire Editing

In Wire Editing you can automatically snap a wire to the intersection of existing and crossing wires on the same net. This basically allows you to create local interconnect wires and avoids non-rectangular shape violations and small wire segments creation. The following figure

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

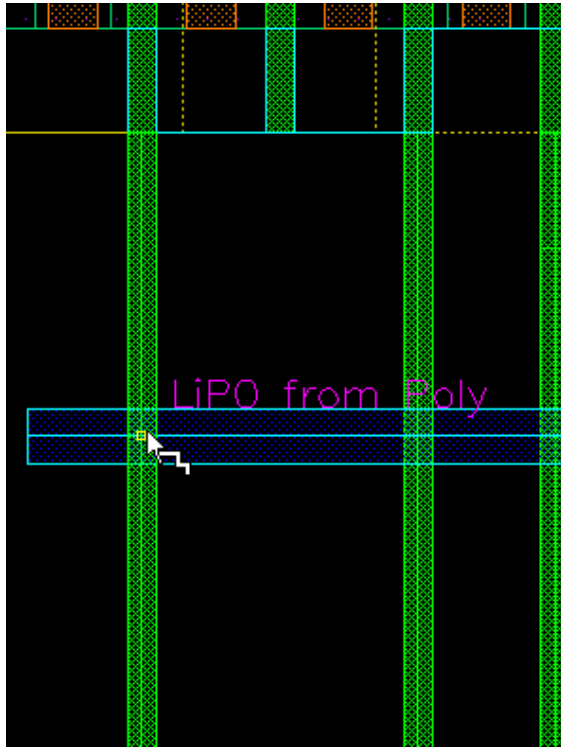
shows an example where two parallel poly wires are connected using `LiPO` over an already existing crossing `Metal1` wire.



Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

Start creating a wire with the current layer as `LiPO`. The cursor is automatically snapped to the intersection between the `Poly` wire and the crossing `Metal1` wires.

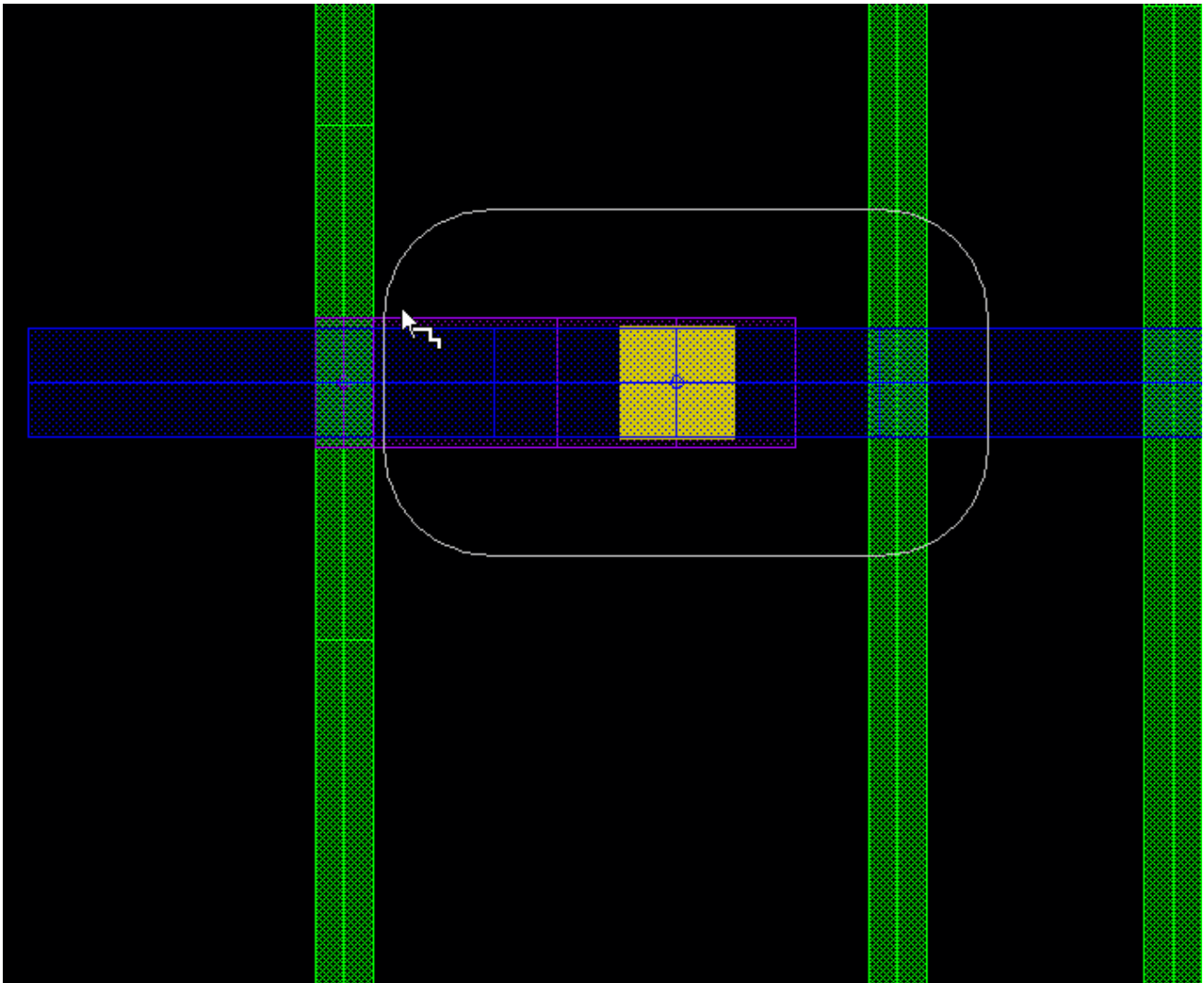


The first click automatically drops a `PO_LiPO` via at the intersection, preventing any `rectShapeDir` violation and creation of small `PO` segments at the intersection. As the `LiPO`

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

wire is being dragged, Wire Editor automatically adds a *LiPO_M1* via as if the connection would terminate on the overlapping *Metal1* wire.

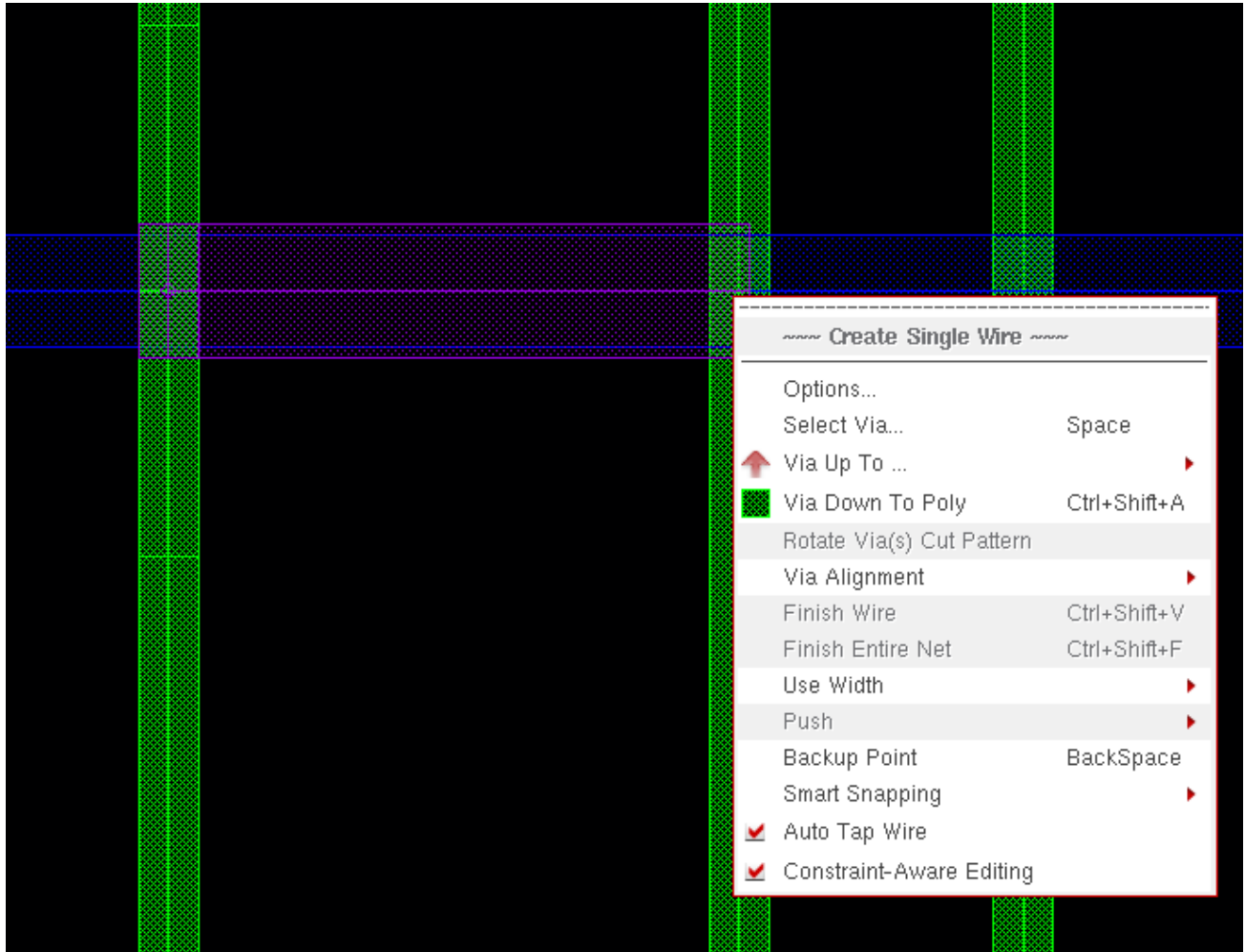


You drag the *LiPO* wire to the right, it overlaps both the *Metal1* wire and the target *PO* wire. The automatically added via then disappears since Wire Editor cannot make an arbitrary

Virtuoso Interactive and Assisted Routing User Guide

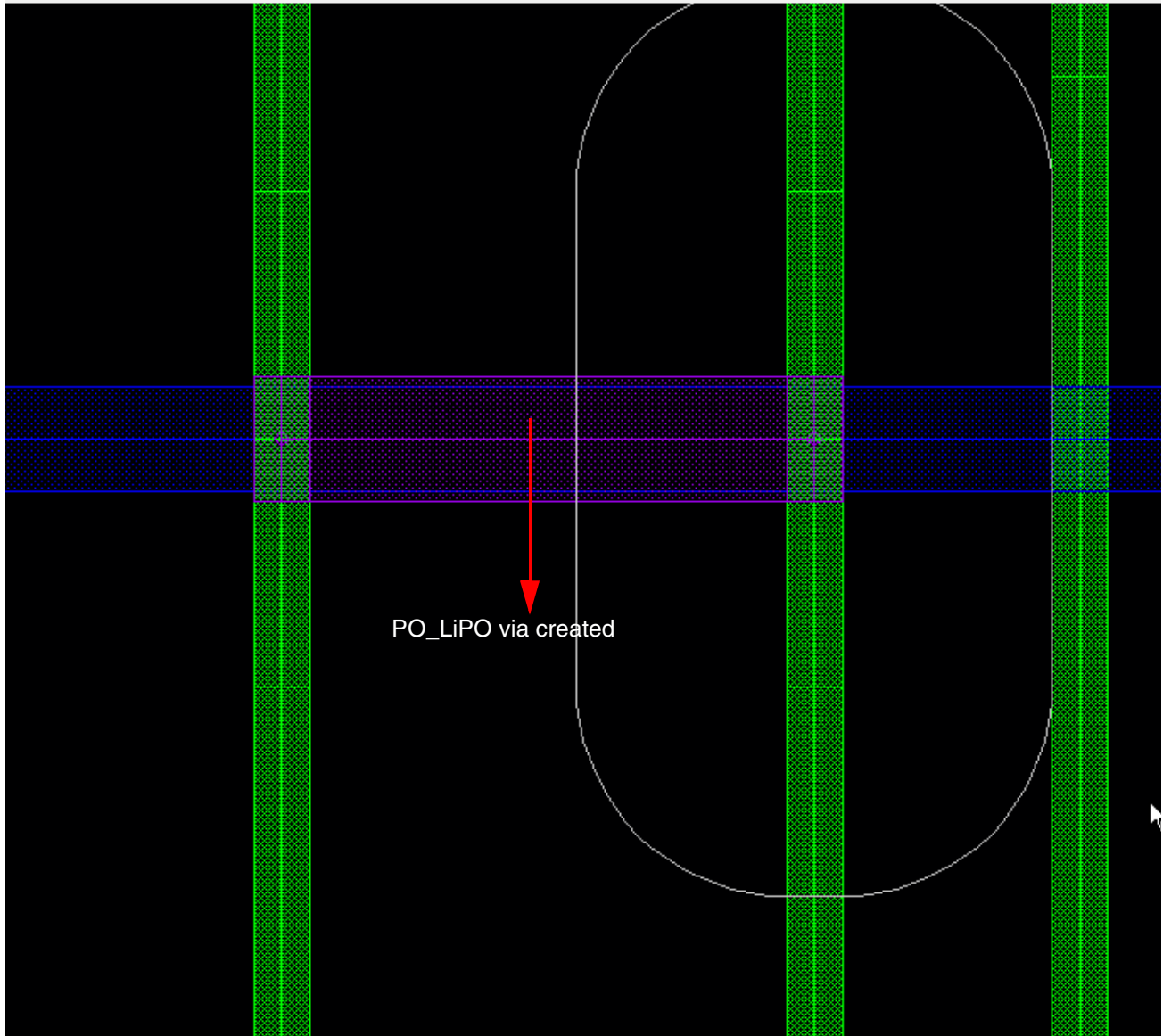
Working with Wires

choice between the two target layers. Therefore, in order to complete the connection to the PO wire, right-click and select *Via Down* to PO in the context-sensitive menu.



The via origin is snapped to the center of the Poly wire. The centerlines of LiPO and Metal1 are aligned while the PO_LiPO via is created and the alignment marker is created to show

that the centerlines are coincident. The via is created on `LiPO` as shown in the following figure.



Smart Snapping in Interactive and Assisted Routing Commands

When you start creating a wire, you can switch on *Smart Snapping* on the Create Stranded Wire Form to enable snapping to the starting and target pins. You can use the smart snapping feature with vias as well (only starting vias) when snapToViaCenter is off (default). By

default, *Smart Snapping* is on. When both Smart Snapping and Gravity are enabled, Smart Snapping is used and Gravity is ignored. To turn off smart snapping, switch off *Enabled* on the *Smart Snapping* sub-menu on the *Create Single Wire* context-sensitive menu.

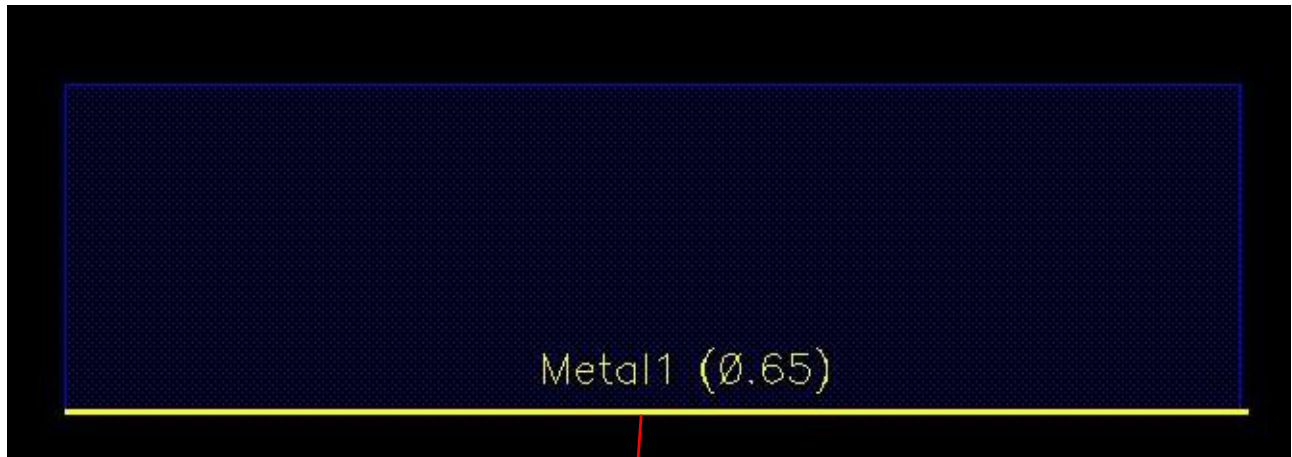
Note: The Smart Snapping is also supported for polygonal pins.

By using the *Cycle Edge* command from the *Smart Snapping* sub-menu, when you bring the pointer close to the potential starting pins, the edge of the pin within the *Aperture* distance of the pointer highlights. You can cycle through the available edges by pressing `Ctrl` and `Spacebar` keys or by using *Cycle Edge*. Each time you press the key combination or choose *Cycle Edge*, the next possible target within the *Aperture* distance highlights. The length of the highlight indicates the width of the wire to be created and depends on the *Use Width* option selected from the Create Stranded Wire Form. To select a highlighted edge for

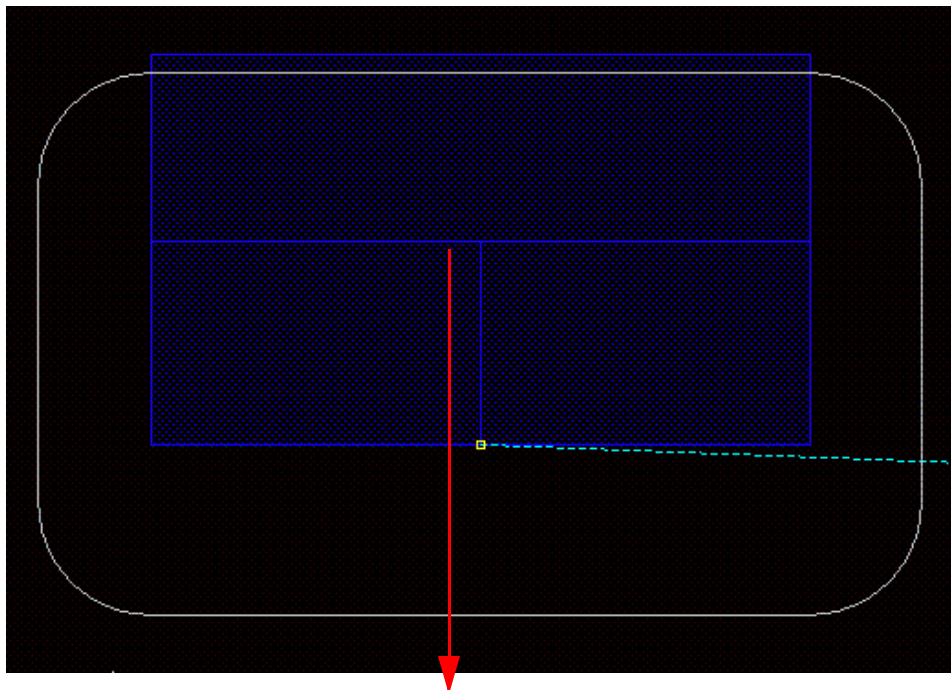
Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

wire creation, click at the pointer location. Digitize subsequent points of the wire. The following figure displays smart snapping on the wider edge of the pin.



Highlighted Edge of the Pin

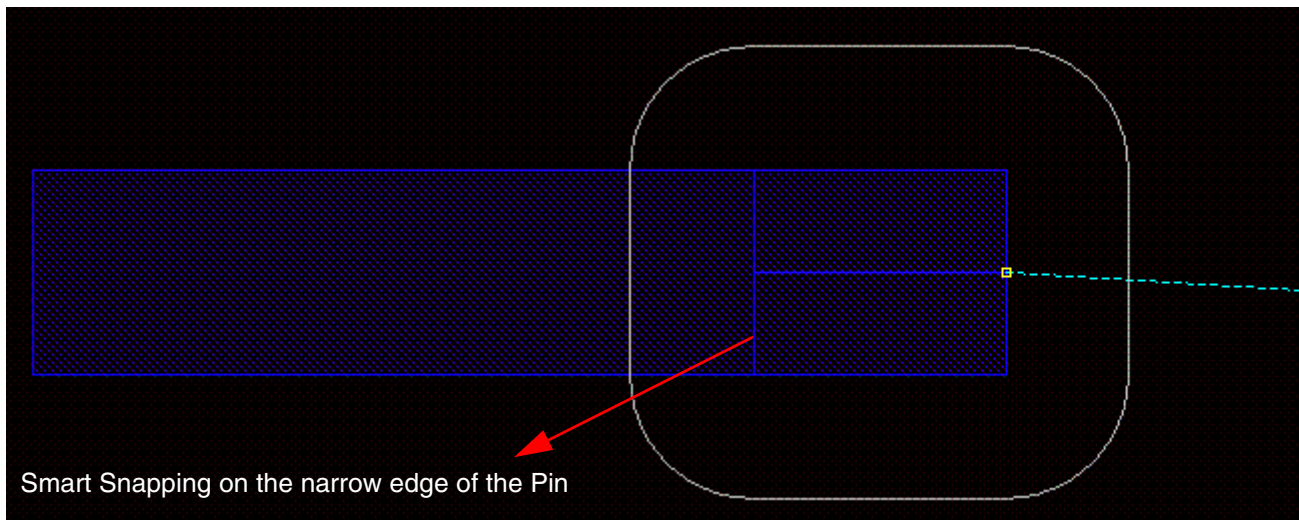
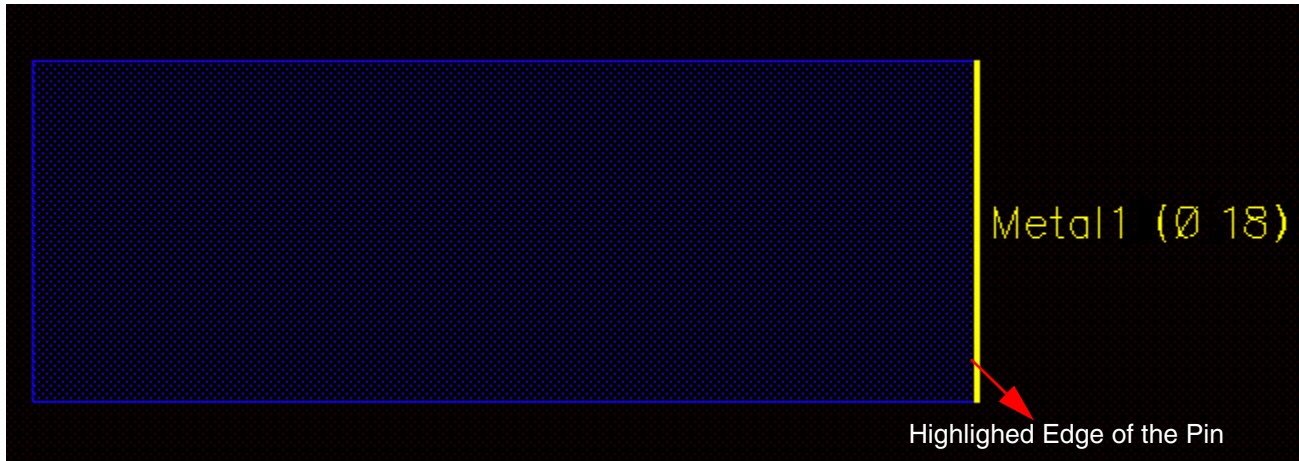


Smart Snapping on the wider edge of the Pin

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

The following figure displays smart snapping on the narrow edge of the pin.

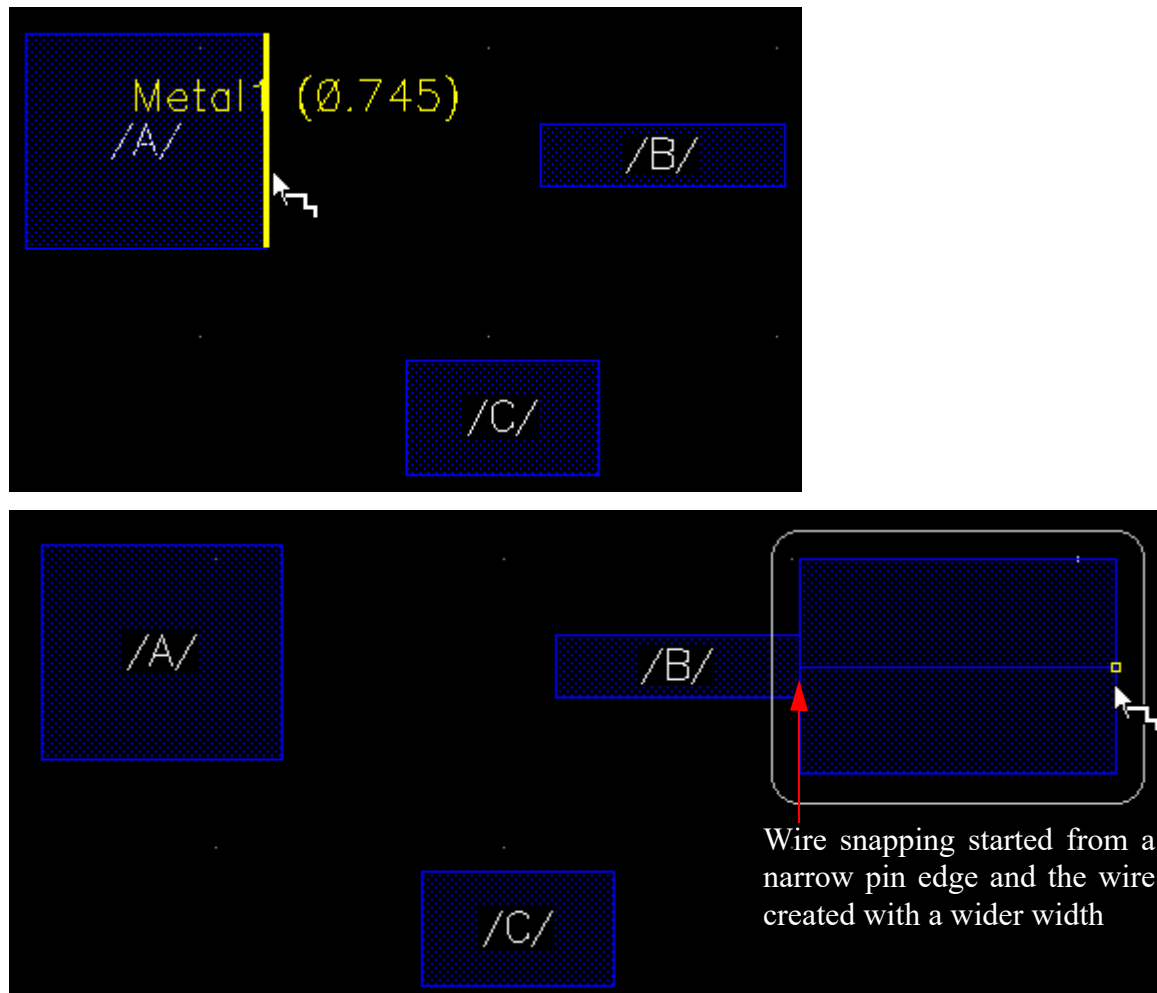


While snapping, you can save the smart snapping object value using the `Ctrl+L` bindkey. You can then use the saved smart snapping value from a different pin edge with a different

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

width value. The following figure shows that a wider wire is created when you start creating it from the pin edge which is less wider than the saved value of the pin edge.



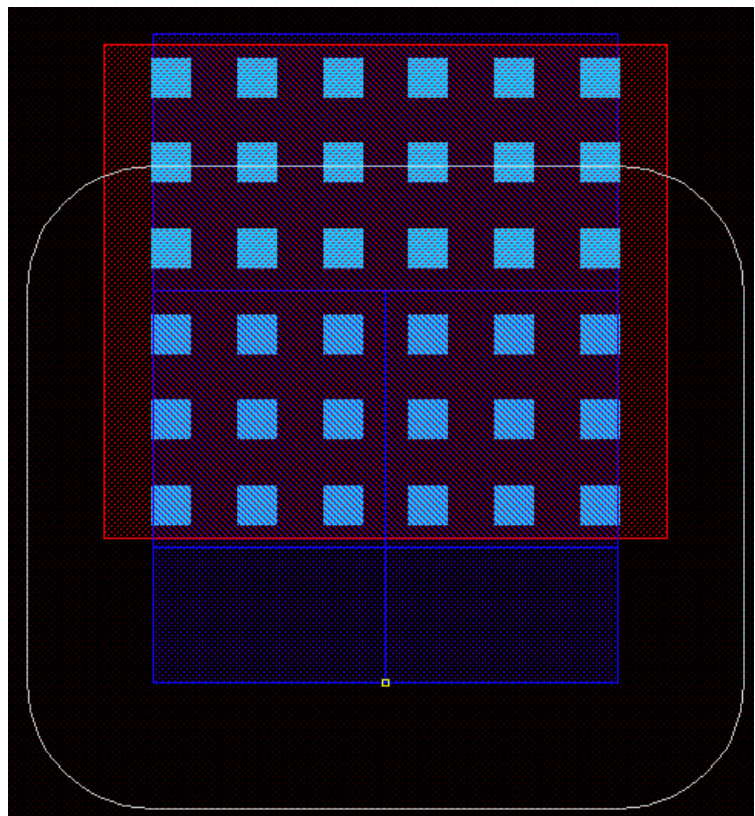
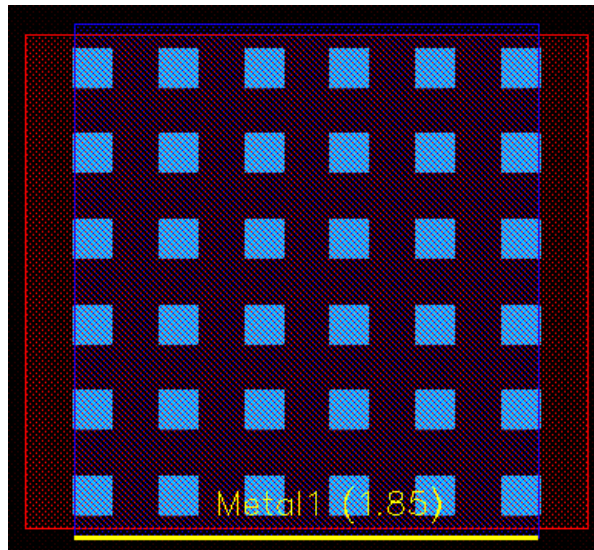
To clear the saved value and return to the original value of the pin edge, press the `Ctrl+L` key again. Also, the saved value is cleared when the mouse pointer does not snap to any pin edge.

You can toggle to the *Use Current Position* mode by pressing together the `Ctrl`, `Shift`, and `Spacebar` keys or by choosing *Use Current Position* from the *Smart Snapping* sub-menu. In this mode, you can start or end a wire in empty space as well as from or at a pin.

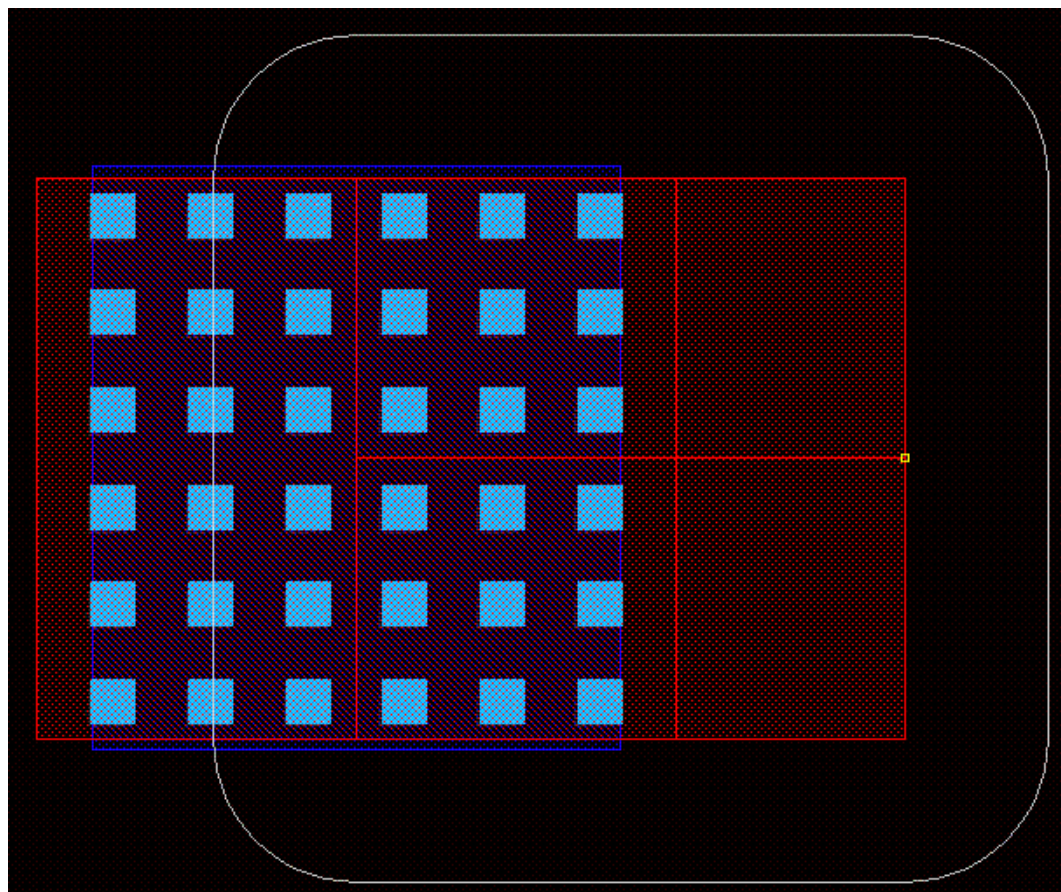
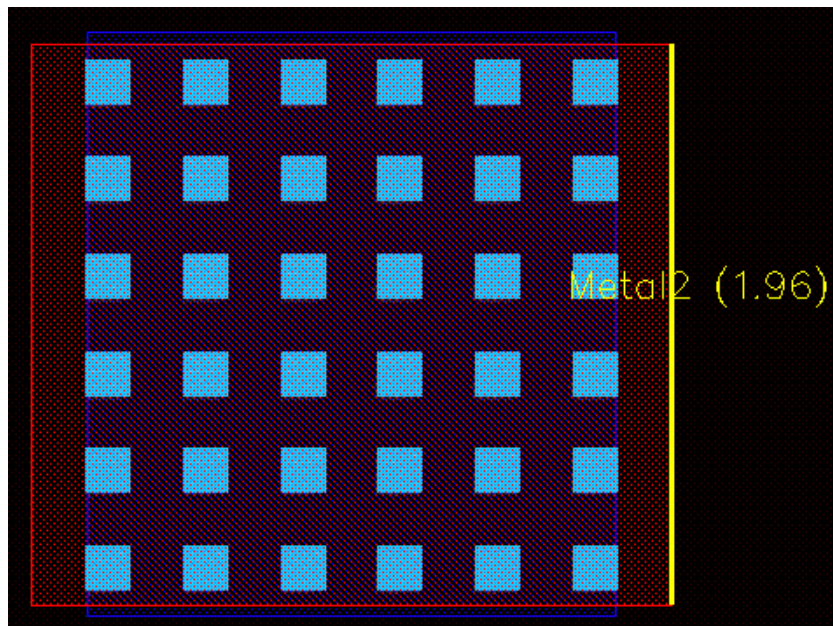
While creating a wire, if *Smart Snapping* is on and the *Snap to Via Center* check box in the *Create Stranded Wire Form* or in the *Interactive Section* of the Wire Assistant is selected, you can cycle through the available via shapes by pressing `Ctrl` and `Spacebar` keys successively. You can toggle to the *Use Current Position* mode by pressing together the `Ctrl`, `Shift`, and `Spacebar` keys or by choosing *Use Current Position* from the *Smart*

Snapping sub-menu. In this mode, smart snapping is turned off and you select a via by using the *Choose object to tap from* form (see [Tapping Vias](#) for more information).

The following figure shows a Metal1_Metal2 via and displays smart snapping on the Metal1 layer of the via.



The following figure shows smart snapping on the `Meta12` layer of the via.

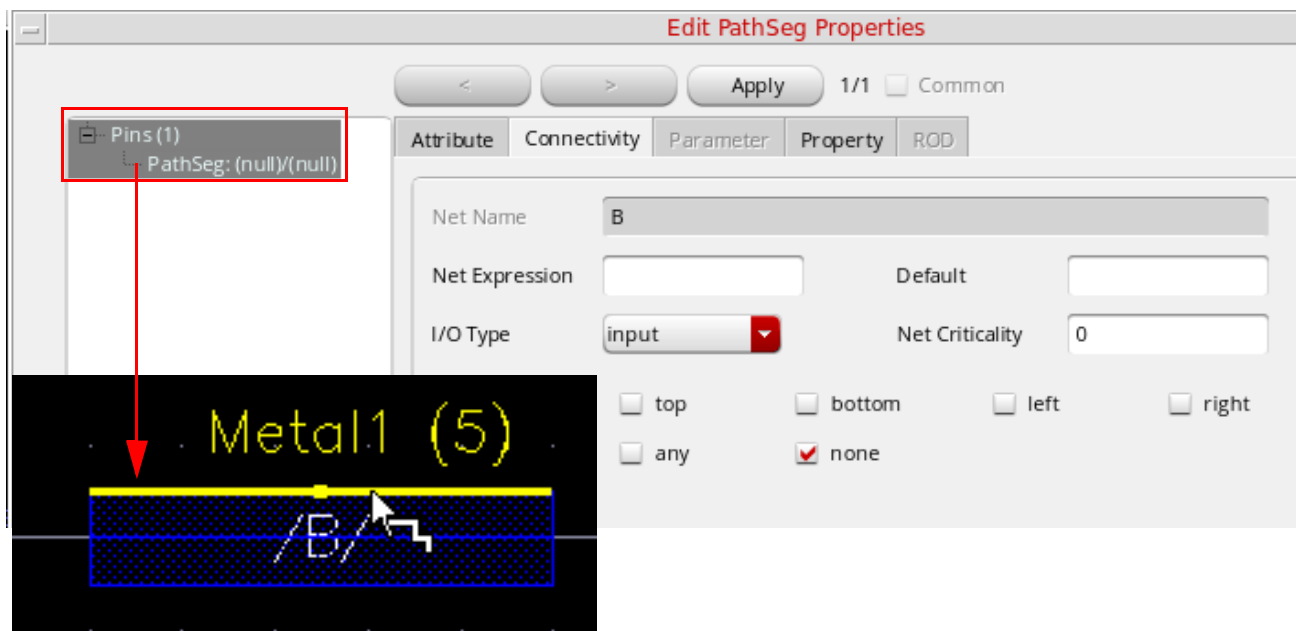


Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

Note: If you select a layer from the *Layers* assistant before clicking a via, you can start wire creation on the layer selected from the *Layers* assistant and bypass the *Choose object to tap from* form. Also, when one of the overlapped layers on which the wire is being created is already selected in the *Layers* assistant, the *Choose object to tap from* form is bypassed. The wire snaps to the via center if the *Snap to Via Center* check box is selected.

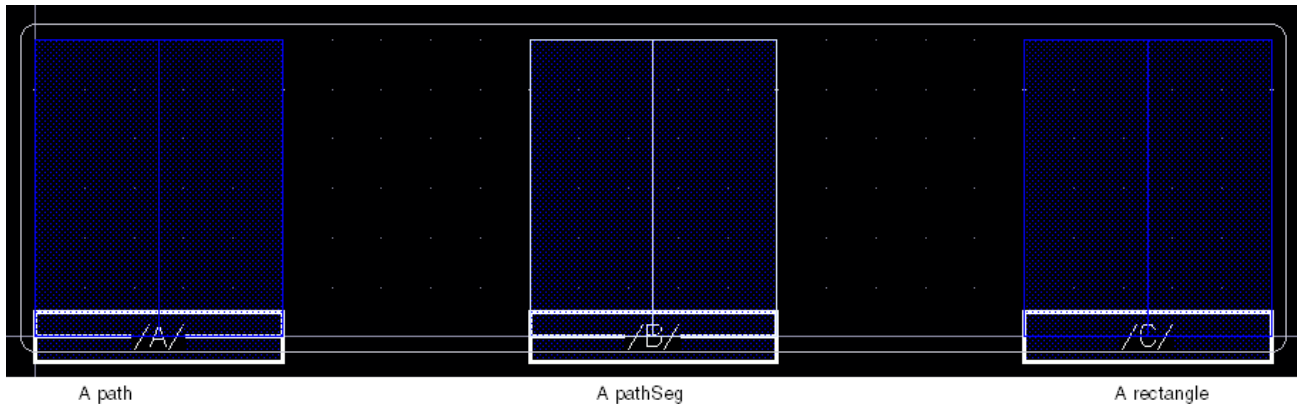
The smart snapping is now supported for path segments. When a path segment is defined as a figure of pins, smart snapping is supported as if it was a rectangle or a polygon. The following figure displays smart snapping on the edge of the path segment that is defined as a figure of pins.



Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

Similarly, *Create Wire* and *Create Bus* is supported when you area select a pin or instance pins regardless of whether their figures are rectangle, pathseg, or path, The following figure shows the area selection of objects where bus can be simultaneously created.



Also, when you start interactive routing from an instance pin that does not have a net assigned to it using either area select or smart snapping, an unassigned wire is created from it.

You can use the smart snapping feature while using the assisted routing commands, *Point to Point* routing and *Guided Routing*. The smart snapping feature, while using these assisted routing commands, enables you to select a starting and ending layer, without bringing up the *Choose object to tap from* form.

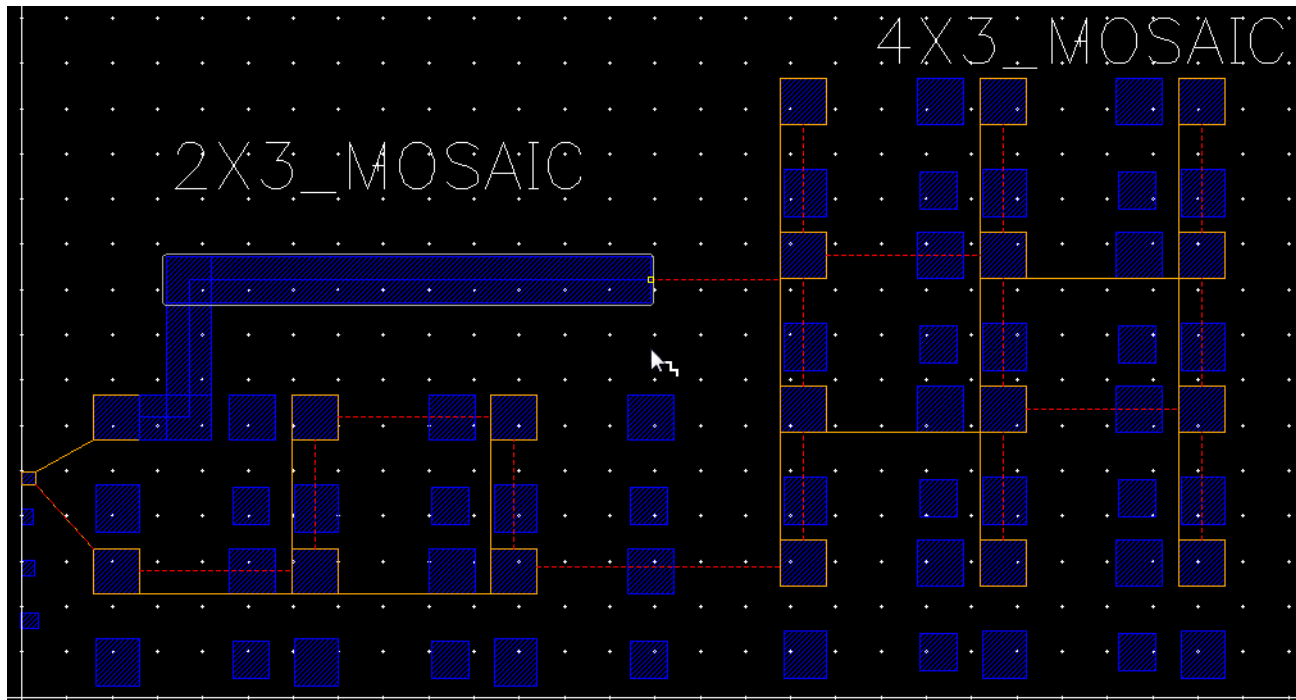
Disabling Layer Tapping, Via Insertion, and Smart Snapping

You can use the `Shift`, `Ctrl`, and a combination of `Shift + Ctrl` keys to temporarily disable layer tapping, via insertion, and smart snapping.

- Before the first click, hold the `Shift` key to disable layer tapping and via insertion. The layer selected in the palette assistant takes precedence over the layer of the tapped object.
- Before the first click, hold the `Ctrl` key to disable layer tapping and enable via insertion. The layer selected in the palette assistant takes precedence over the layer of the tapped object. Also, an additional via or via stack is created to maintain the physical connection between the tapped object and the wire being created.
- Before the first click, hold the `Shift + Ctrl` keys to disable smart snapping. The layer of the tapped object or pin takes precedence over the selected layer in the palette assistant.

Support for Mosaics in Create Wire

The *Create Wire* command supports mosaics. When starting a wire from an instance pin of a mosaic, Create Wire can snap automatically to the instance pin of the other mosaic. The following figure shows a wire automatically snapping to an instance pin of a mosaic. You can also see the flightlines while snapping to an instance pin of a mosaic.



For more information on mosaic support in VLS XL, see [Generating Mosaic Instances](#).

Support for cdsGenVia for Different Purposes

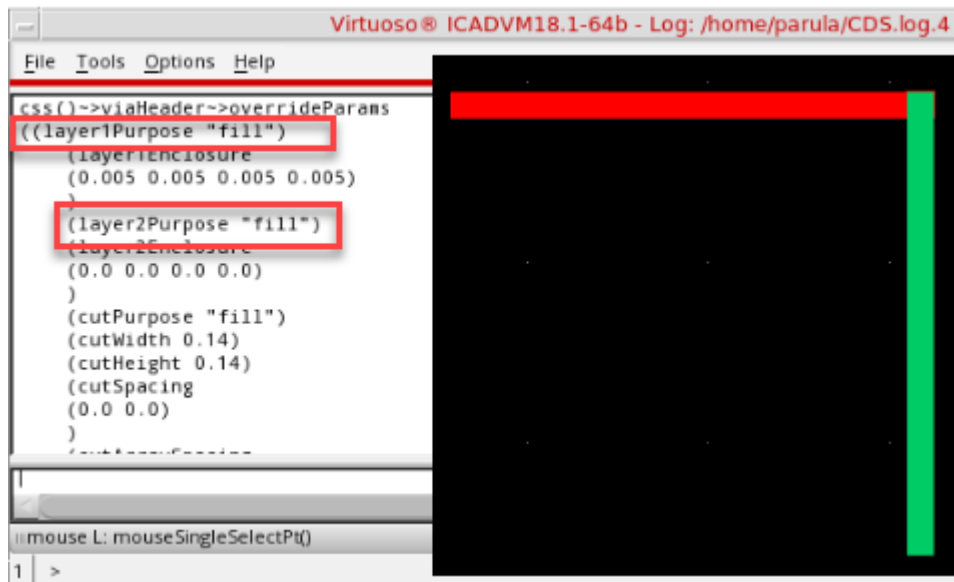
Interactive routing commands support `cdsGenVia` for different purposes other than drawing, for top or bottom layers. This means that the via is created according to the purpose of the input and output pathSeg.

To insert a `cdsGenVia` object:

1. Choose *Create – Wiring – Wire*.
2. Select a layer purpose, for example `Metal1 fill`, in the Palette assistant.
3. Click and drag to create a wire in an empty space.

4. Right-click the wire and choose *Via Upto Metal2*.

The cdsGenVia with purpose fill is inserted between the Metal1 and Metal2 layers.



Setting Dynamic Measurement for the Area Calculation

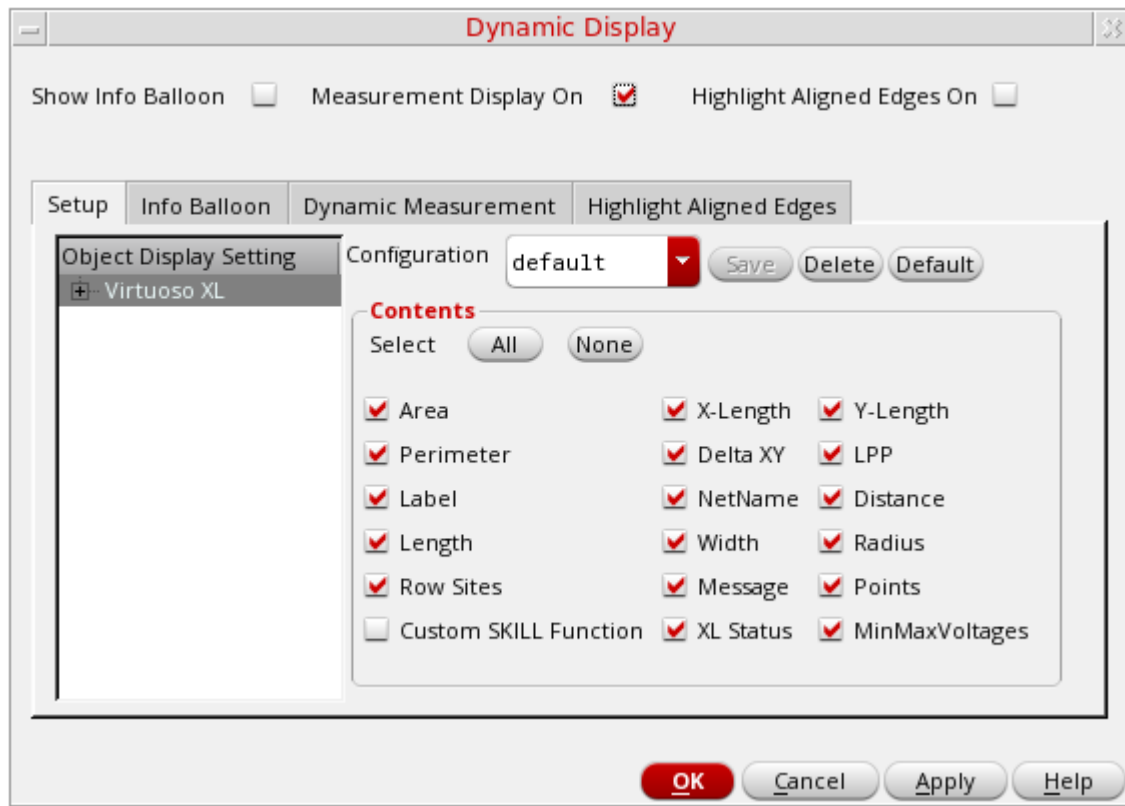
To set dynamic measurement for the area calculation of a wire, do the following:

1. Choose *Options – Dynamic Display*. The Dynamic Display form displays.

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

2. Select *Measurement Display On*.



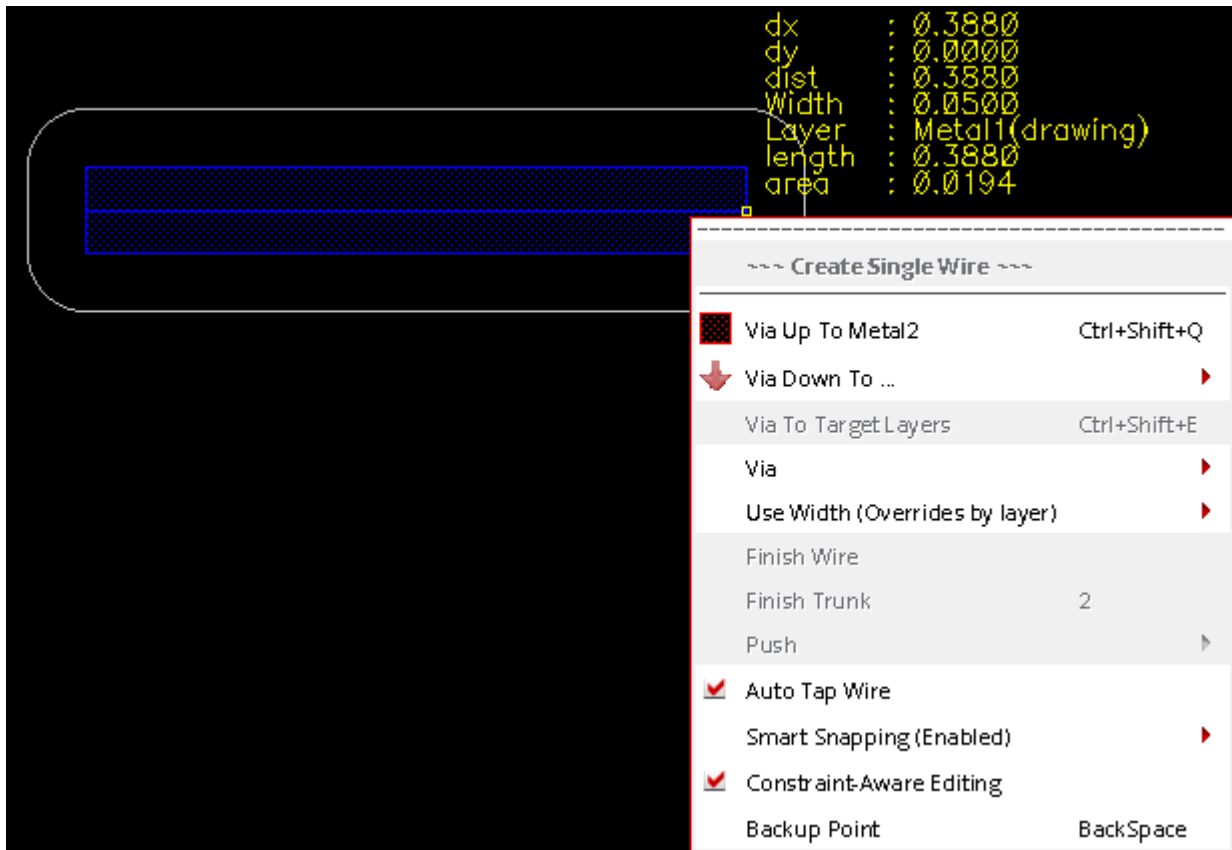
3. Select `Meta11` from the Layers palette

4. Choose *Create – Wiring – Wire*. Now, start creating the wire on `Meta11`.

Virtuoso Interactive and Assisted Routing User Guide

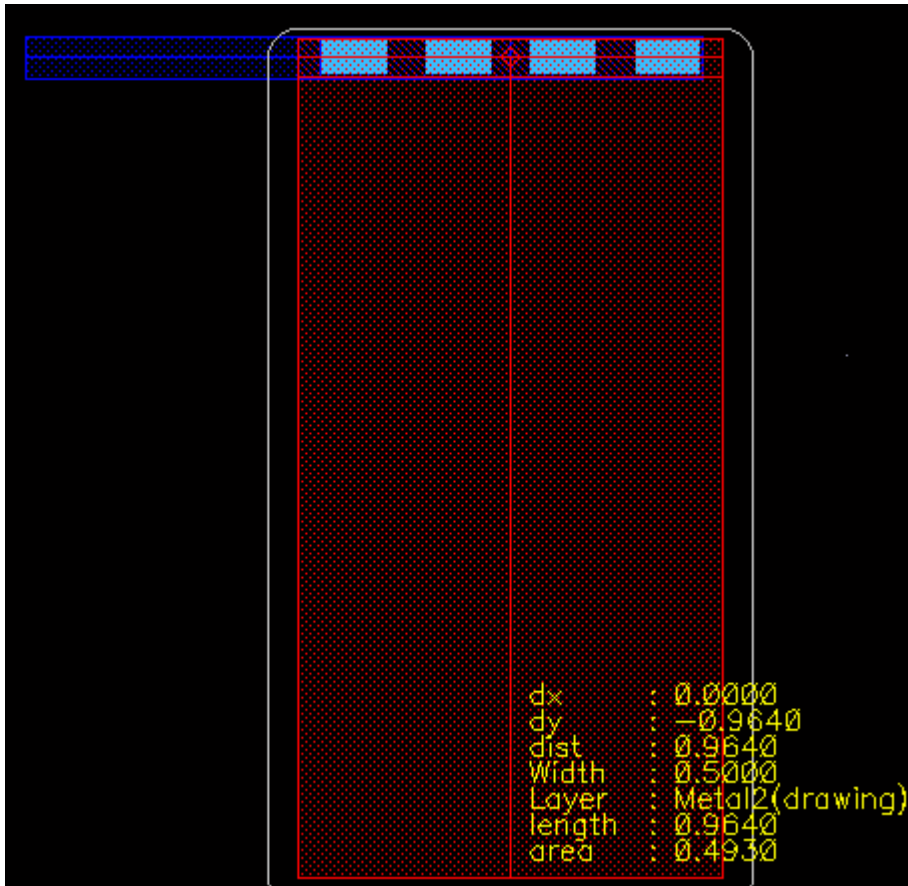
Working with Wires

5. Right-click on the wire being created and select *Via up to Metal2* from the Create Wire context-sensitive menu.



You can see that the area only includes the wire of Metal1.

6. Next, change the width to 0.5 in the *Width* field of the Create Wire form. The Create Wire form is displayed when you press the F3 key. The area includes the placed via and wire of Metal2.



Support for Pixel Based Gravity

Interactive routing commands now support the pixel-based gravity feature implemented in Layout L. The gravity distance setting in the Layout Editor Options form is considered while creating a wire, bus, or while stretching a wire.

Displaying Wire Summary

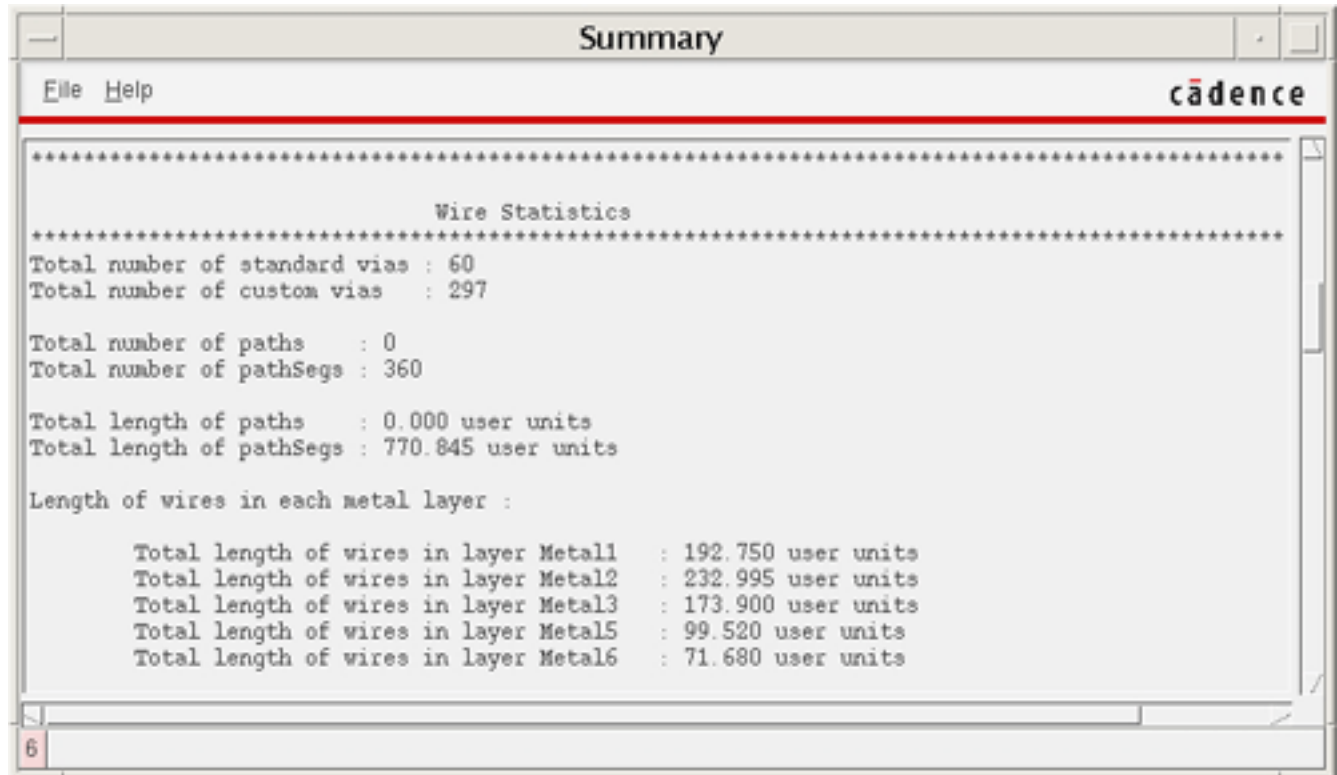
To display a summary of the contents of the cellview:

- ➔ Choose *File – Summary*.

Virtuoso Interactive and Assisted Routing User Guide

Working with Wires

A window containing the summary appears.



Working with Vias

Via definitions must exist in the technology file in order to be available for interactive and assisted routing.

This chapter describes the following topics:

- [Specifying a Valid Via List](#)
- [Specifying Via Parameters Calculation Mode](#)
- [Changing Layers and Selecting Vias](#)
- [Inserting Vias Automatically](#)
- [Fanning Out Wires to Vias](#)
- [Using Via Assistance](#)
- [Creating an Array of Vias](#)
- [Stacking Vias](#)
- [Changing Via Cut Color](#)

In 20nm, the `cdsGenVias` are supported by wire editor. For more information, refer to [cdsViaDevice](#).

Specifying a Valid Via List

The vias that are available for wire editing and creation are determined from the *Via* constraint group specified in the *Create Via* section in the [Layout Editor Options form](#). Only the vias specified in the `validVia` list in the selected constraint group will be used when creating wires. See [Constraint Groups](#) for more information about defining constraint groups.

In addition to standard and custom `viaDefs`, the interactive and assisted routing commands, such as *Create Wire* and *Point to Point*, support standard and custom via variants.

Note: Interactive routing commands does not support table-based `validVia` constraints.

For information about adding customized vias to the technology file, see [Technology File Via Definitions and Via Specifications](#) in the [Virtuoso Technology Data ASCII Files Reference](#).

Specifying Via Parameters Calculation Mode

For creating a via and the via enclosure explanation, the interactive and assisted routing commands, such as *Create Wire* and *Point to Point* routing, use the *Via Parameters Calculation mode* that is specified in the *Wire Editing* section of the [Layout Editor Options form](#). Depending on the via calculation mode, the results of the assisted routing commands are changed.

- When *Min Rules & ViaDef* is specified, the interactive and assisted routing commands use the larger via parameter value found in the minimum rules and the viaDef.
- When *Minimum Rules* is specified, the interactive and assisted routing commands use via parameter values from the minimum rule set. If a parameter value is not found in the minimum rules, value from the viaDef is used.
- When *ViaDef Defaults* is specified, the interactive and assisted routing commands use via parameter values from the viaDef which are legal. Whenever a parameter value in the viaDef is less than the minimum rule, a warning is issued and the value of the minimum rule is used instead.

Changing Layers and Selecting Vias

For more information about creating wires, see [Creating Wires](#).

You can now change a layer at any time during interactive routing. While creating a wire, you can change the layer after dropping a via or a via stack by selecting outgoing wire layer in the Palette Assistant. To change the wire layer while creating a wire:

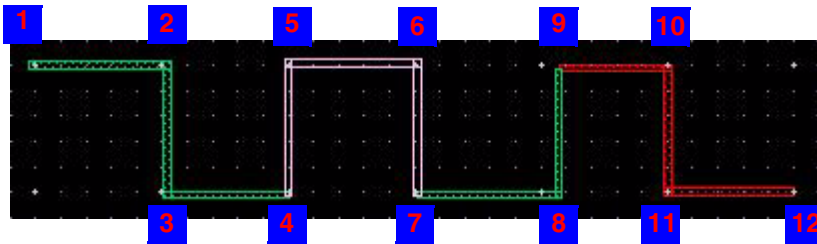
1) Use the *Rotate Via(s) Cut Pattern* and *Via Down to <layer name>* commands on the [Create Stranded Wire Form](#) when the *Create Wire* or *Create Bus* command is active:

1. Start the *Create Wire* command.
2. Start creating the wire by digitizing the wire points.
3. Right-click to select *Via Upto <layer name>* or *Via Down to <layer name>* to change the wire layer to a higher or lower layer, respectively.

Virtuoso Interactive and Assisted Routing User Guide

Working with Vias

4. Continue digitizing points on the new layer.

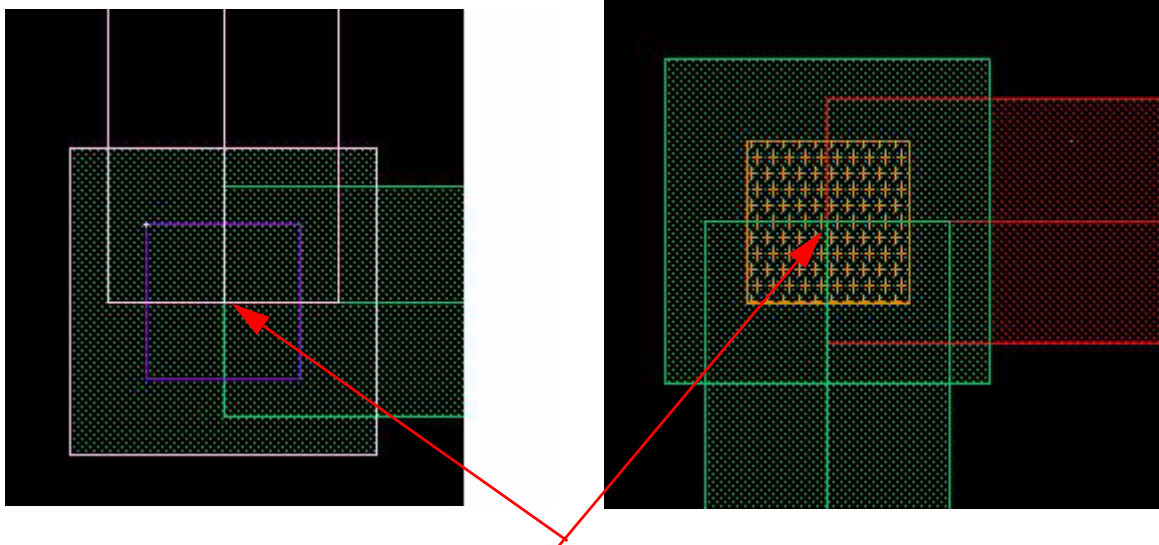


1-12 represent the digitized points of the wire.

4, 7, and 9 represent the via digitization points, where the wire changes to a different layer.

5. Complete the command by pressing `Esc`.

Notice that the center of layer enclosures of the created vias coincides the intersecting centerlines at the endpoints of the two wire pathSegs on different layers. This is shown in the figures below. The first figure shows a zoomed-in view of point 7 of the wire above and the second figure shows the zoomed-in view of point 9.



Center of the via layer enclosures overlapping the intersecting endpoints of the two pathSegs

II) By using the *Select Via* command on the Create Stranded Wire Form when the *Create Geometric Wire*, *Create Wire* or *Create Bus* command is active:

1. Start the *Create Wire* command.
2. Start creating the wire by digitizing the wire points.

3. Right-click to select *Select Via* or press the `Spacebar` key.

The Select Via form appears.

4. In the first column, click the target layer button to select the layer you want to change the wire to.

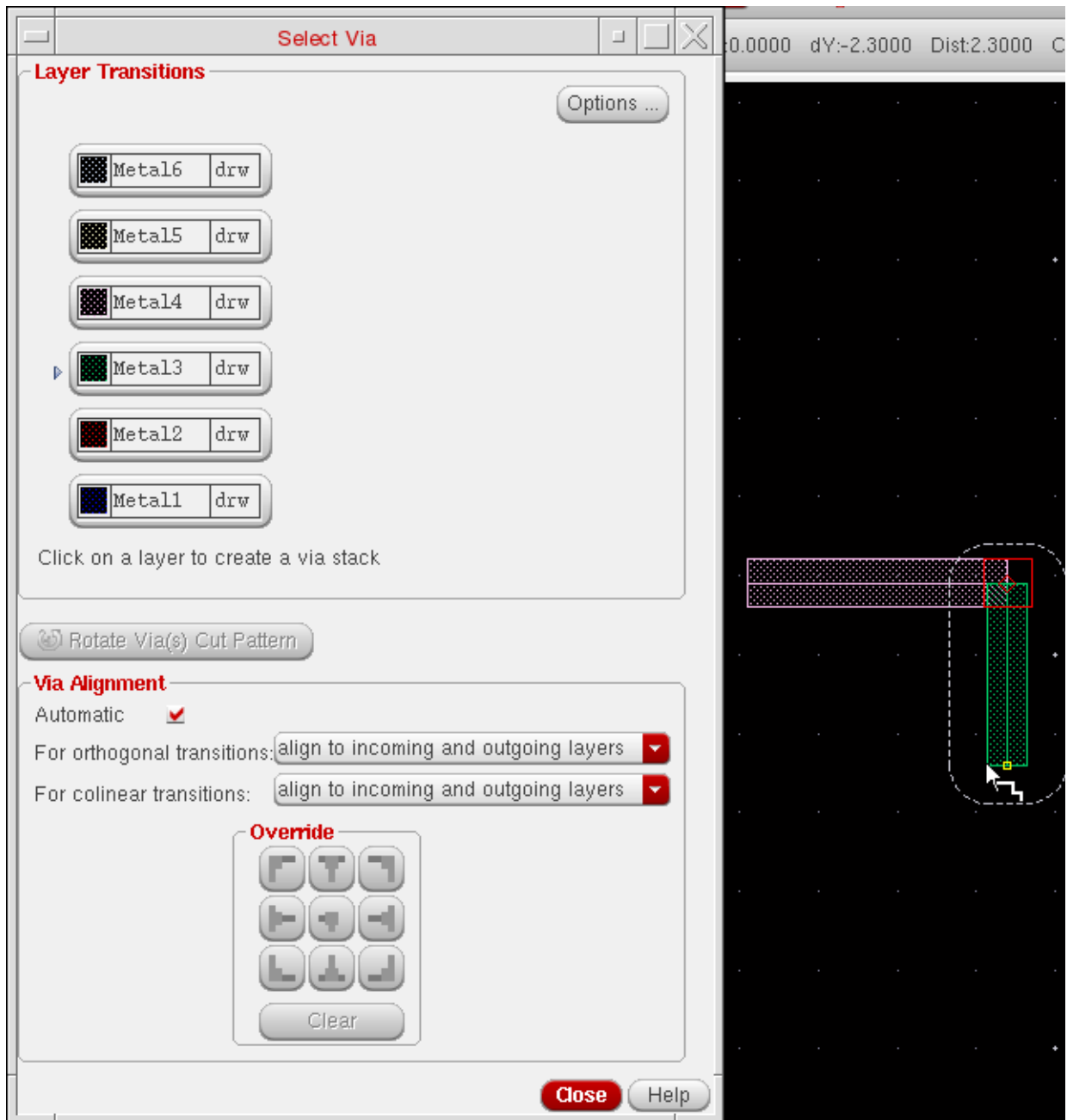
The via in the second column is selected by default when you click the corresponding target layer.

Note: If you select a different via from the cyclic field, click the target layer again to confirm the layer selection.

Virtuoso Interactive and Assisted Routing User Guide

Working with Vias

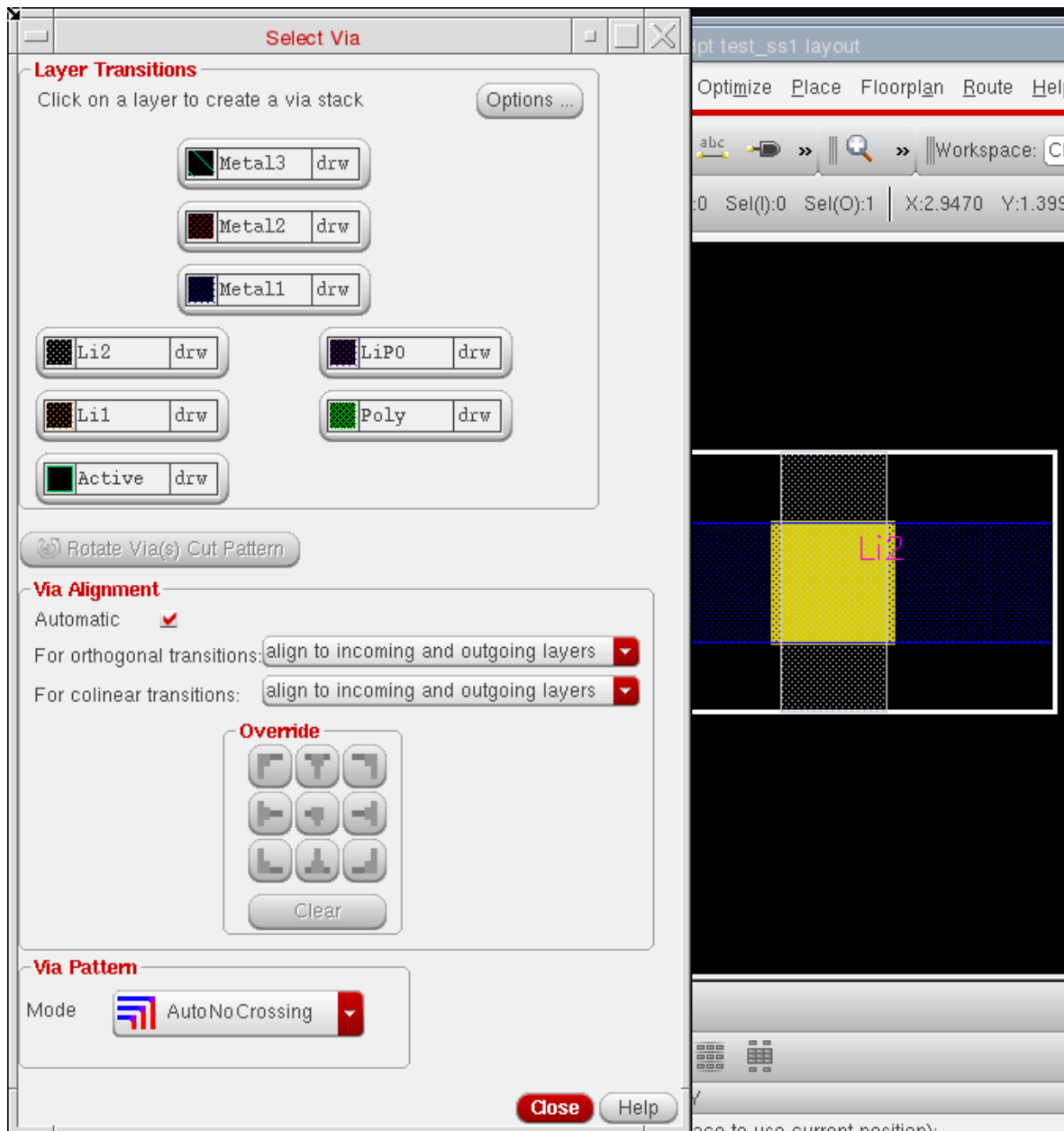
If you move the pointer to the layout window now, a preview of the via appears. The *Select Via* form remains open until you digitize the via.



Select Via Form

Virtuoso Interactive and Assisted Routing User Guide

Working with Vias



Select Via Form

5. While the *Select Via* form is open, you can click other target layers.

Clicking more than one target layer until a via is digitized creates a via stack. At this point, you can do one of the following:

- a. To change the vias in the via stack, click other target layers.

Virtuoso Interactive and Assisted Routing User Guide

Working with Vias

- b.** To remove the via stack, click the current layer button. All vias from the pathSeg end being dragged are removed.
- 6.** To specify the via definition and `cutClass`, click the *Options* button. The Select Via Options form appears as shown in the following figure. For more information about the override variant parameters, cut class and the via definition values, see *Override Variant Parameters*, *Cut Class* and *Via Definitions* in the *Create Via Options Form* in the Virtuoso Layout Suite documentation.

Select Via Options

☐ Override Variant Parameters

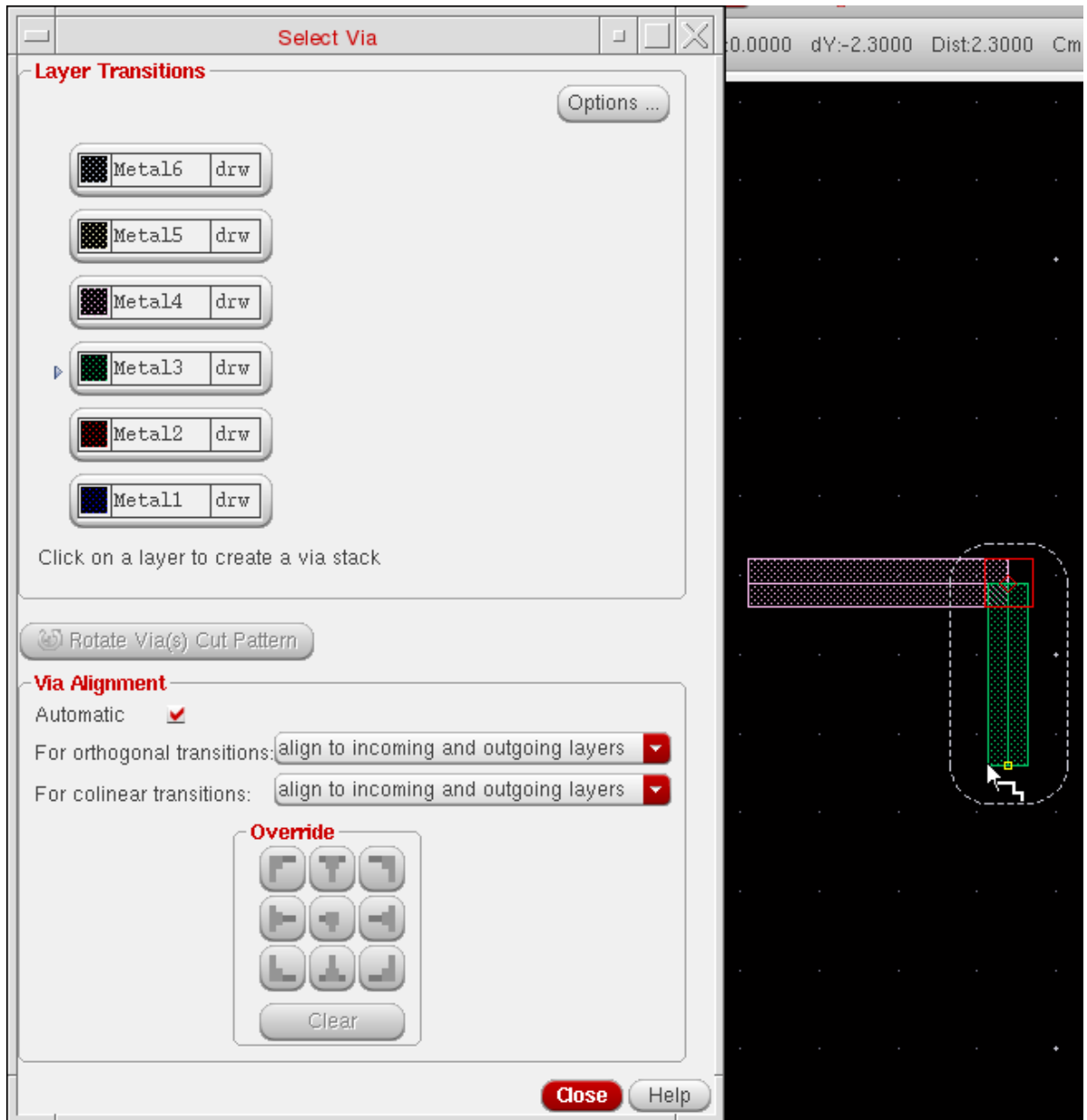
	Via Definitions	Cut Class	User Min Cut Spacing	
			X	Y
Metal10 -> Metal11	M11_M10	None	☐	
Metal9 -> Metal10	M10_M9	None	☐	
Metal8 -> Metal9	M9_M8	None	☐	
Metal7 -> Metal8	M8_M7	None	☐	
Metal6 -> Metal7	M7_M6	None	☐	
Metal5 -> Metal6	M6_M5	None	☐	
Metal4 -> Metal5	M5_M4	None	☐	
Metal3 -> Metal4	M4_M3	None	☐	
Metal2 -> Metal3	M3_M2	None	☐	
Metal1 -> Metal2	M2_M1	None	☐	
Poly -> Metal1	M1_PO	None	☐	
Oxide -> Metal1	M1_NWELL	None	☐	

7. In the *Select Via* form, click the *Rotate Via(s) Cut Pattern* button to rotate the via shape.

Virtuoso Interactive and Assisted Routing User Guide

Working with Vias

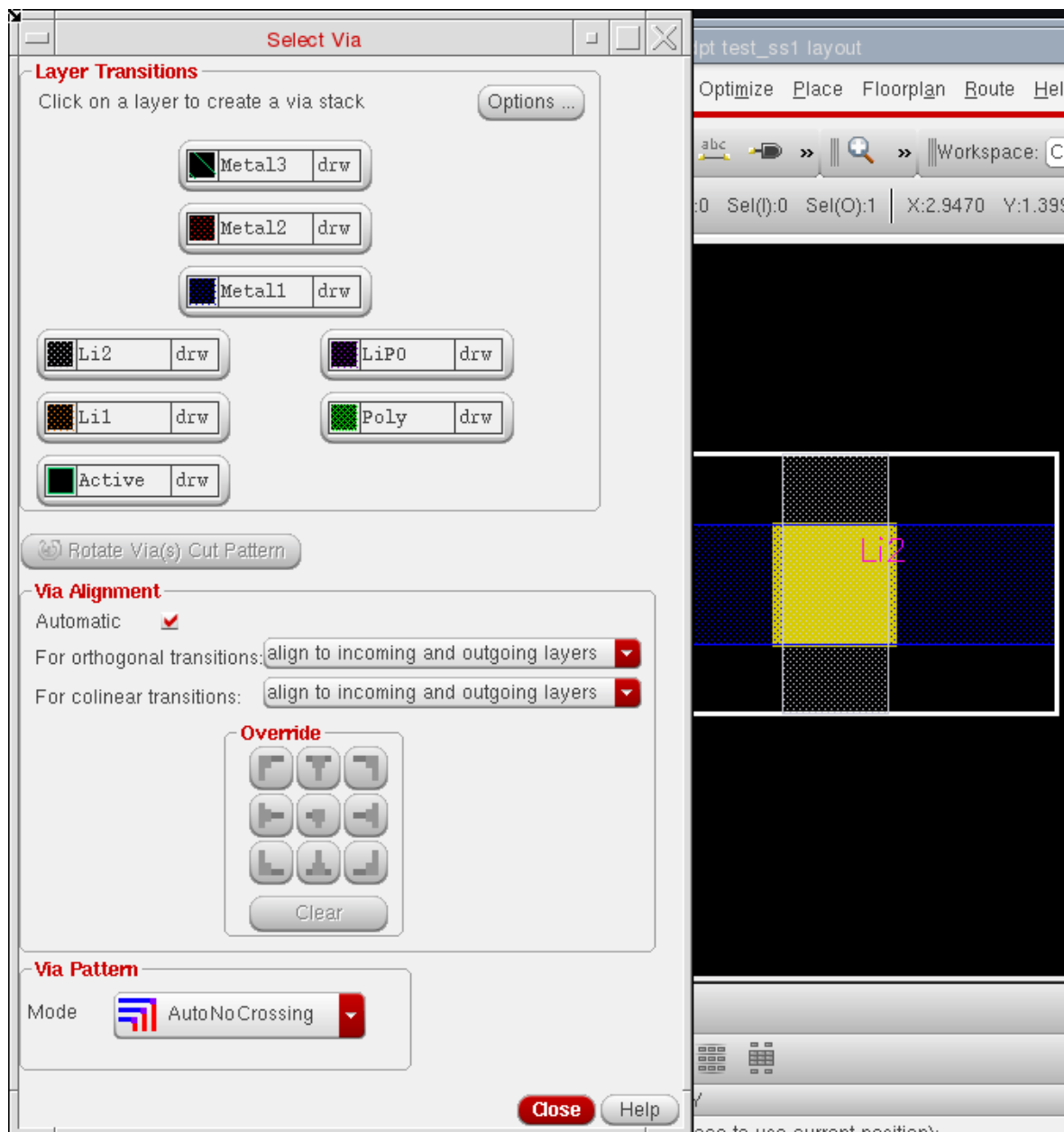
8. In the *Via Alignment* section, select the *Automatic* check box. The *Via Alignment* options are also available on the right-click context menu. Notice that the via now aligns to one side of the existing pathSeg.



Select Via Form

Virtuoso Interactive and Assisted Routing User Guide

Working with Vias



(ICDAV12.2 Only) Select Via Form

9. In the design window, click at a point to digitize the via location.

The *Select Via* form closes. Notice the change in the via alignment with respect to the existing pathSeg when you move the pointer to either direction of the pathSeg.

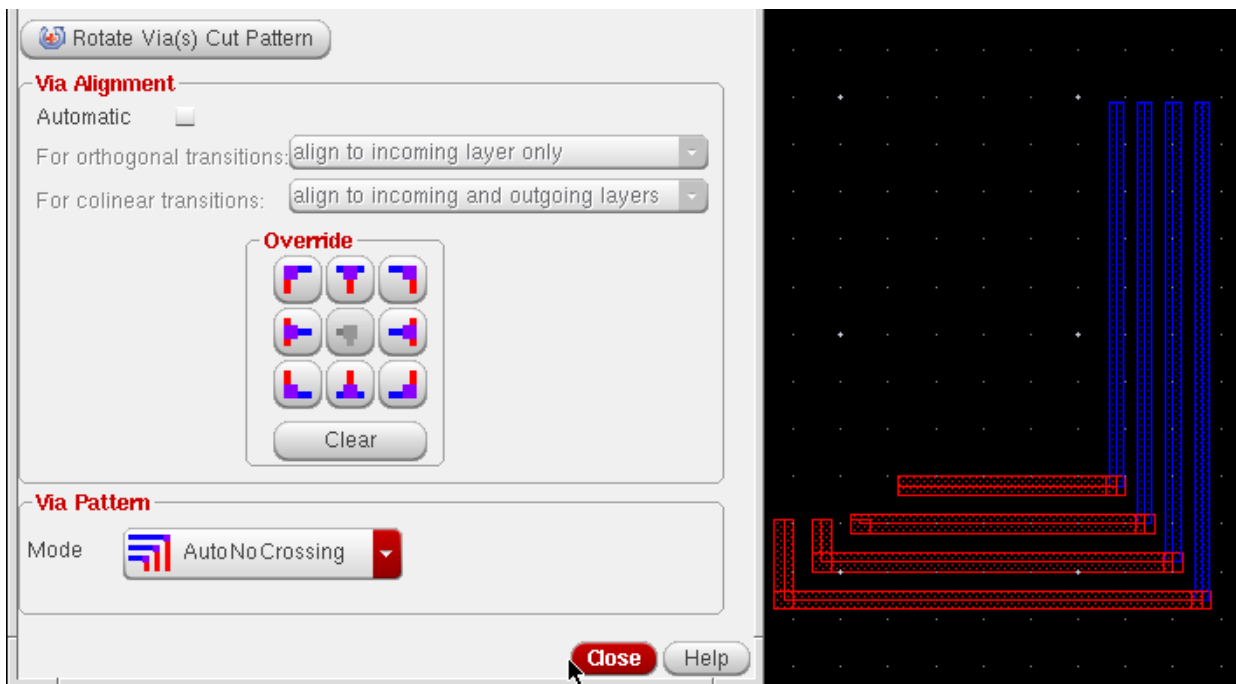
10. Digitize a few wire points.
11. Right-click to select *Select Via* or press the Spacebar key.

Virtuoso Interactive and Assisted Routing User Guide

Working with Vias

The Select Via form appears.

12. In the first column, click the target layer button to indicate the layer you want to change the wire to. In the second column, select one of the available vias from the cyclic field.
13. In the *Override* section, click one of the custom alignment buttons. The *Via Alignment* options are also available on the right-click context menu. Alternatively, you can press the **Ctrl** key and scroll the mouse wheel up or down to cycle through the via alignment values.

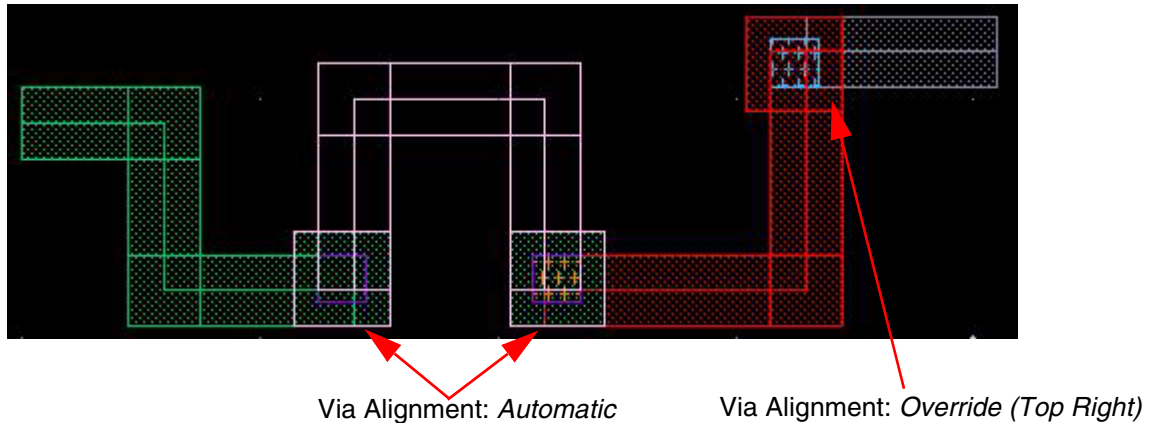


14. In the design display area, click at a point to digitize the via location.

The *Select Via* form closes. Notice the via alignment stays constant with respect to the existing pathSeg when you move the pointer to either direction of the pathSeg.

15. Digitize a few wire points.
16. Complete the command by pressing **Esc**.

The figure below shows the resultant of using automatic and custom via alignment options.



If only one layer and one via are available for selection, you can press the `Spacebar` key to place the vias. The layer changes to the only available layer. If the current routing layer has more than one layer available for selection, the `Select Via` form opens where you can choose a layer or via type. For more information, refer to [Metal Density Options Form](#).

Inserting Vias Automatically

The *Create Wire* command now supports automatic insertion of vias. When a wire which is being created overlaps with same net of wires and between wire and a top-level pin or instance pin, a via is inserted automatically.

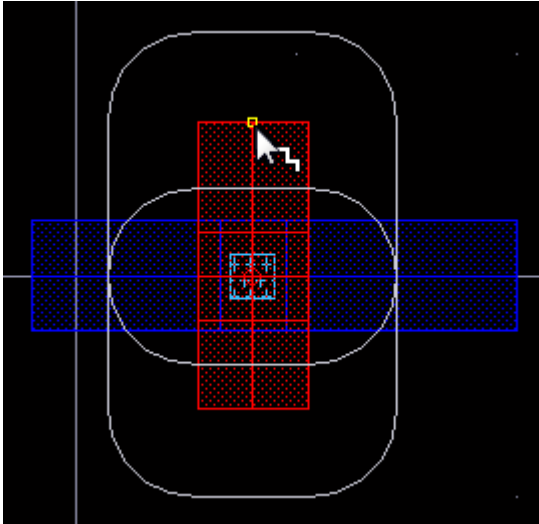
Important

The *Create Wire* command supports automatic via insertion only when the `weAutoDropViaForOverlaps` environment variable is set to `t`.

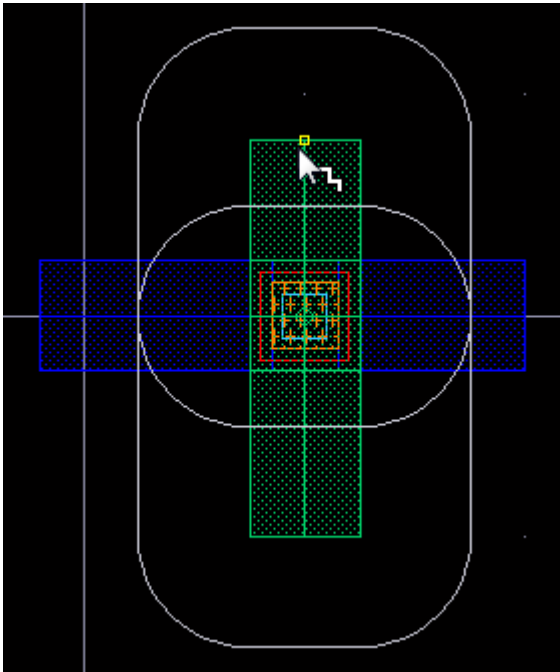
I) By using the *Create Wire* command.

1. To start the *Create Wire* command, choose *Create – Wiring – Wire*.
2. While creating the wire press `F3`. The *Create Wire* form appears.
3. In the *Create Wire* form, specify the net name in the *Net Name* field.
4. After the connectivity for the wire is specified, create the wire on the `Metal1` layer.
5. Now, select `Metal2` layer and again start the *Create Wire* command.

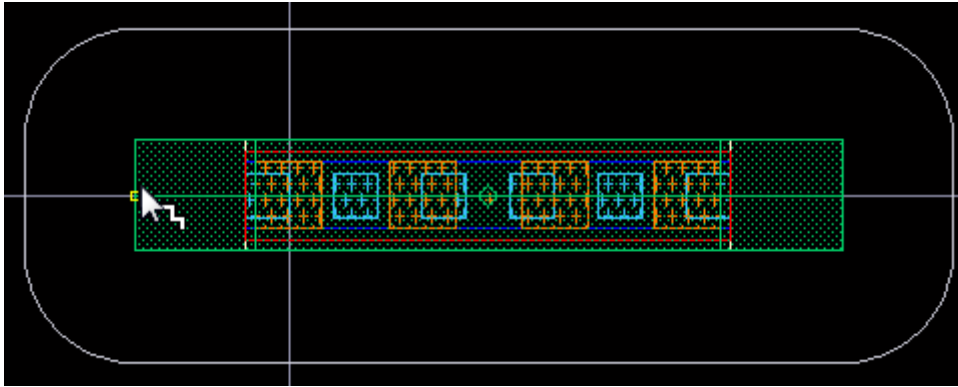
6. Specify the same net connectivity for the wire created on `Meta12` layer as you had specified for the wire created on `Meta11` layer in the *Net Name* field of the Create Wire form.
7. Now, create the wire on the `Meta12` layer such that it overlaps the wire created on `Meta11` layer. A via is inserted automatically on the overlap wires of the same net, as shown in the following figure.



A via stack is inserted when a wire on `Meta11` layer overlaps with the wire on `Meta13` layer. The following figure shows the via stack created on `Meta11-Meta13` layer.

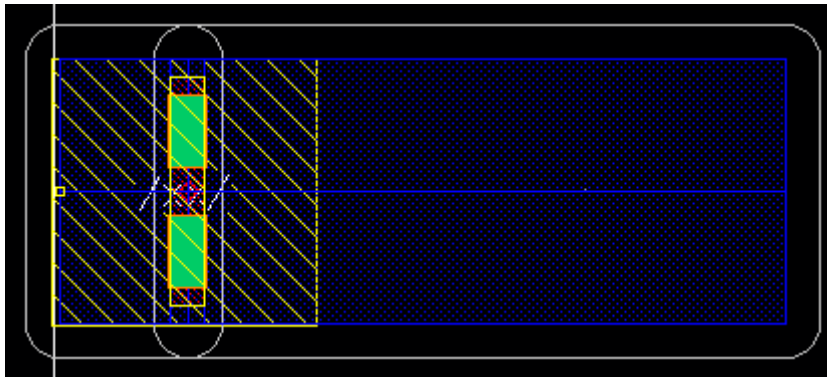


When a wire on `Meta11` layer collinear overlaps with a wire on `Meta13` layer on the same net, a via is inserted automatically and is stretched so that the overlap area is filled with via cuts.



Note: The *Allow Loops* setting is considered while inserting the via automatically. This option is available in the Point to Point Form.

Similarly, a via is automatically inserted on a top-level pin or an instance pin when a wire intersets or overlaps the pin. The following figure shows the via stack created between `Meta11`-`Meta14` layer when a wire on `Meta11` layer overlaps with the top-level pin on `Meta14` layer.



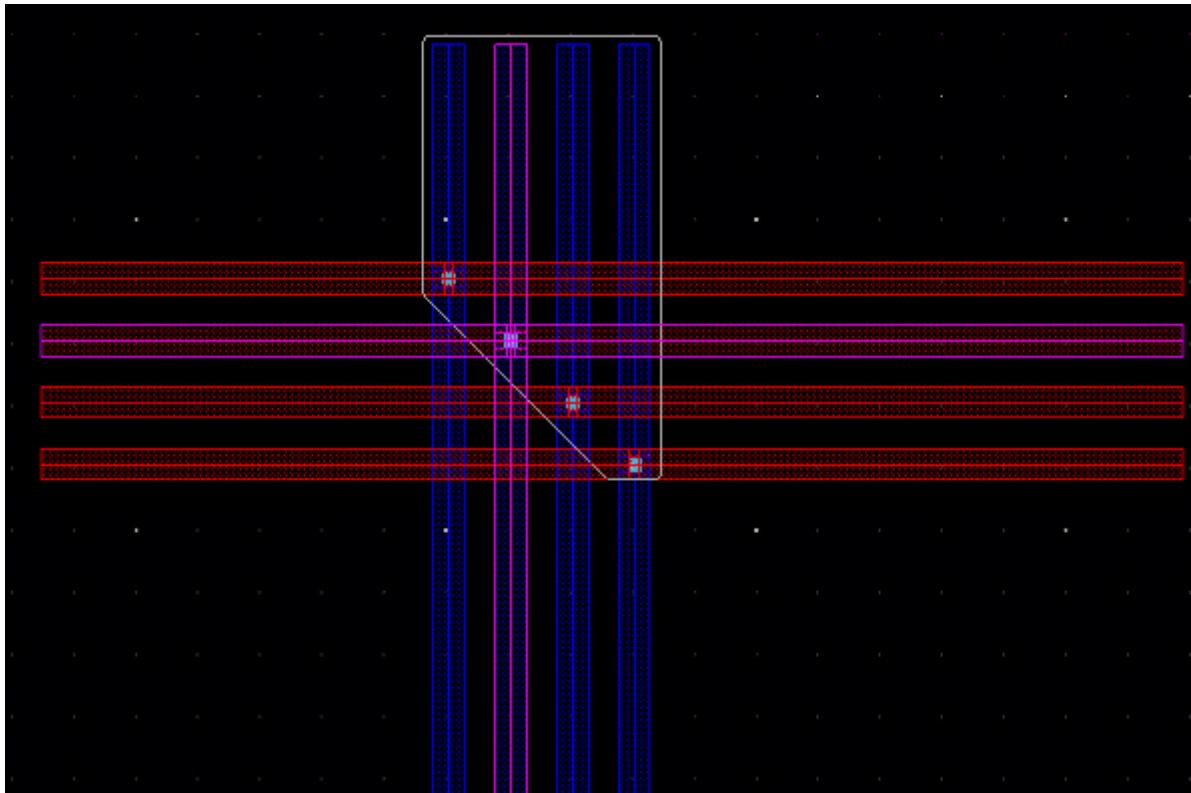
II) By using the *Create Bus* command

The *Create Bus* command also supports automatic insertion of vias. When a bus that is being created overlaps with same net of bus bits and between bus and a top-level pin or an instance pin, a via is inserted automatically. The following figure shows an example when a via is automatically inserted between the bus on `Meta11`-`Meta12` layer when a bus being created on `Meta11` layer overlaps with the bus that exists on `Meta12` layer.

Note: The vias in *Create Bus* are added only if there are no violations.

Virtuoso Interactive and Assisted Routing User Guide

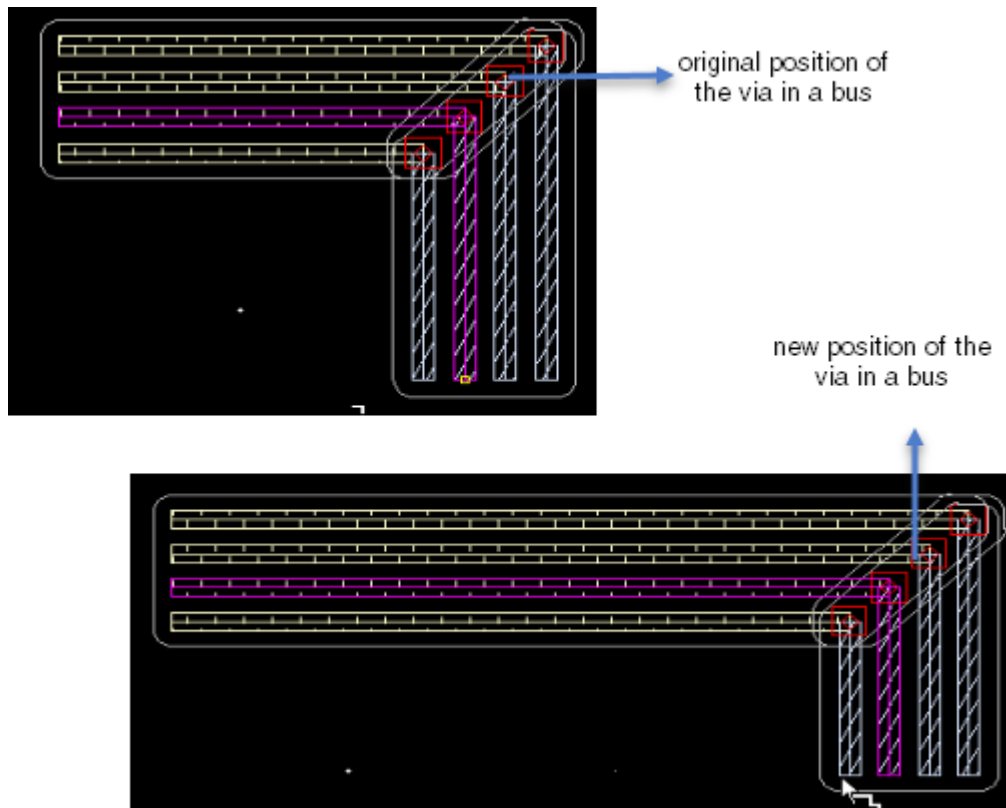
Working with Vias



Virtuoso Interactive and Assisted Routing User Guide

Working with Vias

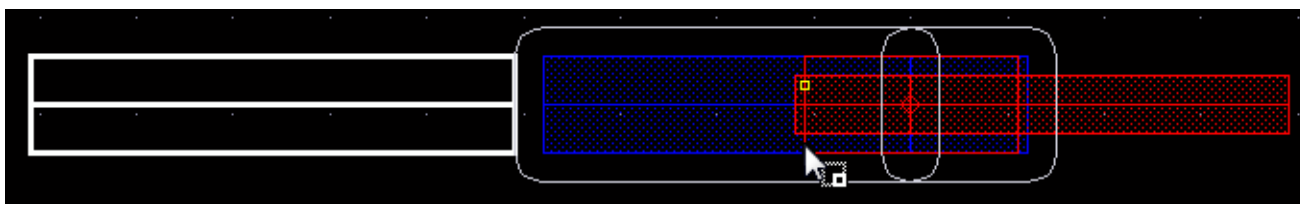
You can slide the bus bits along with the vias. To slide the bus bits, hold the `Shift+Ctrl` keys and move the mouse pointer to the right or left of the current position. The following figure shows the change in the via position when the bus bits slides from left to the right.



Note: The bus bits can also slide across tracks.

III) By using the *Stretch* command.

The Stretch command also supports automatic insertion of vias. The following figures shows that when stretching a wire which overlaps with same net of wires, a via is inserted automatically.

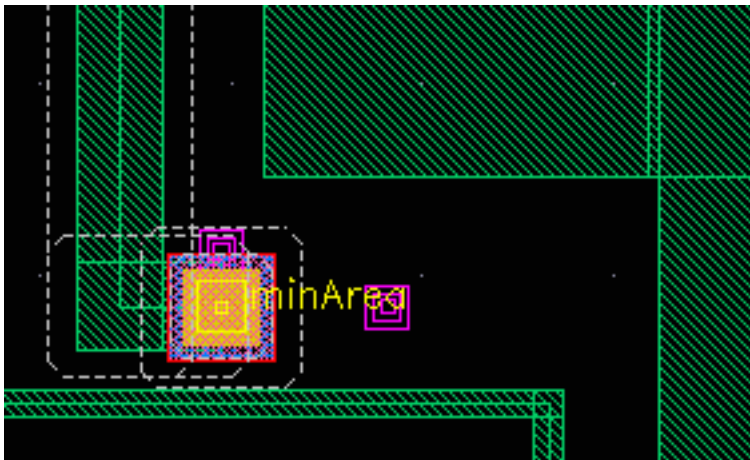


Fanning Out Wires to Vias

When wires are at the minimum wire spacing but the via spacings are larger, the wires can either be fanned out to connect to the vias in the specified pattern or can be back tracked to intersect the previous segment in the wire. This is controlled by using the *Fan Out to Vias* command on the *Use Spacing* sub-menu on the Edit Bus Form.

Using Via Assistance

Via assistant helps determine a legal via site location where you can add a via during routing and not violate design clearance rules.



To enable via assistance:

- The *Create Via* constraint group in the *Options – Editor* form must contain the constraints `horizontalPitch` and `verticalPitch`. For example,

```
routingGrids (
  ( horizontalPitch      "metall"    2.4 )
  ( verticalPitch        "metall"    2.4 )
```

- *Interactive Mode* must be set to *Enforce* in the *Options – DRD Edit* form.
- The last digitized point of the wire being routed must be off the routing grid.

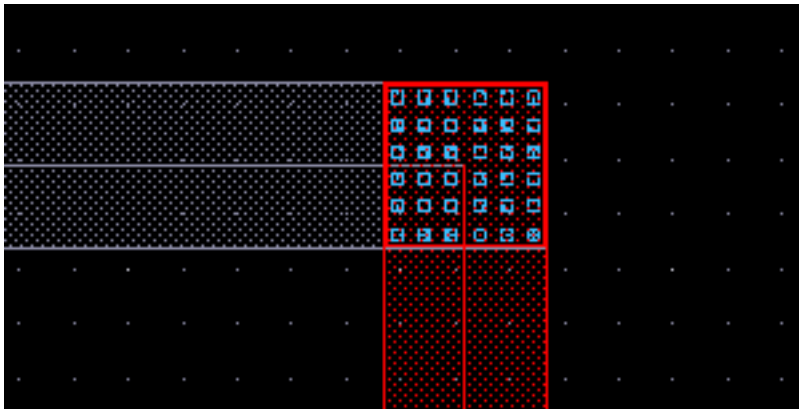
Note: Via assistance does not apply to multiple wire editing and is available only to pathSegs in In Layout XL and higher tiers (*Create Geometric Wire* and *Create Wire* commands).

Creating an Array of Vias

The spacing of cut layers and the minimum number of cuts allowed in a via array can be defined in the technology file. For more information, see [viaSpacing](#) and [minNumCut](#).

Only standard vias can be automatically arrayed to the width of the wire. The number of rows and columns in the array are determined by the width of the wire. The following example display standard via arrays placed on wires.

The figure below shows an array of standard vias placed at a ninety degree intersection of layers.



The software determines the number of cuts that are required based on the width of the wire. The technology file standard via definition allows you to specify the default number of cuts for a via, however, this value is overridden by the number of columns and rows needed for the width of the wire currently being created. For example, if the default number of cuts is 2x2 but the wire width is less than the width of a 2x2 via array, a single via cut is placed on the wire.

Stacking Vias

Note: To move individual vias in a stack you must use the *Move* command. The *Move* command un-stacks vias.

The via stacking limit for specified layers and the minimum area allowed for stacked vias can be defined in the technology file. For more information, see [viaStackingLimits](#).

To stack vias, do the following:

1. Choose *Create – Wiring – Wire*.
2. From the right-click context menu, choose *Select Via*.

The *Select Via form* appears, which displays all the available vias and routing layers that can be reached from the current routing layer.

3. Set the default via for each layer. For each layer, choose the desired via for a target layer from the cyclic field beside each layer.

You can change any number of defaults before clicking on a layer button. You can also raise the *Select Via* form, change the default vias and click OK without selecting a via.

Note: In most cases your design rules require you to place vias with extended overlaps when stacking vias. If the `minArea` constraint is set in the technology file, a via with enclosures that meet the `minArea` constraint requirements is placed for intermediate vias. See [Minimum Area Rules](#) on page 213.

4. From the *Select Via* form select the desired nonadjacent layer.

To cancel any pending vias, click the current layer button. You can rotate pending vias by pressing the right mouse button. This is reflected in the button prompt text in the CIW and on the layout window.

5. Move the pointer and click to place the via stack.

Via stacking adheres to the stack via depth rule, `maxStackVias`.

Inserting Multi-Layer Stack Via

The Create Wire lets you instantiate the multi-layer stack via and allows the wire to exit on any layer of the stack via.

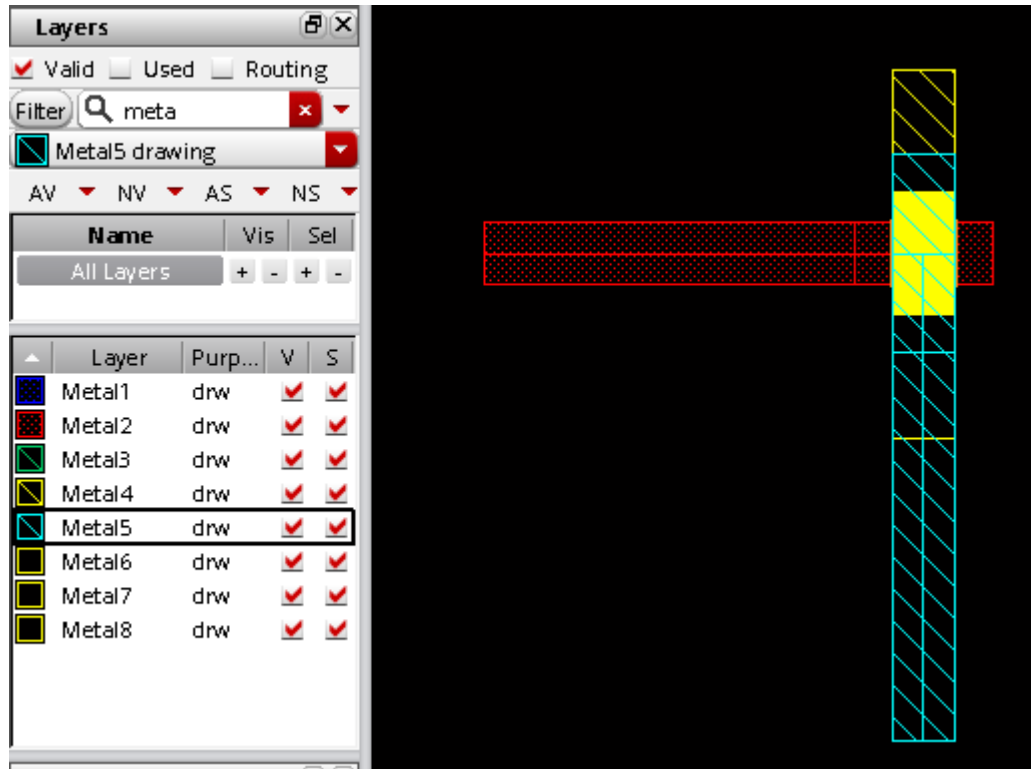
To create multi-layer stack via, do the following:

1. Choose *Create – Wiring – Wire*.
2. Start creating the wire by digitizing the wire points.
3. Right-click to select *Via Up To <layer name>* or *Via Down To <layer name>* to change the wire layer to a higher or lower layer, respectively.
4. Move the pointer and click to place the via.

Virtuoso Interactive and Assisted Routing User Guide

Working with Vias

The following figure shows how the M1–M2 via is inserted.

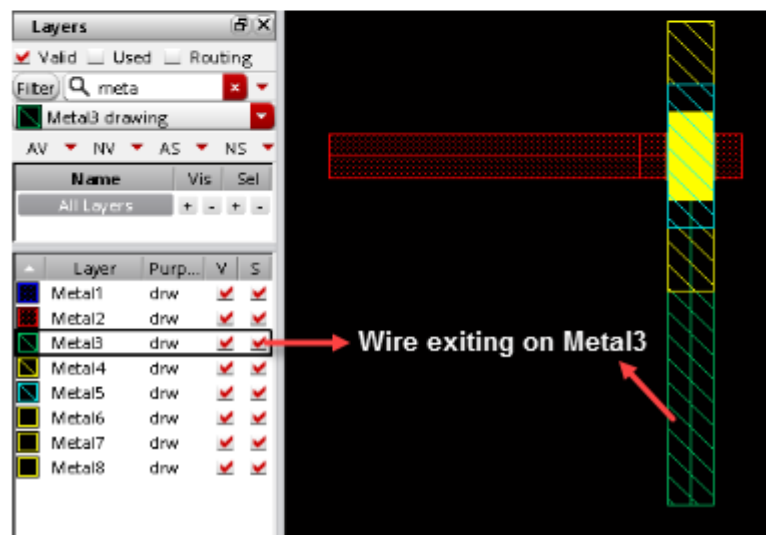
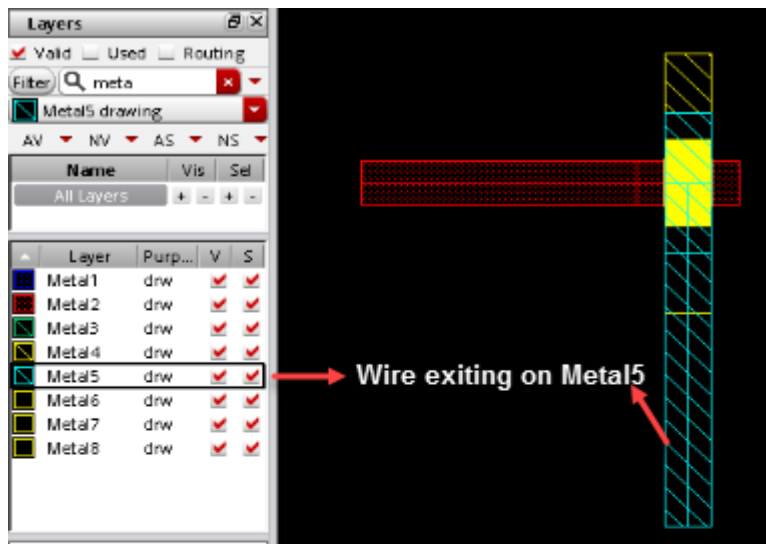


You can also insert a via stack between the metal layers. Consider a case when a stacked via is created between `metal1` and `metal4` layers. The stacked via includes the `metal2` and `metal3` layers as well. In this case, you start a wire on `metal1` then do *Via Down To* and select `metal4`. The M1–M4 via is inserted. After you insert the via, you can choose the desired target layer on which the wire should exit. To do this, select

Virtuoso Interactive and Assisted Routing User Guide

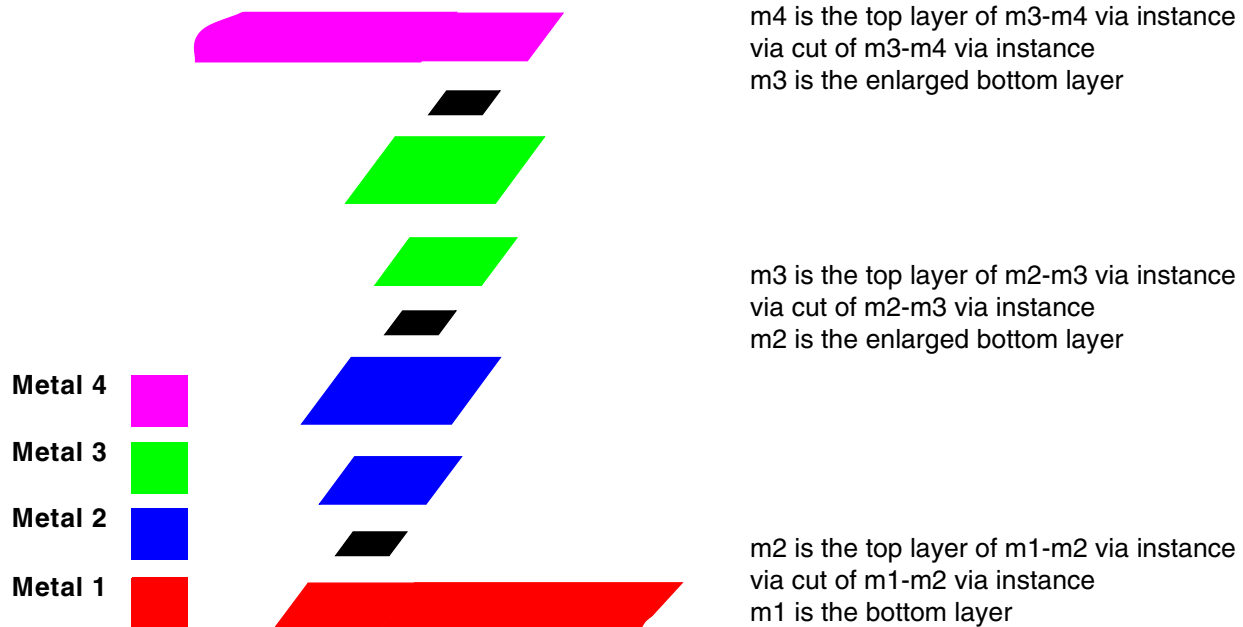
Working with Vias

the target layer from the Palette. The following figures shows a scenario where a M1–M4 via is inserted but the wire exits on the `metal3` layer.



Minimum Area Rules

Extended overlaps (vias with the area of the bottom layer expanded) for stacked vias are used to satisfy minimum area rules (MAR) for intermediate metal.



Changing Via Cut Color

You can now manually change the via cut color for interactive routing commands. The via cut color is supported only for single cut via and for single via stack.

To enable via cut coloring, specify the `viaCutColorMode` environment variable. When the environment variable is specified, the *Cycle Cut Color* option is available in the *Create Wire*, *Create Bus*, and *Create Stranded Wire* context-sensitive menu.

To change the via cut color:

1. Choose *Create – Wiring – Wire*.
2. From the *Layers* assistant, select a layer on which you want to create the wire.
3. To start creating a wire, click at a point in the design display area to digitize the wire.
4. Right-click and select *Via Up To <layer name>* or *Via Down To <layer name>* to insert a via.

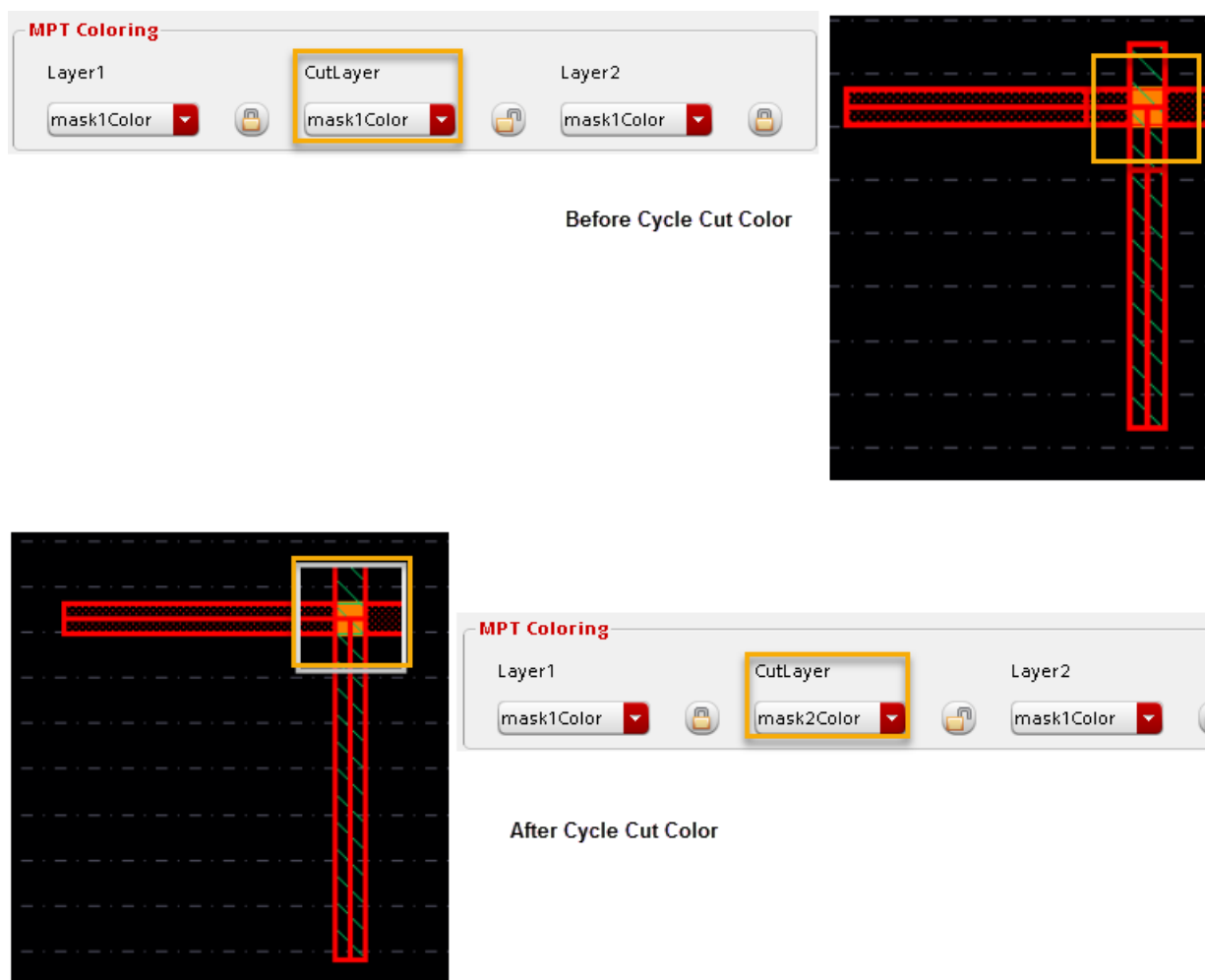
Virtuoso Interactive and Assisted Routing User Guide

Working with Vias

5. To change the via cut color, place the mouse pointer on the *Via* option in the *Create Wire* context-sensitive menu. The *Via* submenu is displayed.
6. Click the *Cycle Cut Color* option. The cycle color option is used to change the via cut color. The specified pattern is not affected.

Note: The color of the via is changed depending on the mask colors available in the technology file.

The following figure shows that the via cut color is changed in the via that has been inserted on the wire.



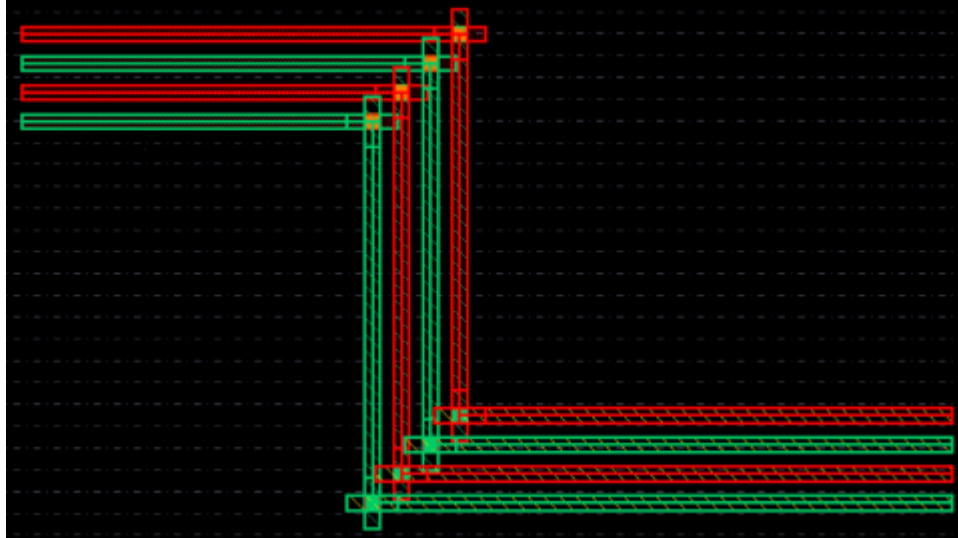
Similarly, you can also change the via cut color for the *Create Bus* command. For *Create Bus*, the via matrix is a row of cut classes. Following are the available via cut color patterns.

```
;; alternate row: row
;; alternate col: col or column
```

Virtuoso Interactive and Assisted Routing User Guide

Working with Vias

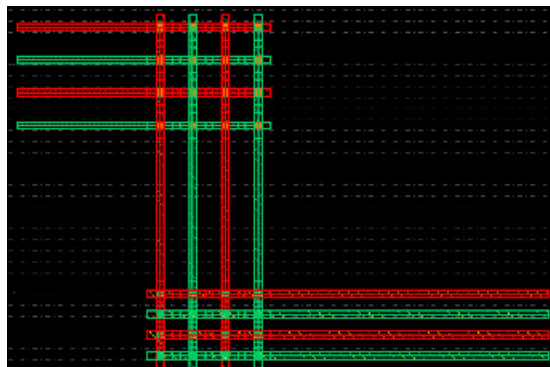
```
;; chessboard: chess or chessboard
;; all others value or no value give uniform cut pattern
```



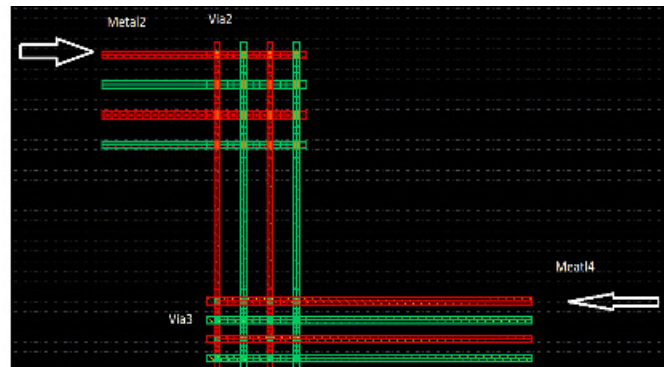
The *Create Stranded Wire* command also supports via cut coloring. By default, via cut pattern is uniform. However, you can change the cut pattern by specifying the weWACmdCalculatedColor environment variable. For example:

```
envSetVal("we" "ViaCutColorPatterns" 'string "Via2 row : Via3 col")
```

The following figure shows the via cut pattern for the *Create Stranded Wire* command, where *Via2* follows the row pattern and *Via3* follows the column pattern.



Before cycle cut color



After cycle cut color

Virtuoso Interactive and Assisted Routing User Guide

Working with Vias

Working with Buses

You can create a bus in the Virtuoso Layout Suite by using the *Create – Wiring – Bus* command. All wires of the bus, referred to as bus bits, are created on the same layer. When you change the bus layer, all bus bits change to the same layer. A bus now supports a mixture of assigned and unassigned nets, as well as repeated nets. Consider an example, where a bus has five bits bus such as:

```
bit0 = net "a", bit1 = net "b", bit2 = unassigned net, bit3 = net "c", bit4 =
unassigned net
```

The corresponding bundle notation is as follows

```
"a" + ", " + "b" + ", " + "" + ", " + "c" + ", " + "" = "a,b,,c,".
```

This means the bus supports a mixture of assigned and unassigned nets.

Another example, where a bus has three bits such as:

```
bit0 = net "a", bit1 = net "b", bit2 = net "a"
```

The corresponding bundle notation is as follows

```
"a" + ", " + "b" + ", " + "a" = "a,b,a".
```

This means the bus supports repeated nets.

For creating a bus, ensure that the technology file contains the minimum information, such as complete foundry rules, and that the non-default or override constraint groups, if any, are specified in the Wire Assistant or in the Edit Bus Form. A wiring constraint group (`virtuosoDefaultSetup` or `LEFDefaultRouteSpec`) that defines the valid layers and the vias allowed for routing must be present in the technology file.

Note: In some cases, the pointer is prevented from moving to a location where the bus would necessarily have violations. This is because the wires within the set are intended to be correct by construction.

You can either directly refer to the procedure described in Creating a Bus.

This chapter covers the following topics:

- Creating a Bus Constraint

- [Autopicking Bus Bits](#)
- [Creating a Bus](#)
- [Restarting a Bus From Existing Routes](#)
- [Using the Control Wire](#)
- [Editing a Bus](#)
- [Avoiding Shorts with Existing Blockages for Create Bus](#)
- [Supporting Multi-Layer Bus](#)

Creating a Bus Constraint

You can create a bus constraint by using the *Constraint Manager* and the *Search* assistant. The bus constraint helps to group all nets to be routed as a bus. The *Create Bus* command recognizes the bus constraint. The *Create Bus* command starts routing the bus if you start digitization from any one of the bus pins. This feature is also known as autopicking the bus bits. For more information, see [Autopicking Bus Bits](#).

The *Constraint Manager* is available only XL tier onwards. So you can use the bus constraint for creating a bus only XL tier onwards. For information about how to create a bus in L tier, see [Creating a Bus](#).

To create a bus constraint in In Layout XL and higher tiers by using the *Constraint Manager*:

1. From the layout cellview, choose *Window – Assistants – Constraint Manager*.

The *Constraint Manager* assistant opens.

2. From the layout cellview, select *Window – Assistants – Navigator*.

The *Navigator* assistant opens.

3. From the *Navigator* assistant, select several nets that you want to route as a bus. Alternatively, you can also search for nets in the design by using the *Search* assistant.

Virtuoso Interactive and Assisted Routing User Guide

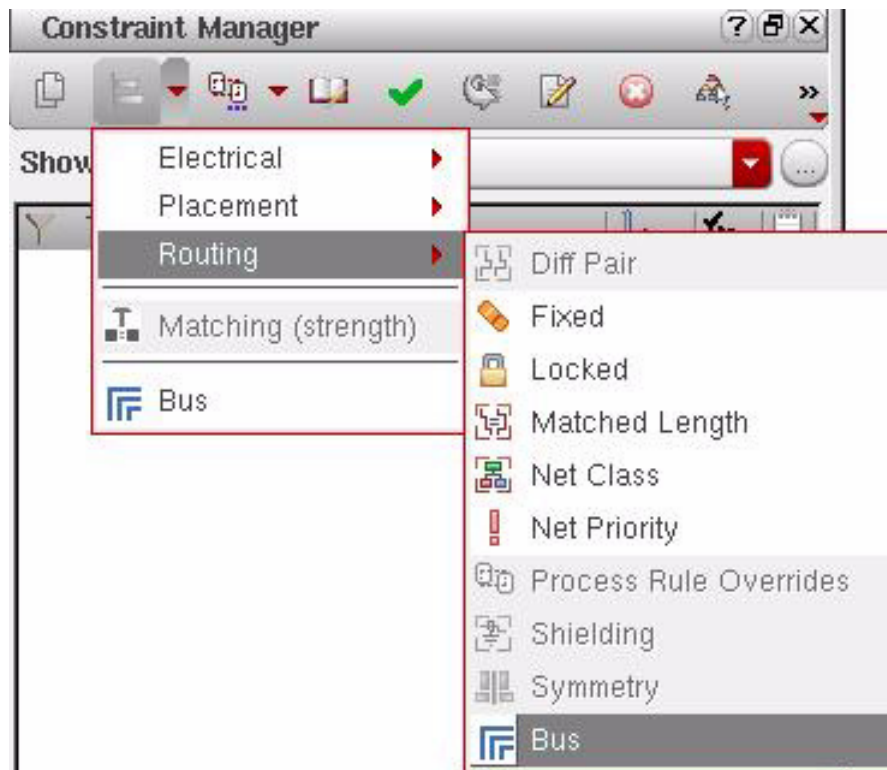
Working with Buses

4. In the *Constraint Manager* assistant, click on the arrow next to the *Constraint Generator* icon.

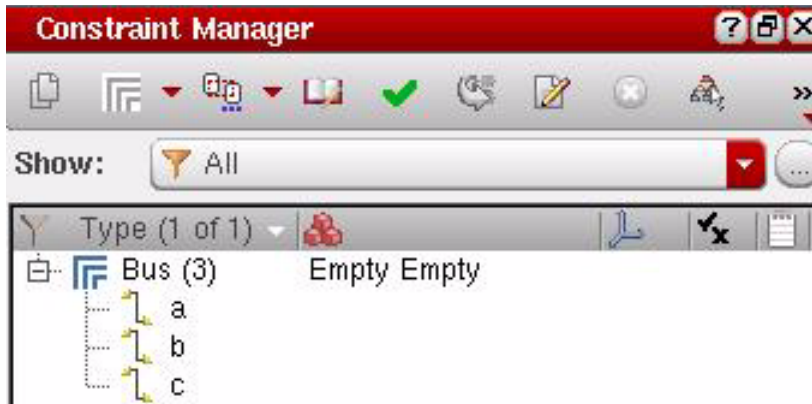


Constraint Generator

5. Select *Routing – Bus*.



6. Expand the created constraint to verify the selected nets are part of the bus.



Autopicking Bus Bits

When a selected net is part of a bus constraint, autopick determines the other nets of that bus constraint on the same layer as the picked net and adds one object of each net to the pick set. If there are pre-routed nets from pins and instPins, autopick picks these wires as well and creates duplicate wires on top of the pre-existing wires.

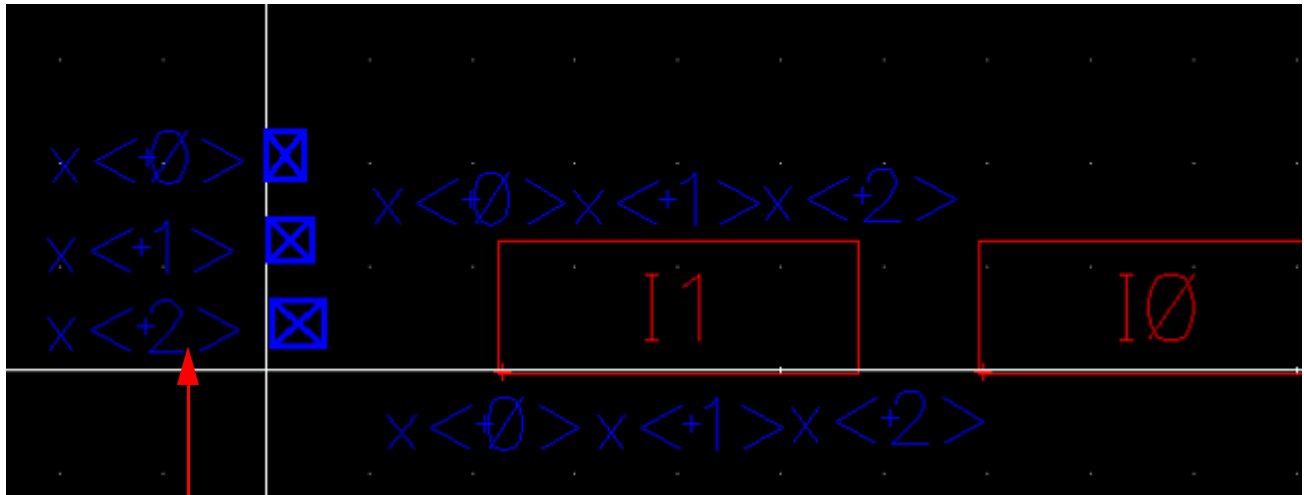
Note: Autopick does not support pre-existing wires or dangling wires even if the associated nets are part of the bus constraint.

For information about how to create a bus constraint, see [Creating a Bus Constraint](#).

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

If you start the *Create Bus* command and click a single pin that belongs to a bus constraint, the command automatically enables bus mode. In the example below, a bus constraint is defined on `netx<0>`, `netx<1>`, and `netx<2>`.

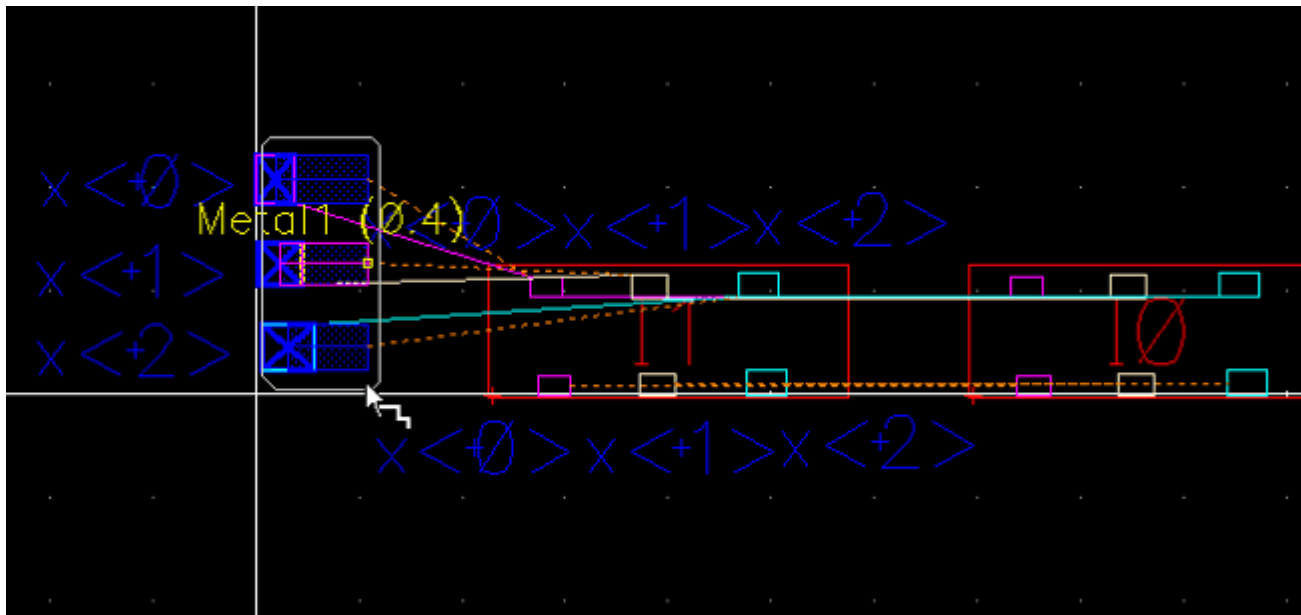


netx<0>, netx<1>, and
netx<2>
Part of a Bus Constraint

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

After starting the *Create Bus* command, if you click on pin `x<0>`, autopick adds pin `x<1>` and pin `x<2>` to the pick set.



If you area select pin `x<0>`, pin `x<1>`, and pin `x<2>`, then pin `x<0>` and pin `x<1>`, being part of the bus constraint, start routing in the bus mode. The other pins are not picked even though they were included in area selection because it does not belong to the bus.

Autopick does not pick instPins that belong to different instances. For example, nets A, B and C are part of a bus constraint. Nets A and B have associated instPins on the instance `I0` and net C has the associated instPin on the instance `I1`. In this case, autopick on net A from instPin on instance `I0` will pick and draw wire for net B on instance `I0`. Autopick will not pick instPin on net C as it belongs to a different instance `I1`, even though net C belongs to the same bus constraint as nets A and B.

To remove nets from an autopicked bus while the *Create Bus* command is running, keep the `Ctrl` key pressed and click on the required pins. To add nets to a bus while the *Create Bus* command is running, keep the `Shift` key pressed and click on the required pins. In the above example, to route only `netA` and `netB`, keep the `Ctrl` key pressed and click on `pinD` to remove it from the pick set.

The *Create Bus* command uses the following criteria to determine the objects that are autopicked:

- Autopicked instance or top-level pins have the following same parameters as the clicked instance or top-level pin:
 - Direction – in, out, or inout

❑ instTerms on different nets

- Autopicked objects are on the same LPP as the clicked object.
- Connectivity is assigned to all nets.
- Number of bends should be same for all selected wires.

In a set of objects that can be autopicked, preference is given to objects in the following order:

- Aligned instPins (x or y axis)
- Closest instPins
- instPins with the same access direction (top, bottom, left, right)

If autopick fails to find adequate objects, you might need to complete the pick set manually.

Creating a Bus

You can start creating buses from the following starting points:

- From a point in free space
- From existing pins
- From pre-existing bus pre-routes (see [Restarting a Bus From Existing Routes](#))

To create multiple floating wires, see [Creating a Floating Wire](#).

To create a bus:

1. Launch Virtuoso Layout Suite.
2. Select *Create – Wiring – Bus* or click on the icon  on the *Create* toolbar.

When you start the *Create Bus* command in Layout EXL, the *Interactive Routing Bindkeys* hint box is displayed at the top-left corner of the layout canvas. The hint box describes the most useful bindkeys and shortcuts for the *Create Bus* command.



3. Press `F3`.

The Edit Bus Form opens.

4. From the *Layers* assistant, select a layer on which you want to create the bus.
5. Specify the values in the *Number of Bits*, *Bit Spacing*, and *Width* fields.
6. To start creating a bus in the empty space in the design display area:
 - a. Click at a point in the design display area to digitize the bus.
 - b. Continue to digitize the bus and create bends.
 - c. Press `Enter` to finish creating the bus.
7. To create a bus that has connectivity in the empty space in the design display area:

- a. In the *Create Bus* form, in the *Net Name* field, specify the list of net names separated by commas or spaces.

You can also specify net names in bus notation. For example, if you specify `bus<0:3>` in the *Net Name* field and start bus creation in free space, the bus wires are created on the nets `bus<0>`, `bus<1>`, `bus<2>`, `bus<3>`.

If the nets specified in the *Net Name* field do not exist in the design, the nets are created. Using this method, you can create floating buses.

- b. Select the *Create Bus Constraint* check box if you want a bus constraint to be created automatically for the picked or the specified nets.
 - c. Click at a point in the design display area to digitize the bus.

- d. Continue to digitize the bus and create bends.
- e. Press `Enter` to finish creating the bus.

8. To start creating a bus from existing pins, use one of the following methods to select multiple pins:

Note: The Bus is created using the *Create – Wiring – Bus* command only. The *Create – Wiring – Wire* command creates single wires and cannot be used to create buses.

a. Selecting Multiple Pins

When you area-select multiple pins that are on the same layer but different nets, the pin nets are used for creating the bus wires; the pin nets are populated in the *Net Name* field in the *Create Bus* form.

The starting pins can be I/O pins or Soft Block pins. You can area-select level-0 pin shapes or visible level-1 instPins (The *Instance Pins* check box should be selected on the *Display Options* form). You can also start routing related nets in bus mode when top-level pin shapes are pre-selected.

If the pins are on different layers, the *Create Bus* command begins wires from the routing layer that has the most number of pins. If some of the selected multiple pins are on the same net, then routing begins from only one pin on the net.

b. Adding Pins to the Selection

Select a single pin and `Shift` + click to add additional pins to the selection.

In both the above methods, you can use the `Ctrl` and `Shift` keys to remove objects from the selection.

Note: The pin is removed only from the picked set; the originally selected set remains intact. You can use the `Ctrl` and `Shift` keys only before the first digitized point. To modify the selected set after the first point is digitized, press `Esc`.

c. Autopicking Pins Belonging to a Bus Constraint (In Layout XL and higher tiers)

By creating a bus constraint, you can define the nets to be routed as a bus. Clicking one of the bus pins starts routing all the bus nets in bus mode. For more information, see [Creating a Bus Constraint](#) and [Autopicking Bus Bits](#). Even pre-selecting a bus pin that belongs to a bus constraint and then starting the *Create Bus* command starts routing all the bus pins that belong to the same bus constraint. For more information about creating a bus from existing routes, see [Restarting a Bus From Existing Routes](#).

d. Adding Instance Terminals

When you select an instance terminal, the wire editor tries to find if there are qualify terminals from other instances that can be used for the bus. The Create bus selects all the qualify terminals from either horizontal or vertical which ever has more qualified terminal. Following are the qualify terminals:

- The instance master of the qualify terminal must be the same instance master of the picking terminal.
- The qualify terminal must be the same terminal on the qualify instance as the picked terminals on the instance of the picked terminals.
- The qualify terminal must intersect with either horizontal or vertical extent regions of the picked terminal.

Note: The bus wires start at pin centers.

e. Adding vias at the start of a bus

When you select a pin to start a bus on a layer that is other than the pin layer, a via is automatically dropped.

- Select a layer in the Palette. For example, M1.
- Choose *Create – Wiring – Bus*.
- Holding the `Ctrl` key, area select the pins that are on layer other than the layer selected in Layers palette. Vias are dropped at the center of each pin. You can also drop a via at the start of a bus by selecting a single pin using `Ctrl` + click.

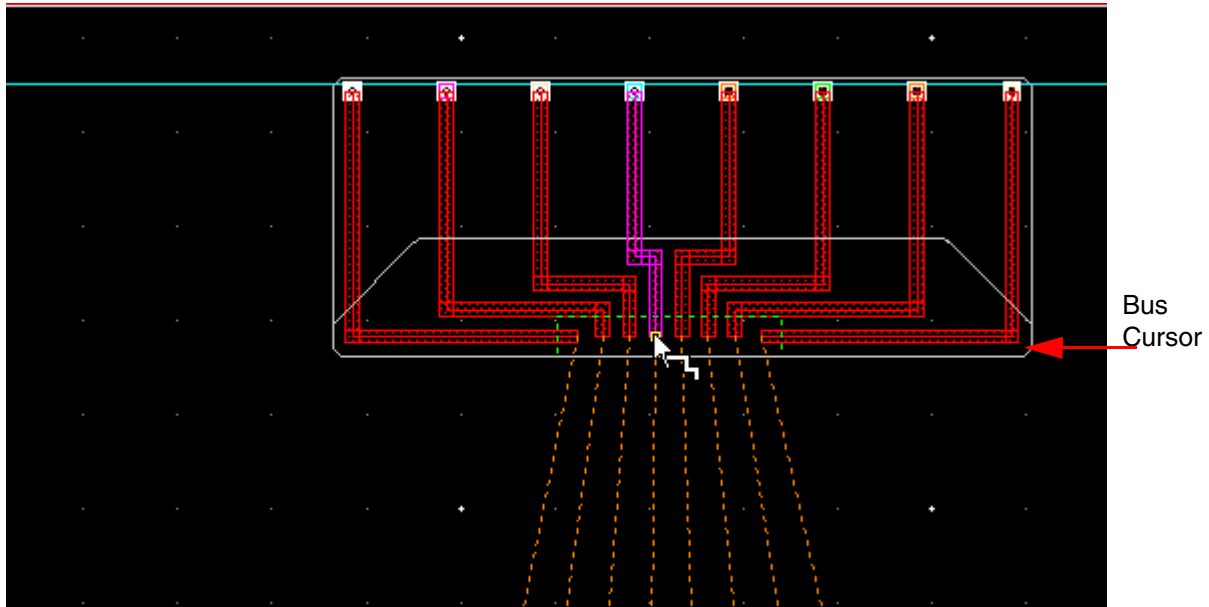
To remove a pin from the selection, hold the `Ctrl` key and area select the pin that you want to remove from the selection. If you want to add the pin to the selection, hold the `Shift` key and area select the pin that you want to add.

9. Right-click to display the Edit Bus Form.

If required, choose *Use Width* from the context-sensitive menu to switch the control wire function to one of the extreme wires or the center wire of the bus. For information about control wires, see Using the Control Wire.

- 10.** You can change width and spacing overrides at any time while creating or reshaping a bus. If you select the *Default Constraints* option from the *Use Spacing* menu, then the spacing after the bend is the `minSpacing` as shown in the figure below. The default

width is based on the `minWidth` and the default spacing value is based on the `minSpacing` constraint for each layer that can be used for routing.



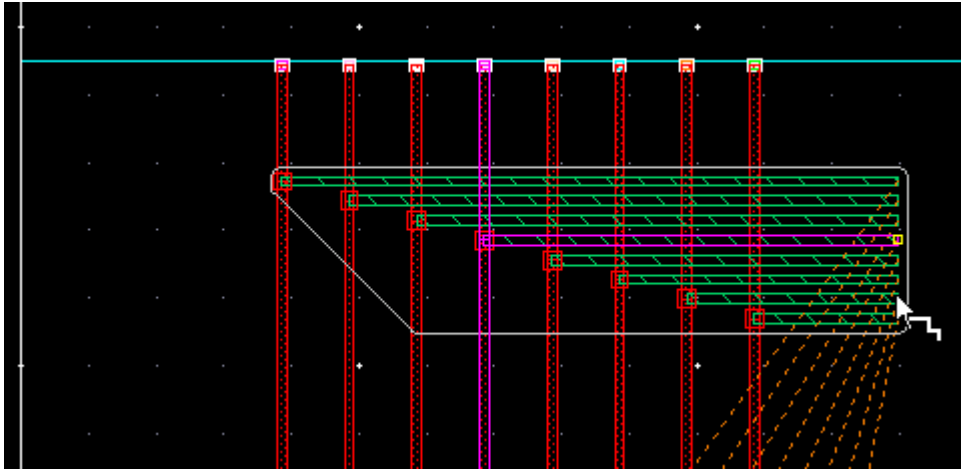
11. Select *Gather/Spread* from the *Use Spacing* menu of the Edit Bus Form to gather or spread bus wires before the first bend of wires.

If you select *Gather/Spread Bus*, the bus wires are surrounded by a dotted box called the bus cursor. Individual wires are gathered or spread inside this box, which can be moved to any desired location. Jogs are added and the wires are gathered to the minimum spacing rule or specified spacing value. The command on the context-sensitive menu toggles to *Cancel Gather/Spread*.

If you select *Gather/Spread Bus* repeatedly and one of the times no gather action is performed because the *Bit Spacing* field in the form is not modified, then the *Create Bus* command restores the wire spacing that was used before the previous gather operation. For example, if you start creating a bus from pins that are non-uniformly spaced and *Bit Spacing* is 0, and select *Gather/Spread Bus* from the *Create Bus* context-sensitive menu, the wires are gathered to `minSpacing`. Click and drag to pass the obstruction. If you select *Gather/Spread Bus* again without changing the *Bit Spacing* value, the *Create Bus* command restores the spacing that was used before the previous spacing, that is, the pin spacing will be restored.

When routing a bus, each time you move the pointer, the locations of the last pathSeg leading up the pointer are computed. If you select *Cancel Gather/Spread Wires* and the wires are not fanned out to a via pattern, only one segment leads to the pointer for each wire. If you select *Gather/Spread Bus*, jogs are added to the pathSegs to bring them closer together if they are spaced farther than the minimum spacing.

Pressing `Backspace` removes the jogs and the wires are not gathered or spread.



12. To change the spacing between multiple wires, press `F3` to open *Create Bus* form. In the *Bit Spacing* field of the *Create Bus* form, specify the new spacing value. You can also change the spacing with the *Use Spacing* menu or the Wire Assistant.

The spacing is updated from the last digitized point.

13. If required, change the bus wire width in the *Width* field in the *Create Bus* form or select the appropriate option from the *Use Width* menu on the Create Stranded Wire Form.

The width is updated from the last digitized point.

14. To remove the last segment of a wire, use `Backspace`.

Bus editing supports *Undo*, *Redo*, and `Backspace`.

Note: Do not draw wires back over previously drawn wires.

15. To change layers and add vias or to fan vias out to connect to the vias, see Changing Layers and Selecting Vias.

16. You can finish creating the bus at any point by double clicking or by pressing `Enter`. The bus ends at the pointer location.

You can check the connectivity of the bus bits by using the *Edit Properties* form or the *Property Editor* assistant.

For finishing the bus connections automatically, see Finishing a Trunk.

Restarting a Bus From Existing Routes

You can use the *Create Bus* command to restart a bus by picking the end-point of any of the floating bus wires that are on nets belonging to the same bus constraint. You can restart a bus with wires on different purposes but same metal layer.

Note: While restarting a bus, If a via is created, then the via purpose is changed according to the layer purpose of the connected wire.

To restart an existing bus, where the bus wires belong to a bus constraint:

1. Choose *Create – Wiring – Bus*.

Note: In L, you can autopick nets of a bus constraint that has been created in XL.

2. Click at the end-point of any of the bus wires.

All the bus wires start routing on the same layer.

To restart an existing bus, where the bus wires do not belong to a bus constraint, use the wire tie-out method, as described below:

1. Choose *Create – Wiring – Bus*.
2. Pick bus wires by area-selecting all the wires of the bus.

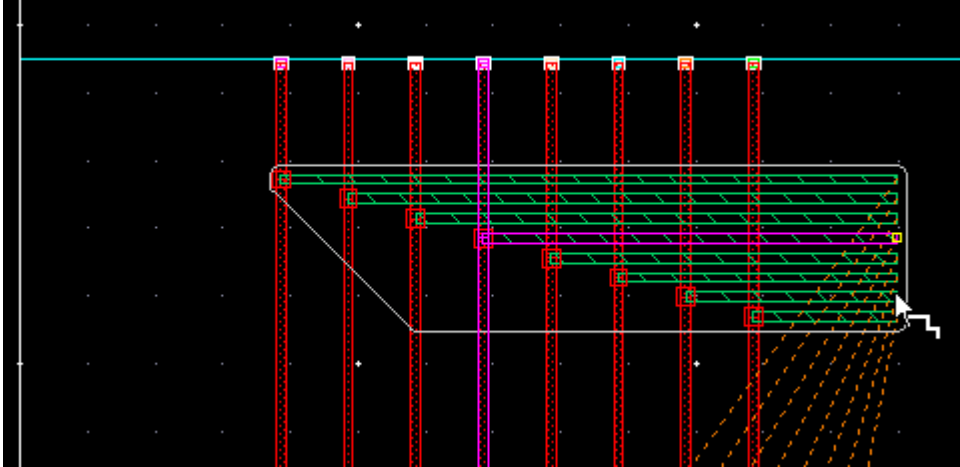
The Select Via form opens. For more information, refer to [Metal Density Options Form](#).

3. Select the appropriate via for changing the bus to the required layer.
4. Specify the required via alignment.
5. Digitize the via.

The *Select Via* form closes.

6. Continue routing and digitizing the bus.

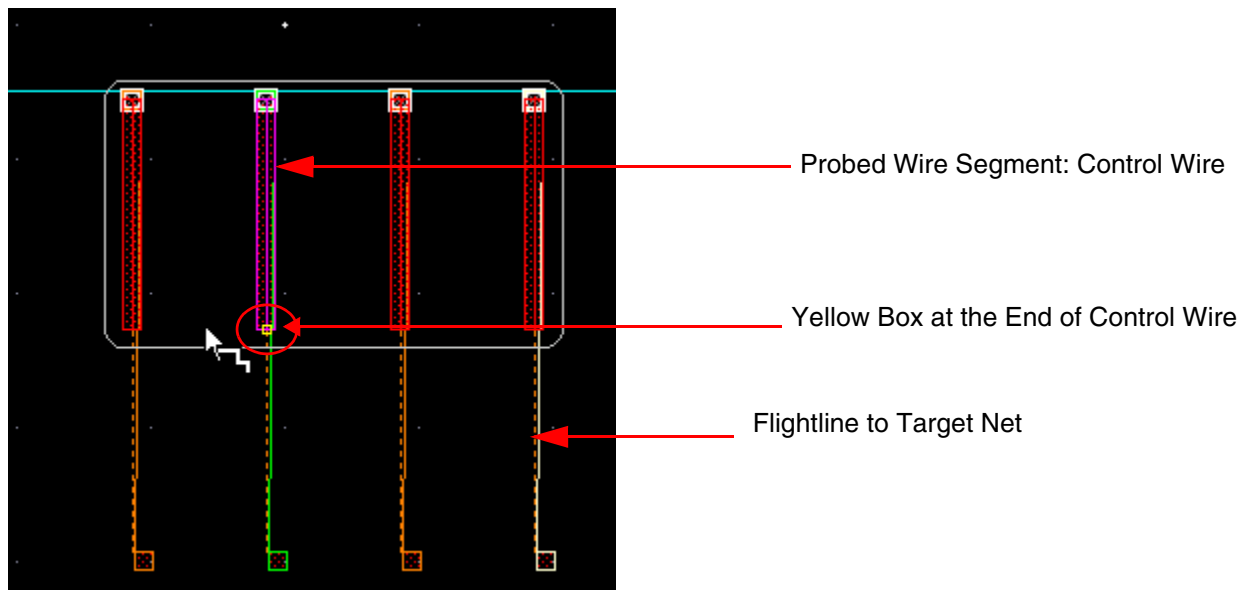
The following figures shows a wire tie-out.



Note: Wire tie-out works only if the distance between the centers of consecutive wires is large enough to create the required vias without creating a spacing violation.

Using the Control Wire

In the set of multiple wires, one of the wire segments is called the control wire. In a bus, it stands out as being the probed/highlighted wire segment with a small yellow box at the end of the wire segment. By default, the center wire is the control wire.

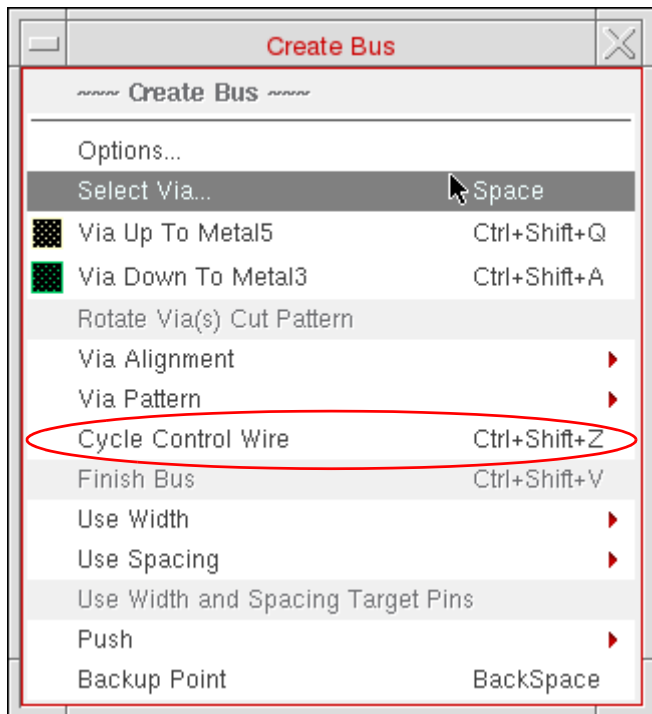


Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

The control wire serves as a pivot point as the multiple wires are dragged into routing position. The other wires of the selected bus set follow the control wire at appropriate spacing.

You can toggle the control wire role to one of the extreme wires of the selected bus. When creating or editing multiple wires, you can cycle the control wire role to the Top-Middle-Bottom (in a horizontal bus) or Left-Middle-Right (in a vertical bus) wires by selecting the *Cycle Control Wire* option from the Edit Bus Form.

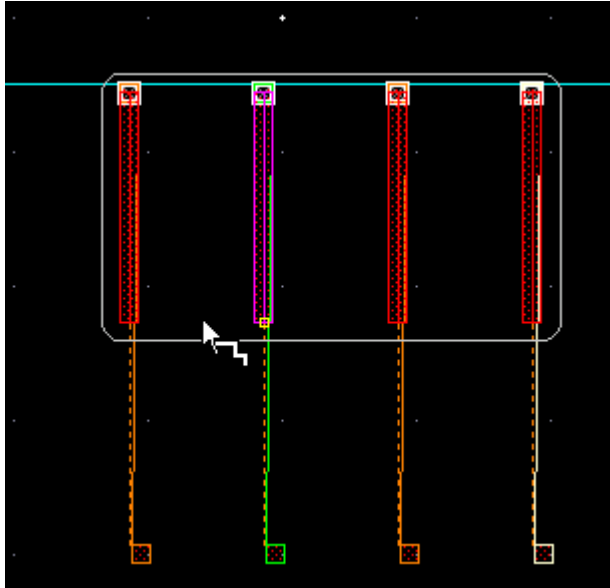


The wire segment assigned as the control wire continues to function as the control wire even if you select the *Backup Point* option from the context menu during multiple wire editing. The control wire, however, defaults to the middle wire if you add or remove (Ctrl + click) any wire segment to/from the bus.

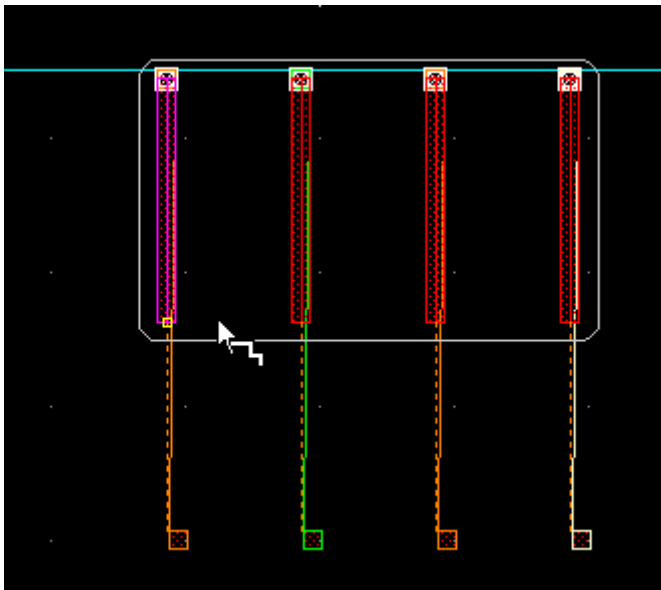
Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

For example, in the following figure, the middle wire is the control wire.



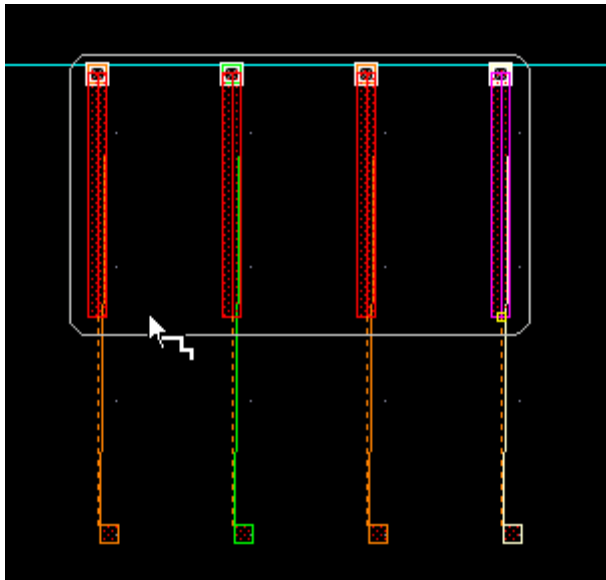
Right-click to select the *Cycle Control Wire* option from the context menu or press Control + Shift + X. The probing/highlighting and the yellow box passes on to the next possible control wire, which is the left one here.



Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

Again, select the *Cycle Control Wire* option from the context menu or press the combination Control + Shift + X. It probes/highlights the right wire as the control wire.

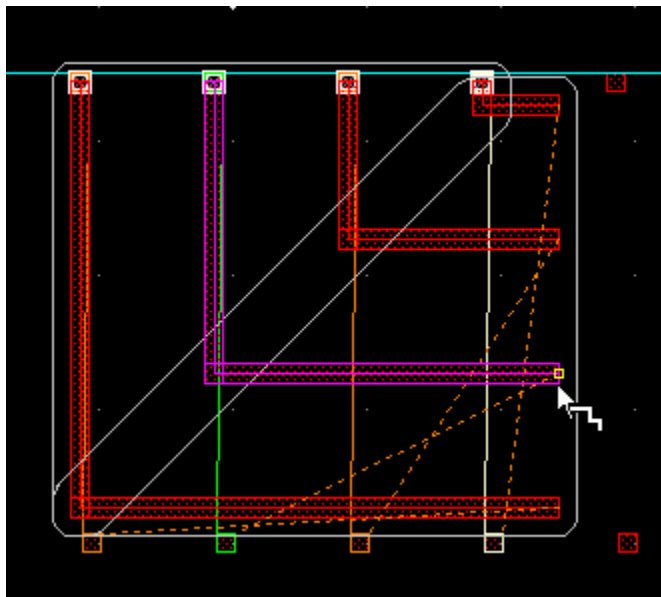
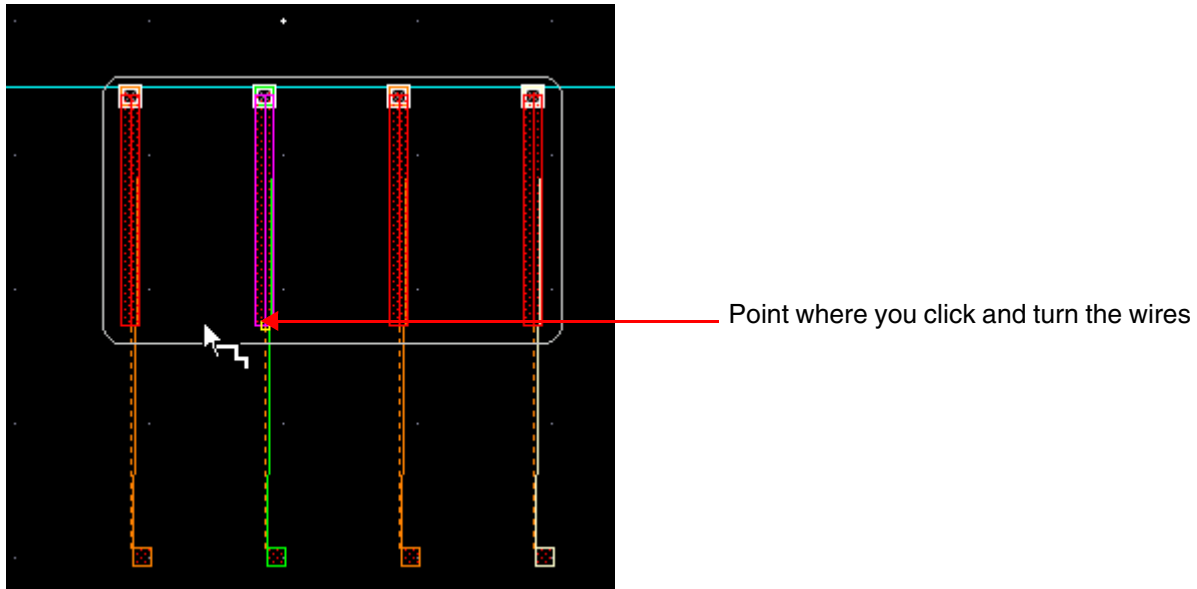


Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

When multiple wires are turned on the same layer, the control wire serves as the pivot point for turning the bus and determining the route. The following figures illustrate this.

Middle Wire Segment is the Control Wire



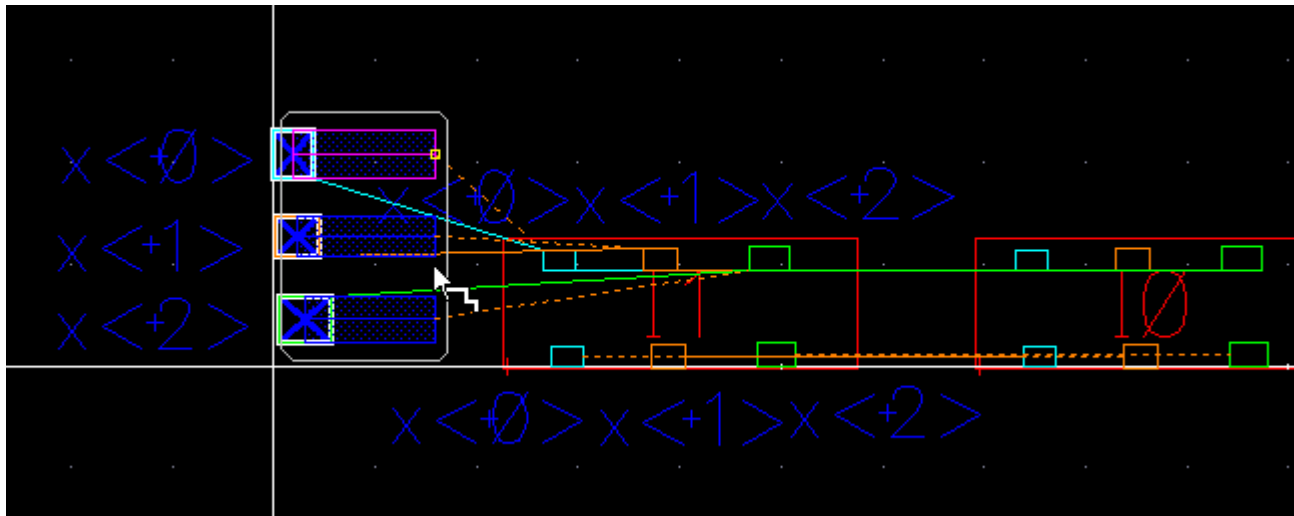
Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

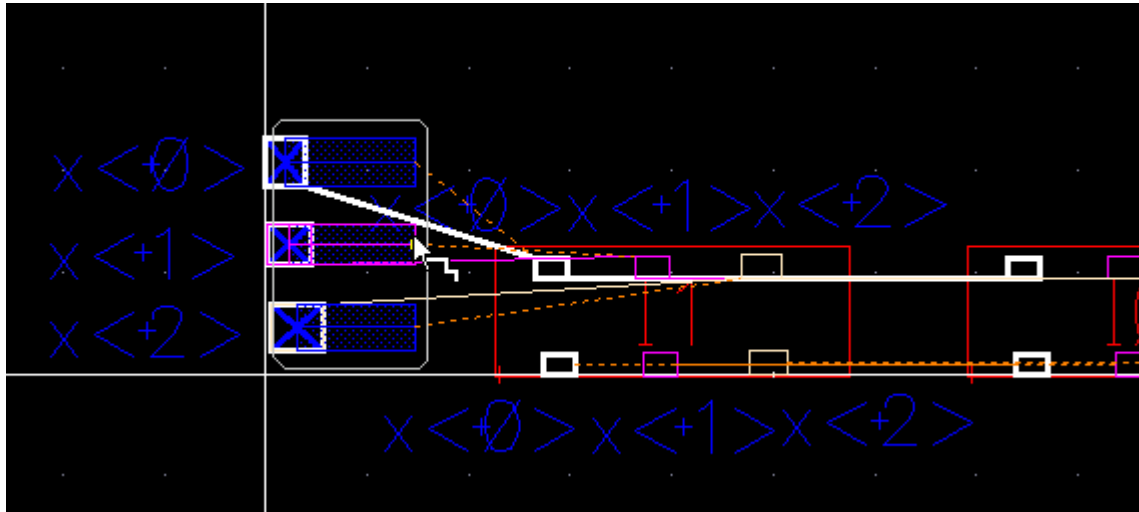
For best results, move the pointer over the yellow box at the end of the control wire when digitizing points for multiple wires. This reduces conflicts and allows better control of wire placement.

Flightlines are drawn from the current edited or created wires to the closest targets. Flightlines in a bus are controlled by the flightline starting from the control wire. The flightline from the control wire always targets the closest object, called the control target. Flightlines from other bus wires target pins that belong to the same instance and are of the same type or LPP as the control target. If no preferred choice target can be determined based on the control target, the closest object is chosen as the target.

In the example below, wire on `netc` is the control wire and its control target is an `instPin` of the instance `busSub`. Instance pins on `neta` and `netb` of the instance `busSub` are not the closest targets for the wires on `neta` and `netb` but they belong to the same instance as the control target. Therefore, flightlines of wires on `netx<2>` and `netx<1>` are also drawn to `busSub` instead of being drawn to `I1`.



If, however, wire on `netb` is the control wire, as shown below, the only appropriate targets for wires on `neta` and `netc` are their respective closest targets. For wire on `neta`, it is the `instPin a` of instance `I1`. For wire on `netc`, the closest target is the `instPin c` of instance `busSub`.



Editing a Bus

The *Edit Bus* command is available in Layout XL and higher tiers and lets you interactively change the width and spacing gap for the selected bus. The selected bus must consist of a set of parallel wires with the same number of bends. You can edit separate portions of a bus, several bits, or the entire bus depending on the selection. Only the selected parts of a bus can be modified while the unselected parts can be edited in certain cases to ensure valid connectivity. You cannot change the width and spacing gap of the bus segments that are directly connected to pins.

The bus bit table in the Edit Bus form is used to set up new width and spacing, either on an individual bus bit or on all the selected bus bits. To edit the width or spacing of the selected part of a bus:

1. Choose *Edit – Wiring – Bus*. Press `F3`.

The Edit Bus form opens displaying the bus bit table.

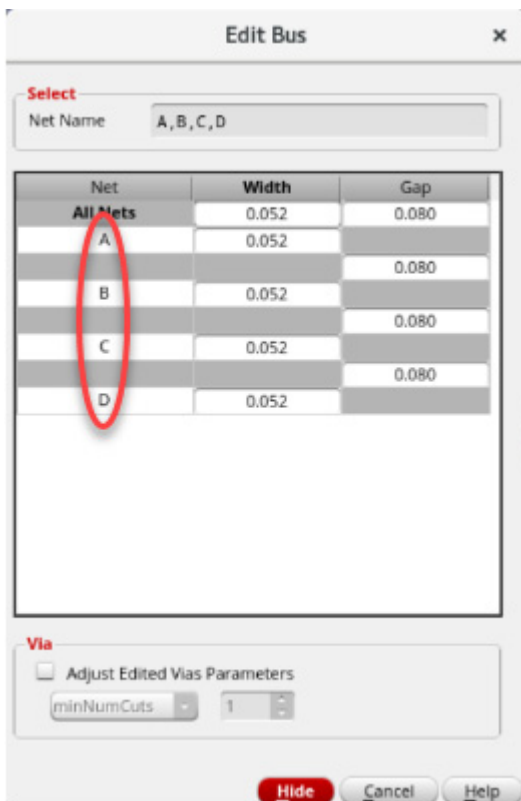
Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

When there is no bus selected, the *Net* column is empty and a message is displayed informing you to select at least two bus bits to change their width or gap space.



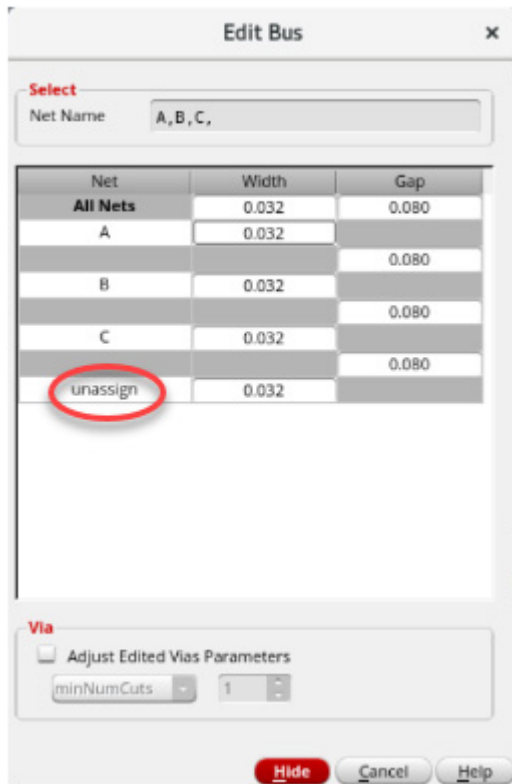
If a bus with net names is selected before running the *Edit – Wiring – Bus* command, the net name corresponding to each selected bus bit appears in the bus bit table as shown in the figure below.



Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

If a selected bus bit does not have a net assigned, *unassign* is displayed in the *Net* column.



2. To change the width, specify the new value in the *Width* column corresponding to the net name of the bus bit.

For example, if you want to change the width of bus bit with net name *d*, specify the new value in the *Width* column corresponding to net name *d*.

Once the width of one bus bit is modified, the *All Nets* field of the bus bit table displays *Various* as the different width values are not homogeneous anymore. The *Various* value in the *Width* and *Gap* columns of the bus bit table appears when the width of all the

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

selected bus bits is not the same or when the spacing between all the selected bus bits is not the same.

Edit Bus [X]

Select

Net Name: d, c, B, A

Net	Width	Gap
All Nets	Various	Various
d	0.240	
		0.300
c	0.240	
		0.200
B	0.080	
		0.200
A	0.240	

Via

— Adjust Edited Vias Parameters

minNumCuts: 1

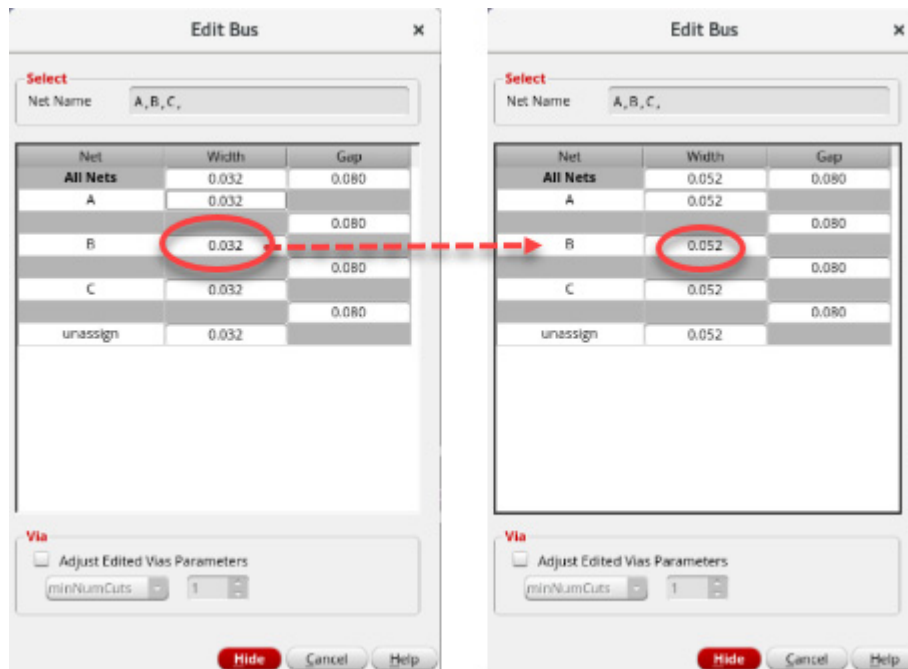
[Hide] [Cancel] [Help]

To change the width of all selected bus bits, specify the new width value in the *Width* column next to *All Nets*. When you specify the new width value, the width of all the selected bus bits is automatically updated and the spacing between the edited bus bits

Virtuoso Interactive and Assisted Routing User Guide

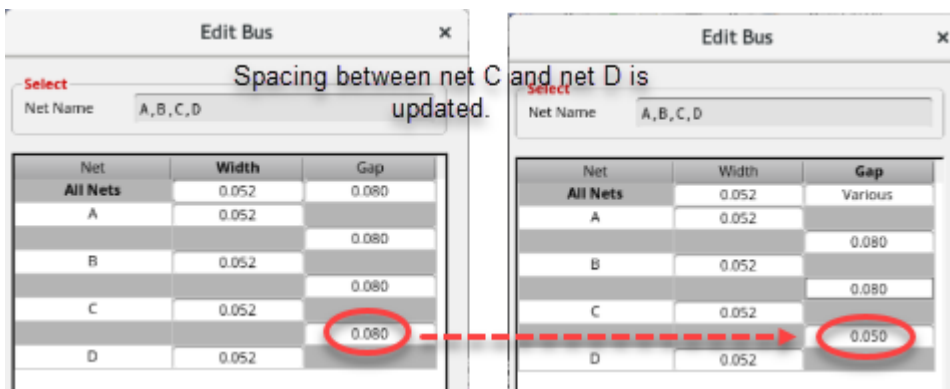
Working with Buses

is automatically adjusted, if necessary, to avoid DRC violations. The following figure shows the edited width of the selected bus bits and the adjusted spacing between them.



- To change the spacing between two selected bits of a bus, specify the new spacing value in the *Gap* column between the two bus bits.

For example, if you want to change the spacing between the bus bits with net name *c* and *d*, specify the new spacing value in the *Gap* column between the net name *c* and *d*. The following figure shows how the spacing is adjusted between the edited bus bits.



Once the spacing between the two selected bus bits is modified, the spacing between all the selected bus bits is not consistent and the common *Gap* field of the bus bit table displays *Various*.

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

To change the spacing between all the selected bus bits, specify the new spacing value in the *Gap* column next to *All Nets*. The following figure shows how the spacing gets adjusted between all the selected bus bits.

The figure shows two instances of the 'Edit Bus' dialog box. Both have 'Net Name' set to 'A, B, C, D'. The left dialog shows a table with 'Width' 0.052 and 'Gap' 0.080 for 'All Nets'. The right dialog shows the 'Gap' value changed to 0.045, which is highlighted with a red circle.

Net	Width	Gap
All Nets	0.052	0.080
A	0.052	
B	0.052	0.080
C	0.052	0.080
D	0.052	0.080

Net	Width	Gap
All Nets	0.052	0.045
A	0.052	0.045
B	0.052	0.045
C	0.052	0.045
D	0.052	0.045

4. In the *Via* section of the Edit Bus form, select *Adjust Edited Vias Parameters*.

The drop-down list is enabled and the *minMumCuts* option is selected by default.

5. Specify the number of via cuts in the field next to the drop-down list. The size of all the selected vias is immediately updated while the width remains unchanged.

By specifying the via cuts as minimum number of cuts, you can override the `minNumCuts` rule value for creating and editing bus bits. If the override value is less than either the technology rule or the current constraint group on the net or on the design, then the value specified in the Edit Bus form resets to the minimum value.

6. Click the drop-down list and select *Row/Col*.
7. Specify the via cuts as the number of rows or the number of columns of via cuts in the fields next to the drop-down list. By doing this, you can override the `minNumCuts` rule with a more explicit configuration for creating and editing bus bits.

The increase and decrease in the number of via rows and columns is dependent on the minimum number of via cuts.

`number of via rows * number of via columns >= minimum number of via cuts.`

The above condition must always be satisfied, otherwise a `minNumCuts` violation is created by the interactive routing commands, such as *Create Wire*. To avoid the violation, the Edit Bus form adjusts the number of via rows and columns whenever it is required to satisfy the condition. For example, if minimum number of via cut is set to 9, then the row and column automatically adjusts to 1 and 9.

When the `minNumCut` value is increased, the column and row value is automatically adjusted to meet the constraint. However, if the `minNumCuts` value is reduced, the

existing larger values for columns and rows still satisfy the rule and therefore no automatic adjustment is made to decrease the values. However, the number of rows and columns can be manually reduced to satisfy the `minNumCuts` constraint. When either rows or columns is updated to a very small number that does not satisfy the existing `minNumCuts` setting, the opposite (for example, rows and columns, respectively), is changed to a higher number in order to satisfy the rule.

Related Topics

[Edit Bus Form](#)

Avoiding Shorts with Existing Blockages for Create Bus

The *Create Bus* command has been enhanced to avoid creating a short with an existing blockage by skipping tracks. To route the net without a short, the *Create Bus* command uses the next nearest track that does not have a blockage. For more information, see [Avoiding Shorts with Existing Blockages](#) described for the *Create Stranded Wire* command.

Supporting Multi-Layer Bus

The multi-layer bus functionality is available only in Layout XL and higher tiers. With multi-layer bus support, you can create and digitize the bits of a bus on different metal layers. A multi-layer bus can be created from the following starting points:

- Existing pins
- Existing bus pre-routes (see [Restarting a Bus From Existing Routes](#))

Important

You cannot create a multi-layer bus starting in an empty space. This is because you need to define the layer for each bus bit. For a single-layer bus, the bus is automatically drawn on the layer set as active in the Palette.

This section describes the following multi-layer bus functionality.

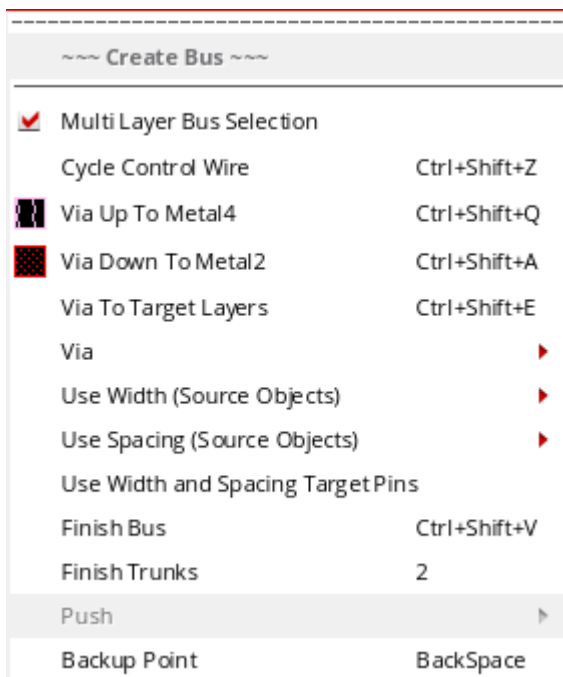
- [Creating a Multi-Layer Bus from Existing Pins](#)
- [Creating a Multi-Layer Bus from an Empty Space](#)
- [Creating a Stacked Multi-Layer Bus from a Single Pin](#)
- [Creating a Stacked Multi-Layer Bus from Multiple Pins](#)

- [Defining a Custom Layer Pattern](#)
- [Transitioning Capabilities in a Multi-Layer Bus](#)
- [Stacked Wires within Multi-Layer Bus](#)
- [Transitioning Capabilities in a Wire Stack within a Multi-Layer Bus](#)
- [Customizing Multi-Layer Bus Transition](#)

Creating a Multi-Layer Bus from Existing Pins

To create a multi-layer bus starting from pins located on different metal layers:

1. Choose *Create – Wiring – Bus* or click on the *Create Bus* icon  on the *Create* toolbar.
2. Right-click anywhere in the canvas to display the [Edit Bus Form](#).
3. Choose *Multi-Layer Bus Selection*. If this option is not selected, the bus is created on a single metal layer even if the selected pins are on different metal layers.

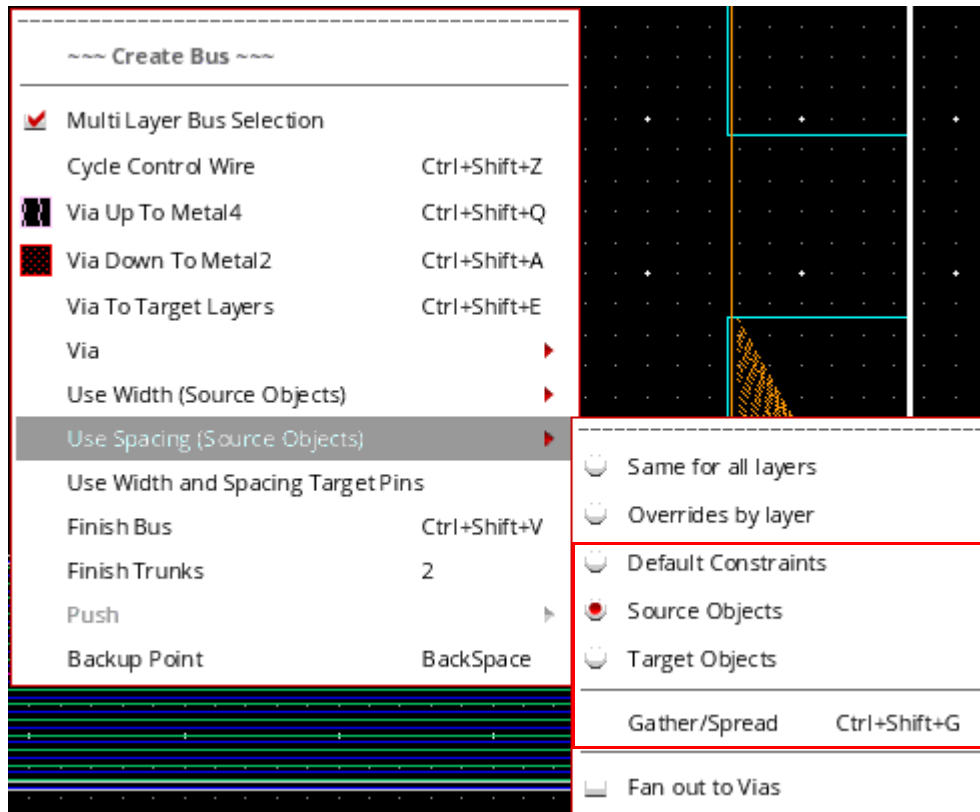


4. To start creating a bus by digitizing pins on different metal layers, select the required pins in the layout design.

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

5. Use the options available on the *Use Spacing* submenu to specify the spacing between the bus bits. The spacing between the bus bits of the multi-layer bus is based on *Default Constraints*, *Source Objects*, *Target Objects*, and *Gather/Spread*. By default, the *Source Objects* option is selected. You can change the spacing between the bus bits on different metal layers at any time while creating or reshaping a bus.

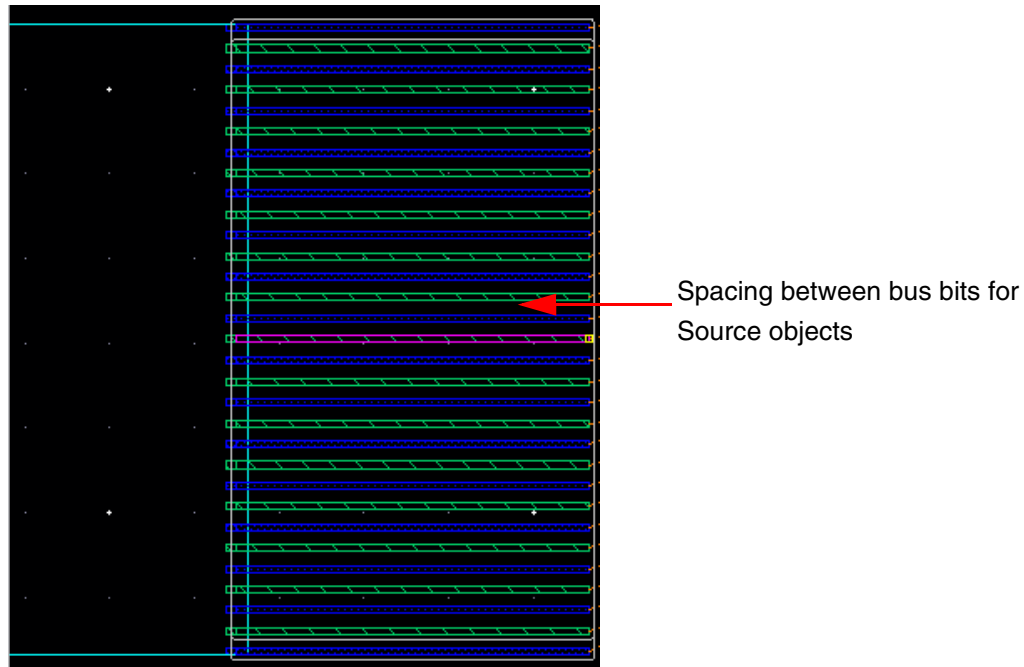


- ☐ Source Objects

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

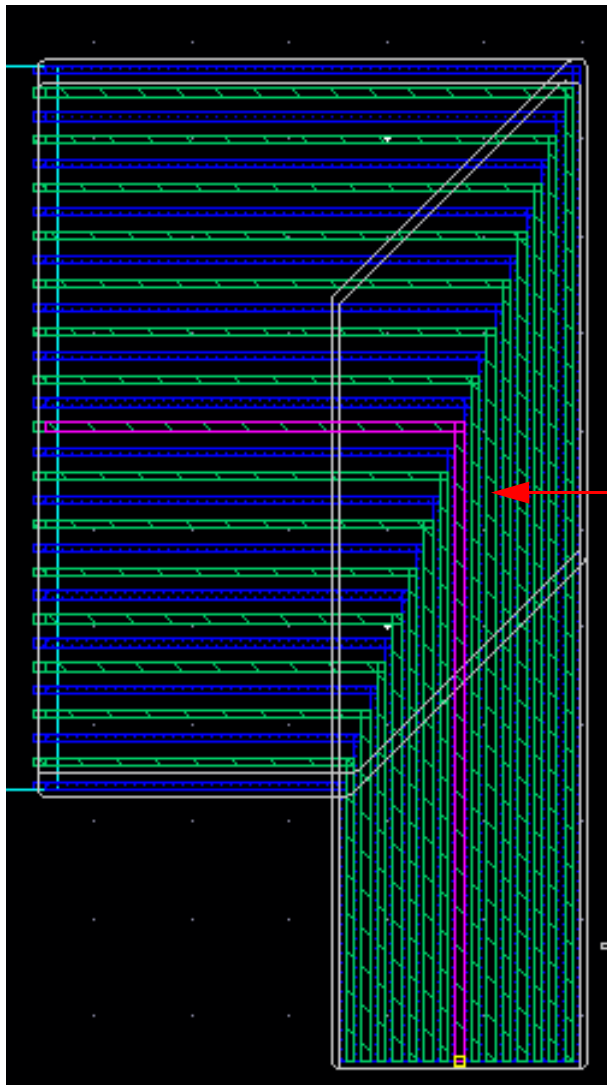
When selected, the spacing between the bus bits on different metal layers is adjusted to align with the spacing of the source object. The following figure shows the spacing between the bus bits when *Source Objects* is selected.



☐ Default Constraints

When selected, the bus bits are abutted because no minimum spacing exist in between the bus bits on two different layers. The following figure shows that the bus

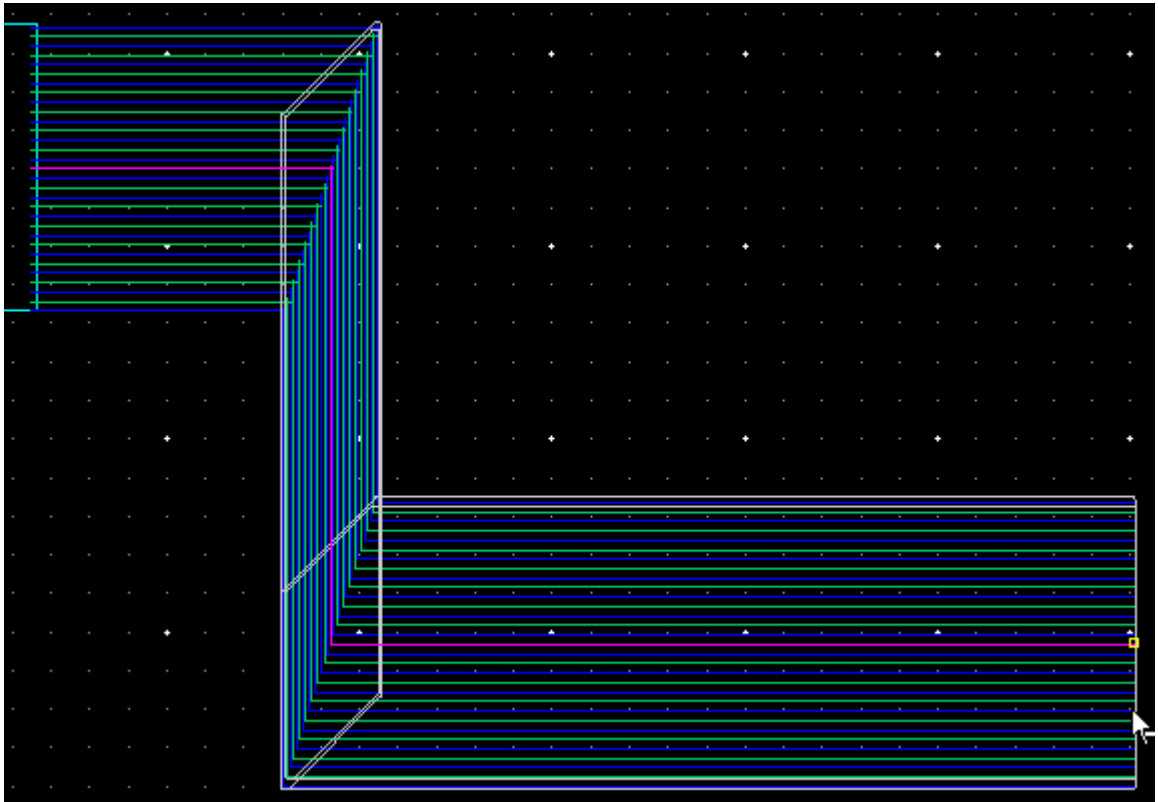
bits of different metal layers are abutted and there is no space in between the bus bits.



The bus bits on different metal layers are abutted without any space in between

❑ Target Objects

When selected, the spacing between the bus bits on different metal layers is adjusted to align with the spacing of the target object. The following figure shows the spacing between the bus bits when *Target Objects* is selected.

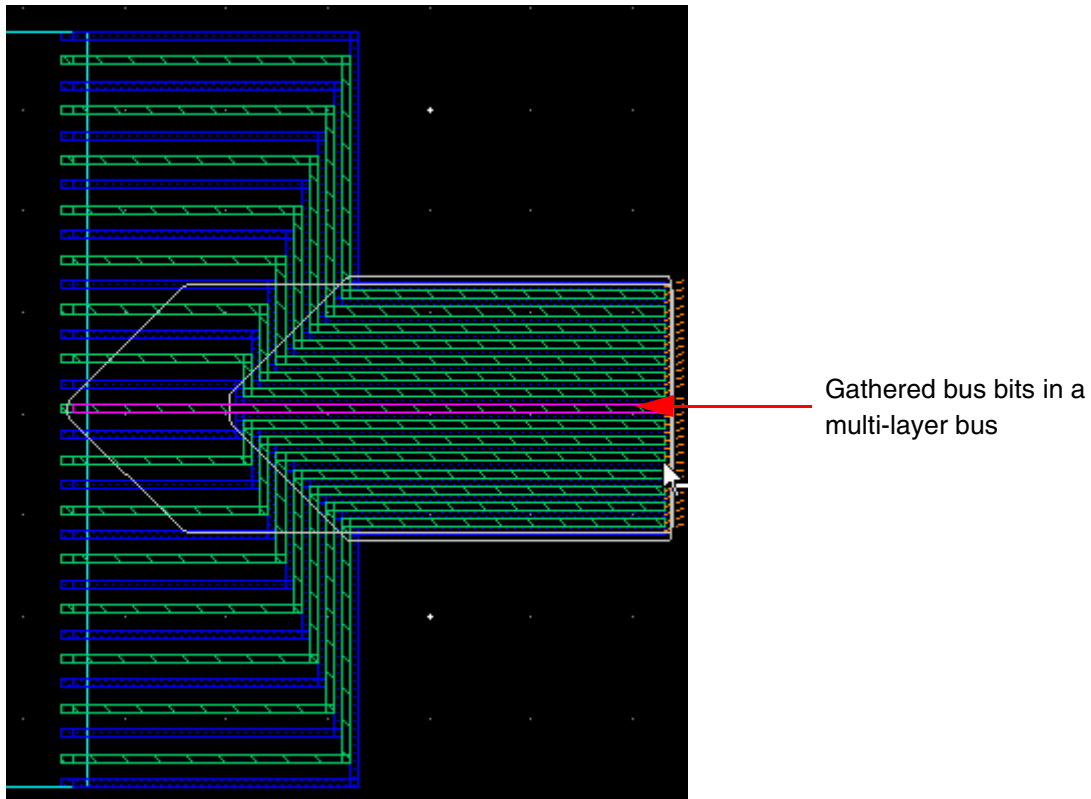


☐ Gather/Spread

To change the spacing between the bus bits that are in the same direction before a bend is inserted, use *Gather/Spread*. Alternatively, you can press the **Ctrl+Shift+G** keys.

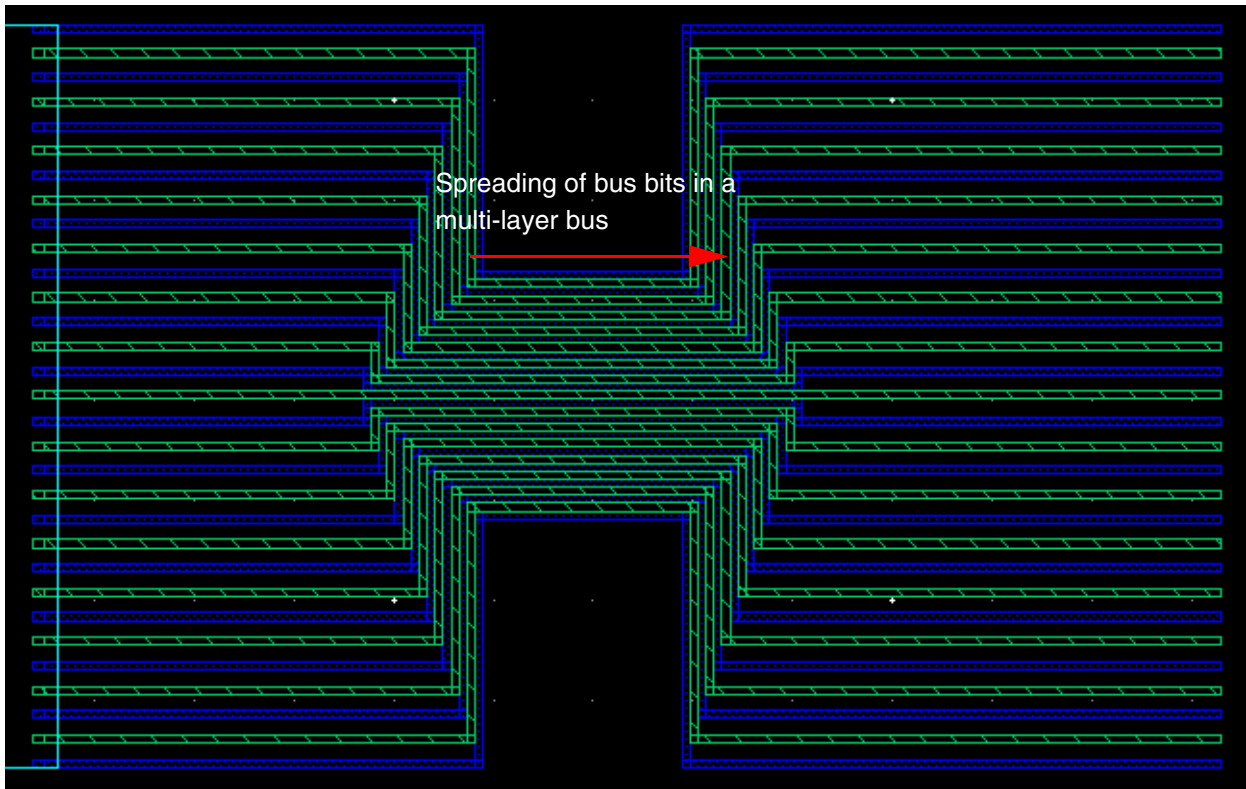
When gathered, the bus bits are spaced with minimum spacing in order to minimize the space used. After the bus bits are gathered, the adjacent bus bits on different layers are abutted; whereas, the adjacent bus bits on the same layer are evenly

spaced at a distance equal to the minimum spacing defined for that layer. In the following figure, you can see a bus with the bits gathered.



When spread, the bus bits are spaced out and placed with the same spacing as that applied before the gather operation or with the same spacing as that of the target objects.

In the following figure, you can see that the bus bits are spread with the same spacing as that applied before the gather.

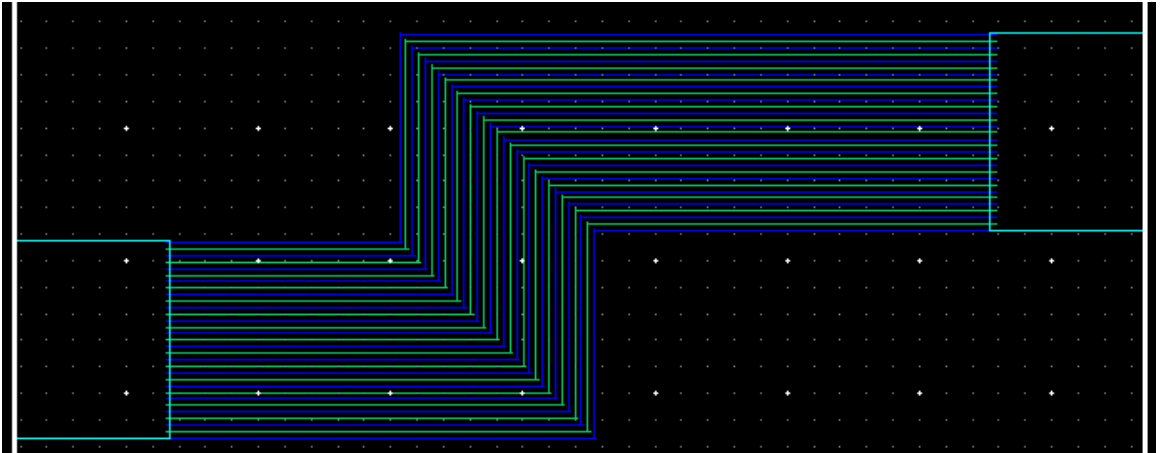


6. Use one of the following methods to complete the creation of a multi-layer bus,
 - a. *Auto Terminate*

Note: For automatic termination of a multi-layer bus, the *Auto Terminate* option available in the Create Bus form and the *Interactive* section of the Wire Assistant must be selected.

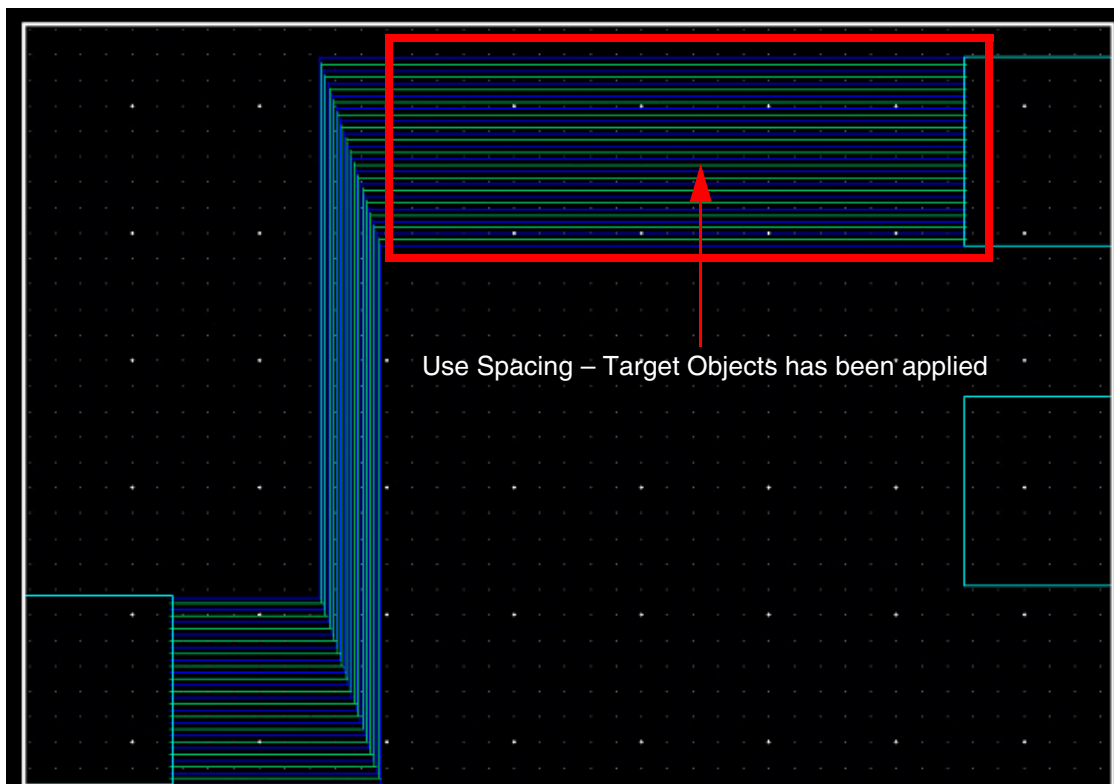
Place the control wire to overlap the target pin, and then double-click or press `Enter` to terminate the bus.

The bus bits are placed automatically at the required spacing. The following figure shows a multi-layer bus that is placed using *Auto terminate*.



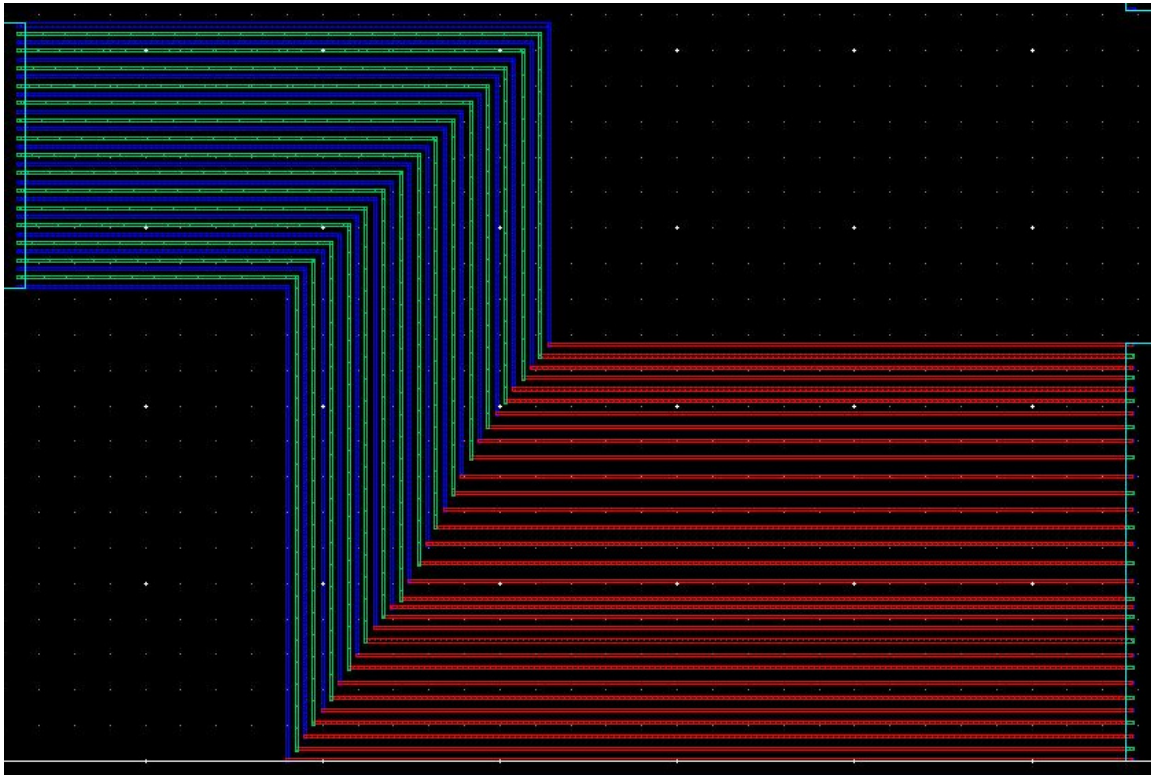
b. Target Objects

The *Use Spacing – Targets Objects* option helps prevent jogs, which get added when a bus is placed using *Auto Terminate*. The following figure shows that the spacing between the bus bits is adjusted based on the spacing between the target pins when the multi-layer bus creation is complete.



c. *Finish Trunk*

When the bus bits are routed, the twigs and trunks get inserted automatically to appropriately route the bus bits to the target pins that are in different order. The following figure shows a multi-layer bus placed using the *Finish Trunk* command. All the twigs are created on `Metal2` to complete the routing of the bus bits to the target pins that are in a different order.



Note: You can check the connectivity of the bus bits by using the Edit Properties form or the *Property Editor* assistant.

You can switch the bus bits between layers, but you can do this only after inserting a via. For more information, see [Transitioning Capabilities in a Multi-Layer Bus](#).

You can also specify a pattern for the vias in a multi-layer bus. The via pattern in a multi-layer bus is similar to the one supported for a single-layer bus.

Creating a Multi-Layer Bus from an Empty Space

To create a multi-layer bus starting from an empty space:

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

1. Choose *Create – Wiring – Bus* or click on the *Create Bus* icon  on the *Create* toolbar.

2. Press F3.

The Edit Bus Form opens.

3. Select the *Custom Layer Pattern* check box to enable a customized layer pattern for creating a multi-layer bus.

4. Specify a pattern for creating a multi-layer bus in the *Custom Layer Pattern* combo box. The specified pattern is added to the drop-down list, along with other available patterns.

You can also select an existing available pattern from the *Custom Layer Pattern* combo box. The specified custom layer pattern is repeated to adjust the number of bits that have

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

been specified for the multi-layer bus. This happens when the number of bus bits is more than the number of layers specified in the pattern.

The screenshot shows the 'Create Bus' dialog box with the following settings:

- Connectivity:** Net Name (empty), ☐ Probe Nets
- Width and Spacing:** All Nets Width: 0.08, All Nets Bit Spacing: [Various], ☐ Fan Out to Vias
- Constraints:** ☐ Create Bus Constraint
- Bits and Justification:** Number of Bits: 4, Justification: center, Offset: 0
- Begin/End Style:** Begin Style: auto, Begin: 0, End Style: auto, End: 0
- Snap Mode:** orthogonal
- Options:** ☐ Snap to Pin Center, ☒ Auto Terminate, ☐ Snap to Via Center, ☐ Cover Pin
- Custom Layer Pattern:** ☒ Custom Layer Pattern, Metal1 Metal2 (Metal2 Metal13)

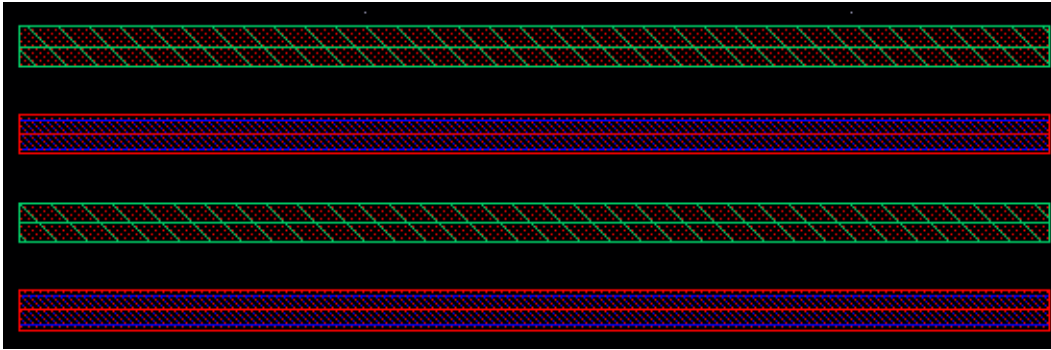
Buttons at the bottom: Hide, Cancel, Defaults, Help.

For more information on how to define a custom layer pattern, see [Defining a Custom Layer Pattern](#).

5. To start creating a digitized multi-layer bus in an empty space, click in the design display area.

The following figure illustrates how a multi-layer bus is created with eight bus bits and a custom layer pattern defined using only four bits (Metal1 Metal2) (Metal2 Metal13). In this case, the custom layer pattern is repeated, (Metal1 Metal2)

(Metal2 Metal3) (Metal11 Metal12) (Metal2 Metal13), to adjust the number of bits that have been specified for the multi-layer bus.



6. Right-click to display the Edit Bus Form.

Use the options available on the *Use Width* submenu to specify the width of the multi-layer bus bits.

Similarly, use the options available on the *Use Spacing* submenu to specify the spacing between the stacks in a multi-layer bus.

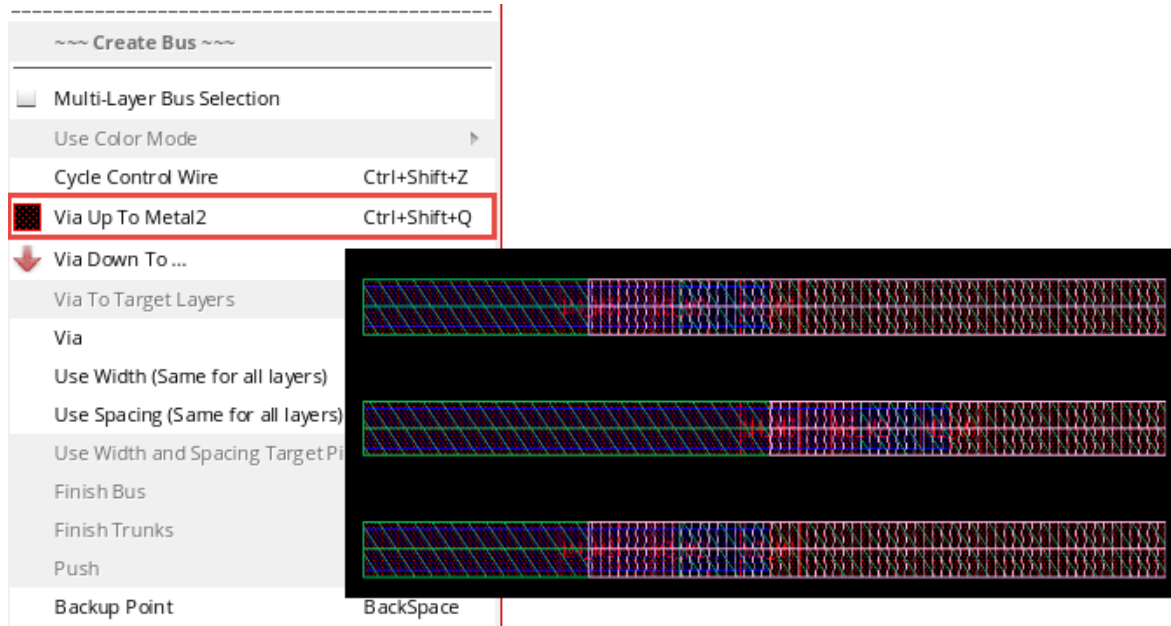
For more information about multi-layer bus support, see Supporting Multi-Layer Bus.

7. You can move the bus bits up or down one or multiple layers. However, you can do this only after inserting a via. To do this:
- Start the *Create Bus* command.
 - Start creating a multi-layer bus by clicking at a point in an empty space in the design display area.
 - Right-click and select *Via Up To <layer name>* or *Via Down To <layer name>* to move the bus bits up or down a layer, respectively. The following figure shows an example in which vias are inserted for a multi-layer bus and the bus bits on different

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

metal layers move up to a higher metal layer when *Via Up To <layer name>* is selected.



You can also move all the bus bits to the same layer, and then switch back to a multi-layer configuration. For more information, see [Switching Bus Bits to a Previously Used Multi-Layer Configuration](#).

8. Press **Enter** to complete the multi-layer bus creation.

Creating a Stacked Multi-Layer Bus from a Single Pin

To create a stacked multi-layer bus from a single pin:

1. Choose *Create – Wiring – Bus* or click the *Create Bus* icon  on the *Create* toolbar.
2. Press **F3** on your keyboard.
The Edit Bus Form opens.
3. Select the *Custom Layer Pattern* check box to enable a customized layer pattern for creating a multi-layer bus.
4. Specify the pattern as (Metal1 Metal2 Metal3) in the *Custom Layer Pattern* field.
5. Select a pin on Metal1.

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

A multi-layer bus on Metal1, Metal2, and Metal3 is created. The bus created on Metal1 layer gets the connectivity from the pin on the layer, whereas the bus on Metal2 and Metal3 does not have any connectivity.



Similarly, when you select pins on different layers with same net connectivity, the multi-layer bus created on metal layers gets the connectivity based on the pins selected on the different metal layers.

The following figure shows an example, where the pins are on Metal1 and Metal2 and the bus created on Metal1 and Metal2 has connectivity, whereas the bus on Metal3 does not have connectivity.



Creating a Stacked Multi-Layer Bus from Multiple Pins

To create a stacked multi-layer bus from multiple pins:

1. Choose *Create – Wiring – Bus* or click the *Create Bus* icon  on the *Create* toolbar.

2. Press F3 on your keyboard.

The Edit Bus Form opens.

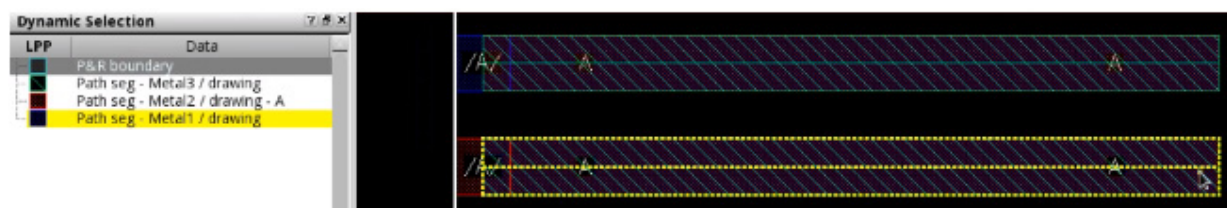
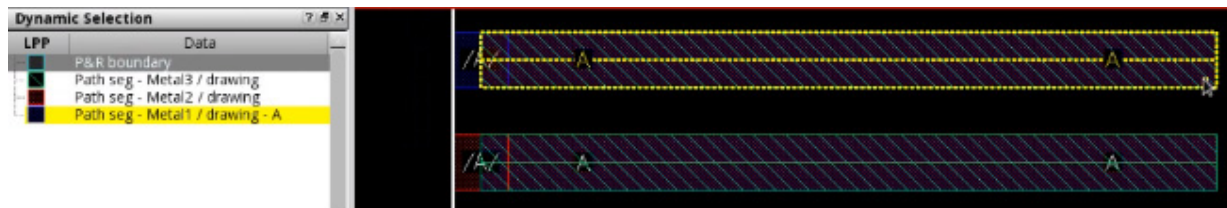
3. Select the *Custom Layer Pattern* check box to enable a customized layer pattern for creating a multi-layer bus.
4. Specify the pattern as (Metal1 Metal2 Metal3) in the *Custom Layer Pattern* field.
5. Select pins on Metal1 and Metal2.

A multi-layer bus on Metal1, Metal2, and Metal3 is created. The bus created from a pin on Metal1 layer gets the connectivity from the pin on the Metal1 layer. Similarly,

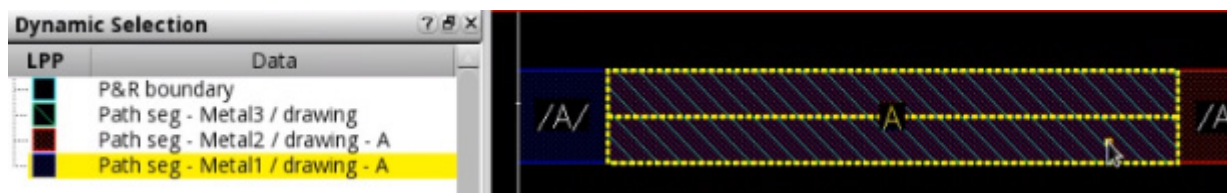
Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

the bus created from a pin on Metal2 layer gets the connectivity from the pin on the Metal2 layer. In this case, the bus on Metal3 layer does not have connectivity.

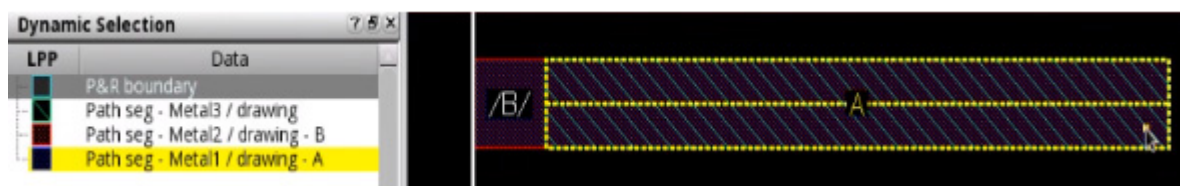


6. To complete the creation of multi-layer bus, select a target pin on a different metal layer. The following figure shows an example, when the source pin is on Metal1 and the target pin is on Metal2. In this case, the multi-layer bus created on Metal1 and Metal2 layers has the connectivity.



You can also start creating a multi-layer bus from pins on different nets. The following figure shows an example when two pins are on different nets and different metal layers.

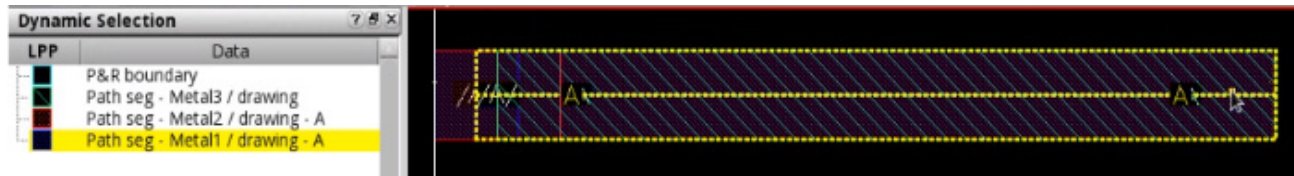
In this case, when you start the *Create Bus* command, the multi-layer bus on Metal1, Metal2, and Metal3 are created. The bus created on Metal1 and Metal2 has connectivity, whereas the bus on Metal3 does not have any connectivity.



Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

In addition, you can also create a multi-layer bus starting from pins with different widths. In this case too, the multi-layer bus created on metal layers with pins gets the connectivity, as shown in the following figure.



However, if the pins are of different vertical widths and are not center aligned, the multi-layer bus cannot be created.

Defining a Custom Layer Pattern

You can define a custom layer pattern for creating a multi-layer bus starting from an empty space.

The custom layer pattern is a string. The syntax to define a custom layer pattern is:

```
(layerName1 [purposeName1] layerName2 [purposeName2] ...) [(layerName3  
[purposename3]) ...]
```

In this format, each bus bit is defined with a layer name or a layer-purpose pair name. For example, if the pattern is defined as (Metal1 drawing) (Metal2 drawing), the first bus bit in a two-bit bus is routed on layer-purpose pair Metal1 drawing and the other bus bit is routed on layer-purpose pair Metal2 drawing.

If an error occurs while defining a custom layer pattern, such as parenthesis mismatch (extra '(' or ')'), unknown layer name, unknown purpose name, or purpose name specified before layer name, a warning message is displayed. The following figure shows an example of the type of message that is displayed.

```
*WARNING* (WE-21215): Extra '(' found in the specified syntax while parsing custom layer  
purpose pattern. Ensure that the syntax is specified in the format, "(layerName1  
[purposeName1] layerName2 [purposeName2] ...) [(layerName3 [purposename3]) ...]".
```

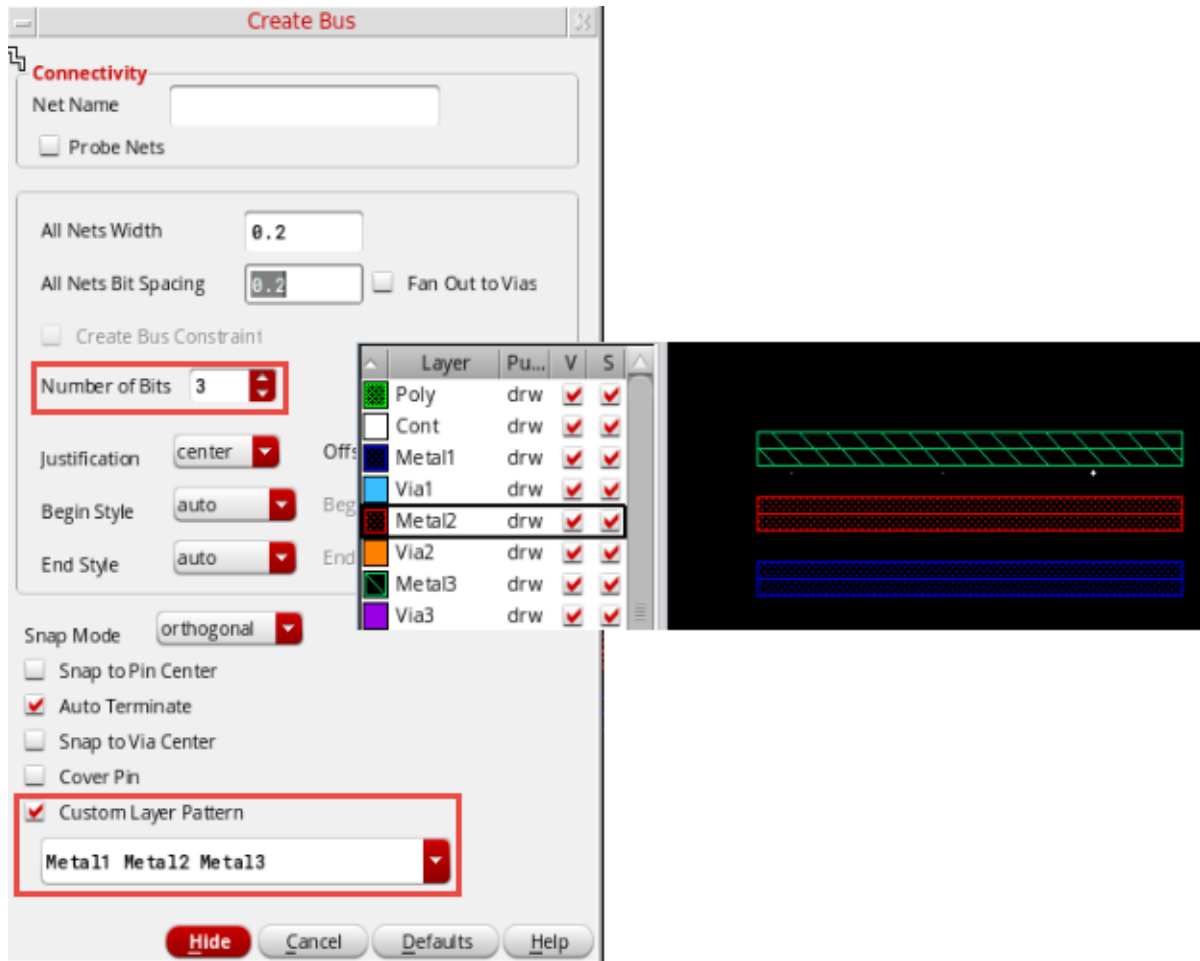
The following are some of the formats in which a custom layer pattern can be defined for a multi-layer bus.

- A custom layer pattern with non-stacked bus bits on different metal layers. For example, Metal1 Metal2 Metal3.

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

The following figure shows an example with three bus bits and the custom layer pattern defined as Metal1 Metal2 Metal3. In this case, the defined pattern has single bus bits on three different metal layers.



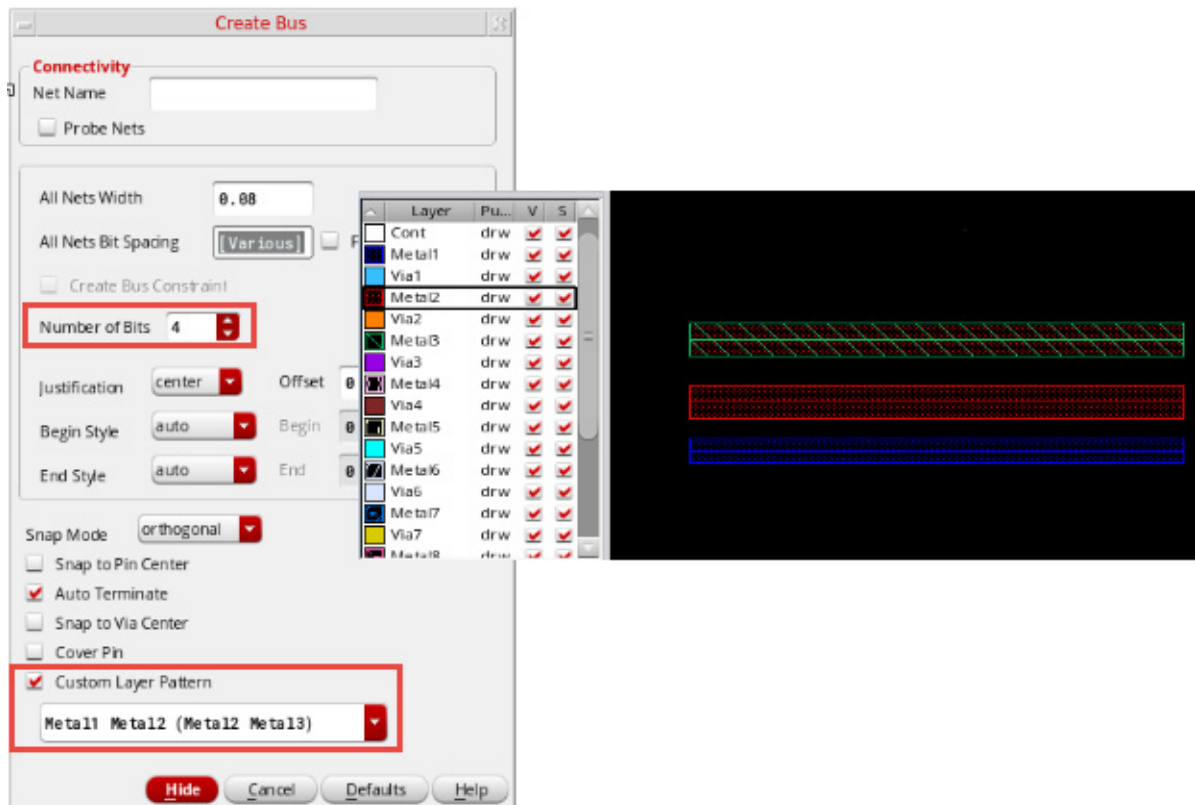
- Multiple layer names specifying a combination of non-stacked and stacked bus bits. For example, Metal1 Metal2 (Metal2 Metal3) or Metal1 (Metal2 Metal3) Metal3.

The following figure shows an example with four bits and the custom layer pattern is combination of non-stacked and stacked bus bits Metal1 Metal2 (Metal2

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

Meta13). In this case, the defined pattern has single bus bits on two different metal layers and a stack of two bus bits.

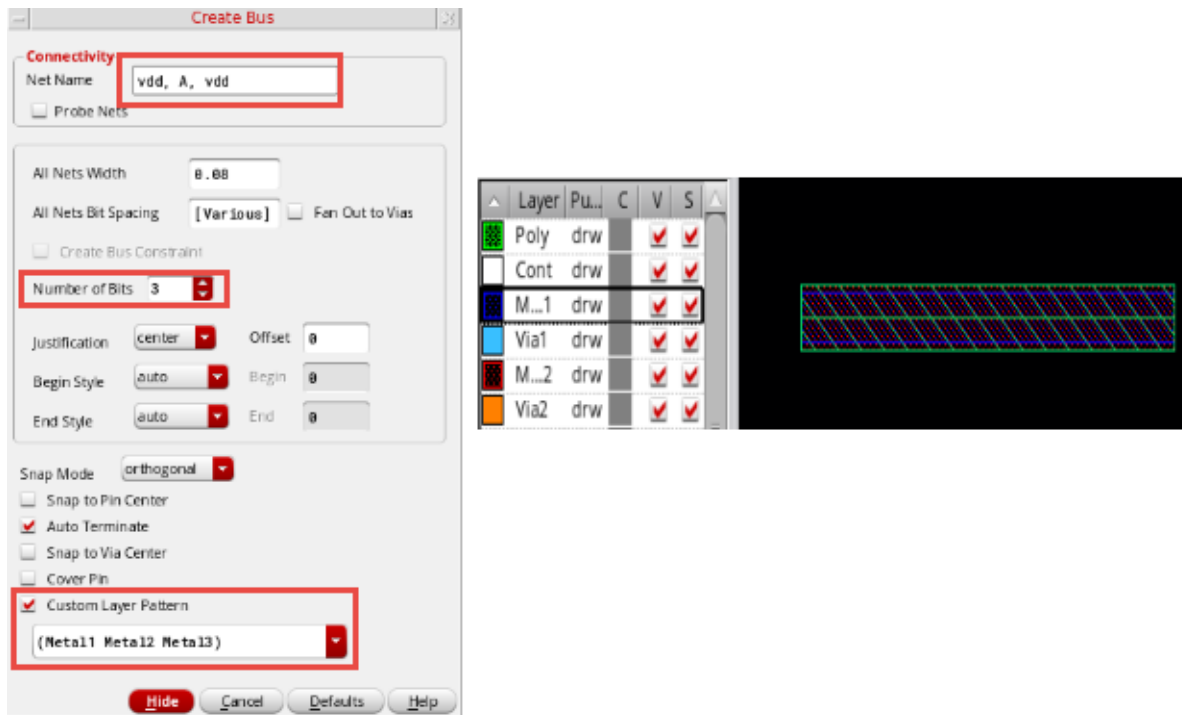


- Multiple layer-purpose pairs enclosed in parenthesis. This defines stacked wires where centerlines overlap each other. For example, (Meta11 Meta12 Meta13) defines a stack of three bus bits.

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

The following figure shows an example with three bus bits and the custom layer pattern that is defined consists of three layer names (Metal1 Metal2 Metal3). In this case, a stack of three bus bits is created.



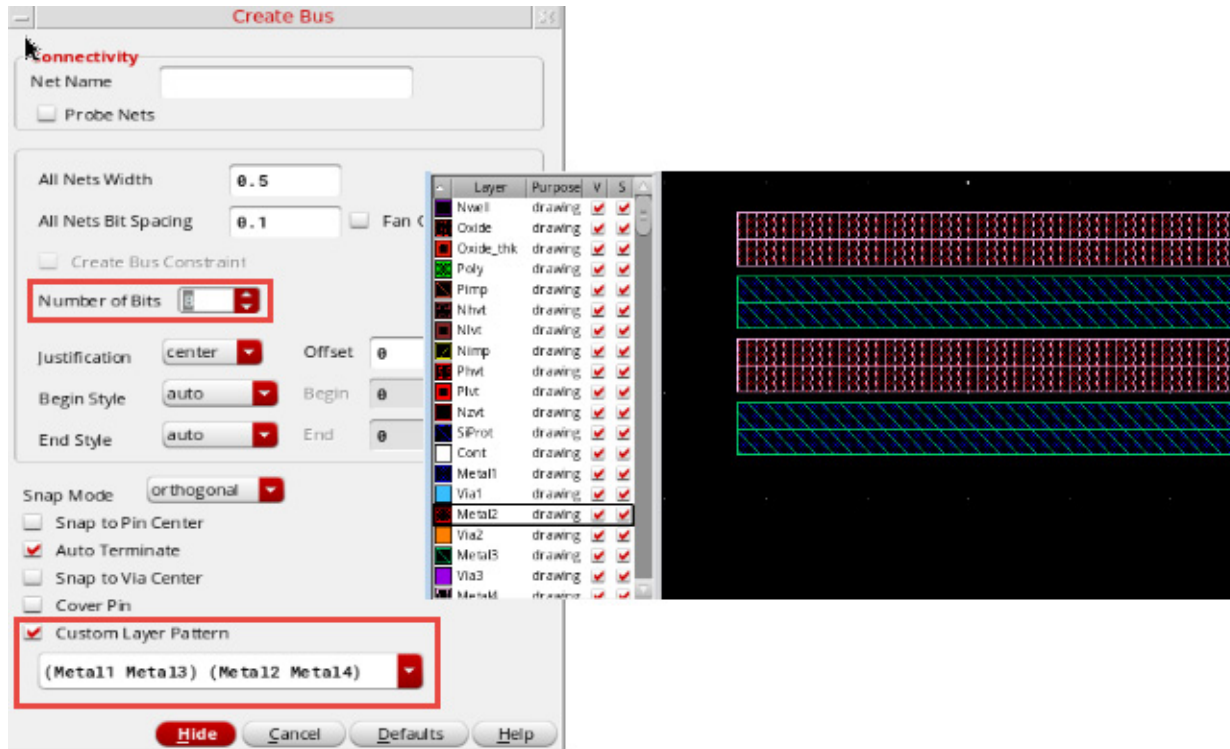
- A custom layer pattern with multiple stacks. For example, (Metal1 Metal3) (Metal2 Metal4) defines a pattern having two stacks and each stack consisting of two-bit pattern.

The following figure shows an example with eight bits and the custom layer pattern consists of two stacks (Metal1 Metal3) (Metal2 Metal4). This custom layer

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

pattern defines two stacks, each with a two-bit pattern. However, the four stacks that are seen is because of the repetition.



Transitioning Capabilities in a Multi-Layer Bus

You can move the bus bits up or down one or multiple layers at a time. However, you can do this only after inserting a via. You can also move all the bus bits to the same layer, and then switch back to a multi-layer configuration.

Switching Bus Bits between Layers

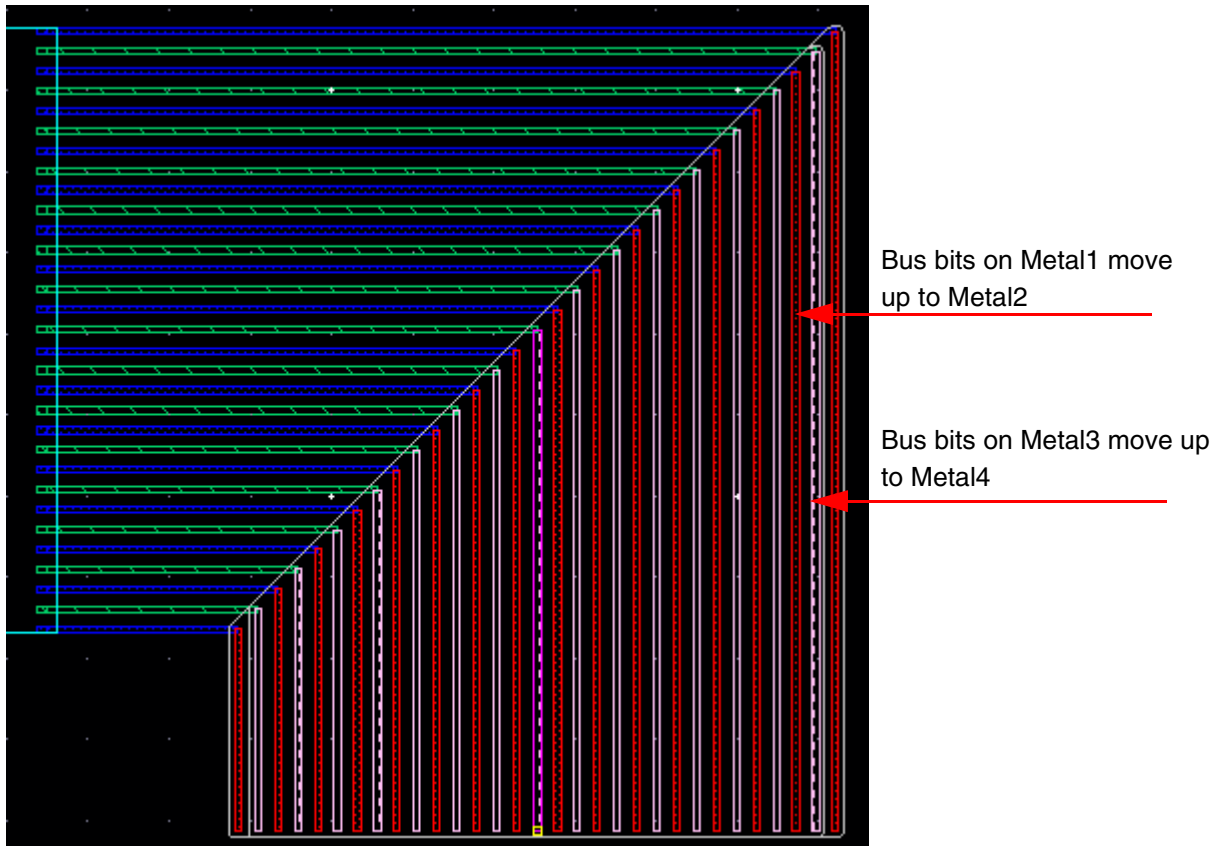
You can switch the bus bits between layers by moving up or down one or multiple layers.

1. Start the *Create Bus* command.
2. Click and drag to select the required pins.
3. Right-click the bus being created and select *Via Up To <layer name>* or *Via Down To <layer name>* to move the bus bits up or down a layer, respectively.

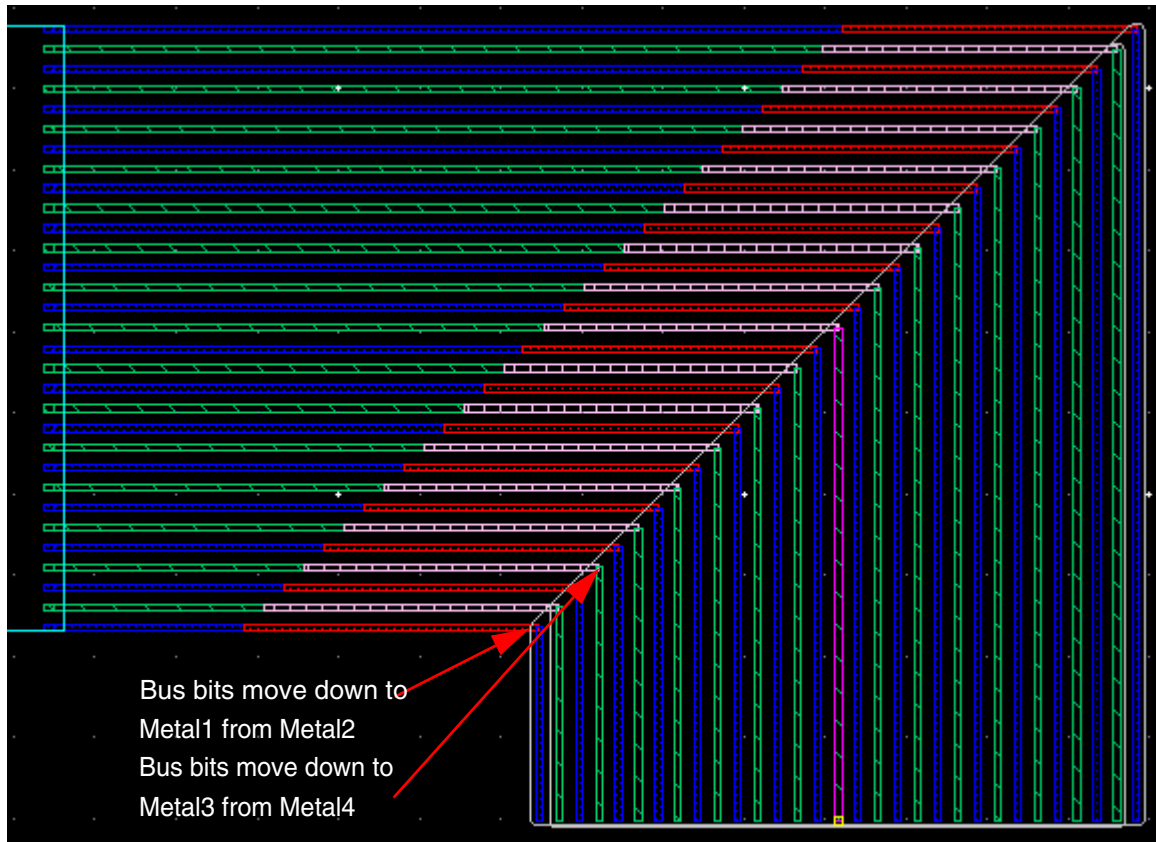
Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

The following figures shows an example where the bus bits on Metal1 move up to Metal2 and the bus bits on Metal3 move up to Metal4 when *Via Up To <layer name>* is selected



Similarly, in the following figure, you can see an example in which the bus bits on Metal2 move down one layer to Metal1 and the bus bits on Metal4 move down to Metal3 when *Via Down To <layer name>* is selected.



Switching Bus Bits to a Target Layer

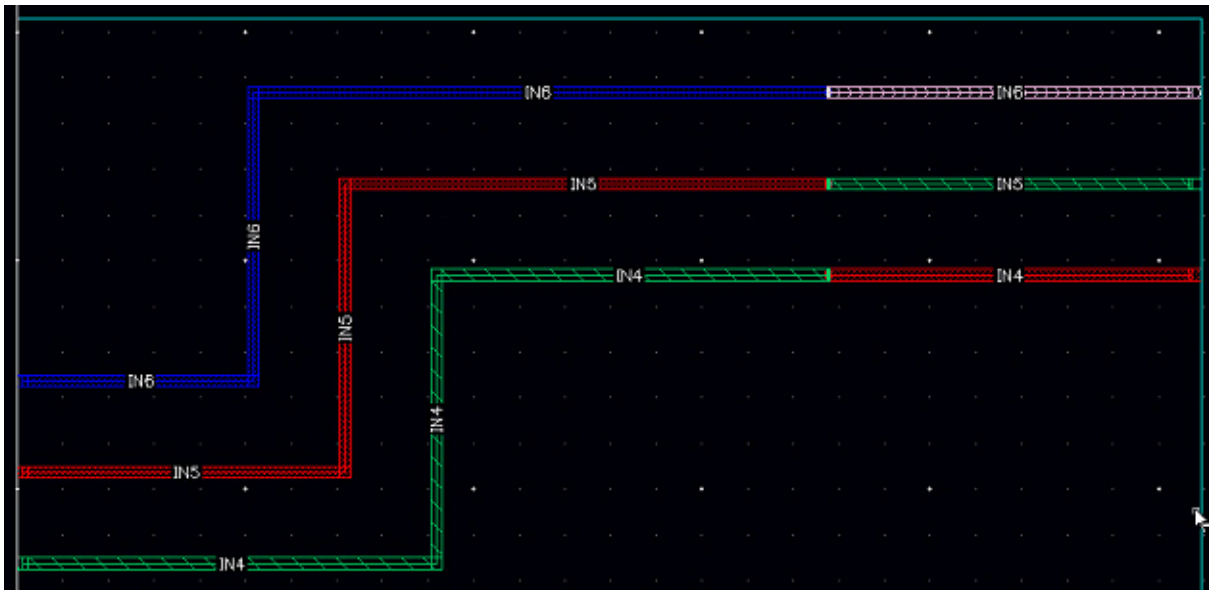
You can add vias to the layer of the target objects. The target objects are all the objects that are pointed to by the flightlines when the *Create Bus* command is run. However, if the flightline option in the Wire Assistant is off, then vias are not added.

1. Start the *Create Bus* command.
2. Click and drag to select the required pins.
3. Right-click the bus being created and select *Via To Target Layer* to move the bus bits to the layer of the target objects. The following figures shows an example where the bus bits on Metal1 move up to Metal4 as Metal4 is the layer of the target objects.

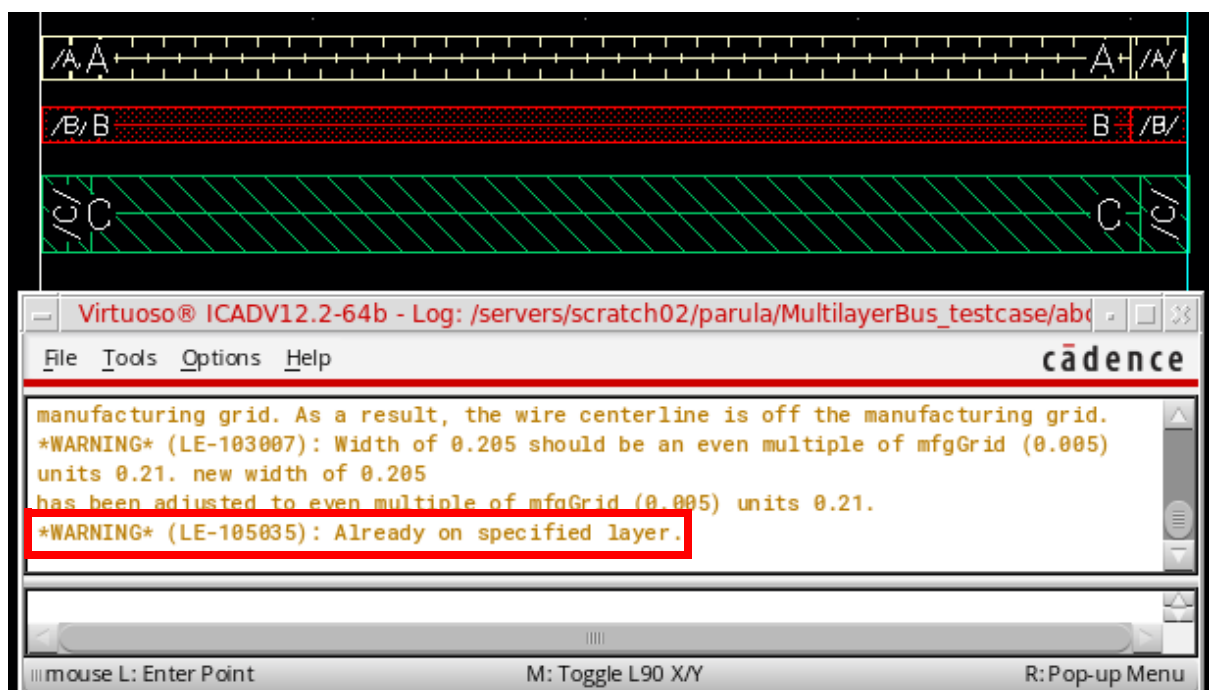
Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

Similarly, the bus bit on Metal 2 and Metal3 move to Metal3 and Metal2, respectively, as they are the layer of the target objects.



When a bus bit is already on the same layer as its target object layer, then no via is added for that bus bit. If all bus bits are already on the same layer as their respective target objects, then a via is not added and a message is displayed in CIW as follows:
"WARNING* (LE-105035): Already on specified layer."



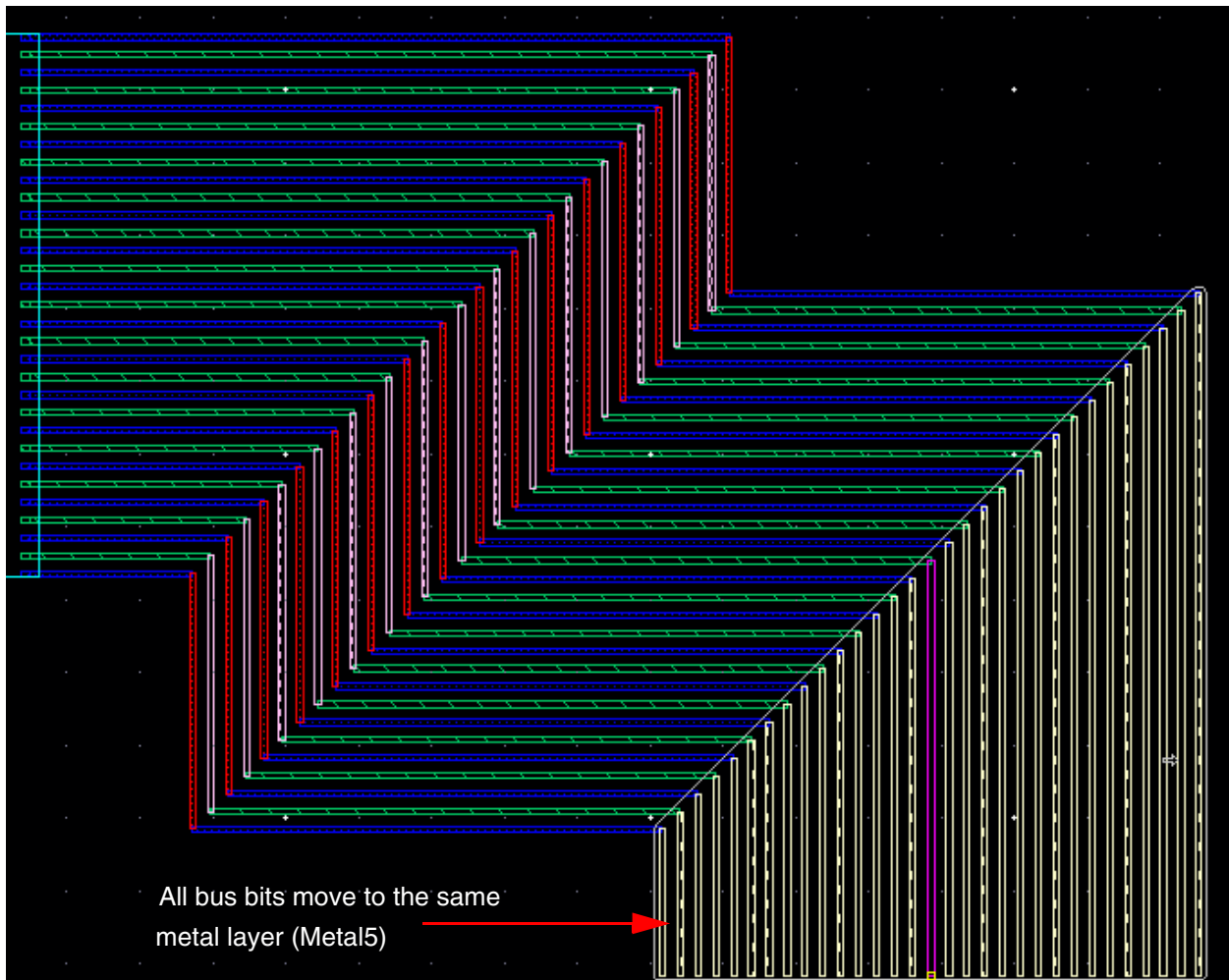
Switching Bus Bits to the Same Layer

To move all bus bits to the same layer:

1. Start the *Create Bus* command.
2. Start creating the bus by digitizing the pins.
3. Right-click the bus being created and point to *Via*.
4. Click *Select Via* or press the *Spacebar* key.

The Select Via form appears.

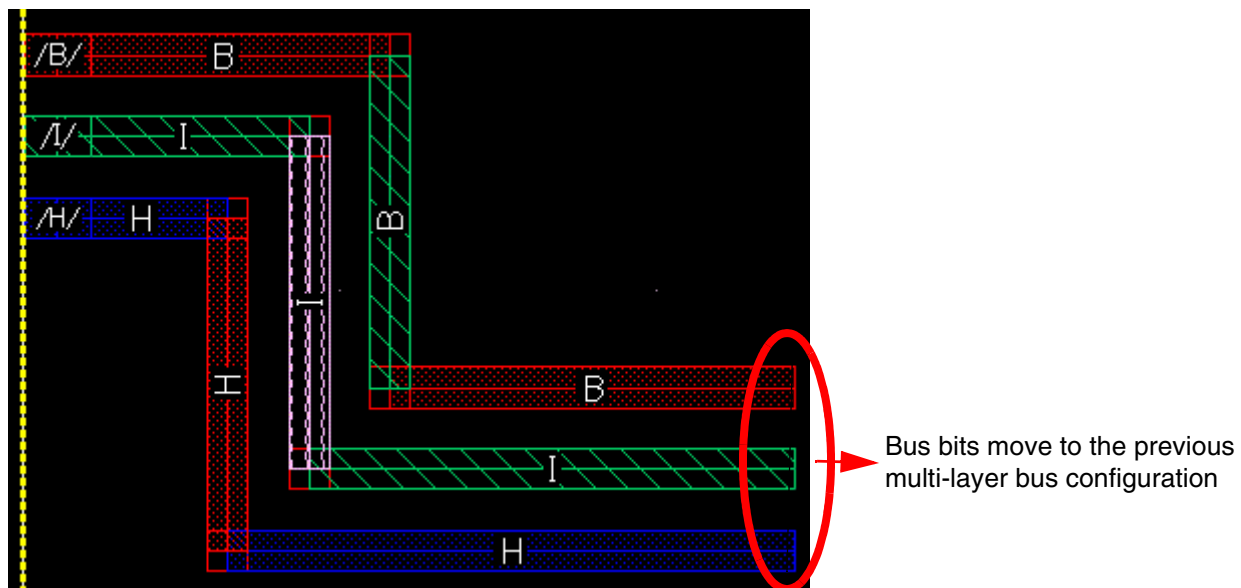
5. In the form, select the metal layer to which you want to move all the bus bits. In the following figure, all the bus bits are moved to *Metal5*.



Switching Bus Bits to a Previously Used Multi-Layer Configuration

Switching the bus bits between a multi-layer configuration and the same metal layer or a target layer provides an easy way to tunnel the bus on one layer, and then quickly switch back to the multi-layer configuration.

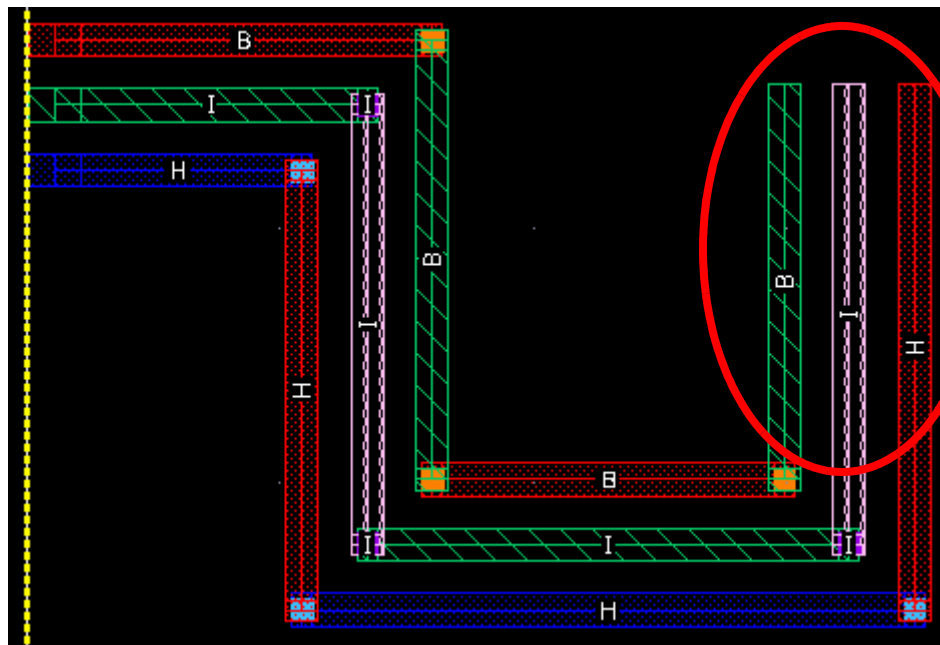
For example, create a bus starting from the bus bits on Metal1, Metal2, and Metal3, and then select *Via Up To <layer name>* to move the bus bits up a layer. The vias are inserted and the bus bits on Metal1, Metal2, and Metal3 are placed on Metal2, Metal3, and Metal4, respectively. If you next select the *Revert To Previous Multi-Layer Bus* option from the *Via* submenu, vias are inserted and the bus bits switch back to Metal1, Metal2, and Metal3, as shown in the following figure.



If you once again select *Revert To Previous Multi-Layer Bus*, the bus bits switch back to Metal2, Metal3, and Metal4, as shown in the following figure.

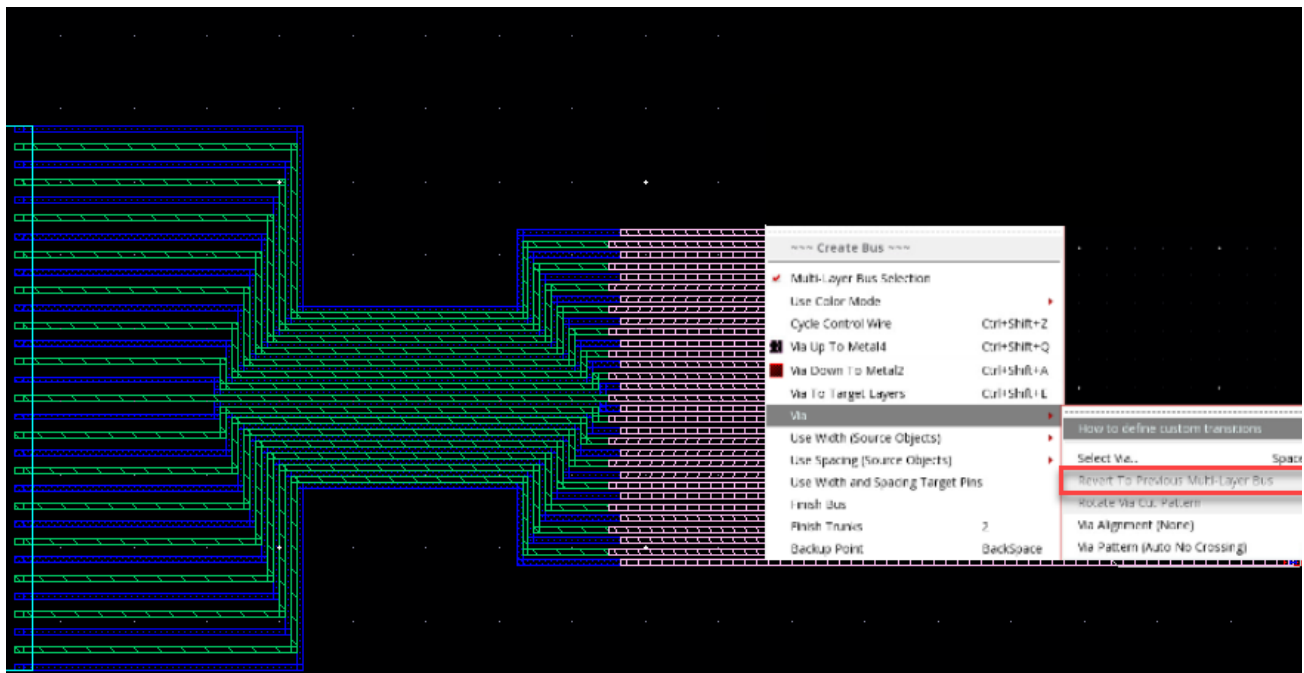
Virtuoso Interactive and Assisted Routing User Guide

Working with Buses



Bus bits again switch back to previous multi-layer Metal2, Metal3, and Metal4

In the same way, when you move all the bus bits to the same layer, you can revert to the multi-layer bus configuration. In the following figure, you can see that the bus bits on Metal15 have moved back to the original multi-layer configuration.



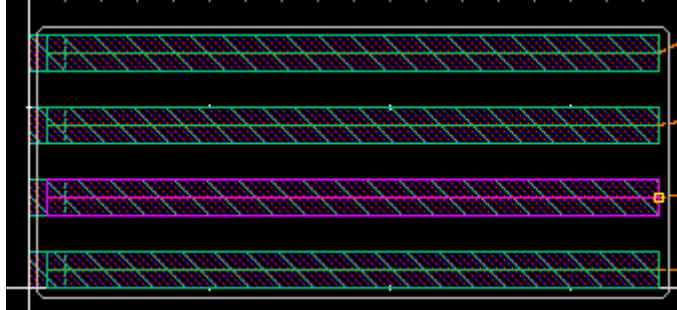
Note: The *Revert to Previous Multi-layer Bus* option is disabled once you have moved back to the original multi-layer configuration from the single layer.

The behavior of the bus bits is based on the control wire. You can cycle the control wire through middle, top, and bottom bits based on the current layer the bus bits are on. This is similar to the cycle control feature in a single-layer bus. For more information, see [Using the Control Wire](#).

Stacked Wires within Multi-Layer Bus

You can create a wire stack by physically stacking pins or wires belonging to adjacent layers on top of each other. This ensures that the metal shapes across all wires in the wire stack coincide. Wires in a wire stack have the same origin and the centerline of the wires in the wire stack should completely overlap.

To create a wire stack within a multi-layer bus, you need to start from existing stacked pins or wires. You can create a wire stack with completely overlapping wires or pins on different metal layers. For example, a wire stack is created when a pin on `Metal1` layer overlaps with the pins on `Metal2` and `Metal3` layers, as shown in the following figure.



Note: You cannot create a wire stack starting in an empty space.

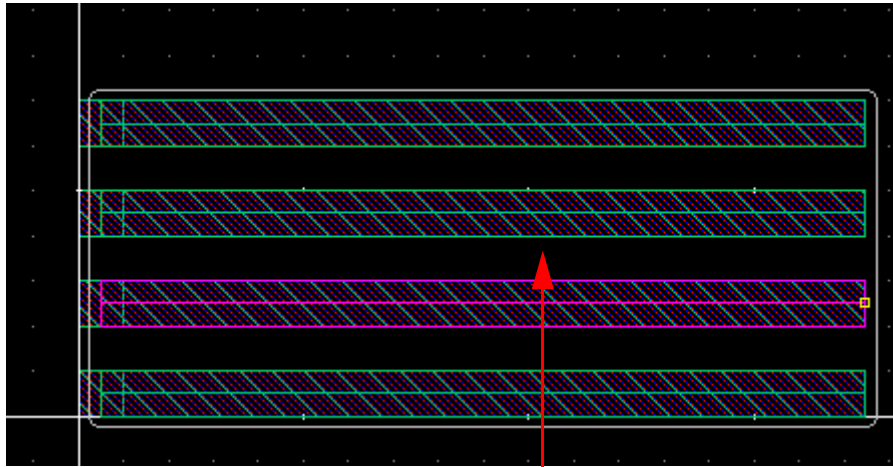
When creating a wire stack, the wires remain stacked throughout. For example, the wires remain stacked at the first bend and even when you change the wire spacing. However, the spacing between wire stacks changes.

The spacing between the wire stacks is based on the *Source Objects, Default Constraints, Gather/Spread, and Overrides by layer* options. These options are available

Virtuoso Interactive and Assisted Routing User Guide

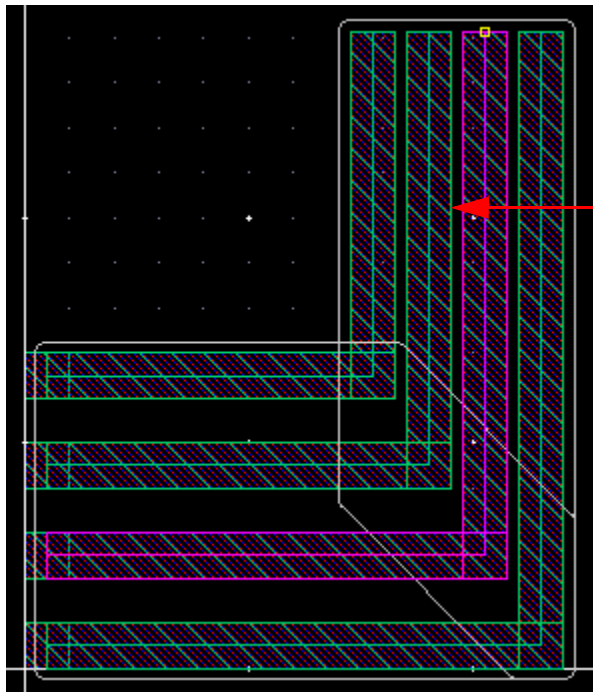
Working with Buses

in the *Use Spacing* submenu. By default, the spacing between the wire stacks is adjusted to align with the spacing of the source objects, as shown in the following figure.



Spacing of wire stacks is aligned with the spacing of the source objects

You can adjust the spacing between the wire stacks by selecting the other options available on the *Use Spacing* submenu. For example, when you select the *Default Constraints* option, the wire stacks are abutted based on `minSpacing`. The spacing between the wire stacks is set to the largest of all `minSpacing` values specified for the metal layers involved in the wire stack, as shown in the following figure.

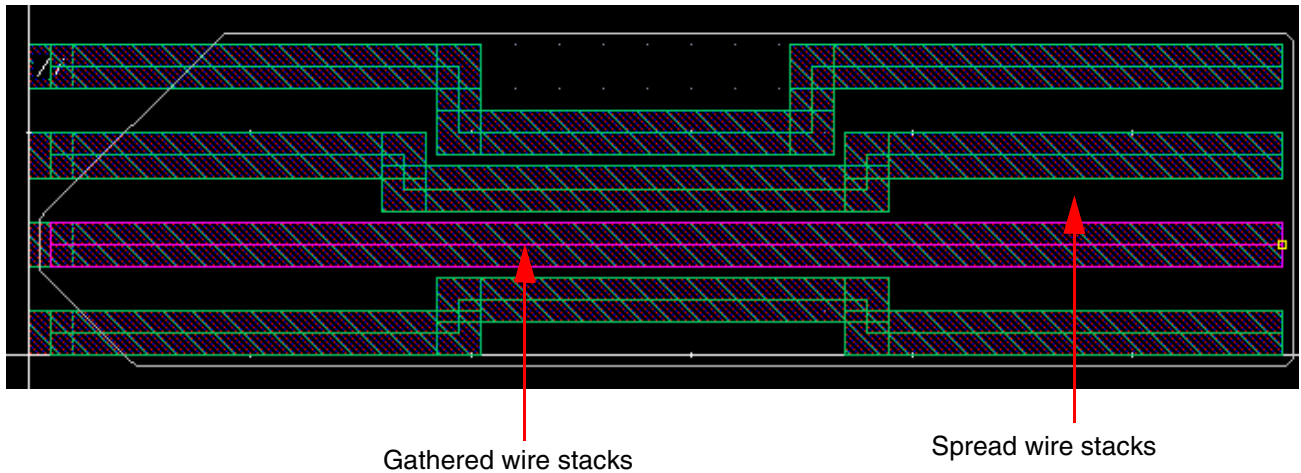


The wire stacks are abutted at the largest of all `minSpacing` values specified for the metal layers.

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

The spacing between the wire stacks is also maintained when they are gathered or spread. The following figure shows how the spacing between the wire stacks is adjusted based on whether they are gathered or spread.

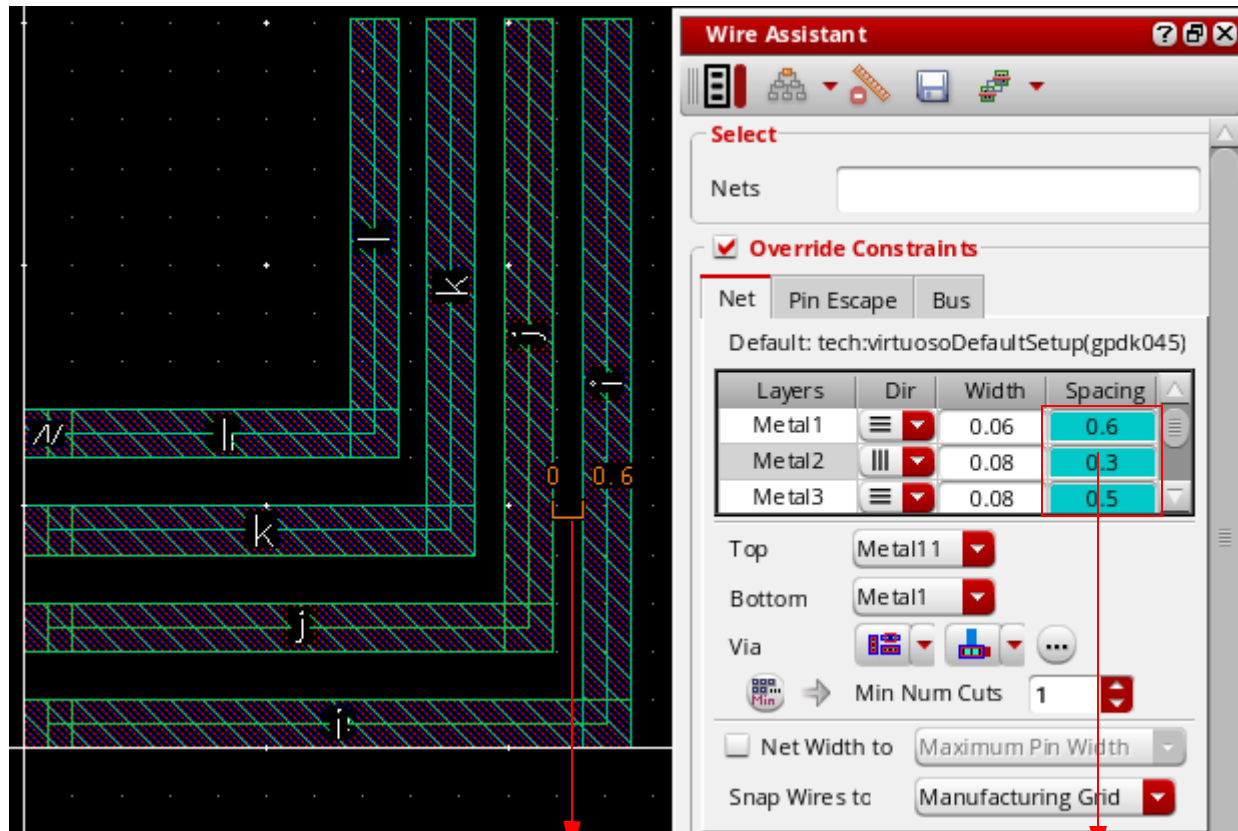


The spacing between the wire stacks can also be specified by using the *Overrides by layer* option in the *Use Spacing* submenu. When the *Overrides by layer* option is selected, the spacing between the wire stacks is set to the largest of all spacing values specified for the

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

metal layers in the width and spacing table on the *Net* tab in the Wire Assistant, as shown in the following figure.

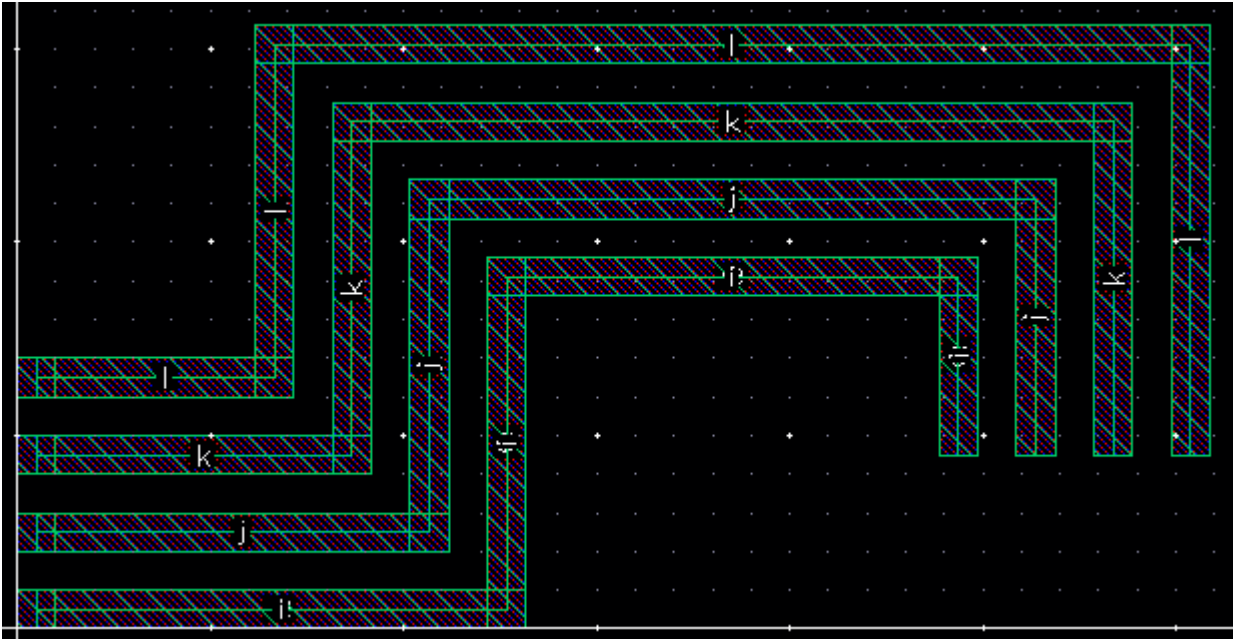


The spacing between wire stacks is the largest of all values specified for the metal layers (0.6)

You can complete creating wire stacks either in an empty space or at a destination pin.

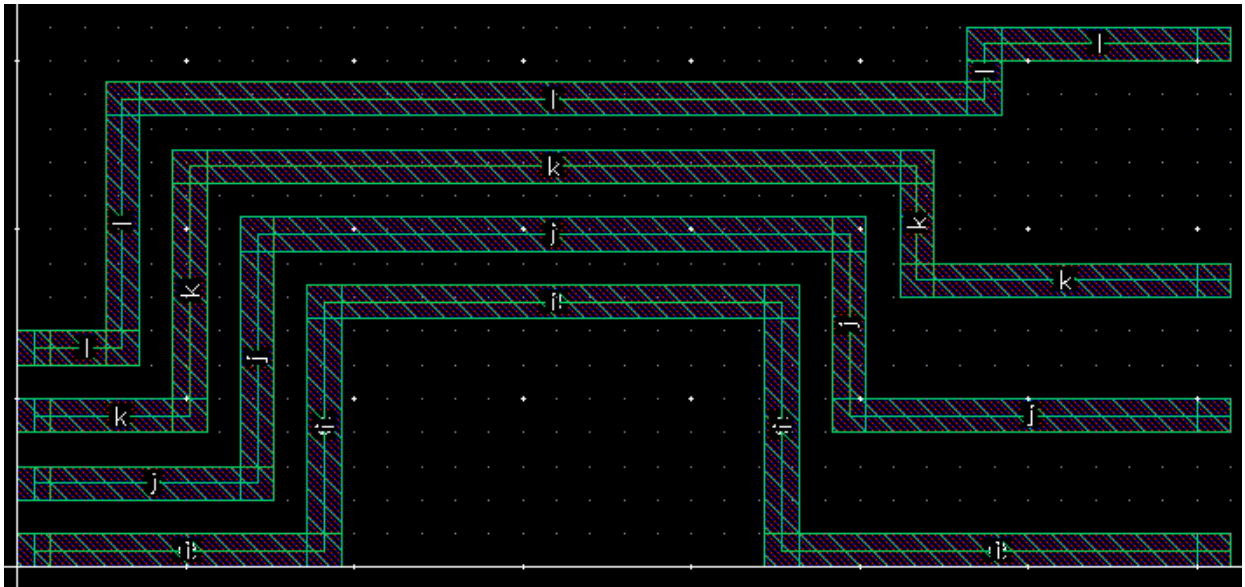
- In an empty space

When the wire stacks are completed in an empty space, floating wire stacks are created, as shown in the following figure.



■ At a destination pin

When wire stacks end at destination pins, they take the spacing from the destination pins, which applies only in the last section of the wire stacks, as shown in the following figure.



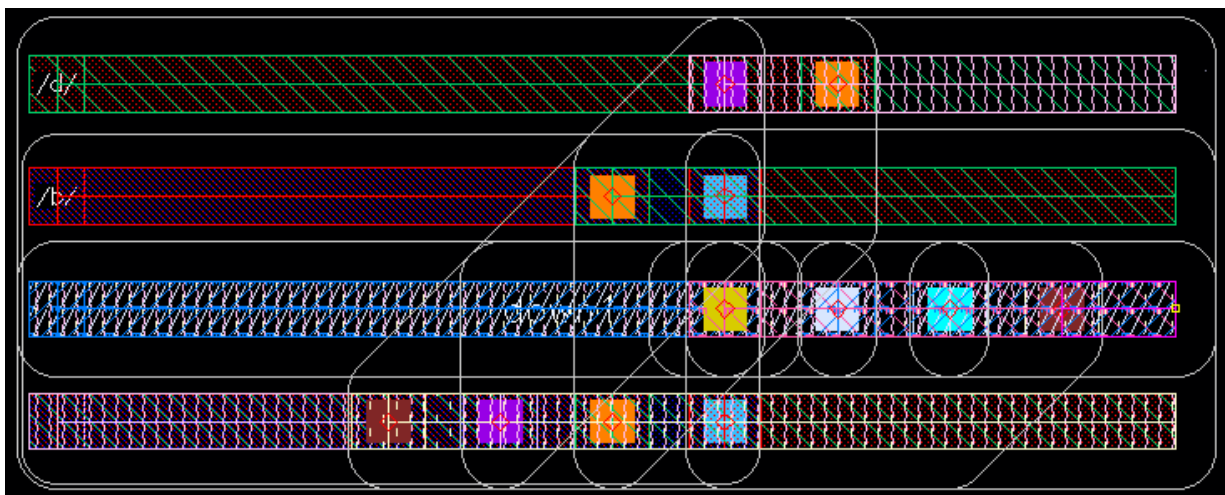
Transitioning Capabilities in a Wire Stack within a Multi-Layer Bus

While creating a wire stack from stacked pins or from already existing stacked wires, you can now insert vias to the wire stacks. You can move a wire stack up or down one or multiple layers at a time by inserting a via. Wires in a wire stack have the same origin and the centerline of the wires in the wire stack completely overlaps even after the wire stack moves up or down. You can also move all the bus bits to the same layer, and then switch back to a multi-layer configuration.

Transitioning a Wire Stack between Layers

You can transition a wire stack between layers by moving up or down to one or multiple layers. To do this:

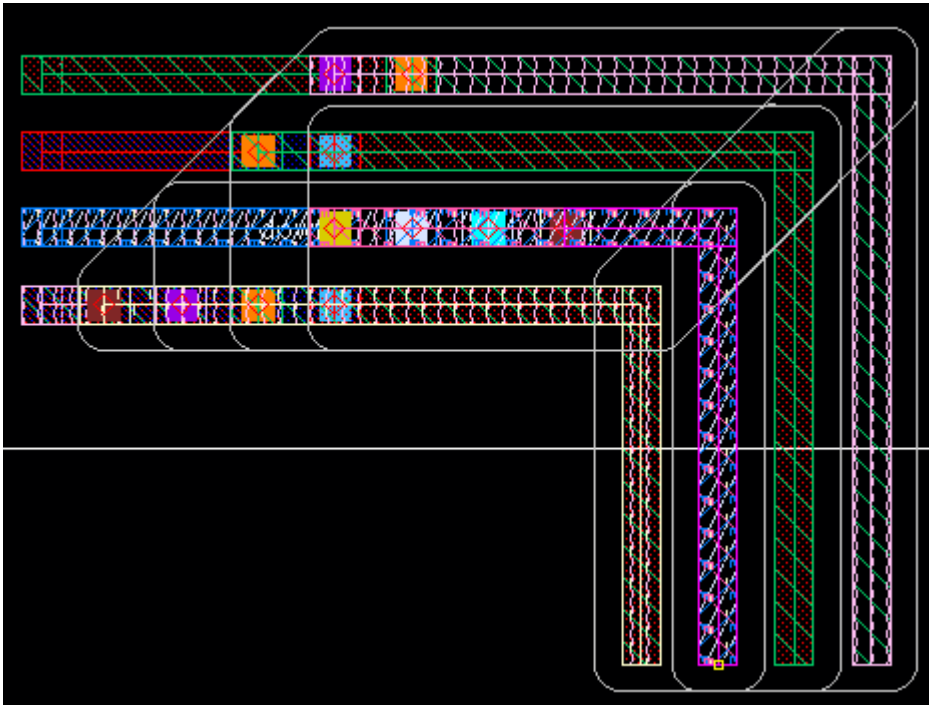
1. Start the *Create Bus* command.
2. Click and drag to select the required stacked pins.
3. Right-click the wire stack being created and select *Via Up To <layer name>* or *Via Down To <layer name>* to move the wire stack up or down a layer, respectively. The following figures shows an example where vias are inserted for each wire stack and the wire stacks on different metal layers move up a higher metal layer when *Via Up To <layer name>* is selected. After inserting a via, the multi-layer bus continues in the straight direction.



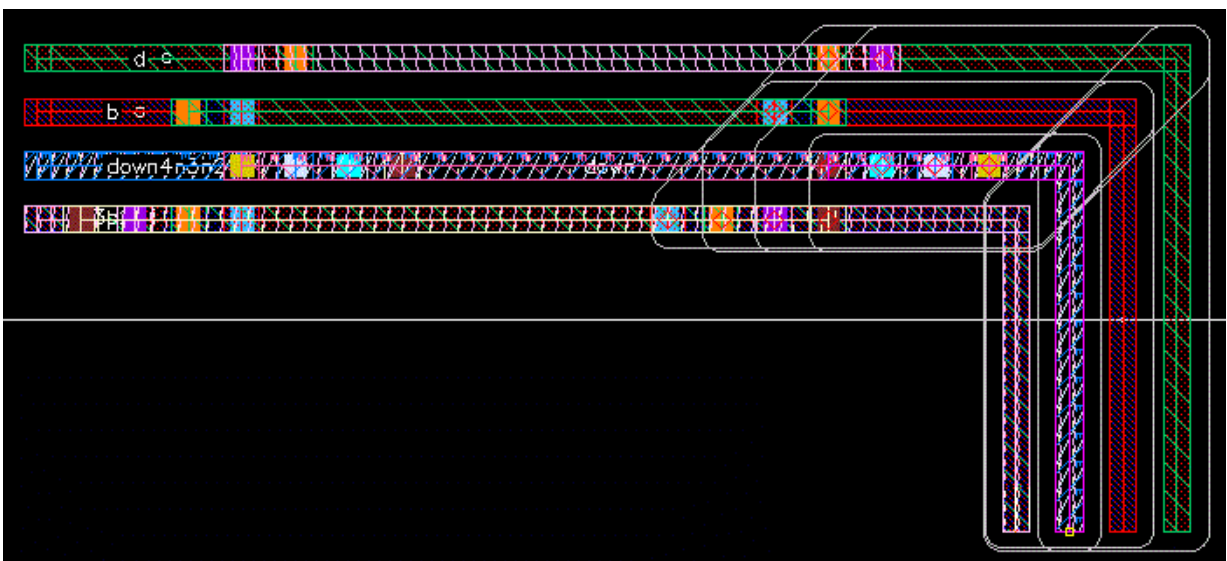
Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

You can create a bend but cannot change the metal layer at the same time, as shown in the following figure.



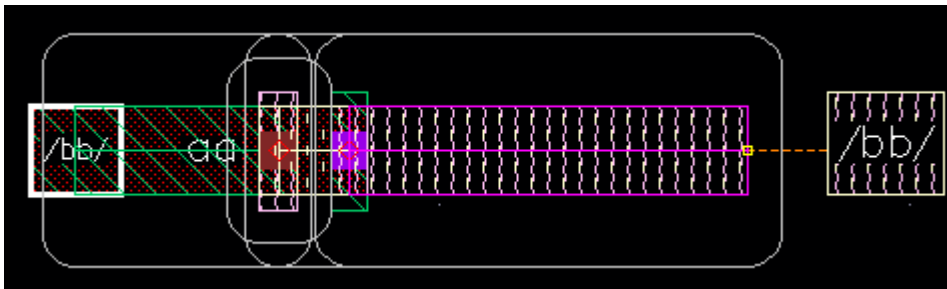
Similarly, in the following figure, you can see an example in which the wire stacks on higher metal layers move down a metal layer when *Via Down To <layer name>* is selected.



Transitioning a Wire Stack to a Target Layer

You can add vias to the layer of the target objects. The target objects are all the objects that are pointed to by the flightlines when the *Create Bus* command is run. However, if the *Show Flightline* option in the Wire Assistant is off, then vias are not added.

1. Start the *Create Bus* command.
2. Click and drag to select the required stacked pins.
3. Right-click the wire stack being created and select *Via To Target Layer* to move the wire stack to the layer of the target wire stack. The following figures shows an example where the wire stack on Metal3 moves up to Metal4 because Metal4 is the layer of the target wire stack.



Transitioning a Wire Stack to the Same Layer

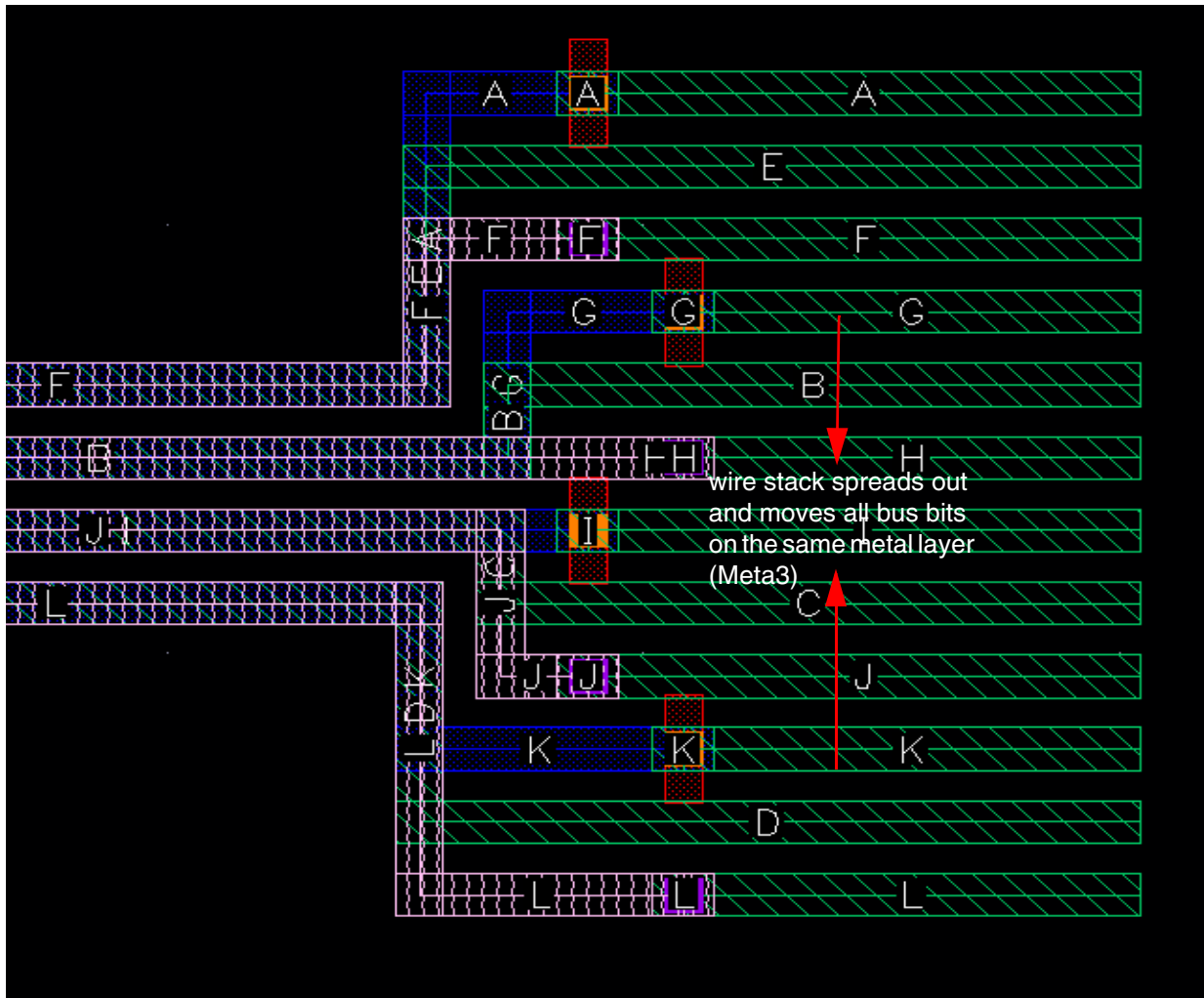
To move the wire stack to the same layer:

1. Start the *Create Bus* command.
2. Start creating the wire stack by digitizing the stacked pins.
3. Right-click the wire stack being created and point to *Via*.
4. Click *Select Via* or press the *Spacebar* key.

The Select Via form appears.

5. In the form, select the metal layer to which you want to move the wire stack. When moving to the same layer, the bus bits in the wire stack are spread, such that there are

no spacing violations. In the following figure, all the bus bits are moved to `Meta15` and the spacing between the bus bits is adjusted to avoid the spacing violations.



Switching Wire Stack to a Previously Used Multi-Layer Configuration

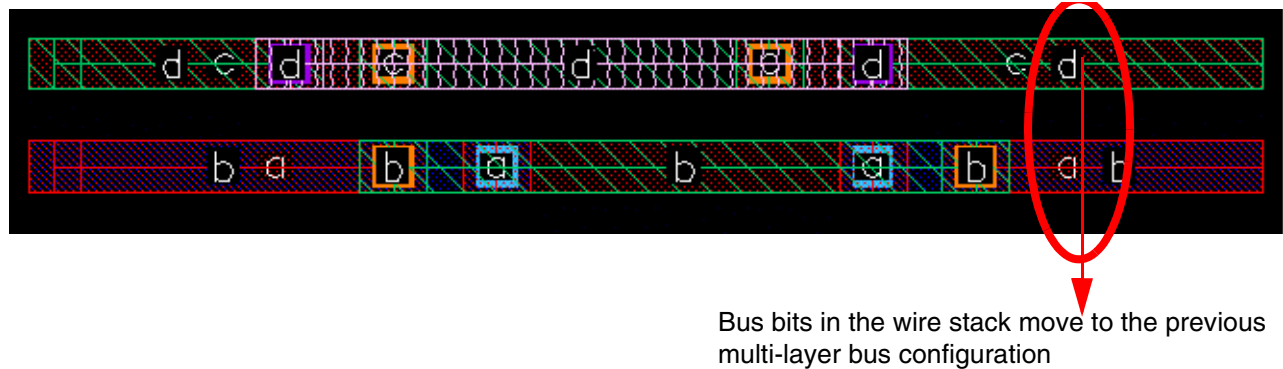
When the wire stack is switched between layers by moving up or down one or multiple layers or by moving to a target layer, same metal layer. you can revert back to a previous multi-layer bus configuration.

When you start creating a bus from stacked pins, and then select *Via Up To <layer name>*, vias are inserted for each wire stack and the wire stack moves up to a higher metal layer. If you now select the *Revert To Previous Multi-Layer Bus* option from the *Via* submenu, vias

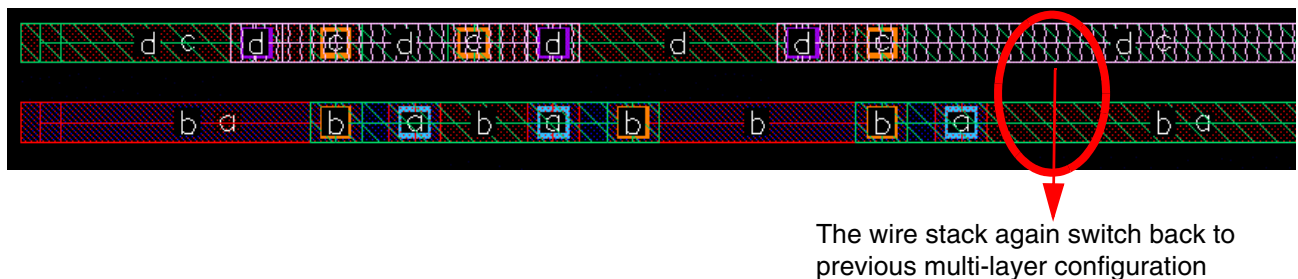
Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

are inserted again and the bus bits switch back to the previous multi-layer configuration of the wire stack, as shown in the following figure.

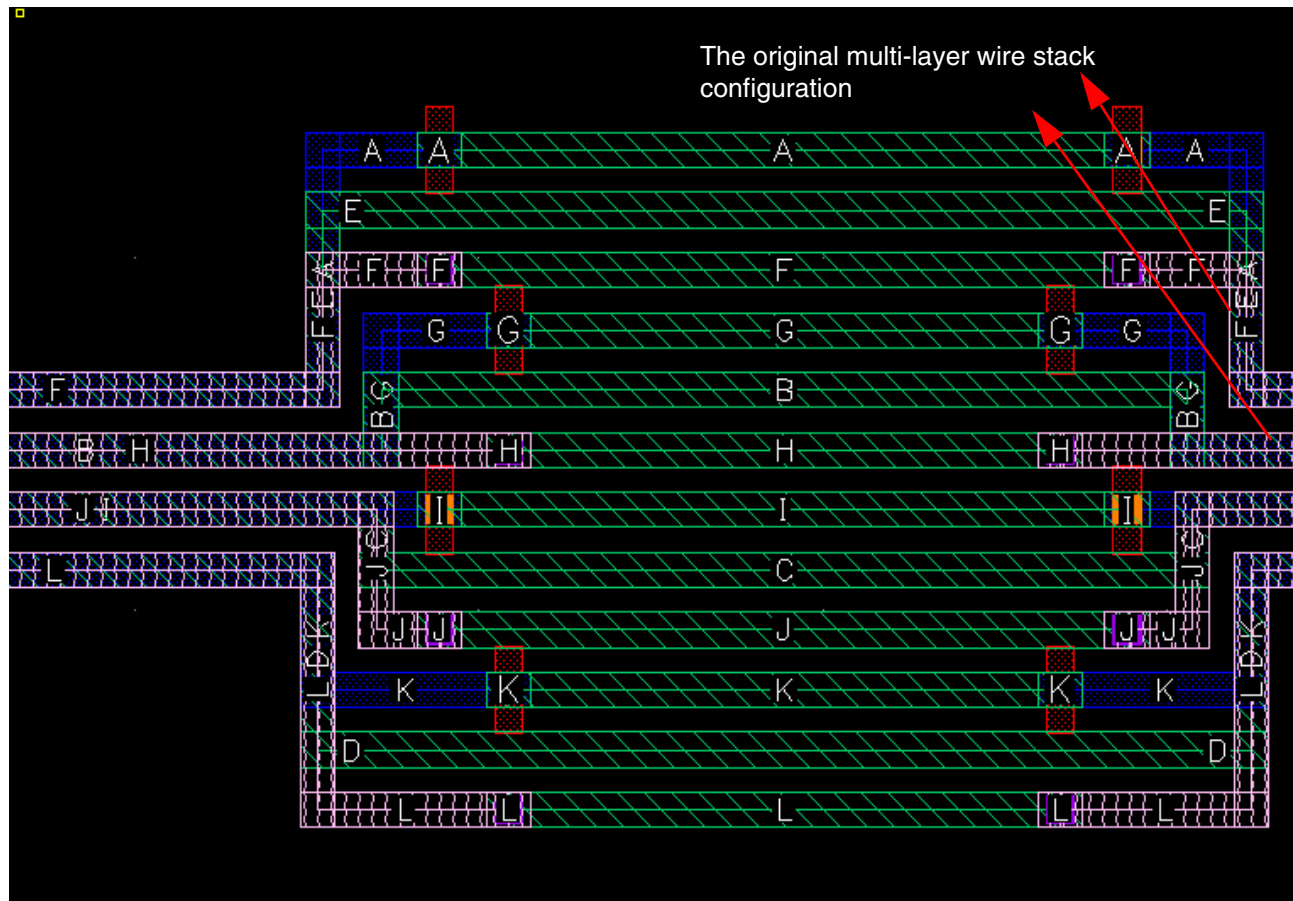


If you once again select *Revert To Previous Multi-Layer Bus*, the wire stack again switches back to previously used multi-layer configuration, as shown in the following figure.



In case of a wire stack on the same layer, you can revert to the previously used multi-layer bus configuration. When you revert to the previously used multi-layer configuration, same spacing between the wire stacks is retained and you continue with the bus bits that are

spread. If you now make a bend and click and continue to create a multi-layer bus, the bus bits are gathered and you move back to the multi-layer wire stack configuration.



Customizing Multi-Layer Bus Transition

You can define customized SKILL procedures to create complex multi-layer bus transition patterns as explained below.

Defining a SKILL Procedure

To define the layers to which the bus bits should be transitioned, you can write a SKILL procedure that returns the list of layers to transition to. When you define the procedure, you need to assign a name to it. The named procedure is then called by the system when you click the associated option from the *Via* submenu or use the assigned bindkey.

The SKILL procedure that you define must have two arguments, the first is the `cellViewId` and the second is a list of net and associated layers (`netId` and `LayerNum`); for example,

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

`list(list(netId1 layer1) list(netId2 layer2))`. The `cellViewId` is used to extract the layer name from the technology file, which makes a layer transition independent of the technology file, and `netLayerPairs` can be used to extract the layers the bus bits are on currently and to specify the layers to which the bus bits would move.

Note: It is not mandatory to use the `cellViewId` and the `netLayerPairs` inside the SKILL procedure. However, they must be a part of the arguments list.

The following example defines a procedure that switches bus bits from the input metal layer to `Metal1` and `Metal2`. The input metal layer is the layer on which the bus bit is currently drawn.

```
(procedure (alternate_Metal1Metal2 cellViewId netLayerPairs)
  (let ()
    (list "Metal1" "Metal2")
  )
)
```

** The `cellViewId` and `netLayerPairs` are automatically provided by the application when the procedure is called.

** `alternate_Metal1Metal2` is the name of the procedure, which is a user-defined value.

The procedure returns a list of valid layer names or valid layer numbers. The second net of the list provided by the system moves to the second layer number or layer name provided by the procedure. This means that the order defined in the procedure is maintained and the list returned by the procedure is read by the application in the same order. For example:

List of list of `netId` and `LayerId` given by the application is `netOne Metal1, netTwo Metal2,..., netX MetalX`.

List of `LayerId` given by the procedure is `Metal3, Metal4, ..., MetalX`.

Then, by default `netOne` moves to `Metal3`, `netTwo` to `Metal4`, and so on.

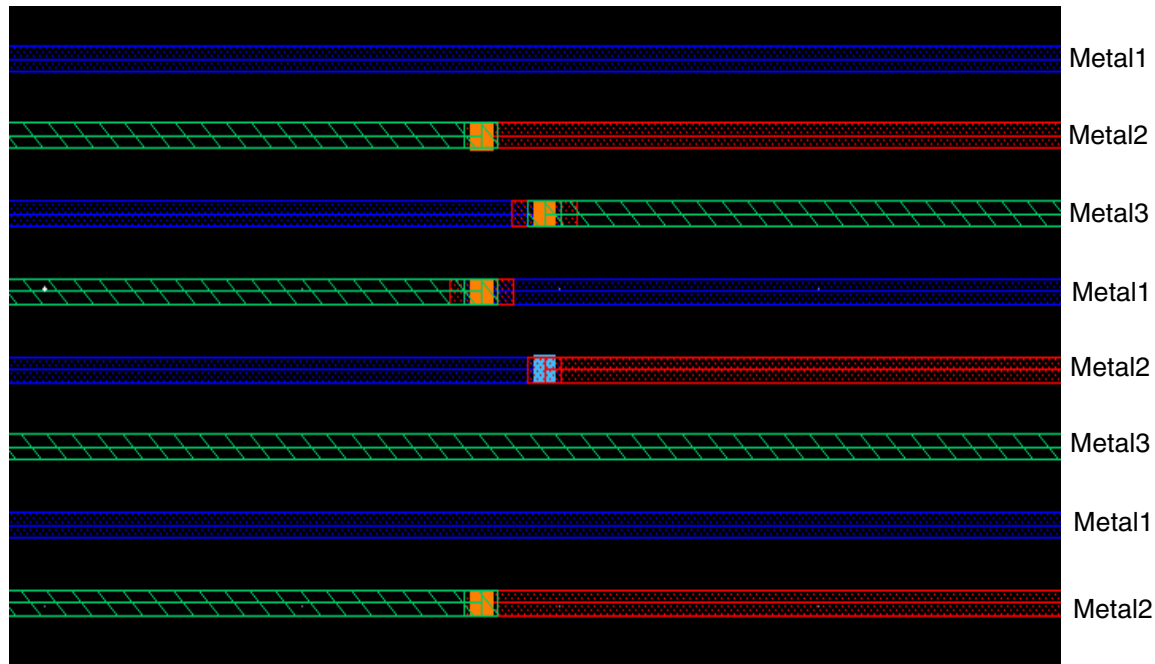
The following are some additional examples that explain how the SKILL procedures to customize layer transition in a multi-layer bus can be defined.

Example1

```
(procedure (pattern_M1M2M3 cellViewId netLayerPairs)
  (let ()
    (list "Metal1" "Metal2" "Metal3")
  )
)
```

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses



When this SKILL procedure is run, bus bits switch to Metal1, Metal2, and Metal3, irrespective of the input metal layer of the bus bit. This alternate transition of bus bits applies to the complete bus whatever is its size.

Example2

```
(procedure (viaUpTwice cellViewId args)
  (let (techFileId netLayerPair layerNum newLayerName newLayerNum layerName
        result)
    result = nil
    techFileId = (techGetTechFile cellViewId)
    (foreach netLayerPair args
      layerNum = (cadr netLayerPair)
      layerName = (techGetLayerName techFileId layerNum)
      newLayerName = layerName
      (case layerName
        ("Metal1"
          newLayerName = "Metal3"
        )
        ("Metal2"
          newLayerName = "Metal4"
        )
      )
    )
  )
```

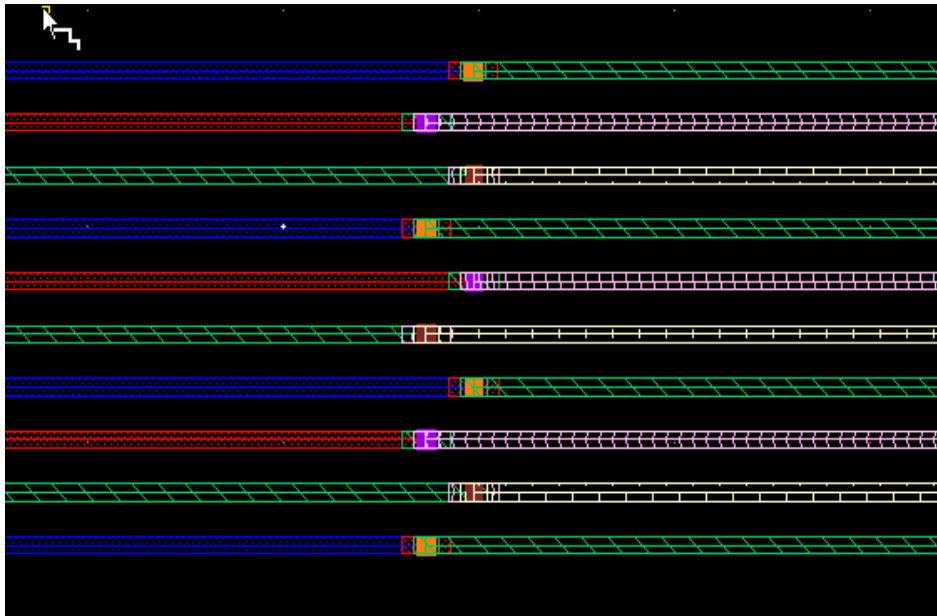
Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

```
        ("Metal3"
         newLayerName = "Metal5"
        )
    )
    result = (tconc result newLayerName)
)
(car result)
)
```

The following figure shows the output when this SKILL procedure is run:

- All bits on Metal11 move up to Metal3
- All bits on Metal12 move up to Metal4
- All bits on Metal13 move up to Metal5
- Bus bits on other metal layers remain unchanged



Example 3

```
(procedure (stitchM1M2M3M4ToM2M1M4M3 cellViewId args)
  (let (techFileId netLayerPair layerNum newLayerName newLayerNum layerName
        result)
    result = nil
    techFileId = (techGetTechFile cellViewId)
```


Virtuoso Interactive and Assisted Routing User Guide

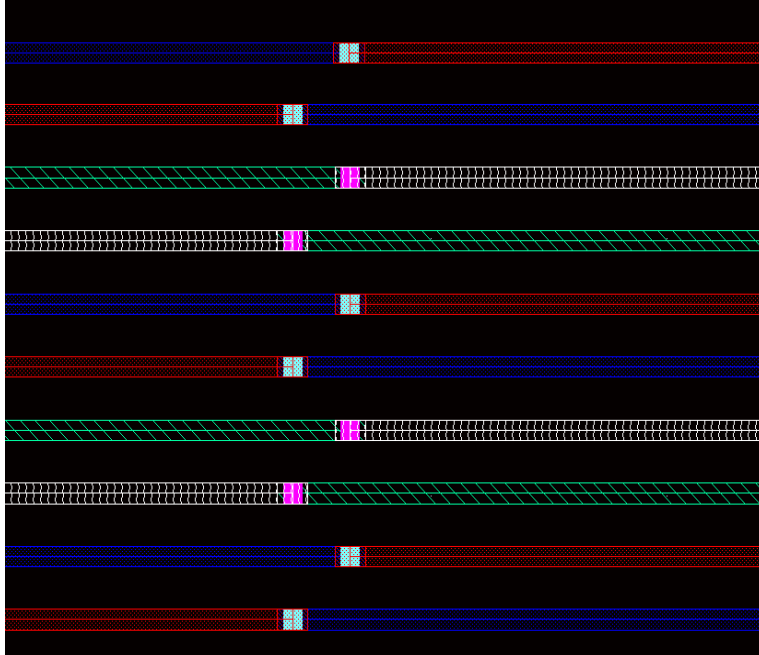
Working with Buses

```
(foreach netLayerPair args
  layerNum = (cadr netLayerPair)
  layerName = (techGetLayerName techFileId layerNum)
  newLayerName = layerName
  (case layerName
    ("Metal1"
      newLayerName = "Metal2"
    )
    ("Metal2"
      newLayerName = "Metal1"
    )
    ("Metal3"
      newLayerName = "Metal4"
    )
    ("Metal4"
      newLayerName = "Metal3"
    )
  )
  result = (tconc result newLayerName)
)
(car result)
```

The following figure shows the output when this SKILL procedure is run:

- All bits on Metal1 move up to Metal2
- All bits on Metal2 move to Metal1
- All bits on Metal3 move up to Metal4
- All bits on Metal4 move to Metal3

- Bus bits on other metal layers remain unchanged



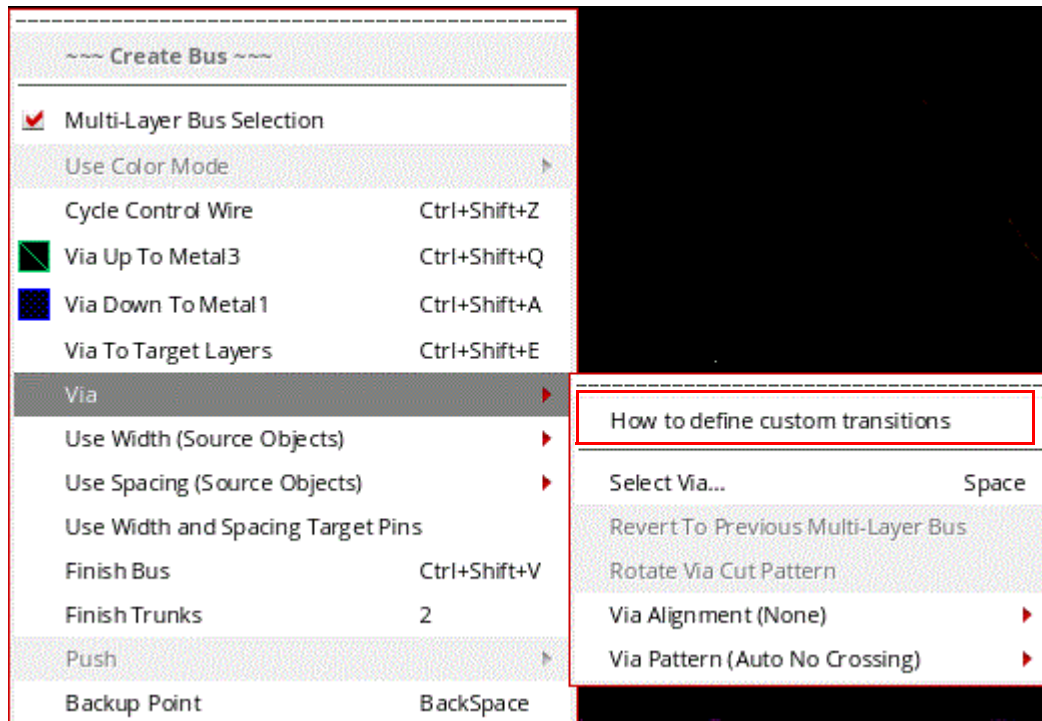
Customizing the Create Bus Context-Sensitive Menu

You can include the SKILL procedure that you define as an option on the *Via* submenu in the *Create Bus* context-sensitive menu and assign a bindkey to it. You can then quickly access and run the procedure by using the bindkey or by clicking the associated option from the submenu.

Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

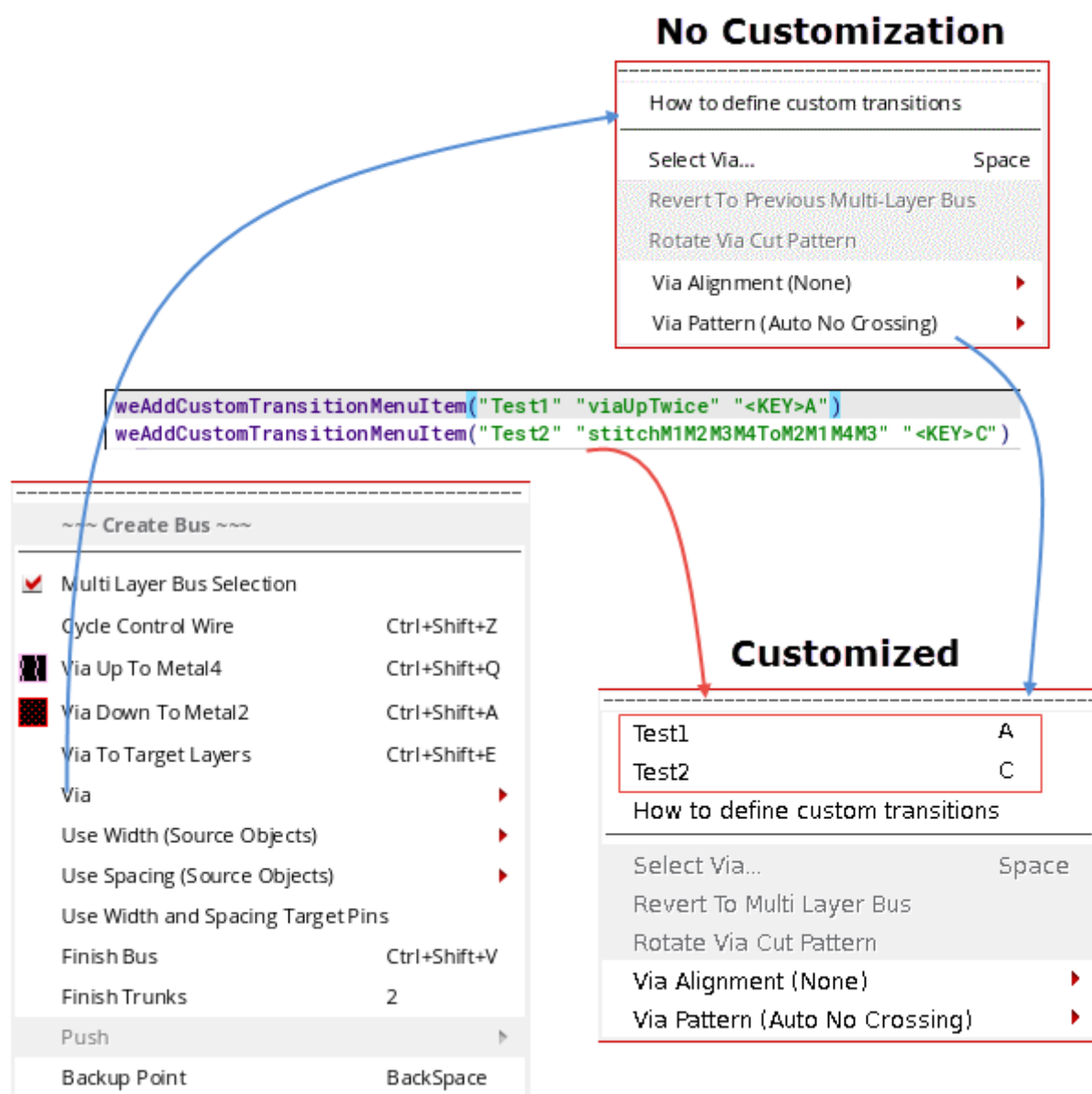
The *How to define custom transitions* options available on the *Via* submenu provides information about how you can customize the multi-layer bus functionality.



Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

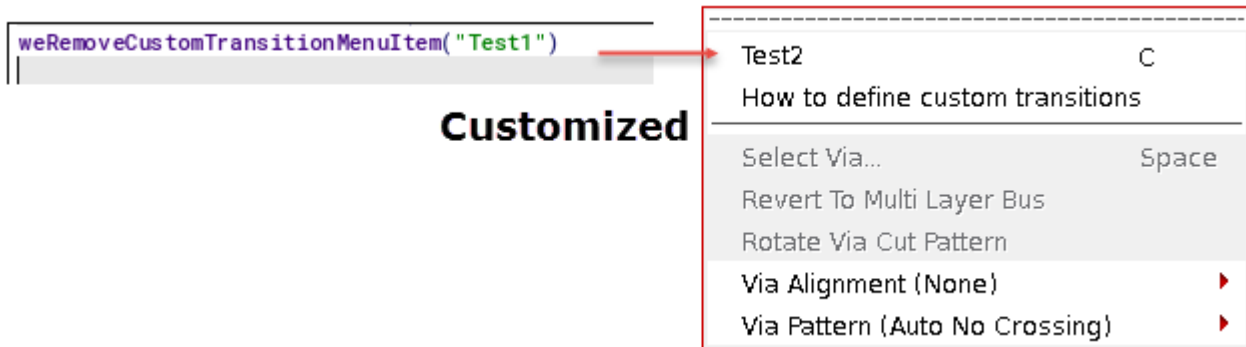
To add the SKILL procedure as an option on the *Via* submenu, use the `weAddCustomTransitionMenuItem` SKILL API. You can add multiple procedures to the *Via* submenu, which are placed one below the other, as shown in the following figure.



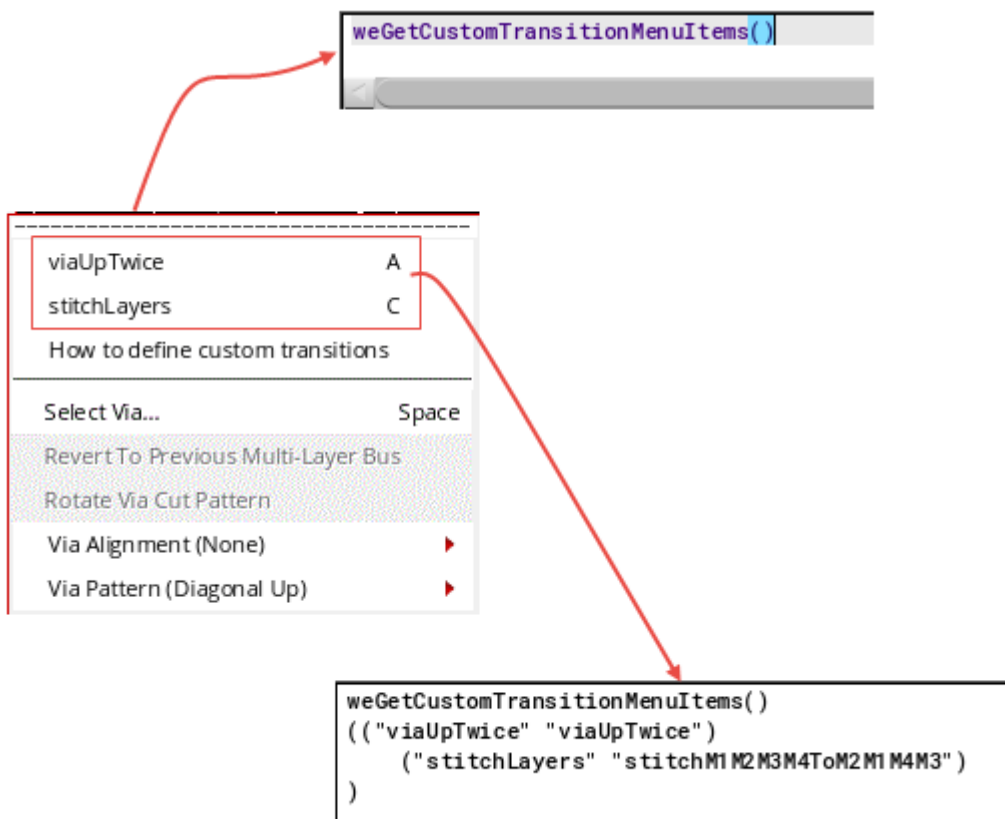
Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

To remove an option from the *Via* submenu, use the `weRemoveCustomTransitionMenuItem` SKILL API. The following figure shows the customized menu after an existing option is removed.



To retrieve the list of customized options added to the *Via* submenu, use the `weGetCustomTransitionMenuItems` SKILL API. The following figure shows a list of customized options added to the *Via* submenu.



Virtuoso Interactive and Assisted Routing User Guide

Working with Buses

Working with Stranded Wires

Stranded routing lets you increase current capacity between two points by creating smaller pathSegs next to each other instead of a large sized pathSeg. At the same time, it also controls resistance and metal density.

The stranded wire feature is available only in Layout XL and higher tiers. The *Create Stranded Wire* command lets you create a wire comprising multiple strands in a single direction on a single net. It does not let you create multiple wires on the same net. This command works without specifying a constraint in the Constraint Manager. A stranded wire can be created from the following starting points:

- An empty space
- An existing single pin
- An existing wire or via
- An existing width spacing patterns

This section describes the following stranded wires functionality.

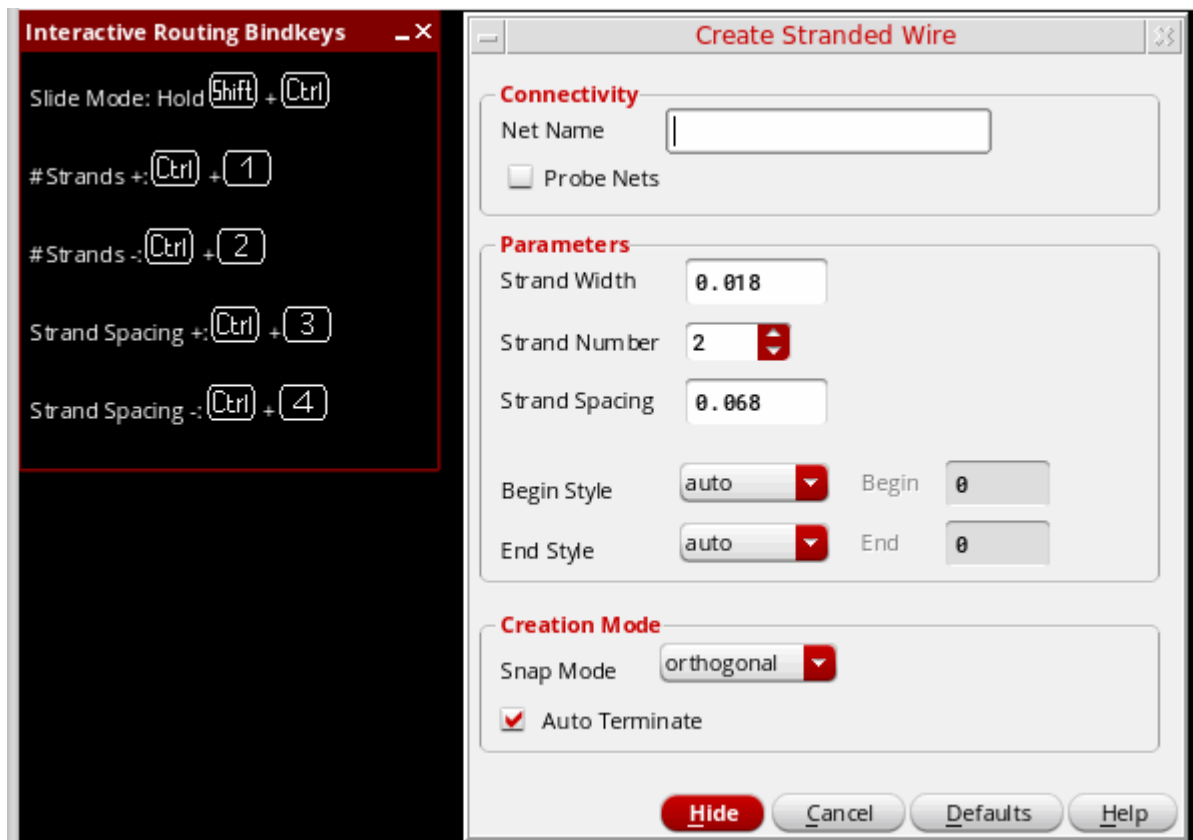
- Creating a Stranded Wire
 - ❑ Creating a Stranded Wire from an Existing Pin
 - ❑ Creating a Stranded Wire from an Existing Wire
 - ❑ Creating a Stranded Wire from Tieout
 - ❑ Creating a Stranded Wire from a Rectangle or PathSeg Covering a Pin
- Inserting Vias Automatically
- Tapering in Stranded Wire
- Enabling Bridge Vias Using WSPs
- Avoiding Shorts with Existing Blockages
- Filtering Objects by Net Name

Creating a Stranded Wire

To create a stranded wire:

1. Choose *Create – Wiring – Stranded Wire* and press F3. Alternatively, use the Ctrl+Shift+S keys to start the *Create Stranded Wire* command and then press F3.

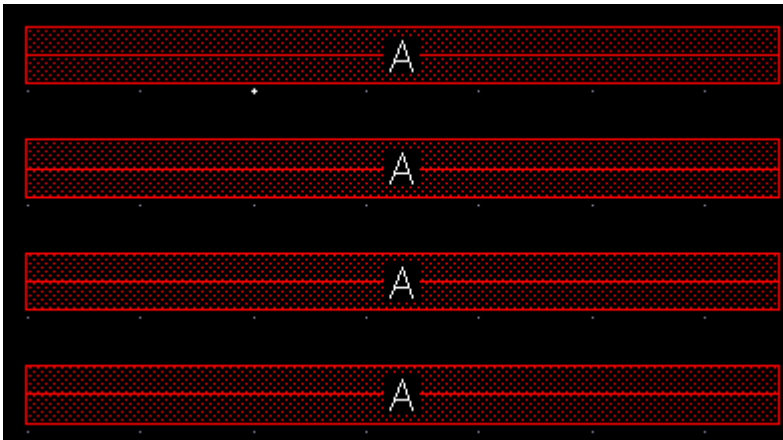
The Create Stranded Wire form opens. You can set the stranded wire parameters in this form or using the bindkeys. The most useful bindkeys and shortcuts for the *Create Stranded Wire* command are displayed in the *Interactive Routing Bindkey* hint box at the top-left corner of the layout canvas. In case, you want to disable the hint box for the *Create Stranded Wire* command, deselect the *Show Hints* option in the Layout Editor Options form.



Note: The *Create Stranded Wire* command supports the `ladderStyle` constraint.

2. From the *Layers* assistant, select a layer on which you want to create the stranded wire.
3. To create a stranded wire in an empty space in the design display area:
 - a. Click at a point in the design display area to digitize the stranded wire.

- b. Continue to digitize the stranded wire and create bends.
 - c. Press `Enter` to finish creating the stranded wire.
4. To create a stranded wire that has connectivity in an empty space in the design display area, in the Create Stranded Wire form, specify a net name for the stranded wire. The stranded wire being created is assigned to the specified net. If the specified net does not exist in the design, it is created.



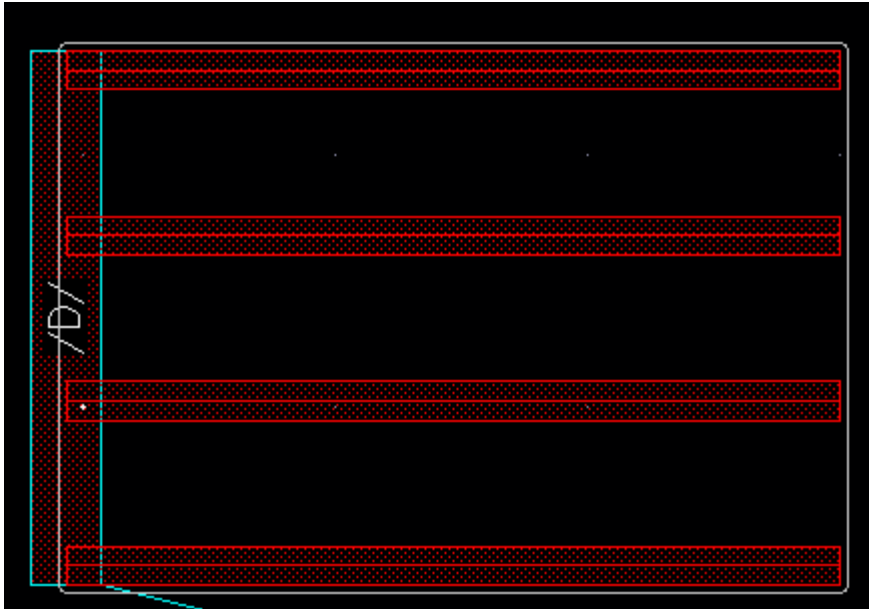
5. Use the *Via Up* or *Via Down* option from the *Create Stranded Wire* context-sensitive menu to insert a via on the stranded wire.



Caution

The via options in the Created Stranded Wire context-sensitive menu are disabled until you click a point in the design display area to digitize the stranded wire.

6. To create a stranded wire from an existing pin, start the *Stranded Wire* command and click at the pin to digitize the stranded wire. The following figure shows a stranded wire being created from a pin on Metal2.



For more information, see [Creating a Stranded Wire from an Existing Pin](#).

7. To create a stranded wire from an existing wire, select the wire and start the stranded wire command.

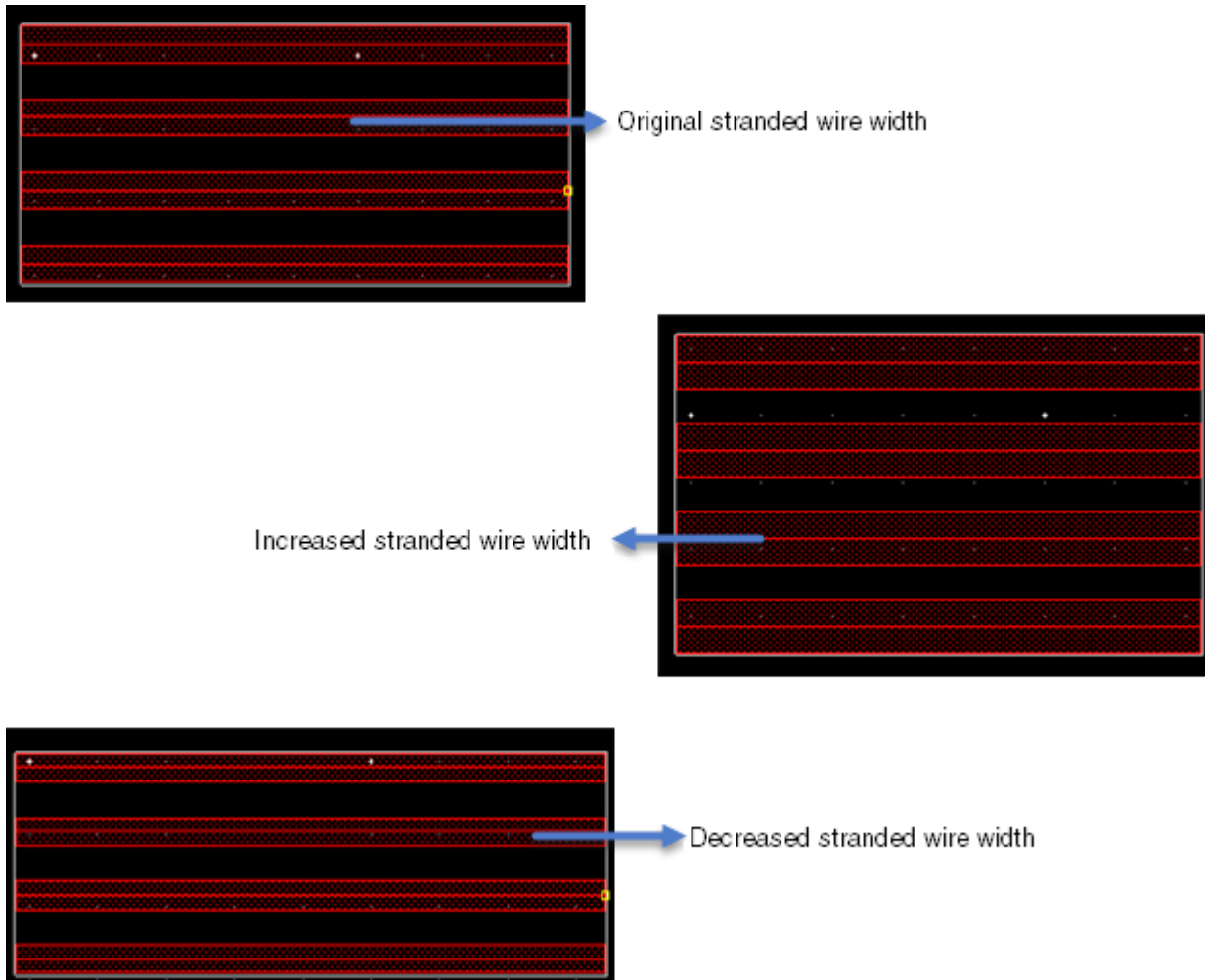
For more information, see [Creating a Stranded Wire from an Existing Wire](#).

8. To change the width of the stranded wire, type the desired width in the *Strand Width* field. Alternatively, use the `Ctrl+Shift+Scroll Up` or `Ctrl+Shift+Scroll Down`

Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

to increase or decrease the width of the stranded wire. The default width displayed in the *Strand Width* field is determined from the `minWidth` constraint for each routing layer.

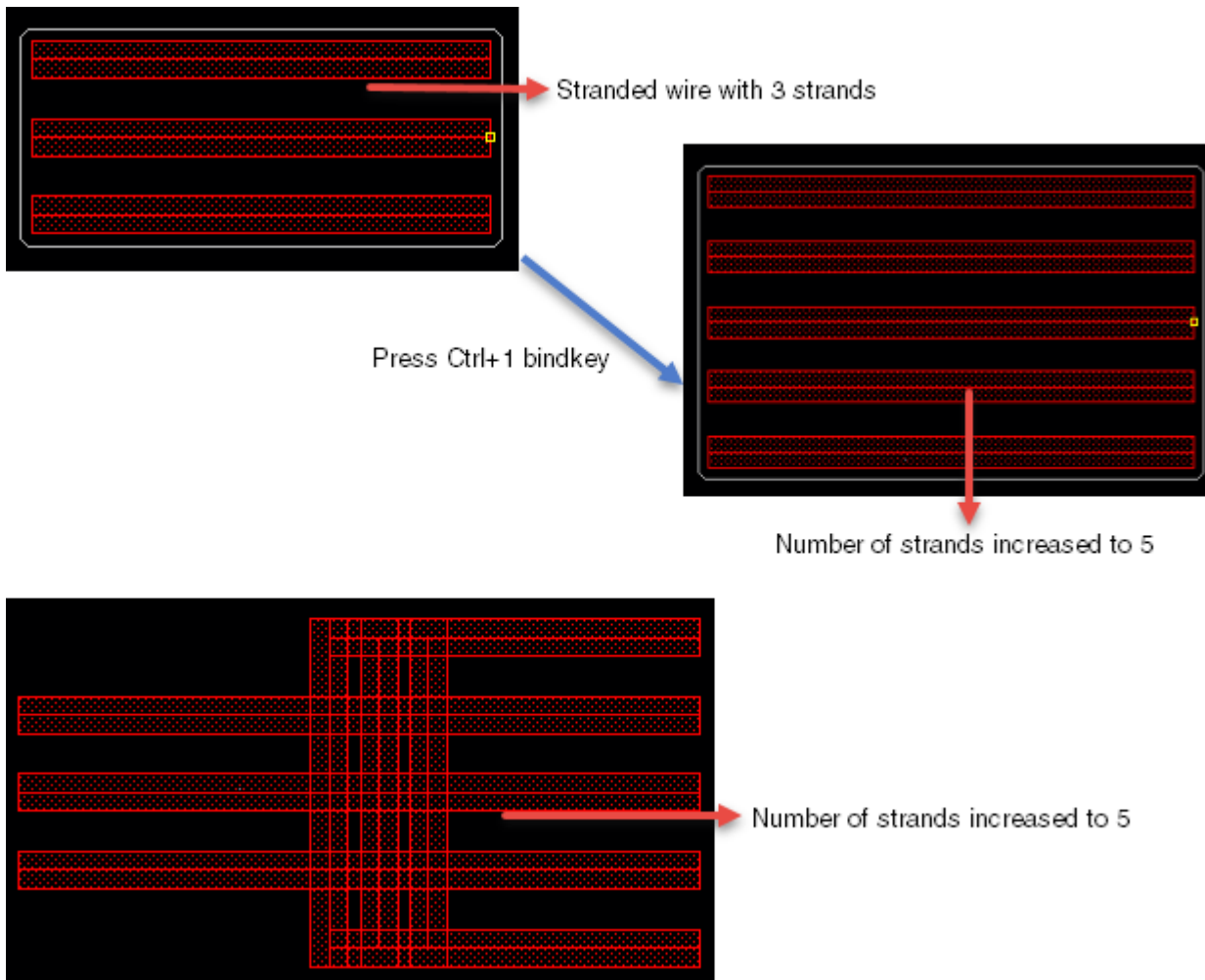


9. To change the number of strands in the stranded wire, type the desired number of strands in the *Strand Number* field. Alternatively, use the `Ctrl+1` bindkey to increase the

Virtuoso Interactive and Assisted Routing User Guide

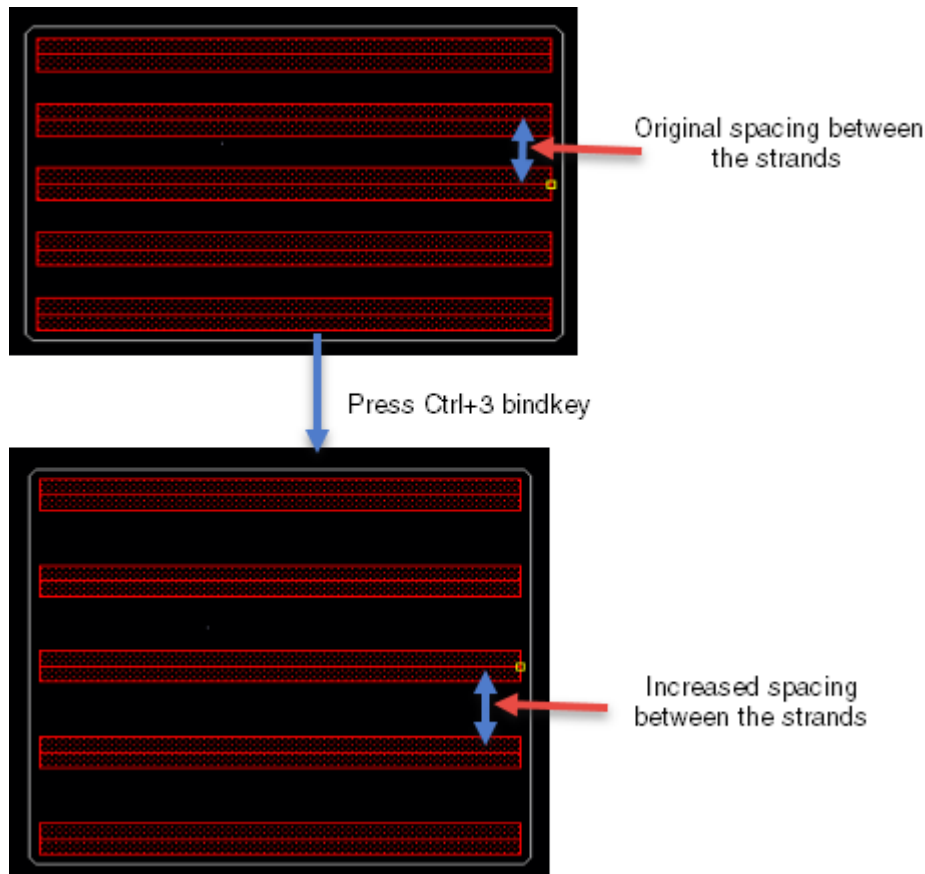
Working with Stranded Wires

number of strands and `Ctrl+2` bindkey to decrease the number of stranded wires. The default number of strands that get created are 2.



10. Change the spacing between the strands by specifying the desired spacing in the *Strand Spacing* field. Alternatively, you can use the `Ctrl+3` bindkey to increase the spacing and `Ctrl+4` bindkey to decrease the spacing between the strands. The default spacing

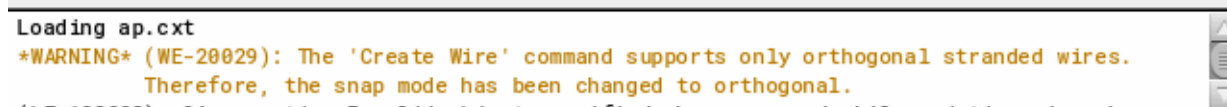
value is based on the `minSpacing` constraint for each layer that can be used for routing. The following figure shows an increase in the spacing between the strands.



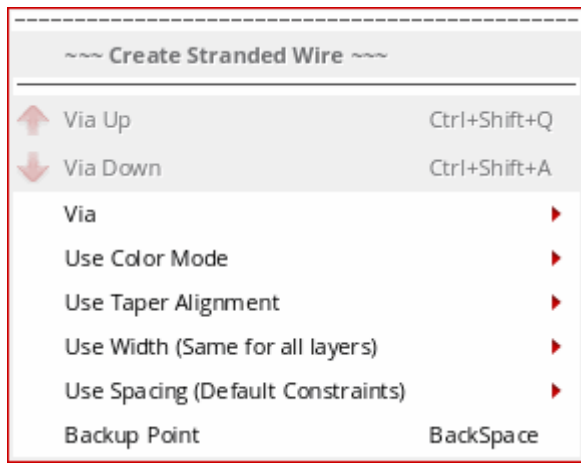
11. Select the begin and end extension style. You can choose *truncate*, *auto*, *extend*, *variable*, or *custom*. The default begin and end extension style is *auto*.

Note: The begin and end style that are finally applied to a stranded wire depends on the begin and end extension values you specify for them. For example, if you choose *custom* as the *Begin Style* and *End Style* but the extension value corresponds to that of *extend*, the stranded wire will have *extend* begin and end styles.

12. Choose a value from the *Snap Mode* cyclic field. If set to *diagonal*, a warning message is displayed in CIW informing that only orthogonal stranded wires are supported. The mode is automatically changed to *orthogonal*.



13. Select *Auto Terminate* to allow the stranded wire you are creating to be terminated with a single click.
14. You can finish a stranded wire at any point by double clicking or by pressing `Enter`. The stranded wire ends at the pointer location.
15. Right-click in the canvas to display the *Create Stranded Wire* context-sensitive menu. For more information, see [Create Stranded Wire Shortcut Menu Commands](#).



Creating a Stranded Wire from an Existing Pin

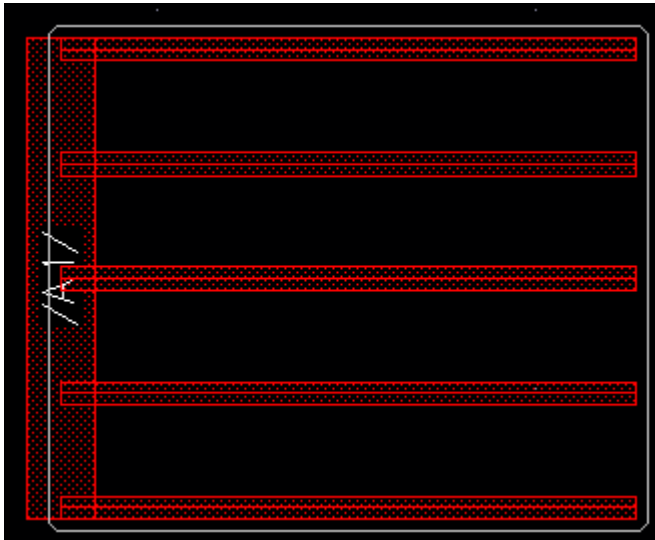
To start creating a stranded wire from an existing pin:

1. Choose *Create – Wiring – Stranded Wire*.

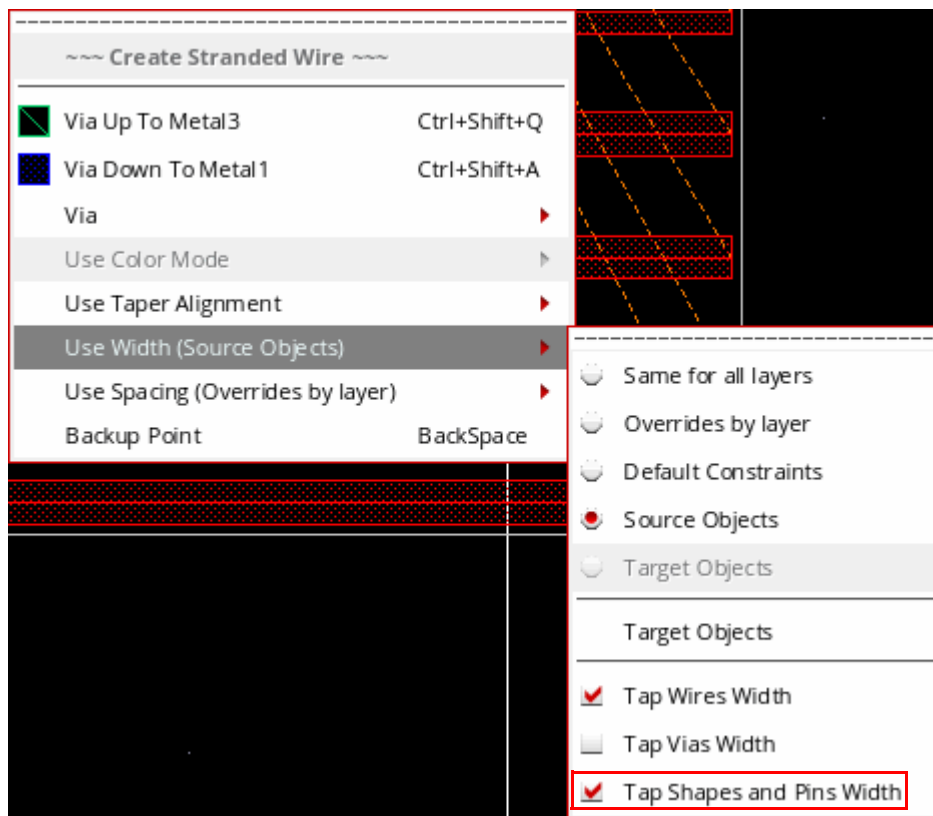
Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

2. Click the pin to digitize the stranded wire. The following figure shows a stranded wire being created from a pin on Metal2.



3. Now, right-click to display the *Create Stranded Wire* context-sensitive menu.



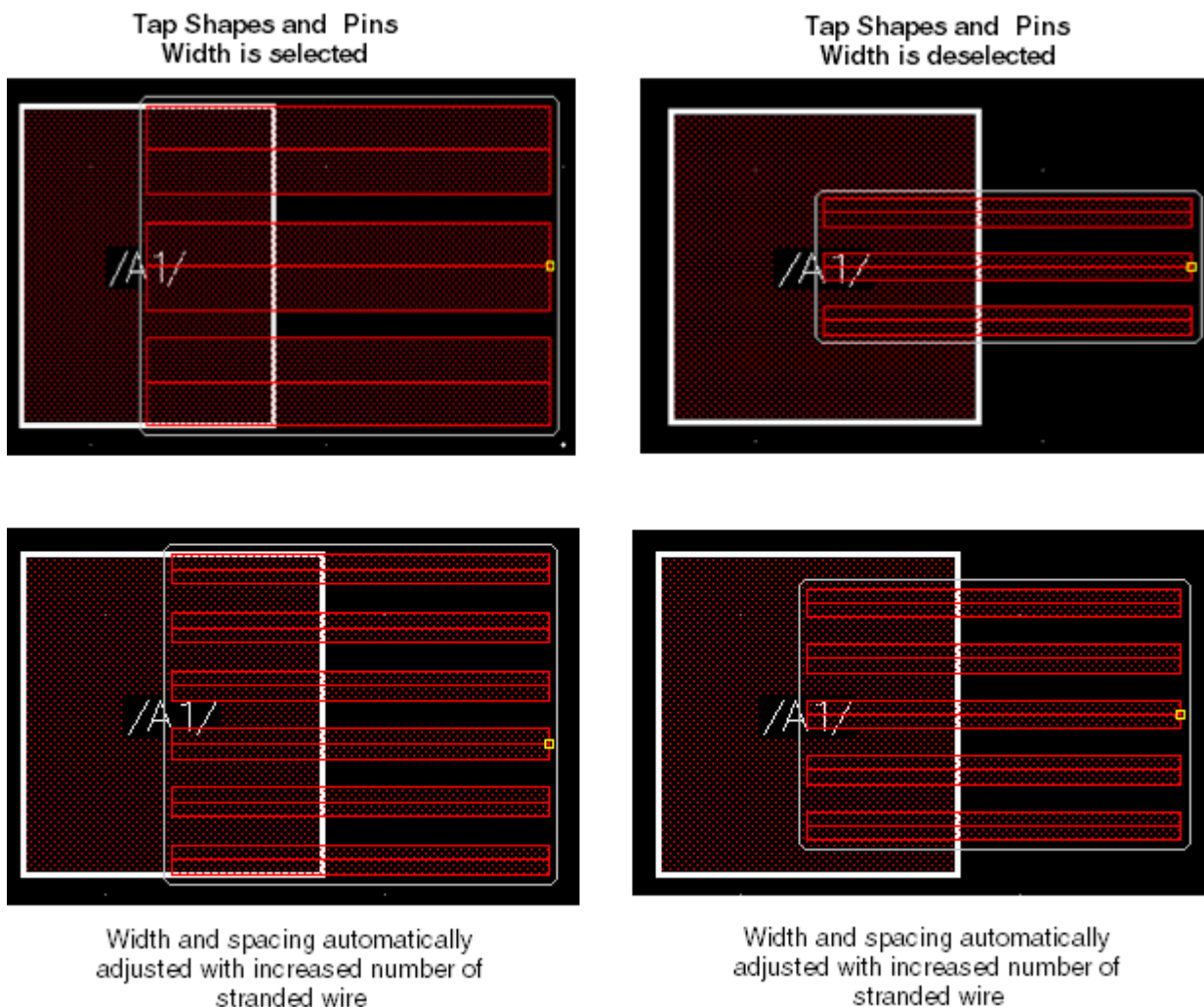
Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

4. Place the mouse pointer on the *Use Width* option. You can see that the *Tap Shapes and Pins Width* option is selected by default. In this case, the stranded wires are evenly spread to cover the existing pin. The width and spacing of the stranded wires are calculated automatically based on the maximum width and minimum spacing of the layer, if not specified in the Create Stranded Wire form.

However, when the *Tap Shapes and Pins Width* option is deselected, the stranded wires are created from the click point. If the width and spacing are not specified, `minWidth` and `minSpacing` are used.

The following figure shows how the stranded wires are created when the *Tap Shapes and Pins Width* option is selected and deselected and how the spacing is adjusted when the number of strands in a stranded wire increases.

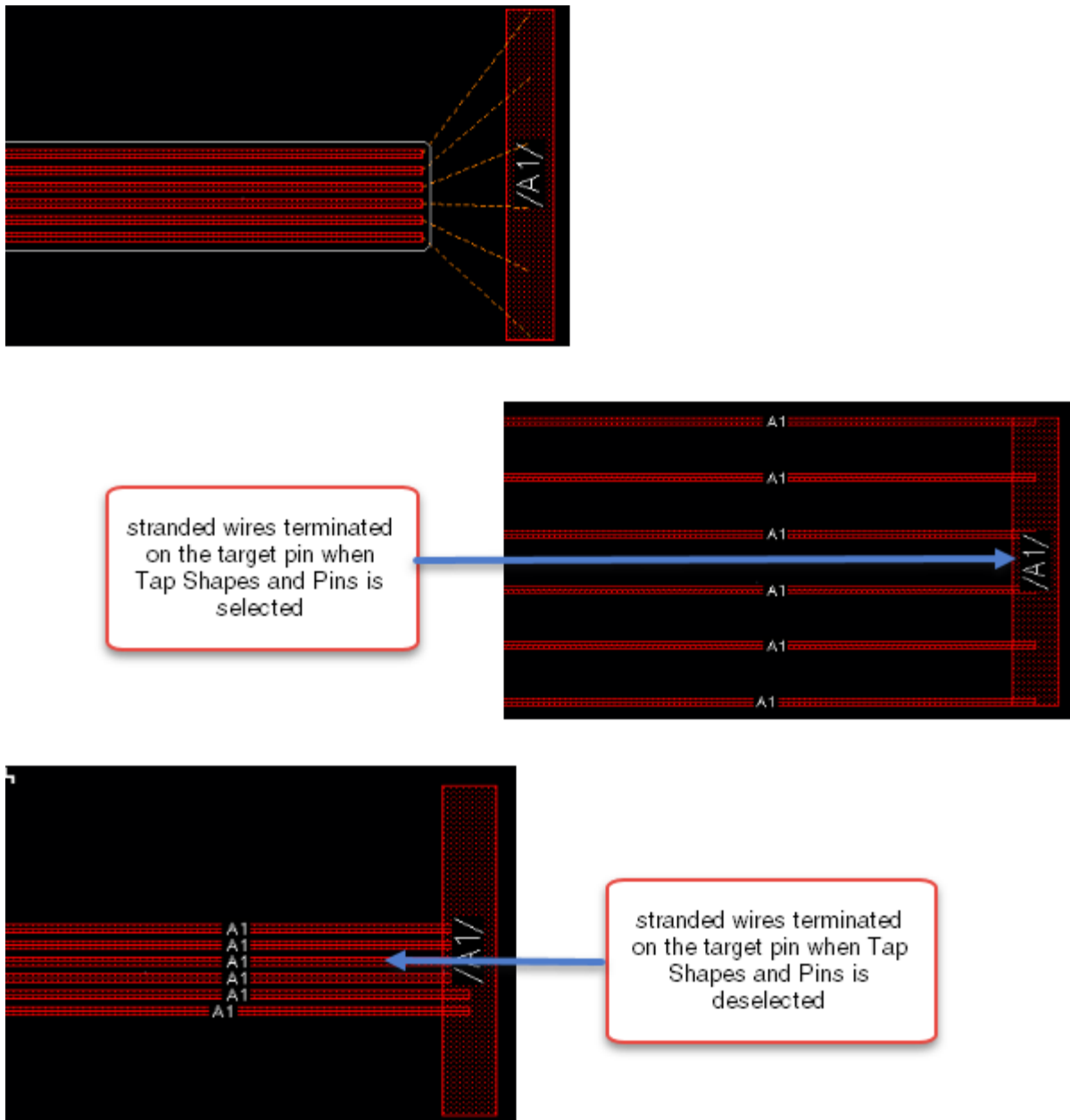


Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

5. To terminate the stranded wires on a target pin, select *Tap Shapes and Pins Width*. The strands are automatically spread along the target pin. However, when the *Tap Shapes and Pins Width* is deselected, the strands does not spread out and are digitized at the click point.

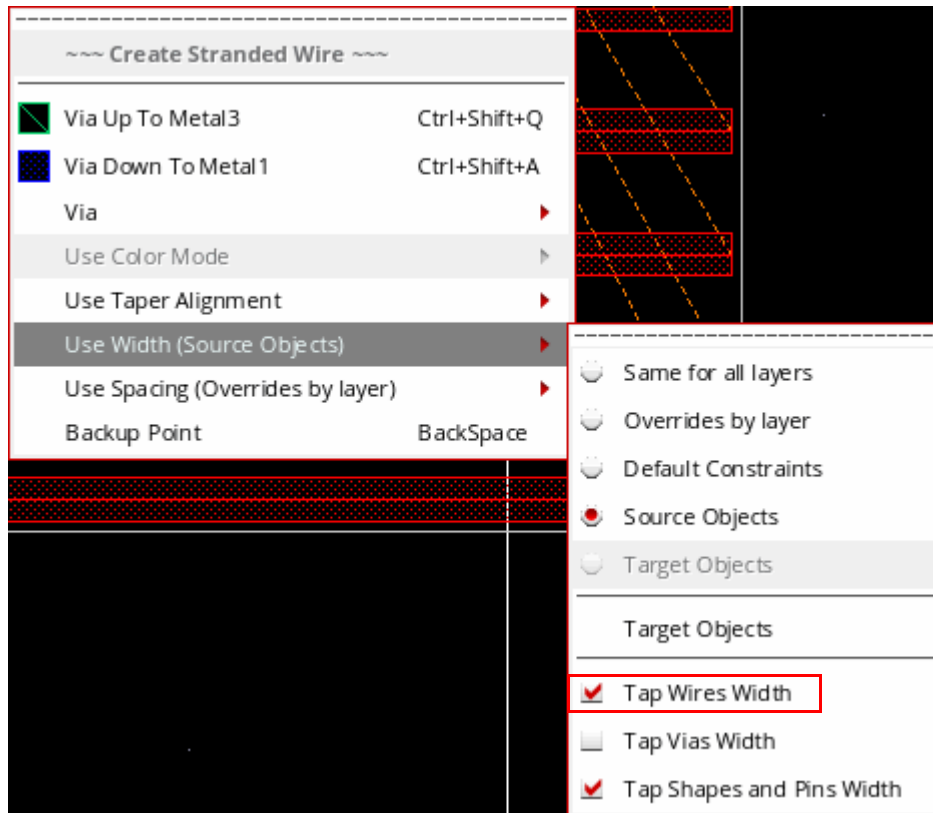
The following figure shows the termination of stranded wires on the target pin when the *Tap Shapes and Pins Width* option is selected and deselected.



Creating a Stranded Wire from an Existing Wire

To create a stranded wire from an existing wire:

1. Choose *Create – Wiring – Stranded Wire*.
2. Select the *Tap Wires Width* option from the *Use Width* submenu.

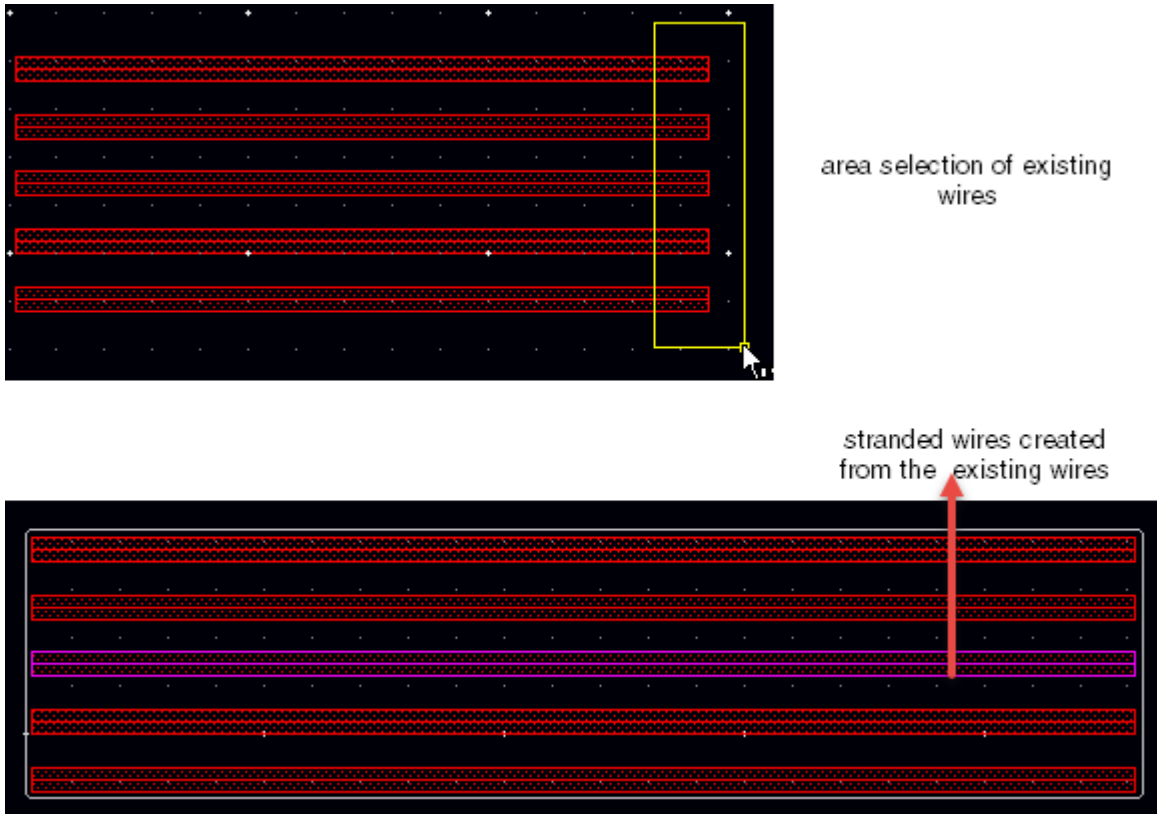


3. Select the end of the existing wires by area selection and continue to create the stranded wire.

Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

The stranded wire is created. For the created stranded wire, the *Use Width* option is automatically set to *Source Object* and the stranded wire continues with the width and spacing of the source object. In addition, the number of strands remains constant.

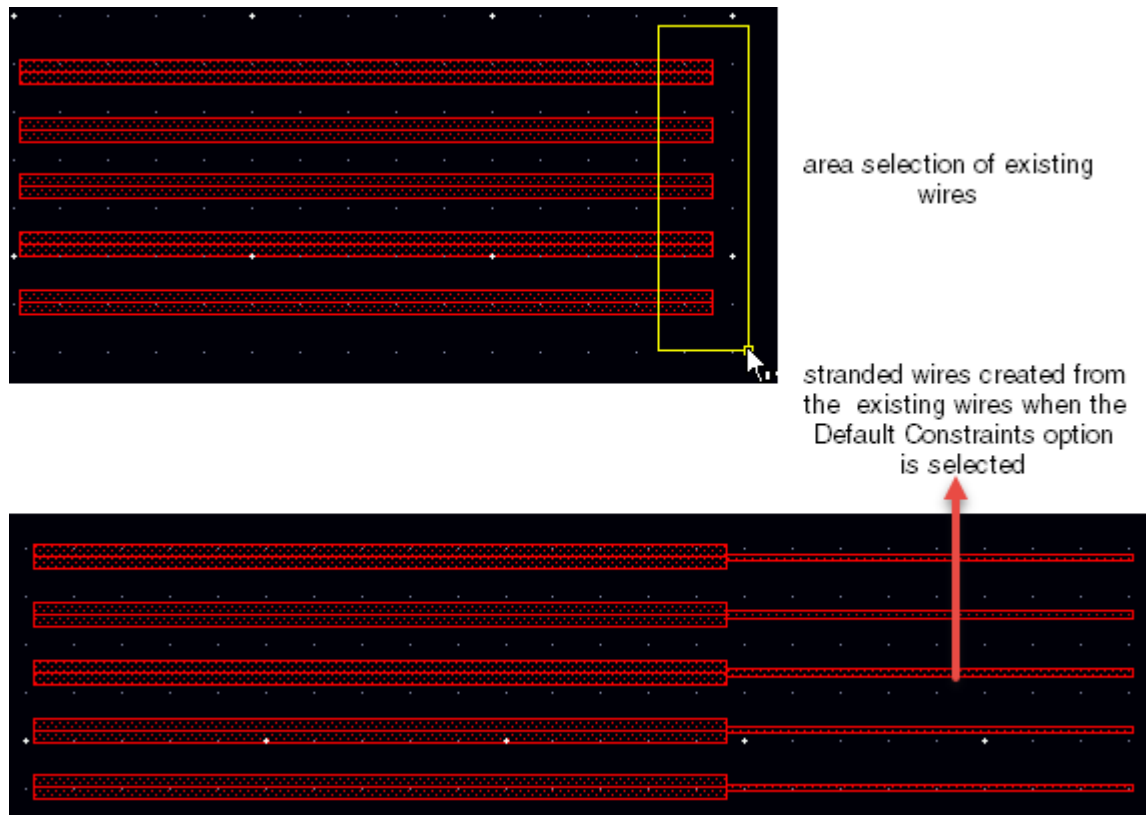


4. Now, deselect the *Tap Wires Width* option from the *Use Width* submenu.
5. Select the *Override by Layer* or the *Default Constraints* option from the *Use Width* submenu.

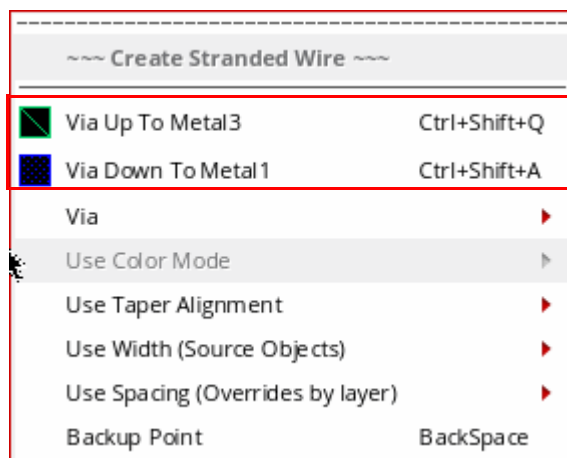
Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

In this case, the width of the stranded wire is changed based on the option that you have selected from the *Use Width* submenu. However, the number of strands remains constant.



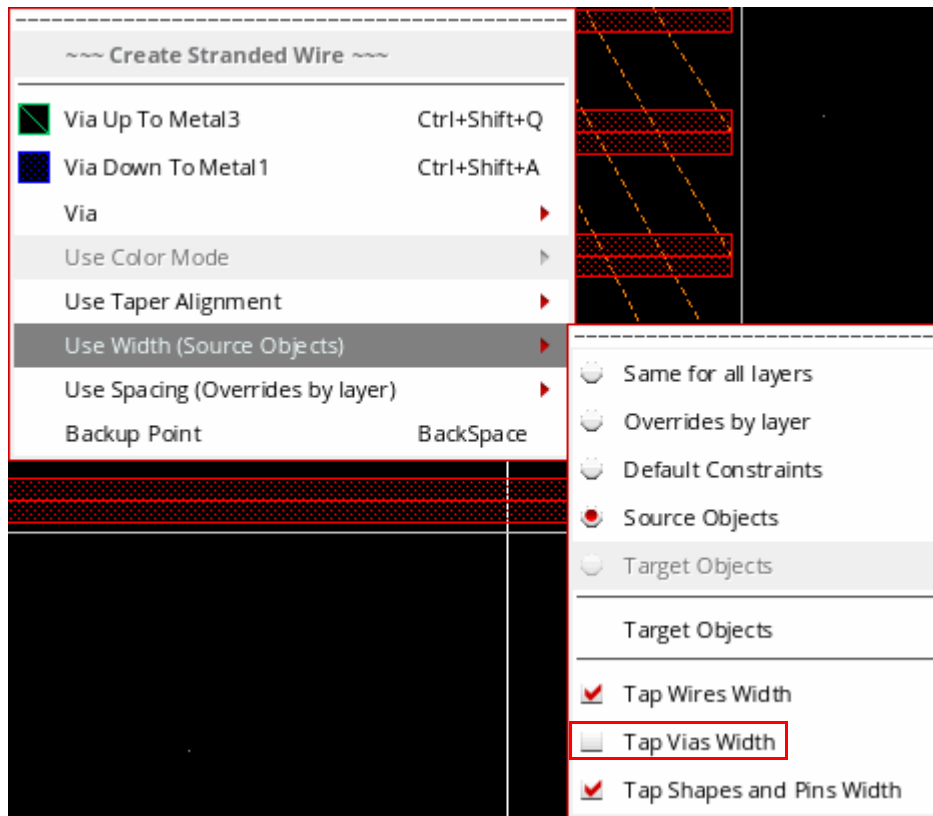
6. Insert vias on the existing wire by selecting the *Via Up To <Metal Layer>* or *Via Down To <Metal Layer>* option from the *Create Stranded Wire* context-sensitive menu.



Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

7. Select the *Tap Vias Width* option from the *Use Width* submenu.

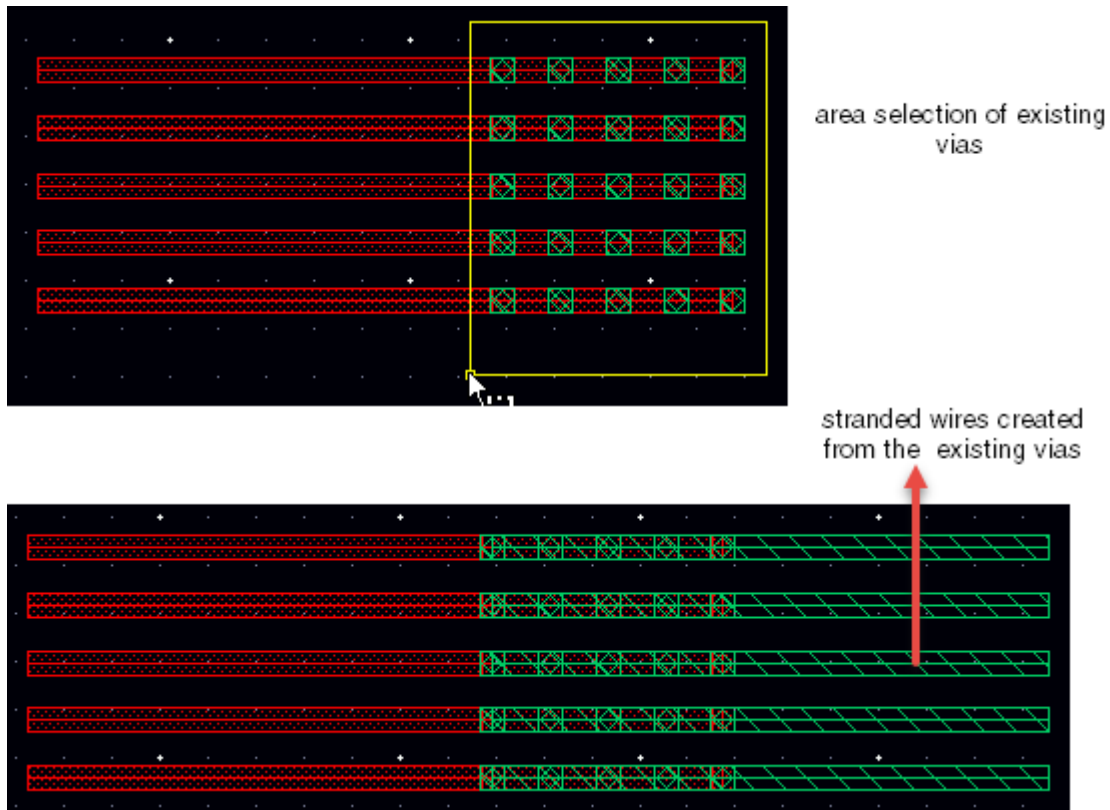


8. Select the inserted vias by area selection and continue to create the stranded wire.

Virtuoso Interactive and Assisted Routing User Guide

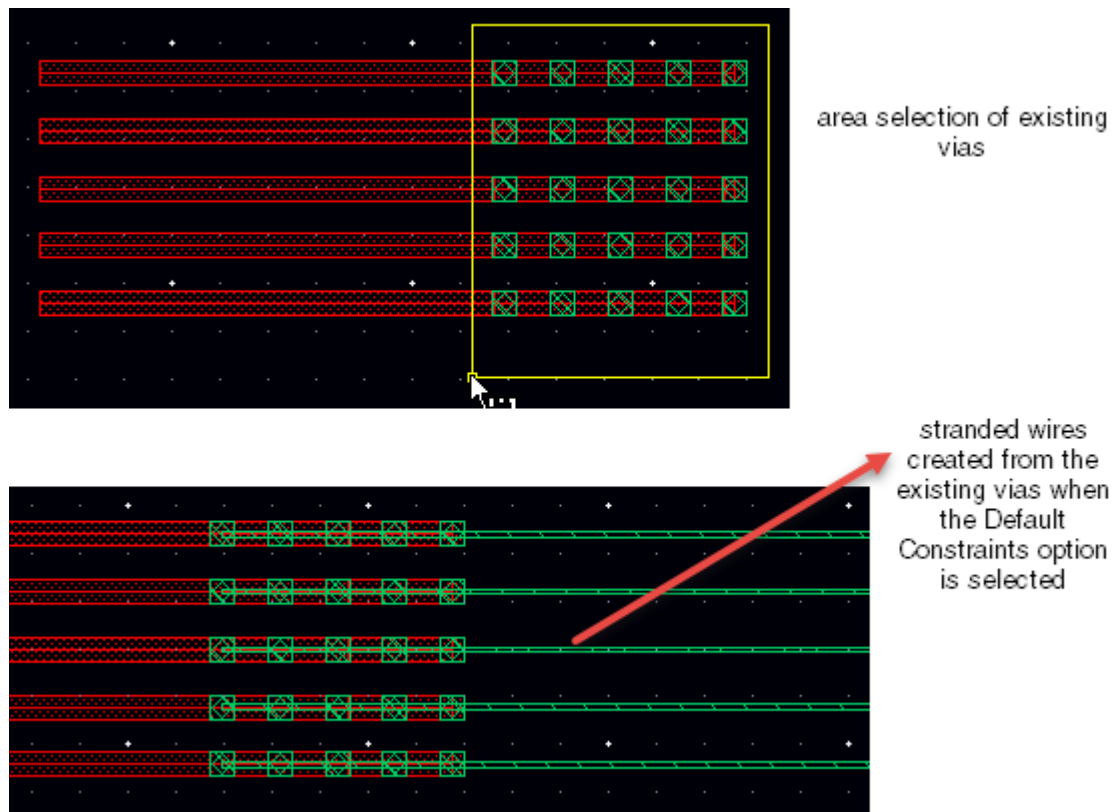
Working with Stranded Wires

The stranded wire is created. For the created stranded wire, the *Use Width* option is automatically set to *Source Object* and the stranded wire continues with the width and spacing of the source object. In addition, the number of strands remains constant.



9. Now, deselect the *Tap Via Width* option from the *Use Width* submenu.
10. Select the *Override by Layer* or the *Default Constraints* option from the *Use Width* submenu.

In this case, the width of the stranded wire is changed based on the option that you have selected from the *Use Width* menu. However, the number of strands remains constant.



Note: Changing the *Use Spacing* to any value does not affect the stranded wire that is created.

Creating a Stranded Wire from Tieout

Sometimes it is necessary to branch from an existing stranded structure. This is what is called as tieout. You can create a stranded wire from a tieout when the existing stranded structure belongs to the same net.

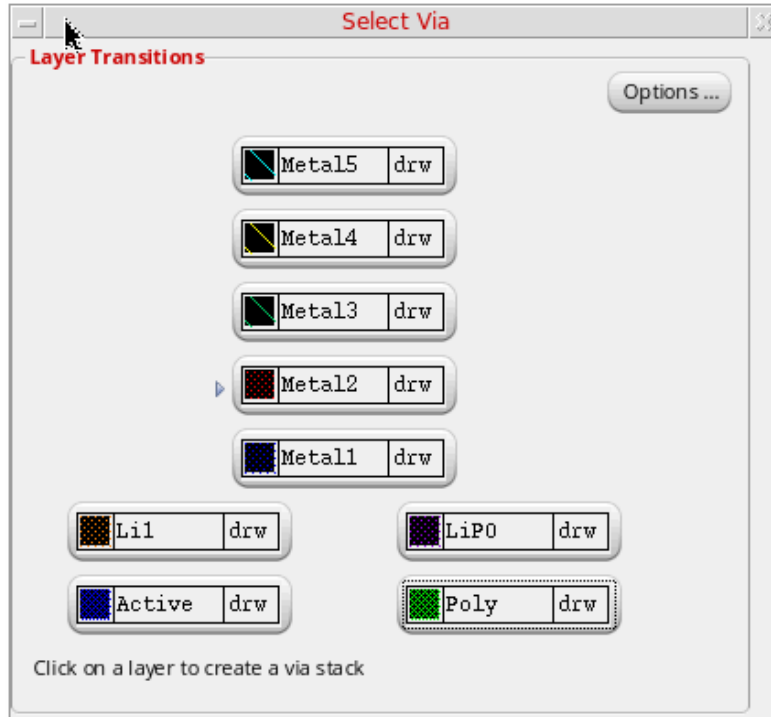
To create a stranded wire from a tieout:

1. Choose *Create – Wiring – Stranded Wire*.
2. In the existing stranded structure, select by area from where you want to start creating the stranded wire.

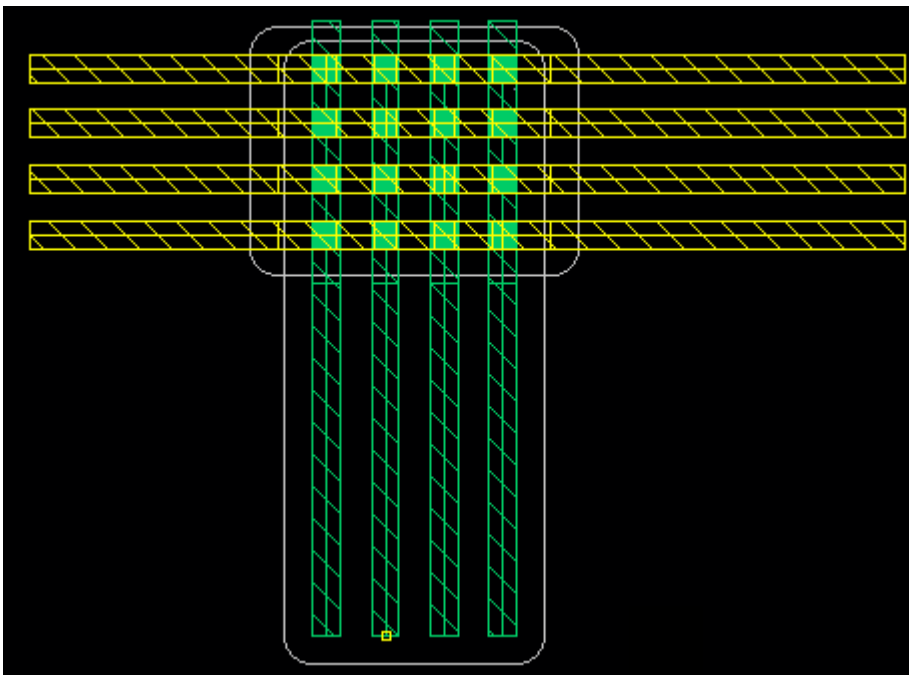
Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

The Select Via form displays.

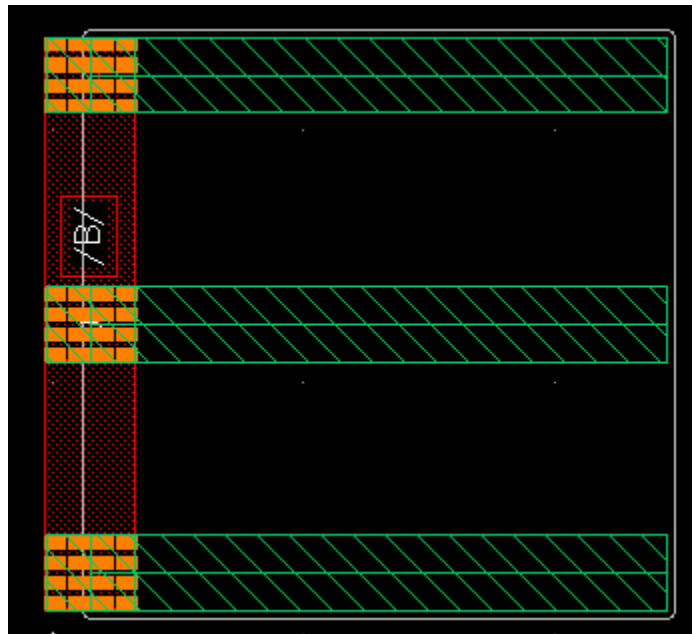


3. From the Select Via form, select the layer on which to create the stranded wires and click **OK**. The following figure shows how the stranded wires is created from tieout.



Creating a Stranded Wire from a Rectangle or PathSeg Covering a Pin

You can create a stranded wire from the rectangle and pathSeg that completely covers a smaller pin shape of the same metal layer and net name. In this case, the rectangle and the pathSeg is considered as a pin and stranded wires can be created, starting from them. The following figure shows how the stranded wires are created from a rectangle that completely covers a smaller pin shape.



Inserting Vias Automatically

The *Create Stranded Wire* command supports automatic insertion of vias on source and target objects. A via is automatically inserted when you start a stranded wire from a different metal layer than the metal layer on which the object exists.

Important

While creating a stranded wire, a via is automatically inserted only when a stranded wire starts from a pin. In case, the stranded wire starts from a rectangle or a pathSeg, a via is automatically inserted when the rectangle or pathSeg is covered by a pin.

To automatically insert vias while creating a stranded wire:

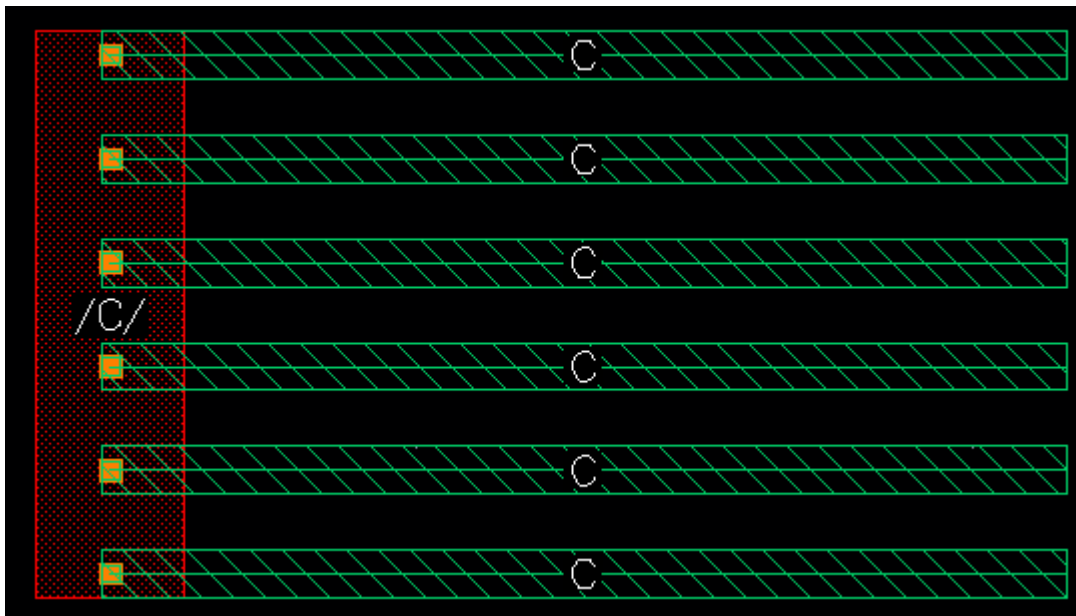
1. Select the desired metal layer from the *Layers* pane.

2. Choose *Create – Wiring – Stranded Wire*.

3. Hold the `Ctrl` key and click the object from which you want to start the stranded wire.

A via is automatically inserted on the source object and the stranded wire continues on the selected metal layer.

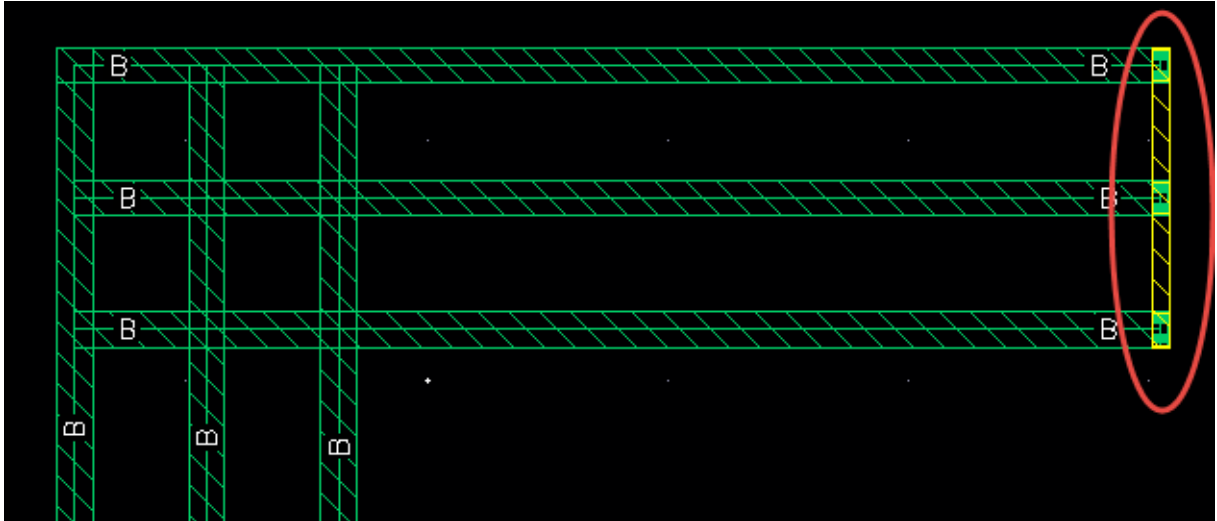
The following figure shows an example where the vertical shape is the source pin which is on `Metal2`. The horizontal stranded wires are on `Metal3`. The `M3M2` via is automatically inserted on the source pin when you start creating the stranded wire.



Virtuoso Interactive and Assisted Routing User Guide

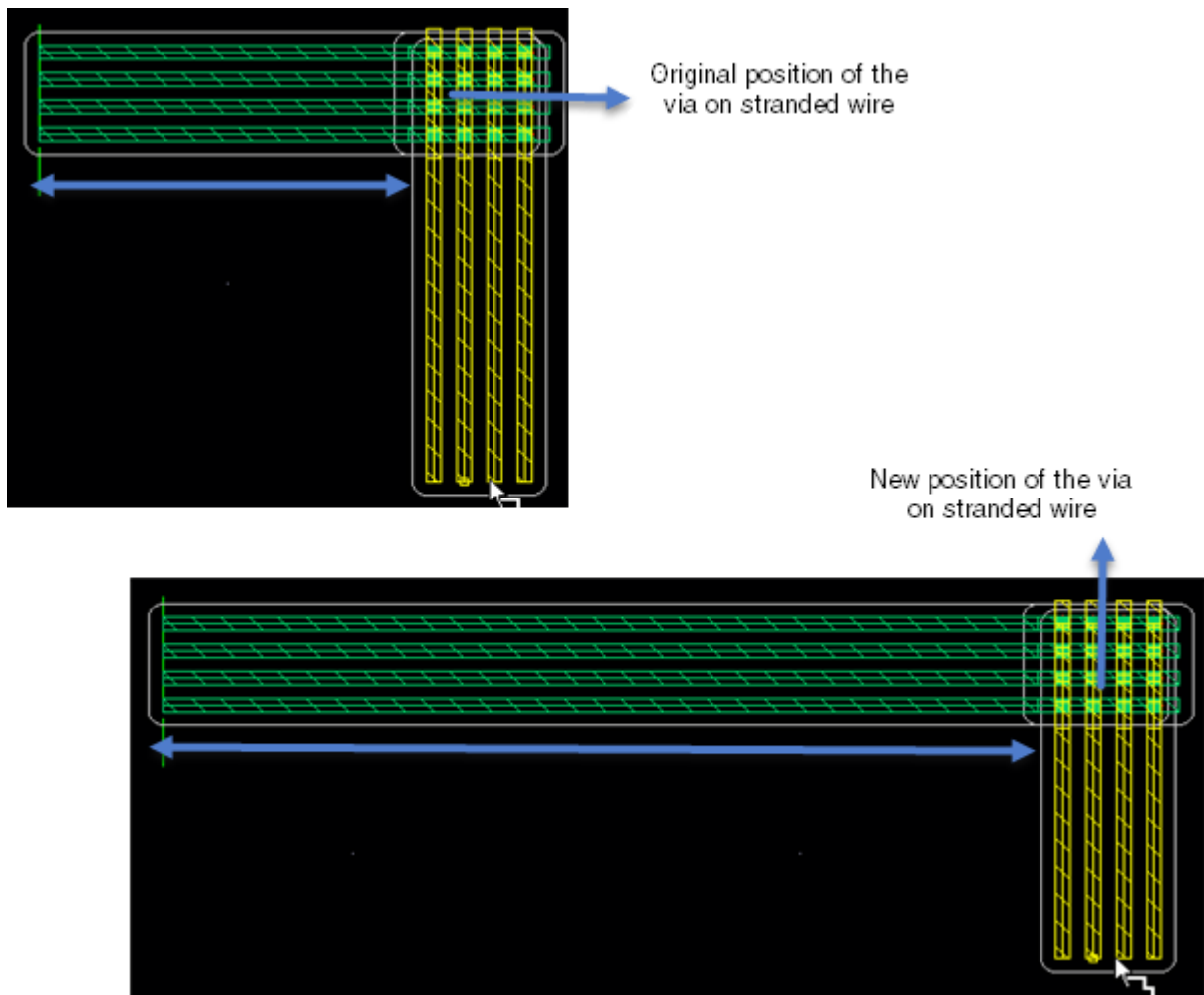
Working with Stranded Wires

Similarly, a via can be automatically inserted on a target object, as shown in the following figure.



You can also slide the stranded wire along with the vias. To slide the current section of the stranded wire, hold the `Ctrl+Shift` keys and move the mouse pointer to the right

or left of the current position. The following figure shows the change in the via position when the stranded wire slides from left to the right.



Tapering in Stranded Wire

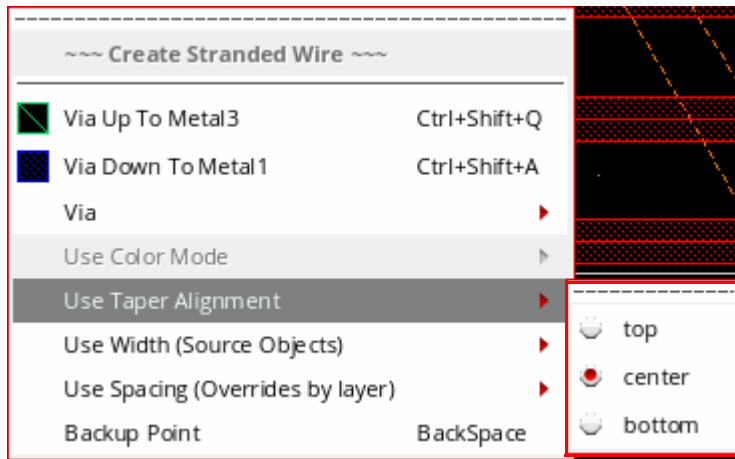
Tapering in stranded wire lets you align the strands in a stranded wire when the number of strands are changed on each section in the same direction for the same layer or different metal layers. To do this,

1. Choose *Create – Wiring – Stranded Wire*.
2. Start creating a stranded wire with four strands from left to right. When the stranded wires are created in the horizontal direction, the taper alignment mode can be selected as top, center, or bottom.

Virtuoso Interactive and Assisted Routing User Guide

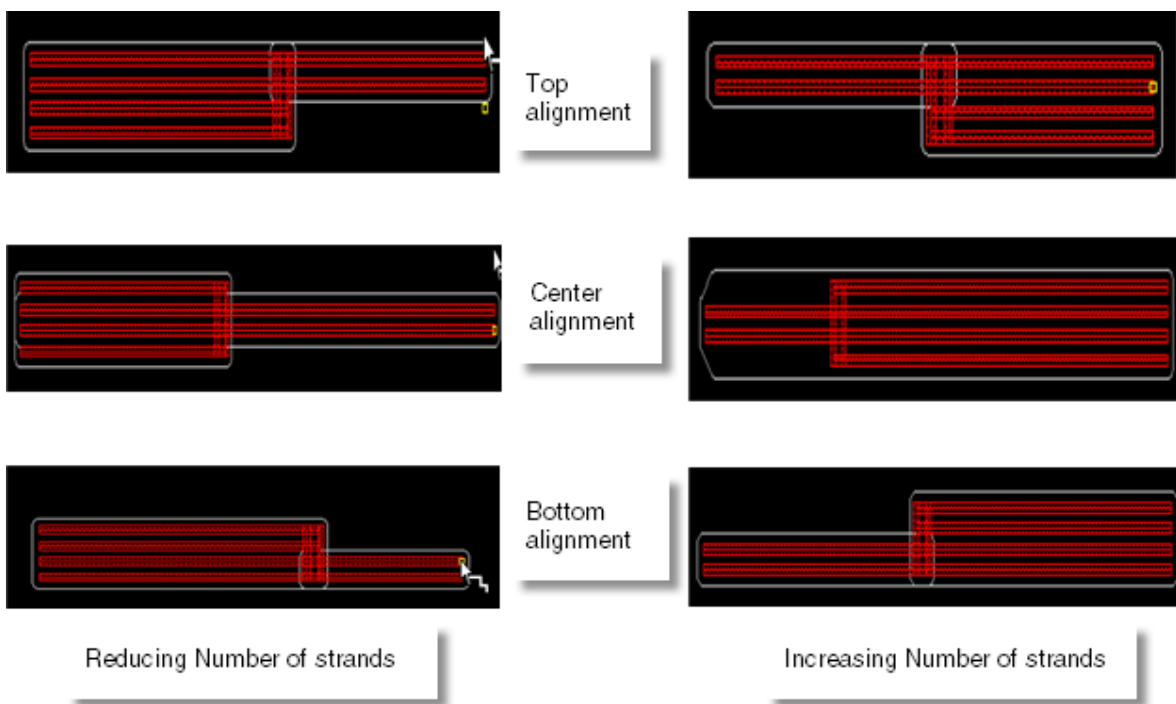
Working with Stranded Wires

3. Right-click the stranded wire being created and place the mouse cursor on the *Use Taper Alignment* option. A submenu is displayed.



4. Select *top*, *center*, or *bottom* as the taper alignment mode.
5. Increase or decrease the number of strands using the Ctrl+1 or Ctrl+2 bindkeys, respectively.

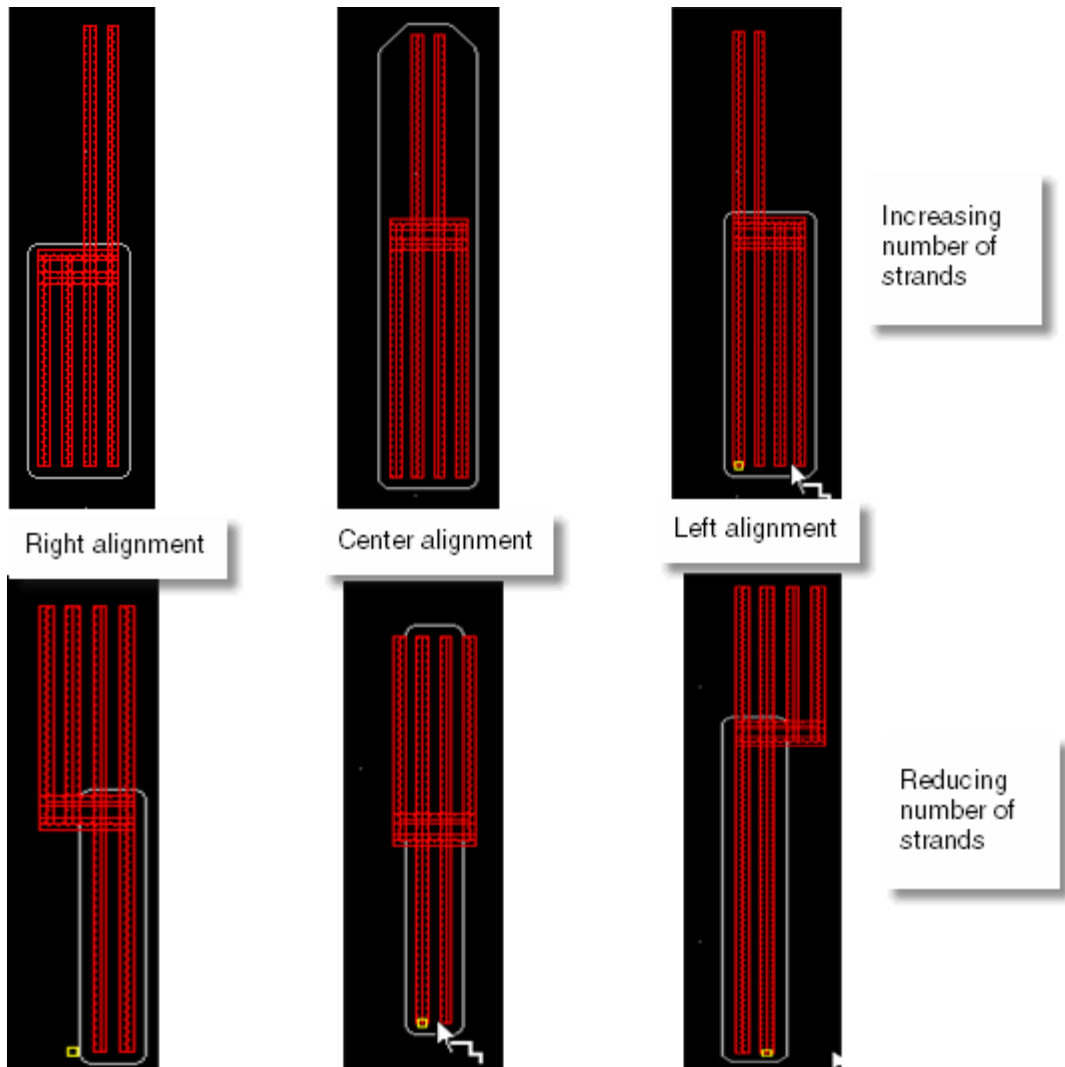
The following figure shows how the stranded wires are aligned when the number of strands is reduced or increased and the taper alignment mode is selected as *top*, *bottom*, or *center*.



Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

When the stranded wire is being created in the vertical direction from top to bottom, the taper alignment mode can be selected as *right*, *center*, or *left*. The following figure shows how the stranded wire is aligned when the number of strands is reduced or increased and the taper alignment mode is selected as *right*, *center*, or *left*.

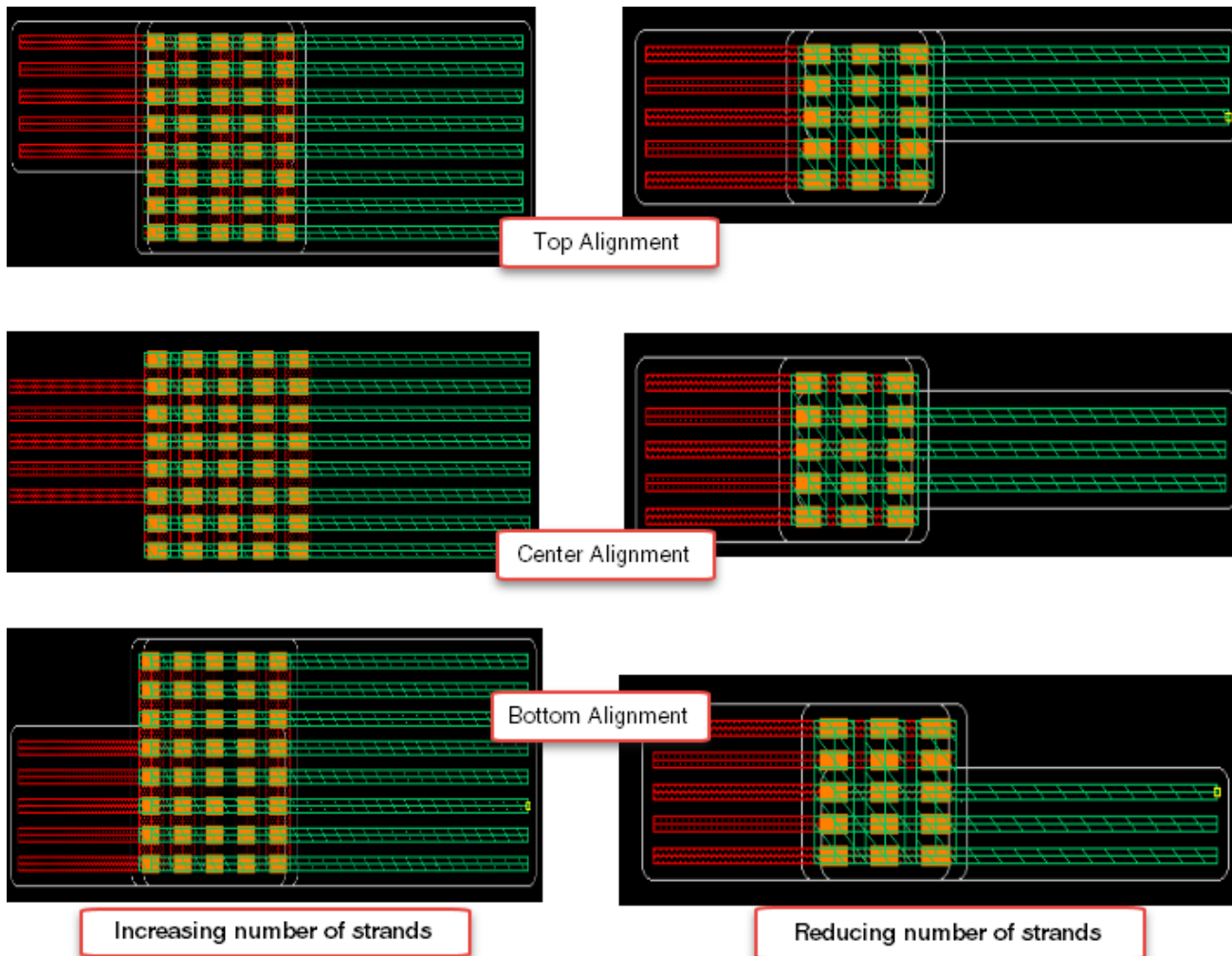


Similarly, you can reduce or increase the number of strands when the stranded wire is on different metal layers. To reduce or increase the number of strands on a different metal layer, select *Via Up To <Metal layer>* or *Via Down To <Metal Layer>*.

Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

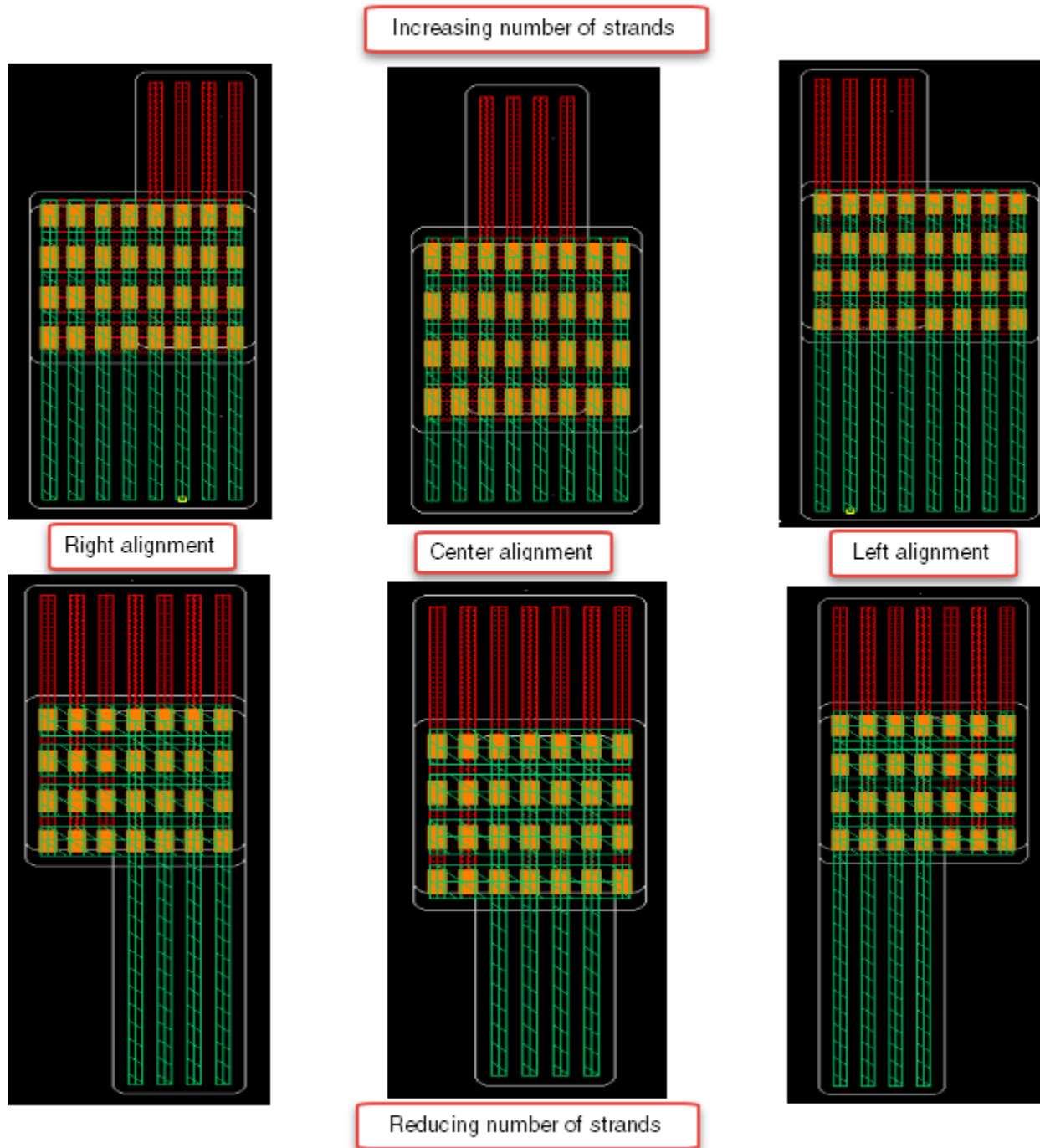
The following figure shows how stranded wires in the horizontal direction are aligned when the number of strands are reduced or increased on a different metal layer.



Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

Similarly, the following figure shows how stranded wires in the vertical direction are aligned when the number of strands are reduced or increased on a different metal layer.



Enabling Bridge Vias Using WSPs

Using standard vias can increase the spacing between the wires and thus reduce density. Therefore, using bridge vias with a stranded wire allows wires to be placed closer to each other to maintain density. A bridge via forms a bridge between two lines in a stranded wire. In a bridge via, the cut layer stretches across both strands to form the connection.



When the WSP track spacing conflicts with the bridge via spacing, you fix it by adjusting the WSP track spacing to match the bridge via spacing.

To enable bridge vias:

1. From the *Wire Snapping* section of the Layout Editor Options form, select *Snap Pattern*.
2. Choose *Create – Wiring – Stranded Wire*.
3. Start creating a stranded wire with two strands.
4. Right-click the stranded wire being created. The *Create Stranded Wire* context-sensitive menu gets displayed.
5. Choose *Via – Select Via*. The Select Via form is displayed.

Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

6. Click the *Options* button in the Select Via form. The Select Via Options form is displayed.

Via Definitions		Cut Class	User Min	Cut Spacing	
			X	Y	
Metal4 -> Metal5	M5M4_stdV	Auto			
Metal3 -> Metal4	M4M3_stdV	Auto			
Metal2 -> Metal3	M3M2_stdV	Auto			
Metal1 -> Metal2	M2M1_stdV	Auto			
LiPO -> Metal1	M1_LiPO	Auto			
Li1 -> Metal1	M1_Li1	Auto			
Li1 -> LiPO	LiPO_Li1	None			
Poly -> LiPO	LiPO_Poly	None			
Active -> Li1	Li1_NActive	None			

7. Select the bridge via definition from the Select Via Options form. The bridge via definition appears in the Select Via Options form only if it is defined in the technology file.

Via Definitions	Cut Class
Metal4 -> Metal5	Auto
Metal3 -> Metal4	Auto
Metal2 -> Metal3	Auto
Metal1 -> Metal2	Auto
LiPO -> Metal1	Auto

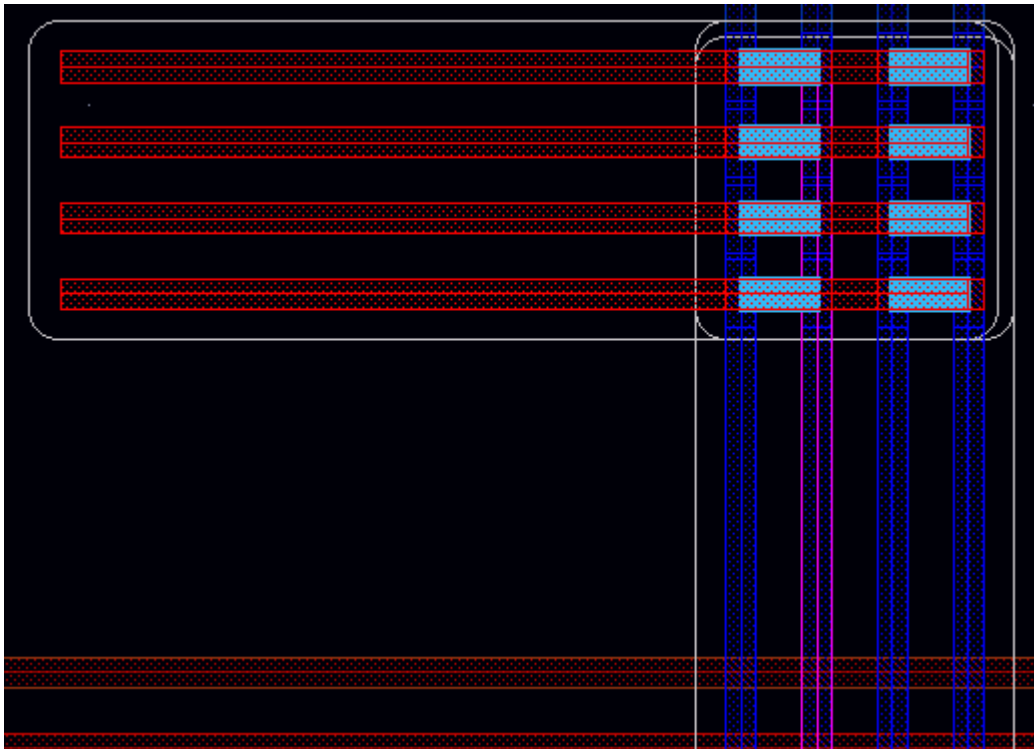
Note: When bridge via is defined in the technology file, you do not need to explicitly

Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

select the bridge via from the Select Via Options form. The bridge via is automatically inserted on WSP tracks.

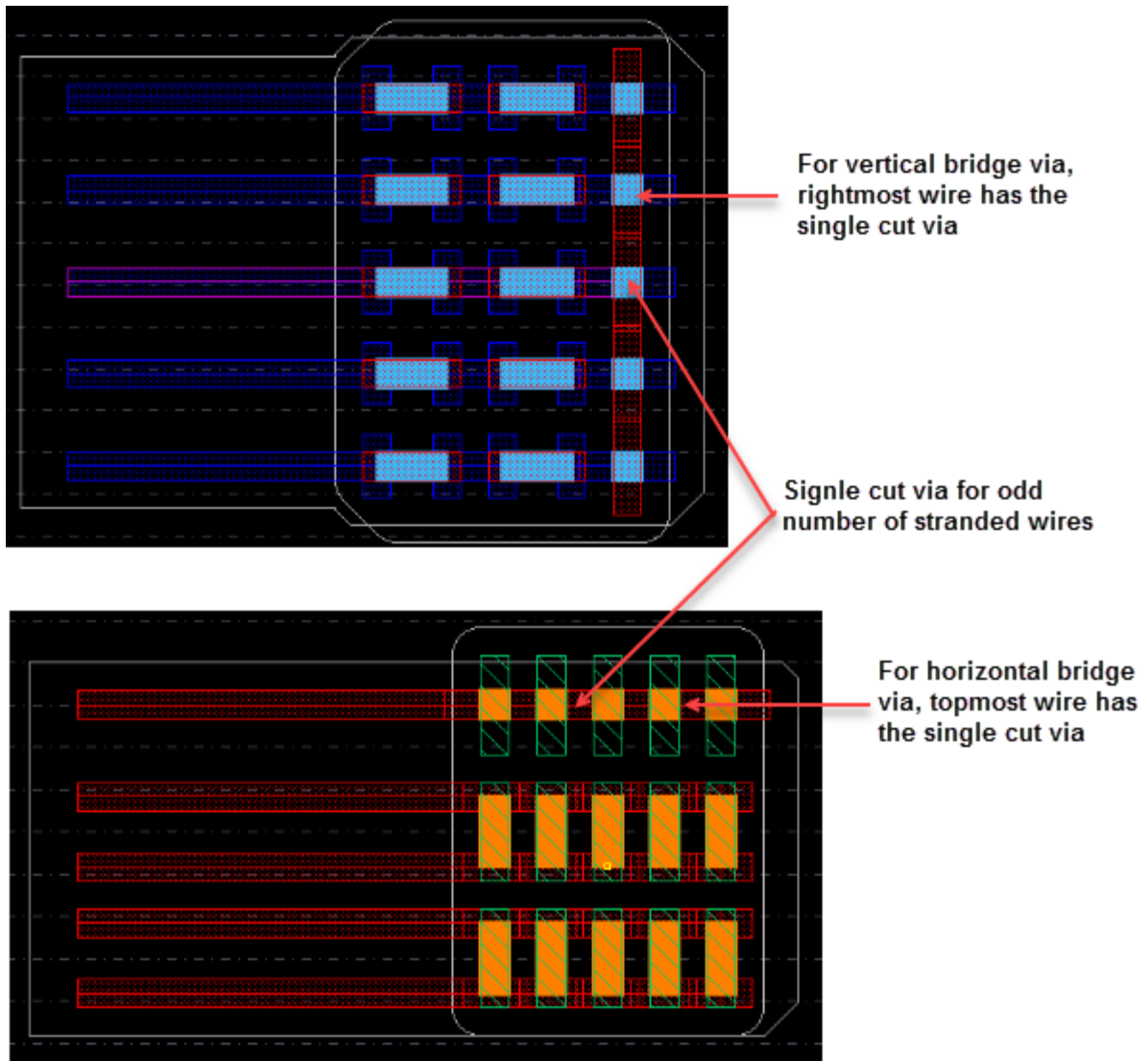
8. Click *OK*. The following figure shows how the bridge via is automatically inserted at the intersection of two tracks on different metal layers.



Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

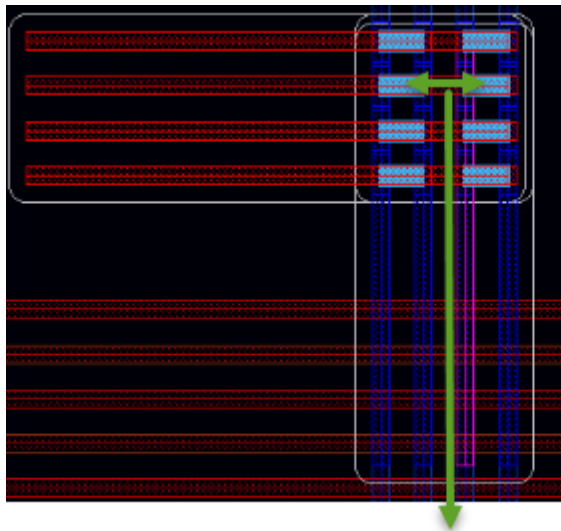
Bridge vias are usually paired. However, if you have odd number of stranded wires, a single cut via is inserted. In case of vertical bridge via, the via of the rightmost wire is the single cut via. In case of horizontal bridge via, the via of the top wire is a single cut via.



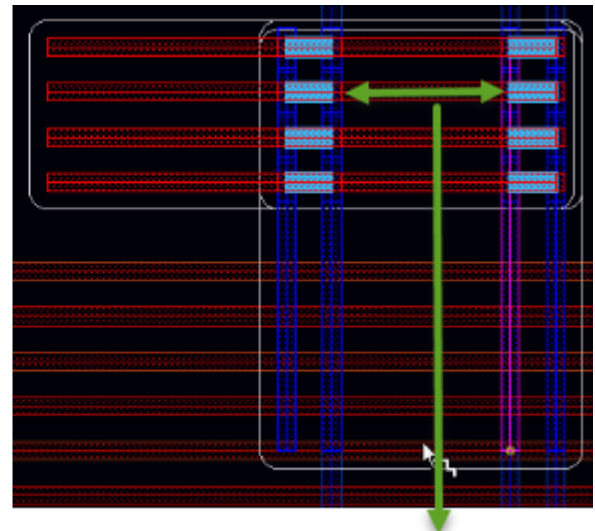
Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

When you modify the spacing between the stranded wires, strand spacing having the bridge via pair remains constant. However, the strand spacing between the other stranded wires change, as shown in the following figure.

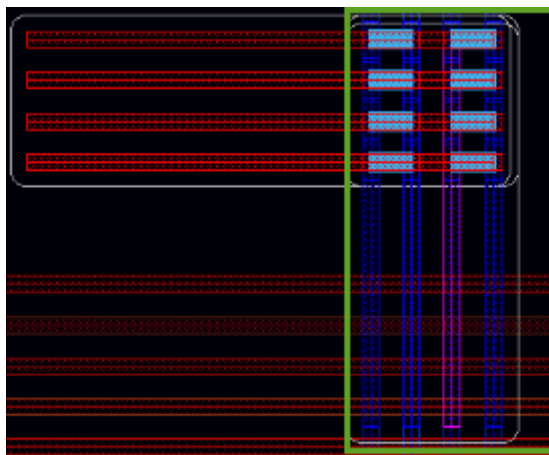


Original spacing between stranded wires

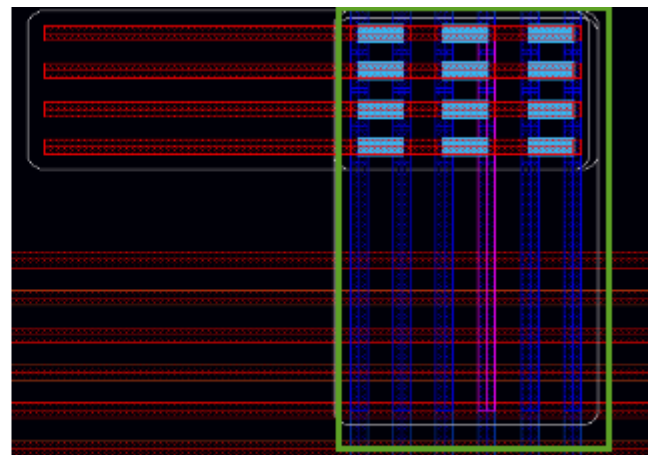


Increased spacing between stranded wires

You can increase the number of strands when a bend is made and a bridge via is inserted. The following figure shows an increase in the number of strands. The strand is added to the right and the existing strands move to the left.



Original number of strands after a bend are 4

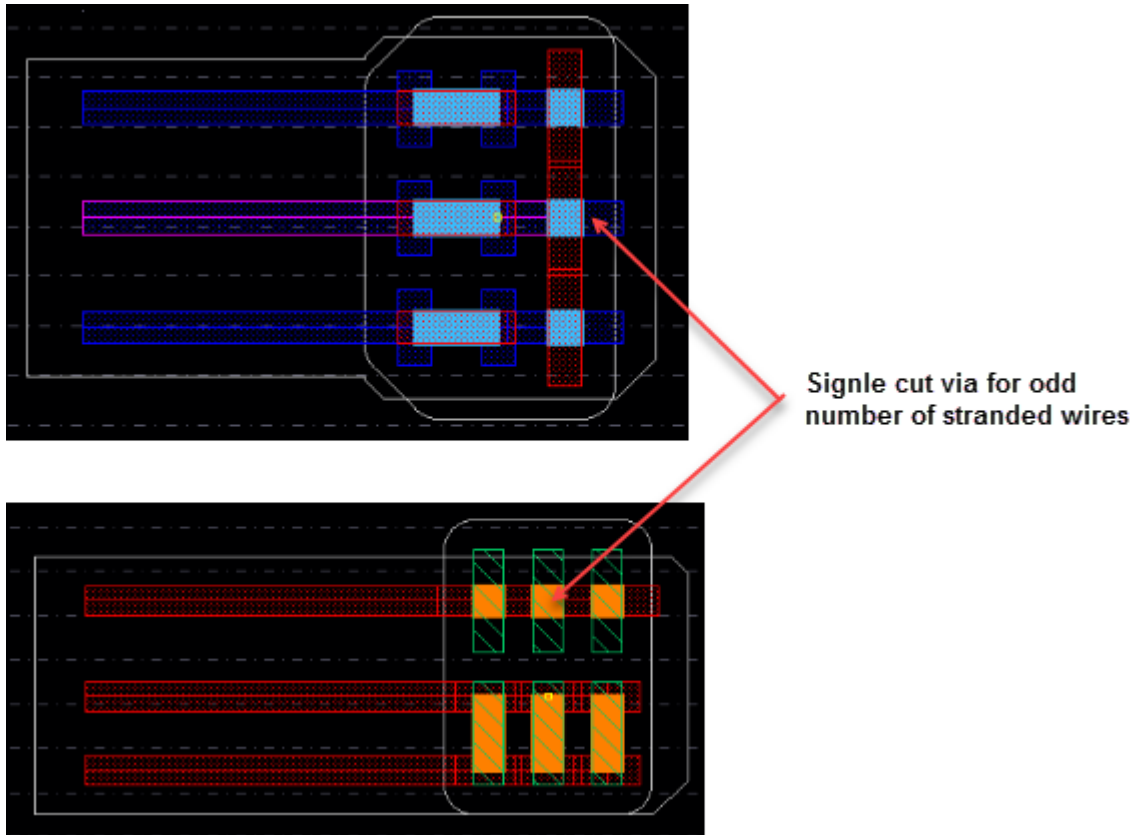


Number of strands are increased to 6 after a bend

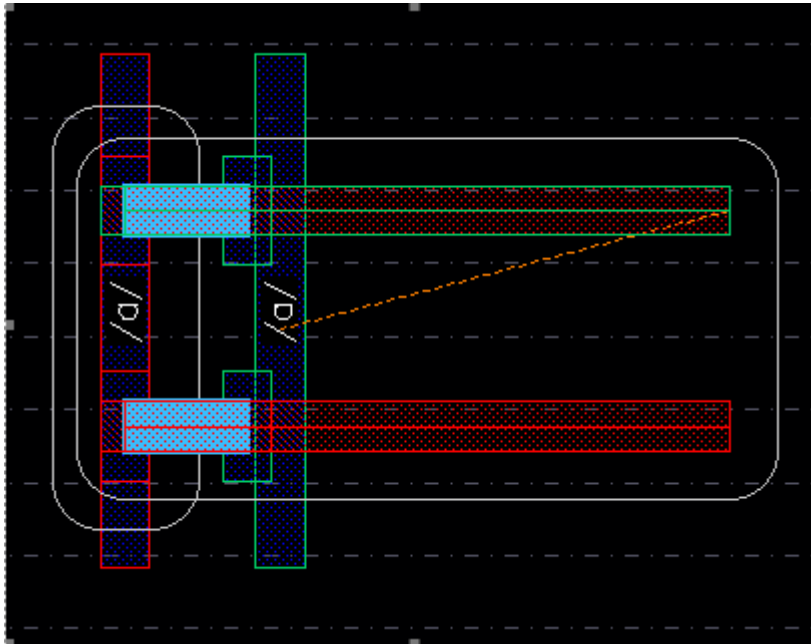
Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

When the number of strands are increased or decreased from an even number to an odd number, then the topmost or the rightmost via is modified to a single cut via.



You can also add a bridge via on a pin. For this, you must adjust the pin spacing to match the bridge via spacing. The following figure shows the result when the pin spacing is not adjusted to match the bridge via spacing.



Important

When MPT is selected and bridge via is created, then the color mode must be *optimized*. You cannot create a bridge via when the color mode is *uniform*. The following warning message is displayed when you try to create a bridge via in *uniform* color mode.

WARNING (WE-27603): Cannot create bridge via in uniform color mode. Switch to mode before creating the bridge via.

For more information on color modes in stranded wire, see [Specifying the Color Mode in Stranded Wire](#).

Avoiding Shorts with Existing Blockages

The stranded routing feature has been enhanced to avoid creating a short with an existing blockage by skipping tracks. Any shape on the same layer but with a different net name between previous bend and current point is considered a blockage. Blockages can be avoided only when creating a stranded wire starting from a space. In addition, only blockages on the same layer with a different net name are avoided. To route the net without a short, the *Stranded Wire* command uses the next nearest track that does not have a blockage.

Virtuoso Interactive and Assisted Routing User Guide

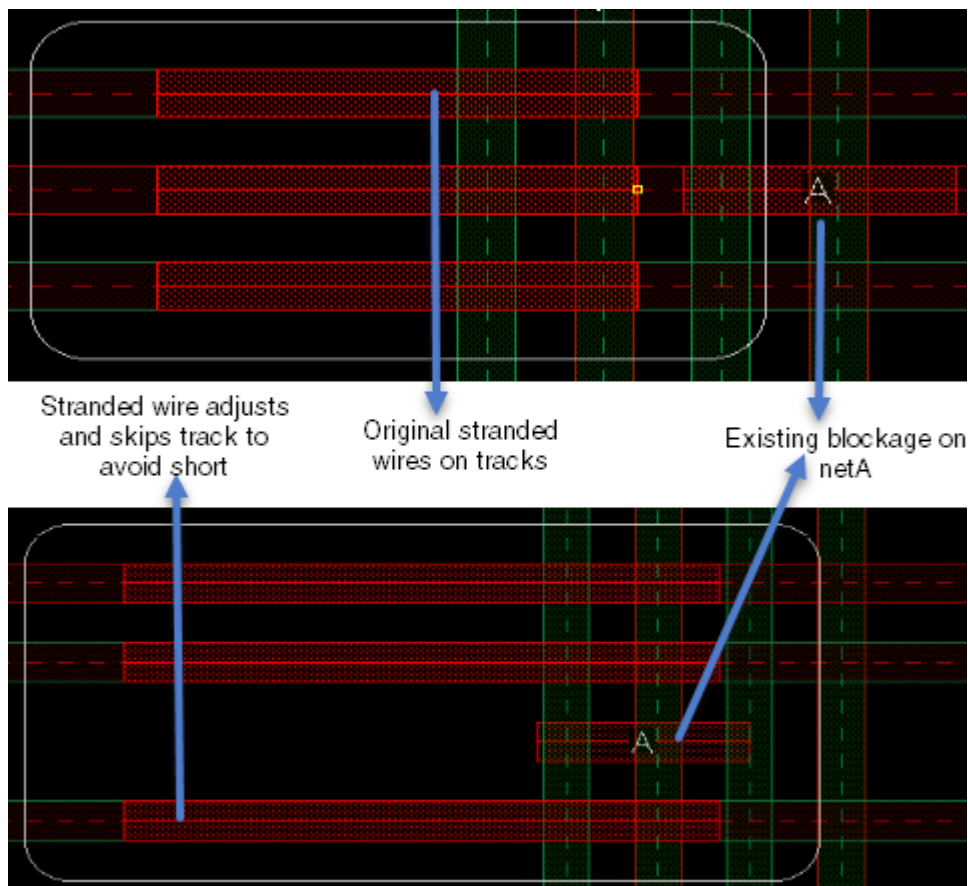
Working with Stranded Wires

The following scenarios illustrate how a stranded wire skips the track on which a blockage appears and accordingly adjusts to the next nearest track that does not have any blockage.

Scenario 1: Blockage is on layer `Metal2` with net name `A`.

1. Choose *Create – Wiring – Stranded Wire* and press `F3`.
2. Specify the net name `B` in the *Net Name* field.
3. Move the stranded wire in the direction in which a blockage with net name `A` appears.

The stranded wire with net name `B` skips the track that has a blockage with net name `A` and accordingly adjusts to the next nearest track that does not have any blockage.



Scenario 2: Blockage is on layer `Metal2` with net name `A`. Increase or decrease the number of stranded wires.

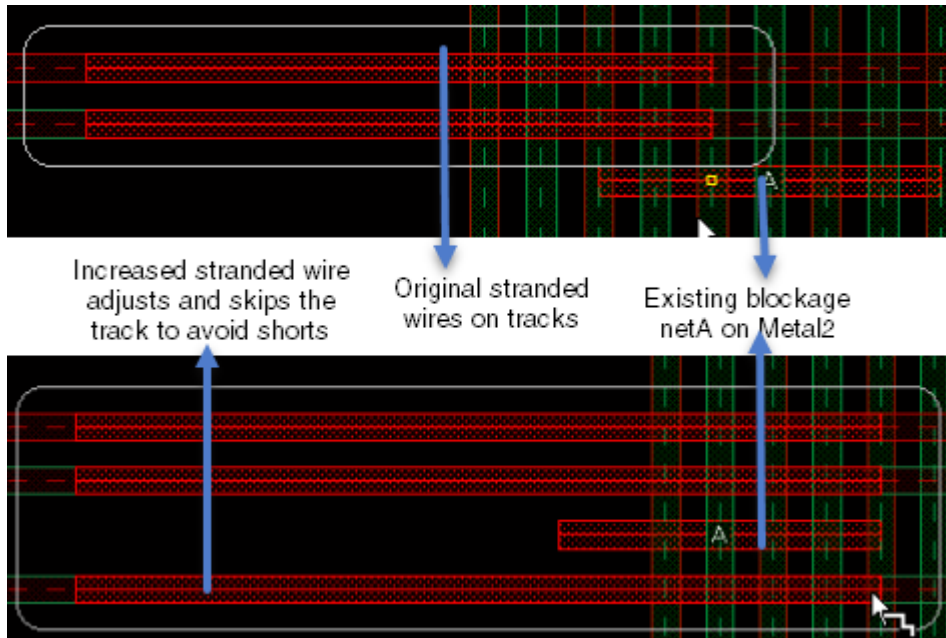
1. Choose *Create – Wiring – Stranded Wire* and press `F3`.
2. Specify the net name `B` in the *Net Name* field.

Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

3. Use the `Ctrl+1` bindkey to increase the number of strands.

The increased number of stranded wires skip the track that has a blockage with net name `A` and accordingly adjusts to the next nearest track that does not have any blockage.



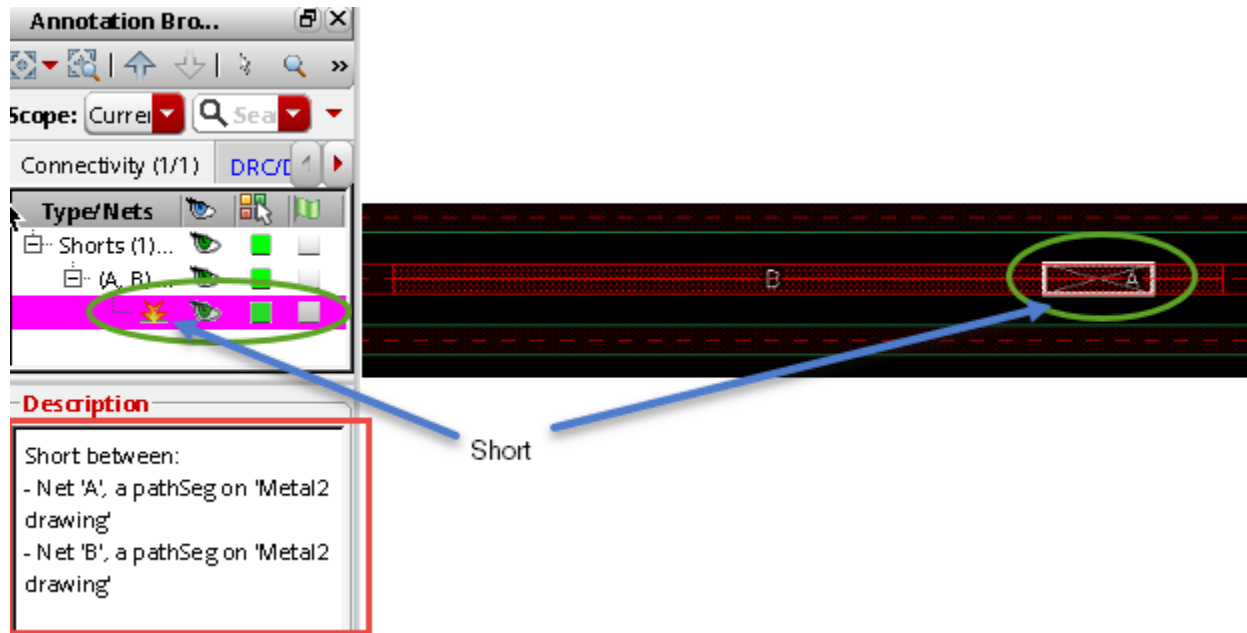
Scenario 3: Blockage is on layer `Metal2` with net name `A`. In addition, a wire exists with net name `A`. Create a stranded wire starting from an existing wire.

1. Choose *Create – Wiring – Stranded Wire*.
2. Click the existing wire to digitize the stranded wire and continue creating the wire. The stranded wire is automatically assigned the net name `B`.
3. Move the stranded wire in the direction in which a blockage with net name `A` appears.

Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

In this case, the stranded wire does not start from an empty space and is, therefore, considered a normal wire. A short is created when the stranded wire overlaps the blockage with net name A.



Scenario 4: Blockage is on layer `Metal2` with net name A. In addition, a wire with net name A exists. Create a stranded wire starting from an empty space and overlapping with another wire with the same net name A.

1. Choose *Create – Wiring – Stranded Wire*.
2. Specify the net name B in the *Net Name* field.
3. Move the stranded wire towards the direction in which another wire with the same net name appears.

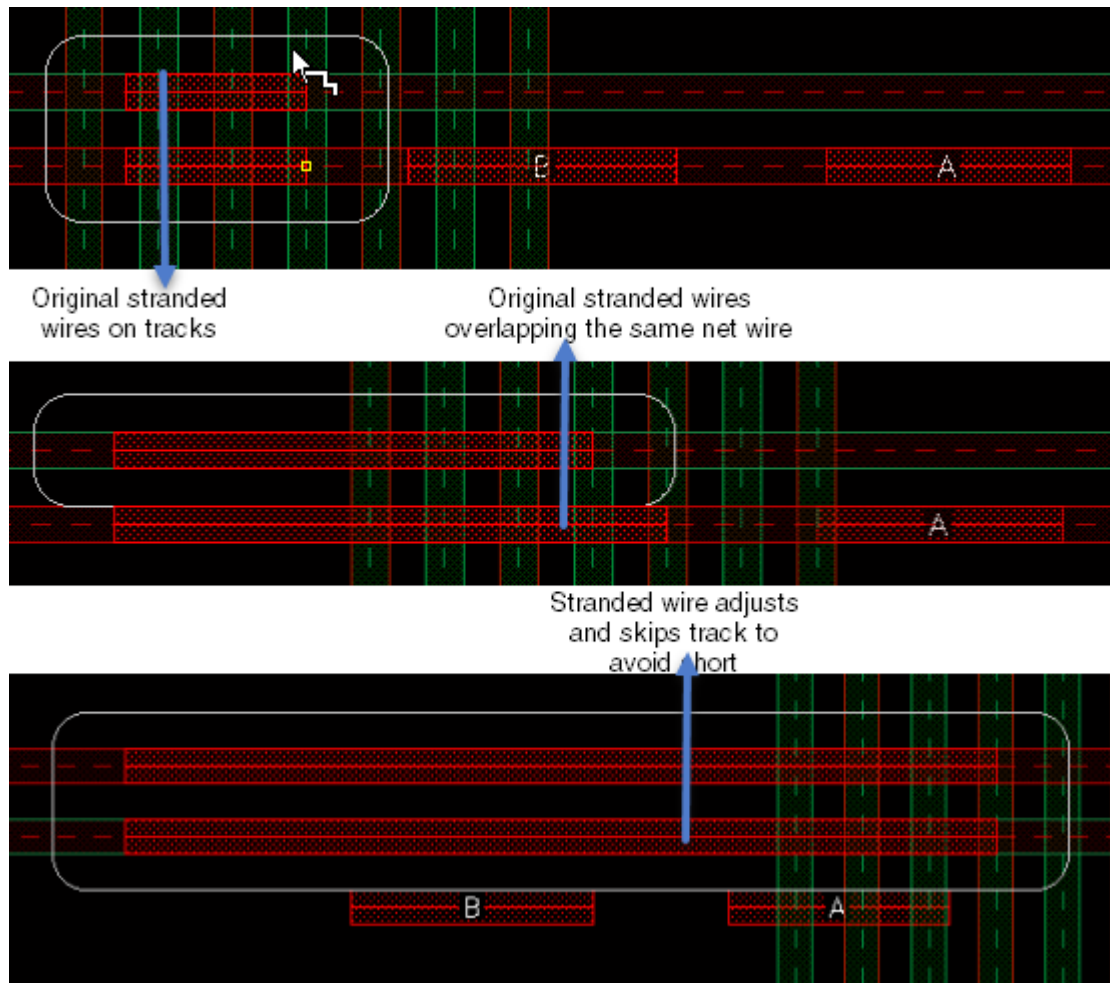
The stranded wire overlaps the existing wire with the same net name.

4. Continue to move the stranded wire in the direction in which a blockage with net name A appears.

Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

Now, the stranded wire skips the track on which the blockage with net name **A** appears and accordingly adjusts to the nearest track that does not have any blockage.



Scenario 5: Blockage with net name **A** exists on layer `Meta13`. Create a stranded wire starting from an empty space on layer `Meta12` and then change the direction by inserting vias.

1. Choose *Create – Wiring – Stranded Wire*.
2. Specify the net name **B** in the *Net Name* field.
3. Choose the *Via Up* or *Via Down* option from the *Create Stranded Wire* context-sensitive menu to insert a via on the stranded wire.

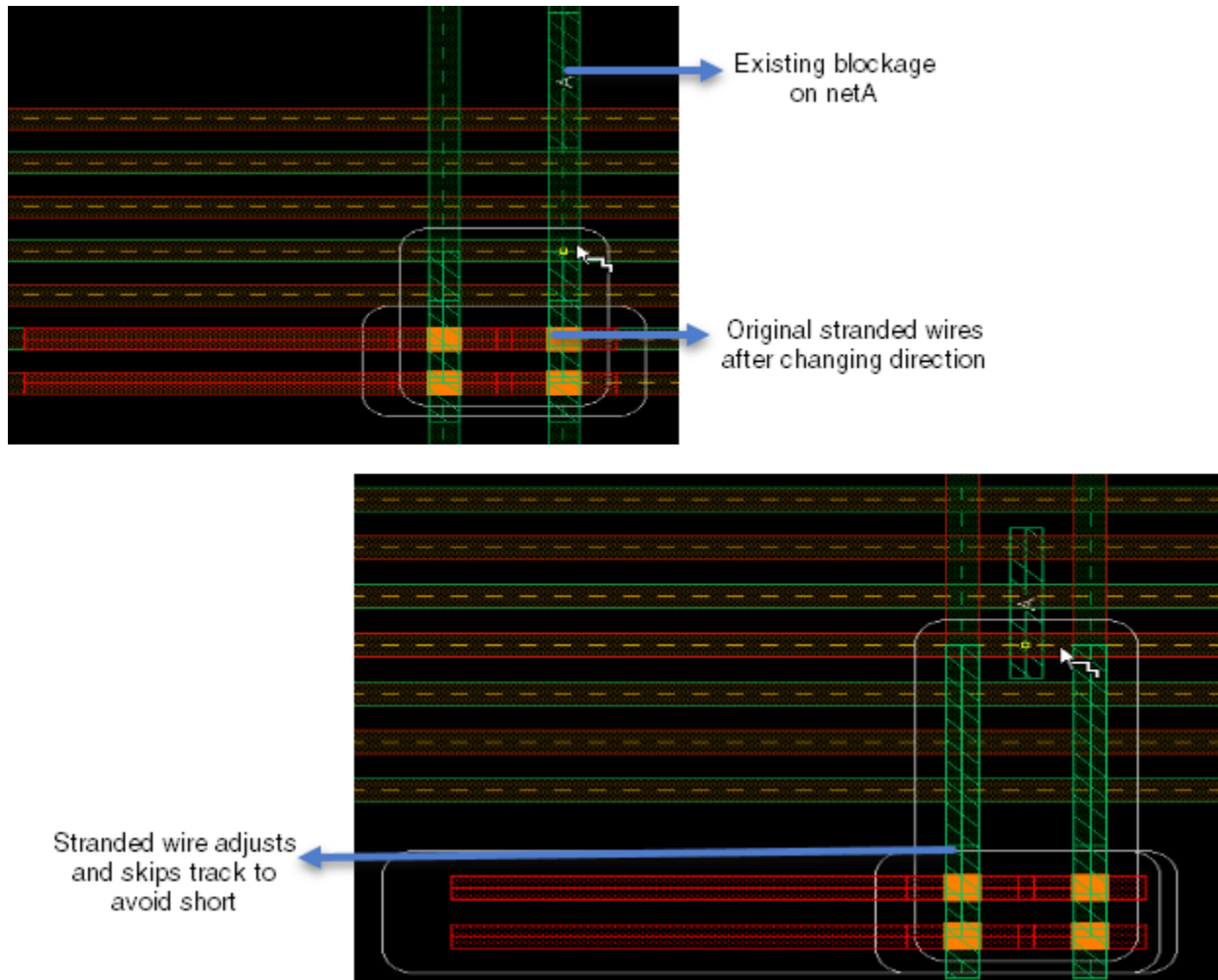
When the via is inserted, the direction of the stranded wire is changed.

4. Move the stranded wire in the direction in which `Meta13` blockage with net name **A** appears.

Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

In this case, the stranded wire skips the track on which `Meta13` blockage appears and accordingly adjusts to the nearest track that does not have any blockage.



Scenario 6: Blockage with net name `A` exists on `Meta13` layer. Increase the space between the stranded wires.

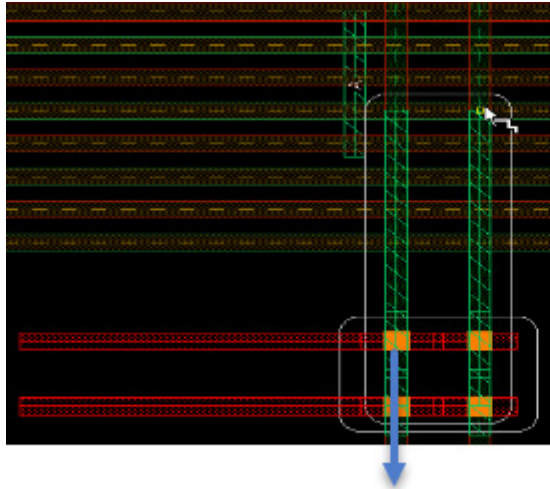
1. Choose *Create – Wiring – Stranded Wire*.
2. Specify the net name `B` in the *Net Name* field.
3. Choose the *Via Up* or *Via Down* option from the *Create Stranded Wire* context-sensitive menu to insert a via on the stranded wire.

When the via is inserted, the direction of the stranded wire is changed.

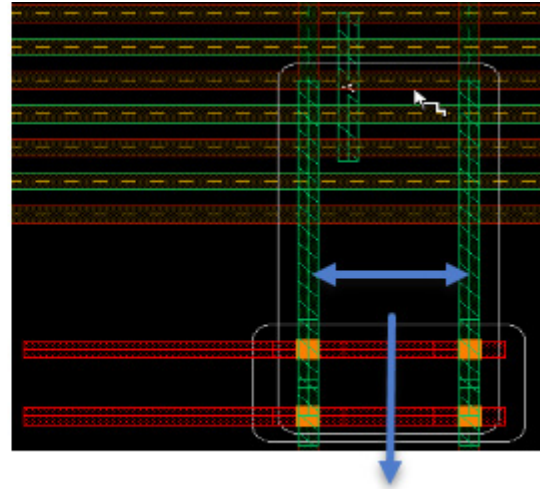
4. Move the stranded wire in the direction in which `Meta13` blockage with net name `A` appears.

5. Use the `Ctrl+3` bindkey to increase the spacing between the stranded wires.

The increased spacing between the stranded wires skips the track that has blockage with net name A and accordingly adjusts to the next nearest track that does not have any blockage.



Original stranded wires on tracks



Increased spacing between strands

Scenario 7: Two blockages exist both having net name A, one on `Meta12` and the other one on `Meta13`. The stranded wire is created from an empty space on layer `Meta12` and changes the direction by inserting vias.

1. Choose *Create – Wiring – Stranded Wire*.
2. Specify the net name B in the *Net Name* field.
3. Choose the *Via Up* or *Via Down* option from the *Create Stranded Wire* context-sensitive menu to insert a via on the stranded wire.

When the via is inserted, the direction of the stranded wire is changed.

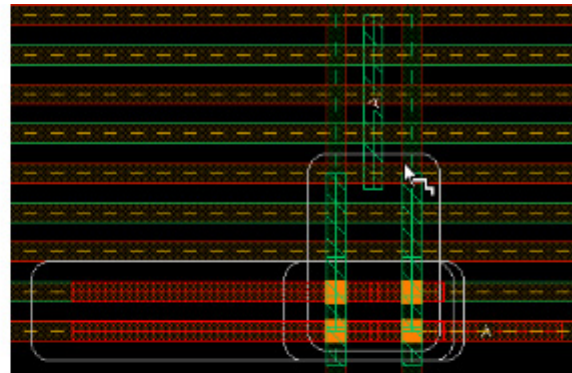
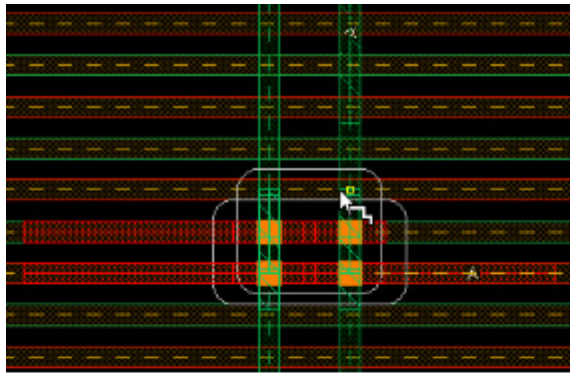
4. Move the stranded wire in the direction in which `Meta13` blockage with net name A appears.

The stranded wire skips the track on which `Meta13` blockage with net name A appears and accordingly adjusts to the nearest track that does not have any blockage. While avoiding the

Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

blockage on Metal3, the stranded wire creates a short with the blockage on Metal2 with net name A.



Short created with Metal2 blockage while trying to avoid Metal3 blockage

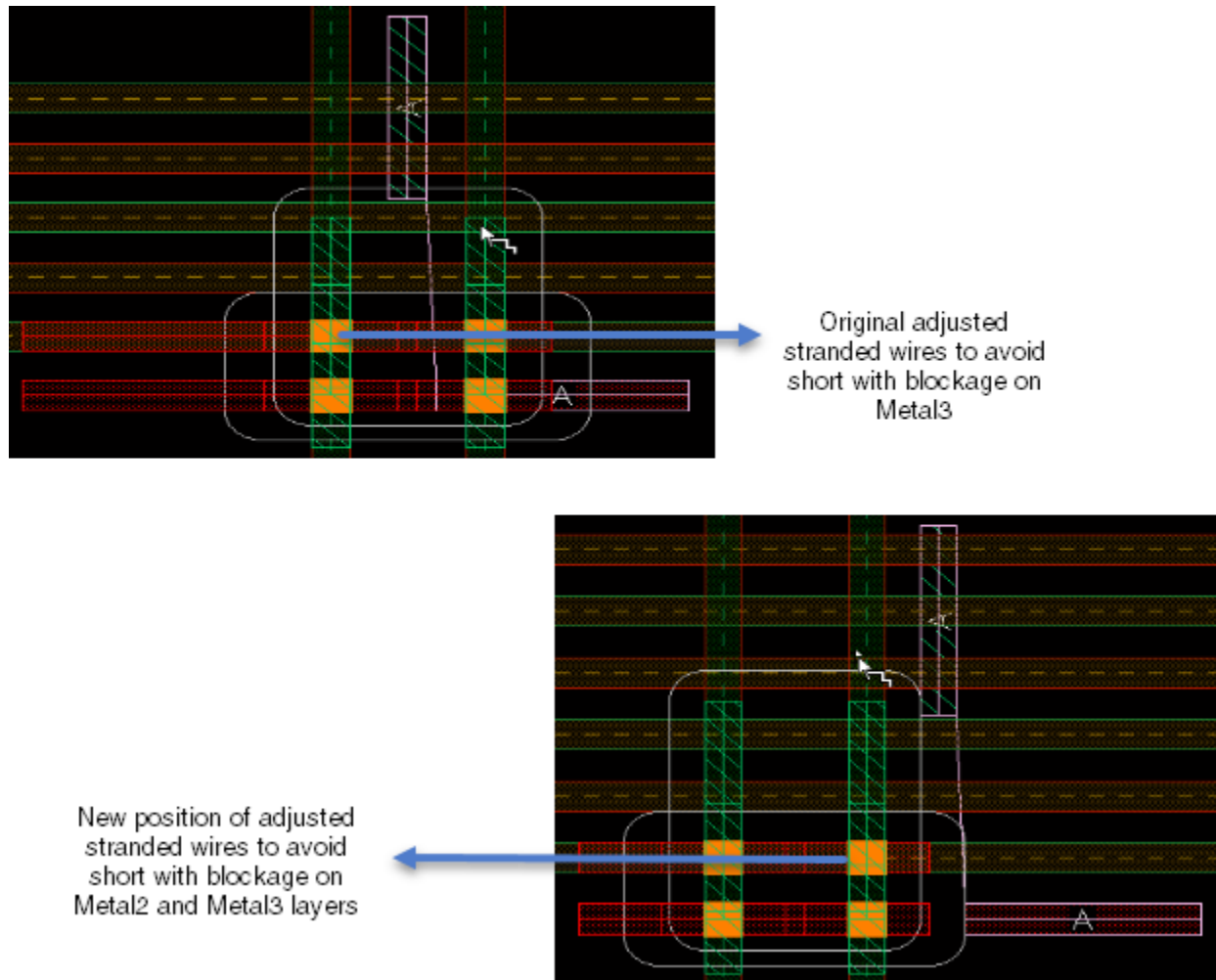


To avoid the short between the stranded wire and the blockage on Metal2 with net name A, slide the stranded wire to the right or left along with the vias. To slide the current section of

Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

the stranded wire, hold the `Ctrl+Shift` keys and move the mouse pointer to the right or left of the current position.



The stranded routing skips tracks to avoid creating a short with an existing blockage on colored WSPs. For more information, see [Avoiding Shorts with Existing Blockages on Colored WSPs](#).

Filtering Objects by Net Name

To start a stranded wire from a desired net without any difficulty, the Stranded Wire command has been enhanced to provide the capability of filtering by net name and by active layer-purpose pair.

To filter objects by net name, perform the following steps:

Virtuoso Interactive and Assisted Routing User Guide

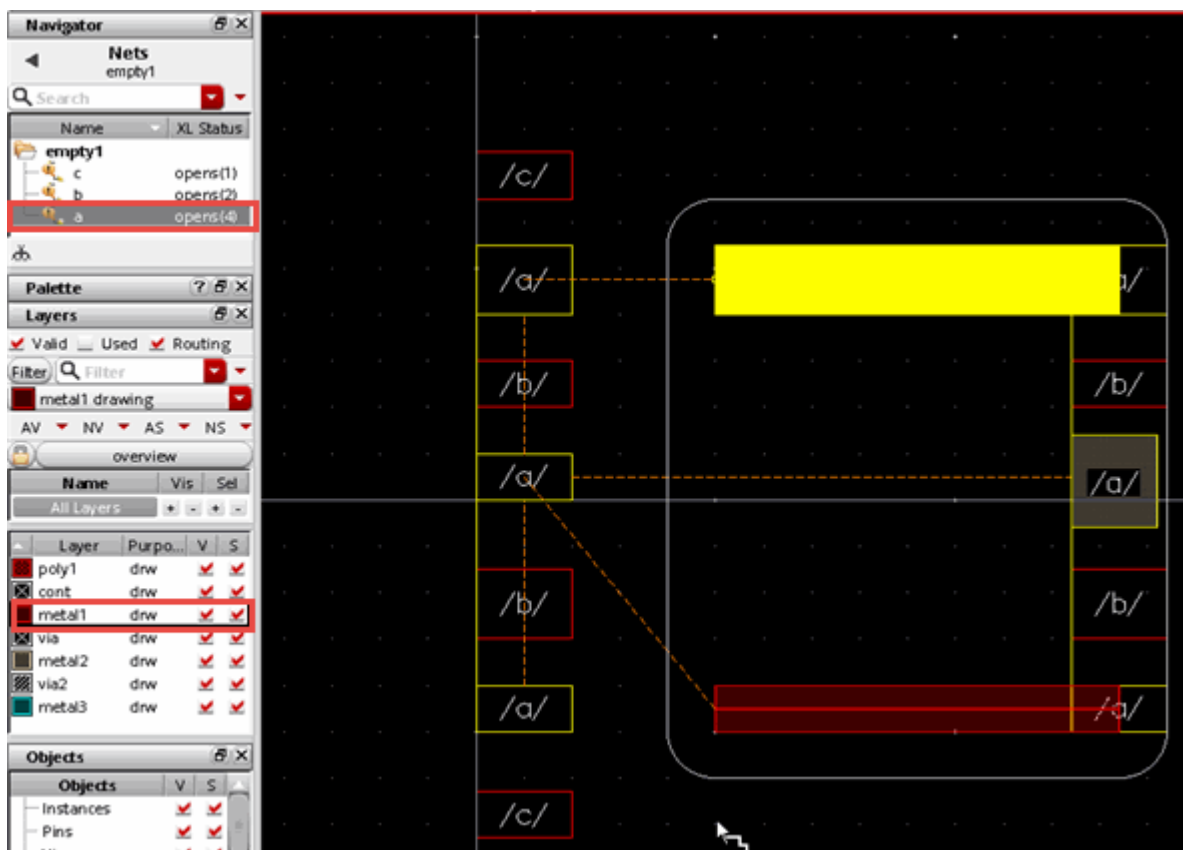
Working with Stranded Wires

1. From the Navigator Assistant, select the net from which you want to start the stranded wire.
2. Choose *Create – Wire – Stranded Wire*. If a net is selected in the Navigator Assistant, it is automatically synchronized and imported to the *Net Name* field of the Create Stranded Wire form.

Note: Ensure that a net is selected in the Navigator Assistant before the *Create Stranded Wire* command is started.

3. Drag to draw a bounding box around the pins for creating a stranded wire. If the net name appears in the Create Stranded Wire form, the stranded wire starts from the pins connected to the selected net. The other pins that are not connected to the selected net are filtered.

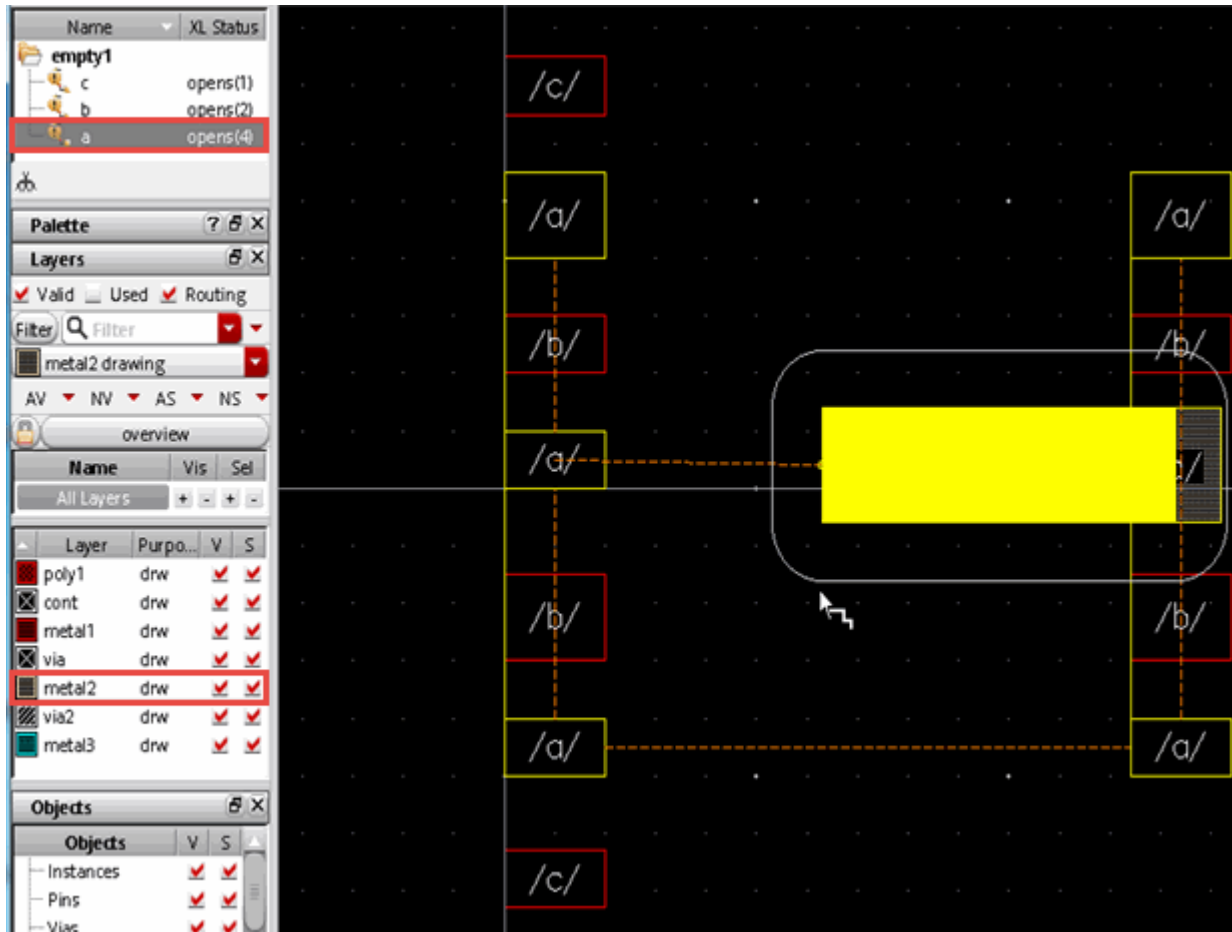
The following figure shows an example where net *a* is selected in the Navigator Assistant and *metal1 drw* is the active layer-purpose pair. In this case, the stranded wire starts from the pins on net *a* of layer *metal1*.



Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

Here is another example where net `a` is selected in the Navigator Assistant and `metal2 drw` is the active layer-purpose pair. In this case, the stranded wire starts from the pins on net `a` of `metal2` layer.



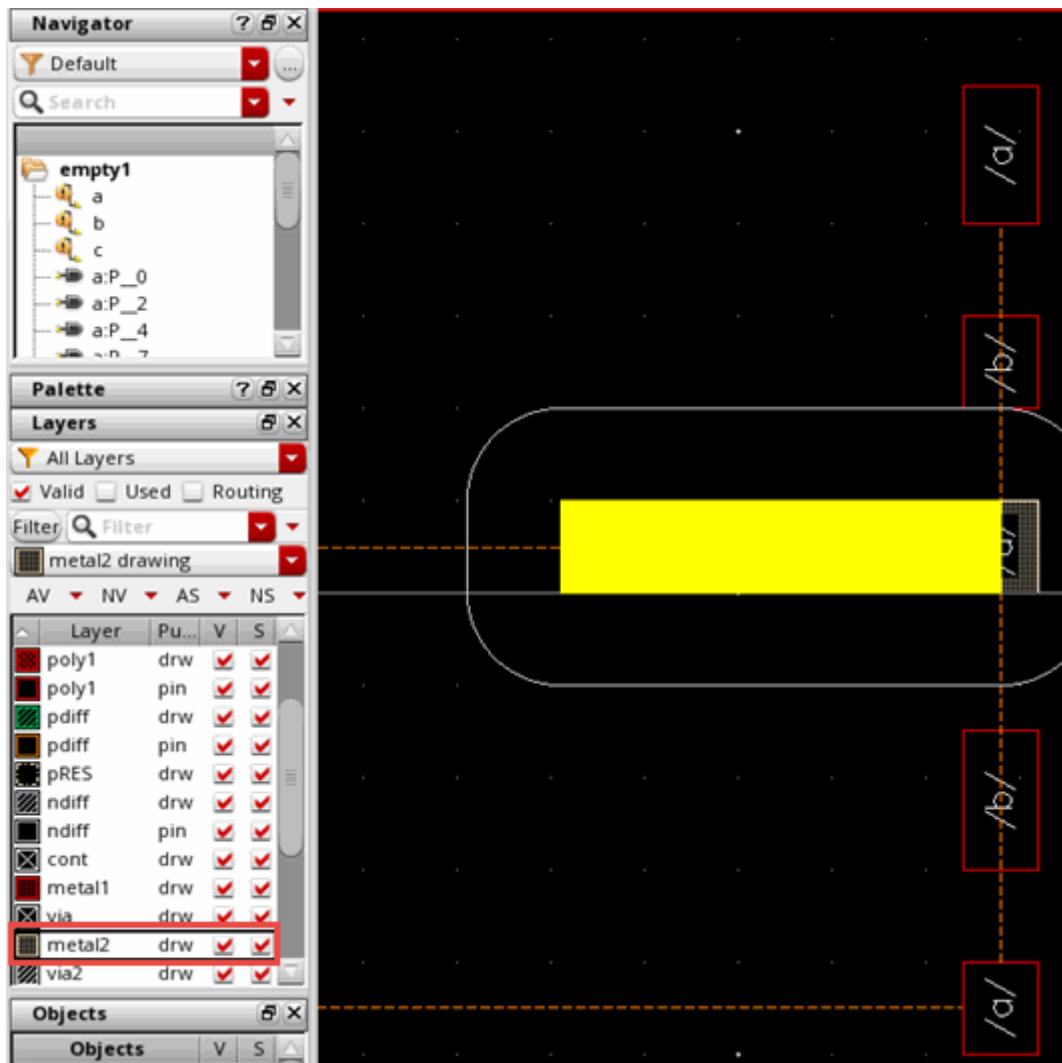
4. To filter objects by the active layer-purpose pair in the palette, select a layer-purpose pair in the palette. The selected object that has the same layer-purpose pair as the active layer-purpose pair in the palette has a higher priority.

The following figure shows an example where `metal2 drw` is selected as the active layer-purpose pair, and a net is not selected in the Navigator Assistant. In this case, the

Virtuoso Interactive and Assisted Routing User Guide

Working with Stranded Wires

stranded wire starts from the pin that is on `metal2 drw`. Also, `metal2 drw` is the active layer-purpose in the palette.



Objects filtered by net names are at a higher priority than by active layer-purpose pairs.

Using Assisted Routing Commands

This chapter describes the assisted routing commands.

- Point to Point Routing
 - Setting Point to Point Wires to Match Pins
- Supporting Track-based Routing
- Finishing a Trunk

Point to Point Routing

Point-to-point routing is available in Virtuoso Layout Suite XL and higher tiers.

The *Point to Point* command is an interactive routing command that automatically generates routing between two digitized points in a given cellview. The two specific locations can be the two click points. The two points can overlap a pin, instPin, pathSeg, path, existing wires or vias, or be in empty space. The *Point to Point* command uses the second digitized point to start the consecutive *Point to Point* command. A point-to-point route completes only when you double-click or press `Enter`.

Point-to-point routing is an acceleration of interactive wire editing when creating short local connections, while also allowing you to maintain tight control of the routing. Long connections can be broken down into several shorter point-to-point connections or can be completed automatically using the *Automatic Routing* command to create routing that is optimal in a global perspective.

Note: By default, Point to Point uses all available layers in the *Wire* default constraint group. You can limit the layer range through the *Wire Assistant*.

The *Point to Point* routing command generates only orthogonal wires, which includes pathSegs and vias created in routes. Diagonal routing is not supported.

Note: You can use the point-to-point router to make a connection to geometric data (paths or pathSegs not in routes).

Important

The *Display Levels* settings *Start* and *Stop* control the hierarchy depth for point-to-point routing.

To create point-to-point wiring:

1. Launch Layout XL or a higher tier of the layout suite.
2. From the design window, select *Create – Wiring – Point to Point* or click on the icon  on the *Create* toolbar.

You are prompted to click at the first location.

3. Press **F3** to open the Point to Point Form.
4. Specify a net name for the wire if you are starting from a point in open space.

When starting from a pin or shape that has connectivity, the net name of the pin or shape is automatically picked.

For more information about selecting the starting and ending targets, see Finishing a Trunk.

You can use the options on the context-sensitive menu to change the width of the wire to match the width of the starting pin. Right-click in the cellview to display the Point to Point Shortcut Menu Commands. See Setting Point to Point Wires to Match Pins.

You can also configure *Point to Point* settings using options in Point to Point and Automatic Section sections in the Wire Assistant.

5. If required, you can override the default width by specifying a new value in the *Width* field.
6. Use the *Top* and *Bottom* cyclic fields in the *Routing* column to limit the routing layers to a subset of the full range of metal and poly layers.

Note: Taper layers can be excluded or included in the routable layer range. However, if a layer, such as poly, is excluded and a poly pin is set up in such a way that the router cannot ascend to metal1 immediately at the pin through a metal1-poly via, the router will fail to connect to the poly pin. For more information about taper layers, see virtuosoDefaultTaper.

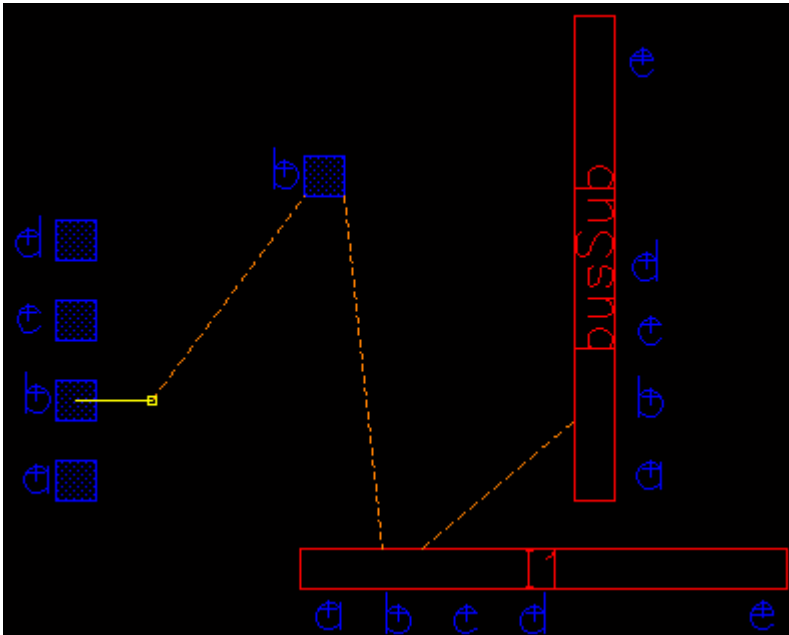
7. Set up the valid and taper vias using the *Vias* buttons.
8. Select the *Connection Type*. *Exact Location* creates routing between the digitized points. *Shortest Connection* creates routing as the shortest connection between the digitized points using any pre-existing wires.

Virtuoso Interactive and Assisted Routing User Guide

Using Assisted Routing Commands

9. In the cellview, click on a pin or a routing element such as a pathSeg, path, or via. Optionally, you can click in an empty area of the design. You are prompted to click at the next location.

A flightline is drawn from the clicked point to the closest target on the specified net.



10. Click on subsequent points.

Routing is generated between consecutive digitized points.

After you digitize the second or more points:

- ☐ The *Net Name* field in the Point to Point Form becomes uneditable.
- ☐ You can edit *Routing Layers*. If editing the layer limit results in making the previous point-to-point route invalid, the *Point to Point* command creates stacked vias between the previous click point and the valid routing layer.

11. To specify the ending location of the wire, double-click or press `Enter`. The ending location can be a pin or a routing element such as a pathSeg, path, or via. Optionally, you can click in an empty area of the design. A highlighted line is displayed to show the connection between the first clicked point and the current pointer location.

This completes a point-to-point route. You are prompted to specify a new starting location for the *Point to Point* command.

Note: Ensure that the start and end points of the route are within the P&R boundary because routing is not allowed outside the P&R boundary.

The *Point to Point* command automatically stops for the route if:

- ❑ While routing an incomplete net, you tap a shape (pin/route) that completes the net.
- ❑ While routing a complete net, you tap a shape on the net.
- ❑ No net is being routed (starting from empty space or unassigned shapes) and you tap a shape on a complete net.

Routing paths are found in empty areas around objects with valid spacing rules. No pushing is performed to make room for routing.

12. To end the command, press Esc or click *Cancel* in the Point to Point Form.

Note: The point-to-point router does not follow `minEndOfLine` spacing in cases where the wire is floating and consequently forms an incomplete connection, as the data is still in the midst of changes.

If the point-to-point router cannot complete a route and the *Interactive Edit* mode is set to *Post-edit* or *Notify* in the *DRD Options* form, the router forces a route creation with violations. The router also issues a warning message to indicate that there were violations and that you can run *Verify – Design* to view the markers for the violations. This enables you to analyze the problem areas where the router could not produce a clean route.

If the point-to-point router cannot complete a route and DRD *Interactive Edit* mode is set to *Enforce*, a warning message is issued to indicate if the router is stuck at the start or end point or pin or if there is insufficient space to complete routing.

Important

The point-to-point router does not support track patterns and routing grids. The routing is performed on the manufacturing grid.

There is no color assigned to the routing created by point-to-point router even if you have WSPs that are defined with color information. In order to have color from WSPs, you need to enable coloring before running the point-to-point routing.

Setting Point to Point Wires to Match Pins

You can use the options on the *Use Width* command on the Point to Point Shortcut Menu Commands to set point-to-point wire widths to match pins. The selected match pin width applies to all applicable routing layers of the current effective constraint group, where it does not violate the corresponding `minWidth` and `maxWidth` constraints. The width change is not implemented if it introduces a clearance violation.

The point-to-point pin matching option remains effective throughout the entire layout session. Point-to-point requires two clicked points. When the pin matching option is on, the width of the first pin selected is used. The object selected must be a pin, otherwise the width constraints in the current effective constraint group are applied.

Supporting Track-based Routing

Using the assisted routing commands, such as point to point, you can now perform routing on track grids. To enable the assisted routing commands for track-based routing, the following conditions must be satisfied:

- Before starting a assisted routing command, select the *Track Pattern* option from the *Snap Wire To* drop-down list in the Wire Assistant.
- Each routing layer that has a track pattern must have a defined routing direction, which is either horizontal or vertical.
- Pin shapes must be on a track grid.

Finishing a Trunk

The *Finish Trunk* command on the *Create Bus* and *Create Wire* context-sensitive menu automatically converts all the pathSegs except for the first and the last pathSeg into trunks, regardless of whether they are connected to a pin or not. The first and the last pathSeg is considered as a regular pathSeg if it is connected to a pin at one end and to a trunk at the other end. Else, it is converted to a trunk. The default bindkey for running the *Finish Trunk* command is 2. For more information, see [Using the Finish trunk Command](#) in the Virtuoso Space-based Router User Guide.

Virtuoso Interactive and Assisted Routing User Guide

Using Assisted Routing Commands

Editing Routed Connections

This chapter describes the following topics.

- [Stretching Wires and Vias](#)
- [Moving Wires](#)
- [Copying Wires](#)
- [Re-routing a Wire](#)
- [Reshaping Wires](#)
- [Removing Loops From Wires](#)
- [Merging Wires](#)
- [Self-intersecting Objects](#)
- [Deleting Wires](#)
- [Copy Route Command](#)
- [Setting Copy Route Options](#)
- [Copying Pin-to-Pin Routes](#)
- [Copying Pin-to-Wire Routes](#)
- [Handling Wire Copies for EM Analysis](#)

Stretching Wires and Vias

If you stretch segments or vias, DRD-Enforce with Push works on prerouted paths and segments. Prerouted paths everywhere in the design, not just those pushed, are split at t-junctions and vias to conform to the routing connectivity model. New paths introduced by split are written as segments. As a result, some parts of prerouted paths are converted to segments.

If you stretch other object types such as paths, rectangles, polygons, MPPs, Pcells, etc., DRD-Enforce with Push does not work.

- [Variables that Control Stretch](#)
- [Controlling the Angle of Wires](#)
- [Using the Stretch Command](#)
- [Preserving Paths](#)
- [Stretching Wires Connected to Pins](#)
- [Stretching Wires Connected to Vias](#)
- [Stretching Wires Across Tracks](#)

Variables that Control Stretch

- Move Connected Instances and Wires (Layout EXL and higher tiers)

This option lets you maintain connectivity of the instances and pathSegs to the selected object while moving or stretching the instances or pathSegs. When this option is selected, the *Keep Wires Connected to* and *Lock Unselected Vias* options are disabled.

- Keep Wires Connected to option

The *Keep Connected* option on the *Stretch* form will allow wires that are stretched to remain connected. See [Metal Density Options Form](#).

- Angles of segments

Often determines when an extra segment is needed. See [Controlling the Angle of Wires](#).

- Selection of objects

Selecting one vertex in a segment leaves the other vertex alone. Selecting both vertices in a segment causes the segment to move.

The length of attached segments may change to keep them connected or additional segments may be added to keep them connected.

Selecting a via may cause the addition of a segment.

See [Wire Objects](#) for more information.

- Constraint-aware editing

The *Constraint-aware editing* option on the *Configure* tab of the Connectivity form allows you to perform the initial constraint checks. This is recommended for the interactive routing commands to improve the runtime and also the use model.

Controlling the Angle of Wires

To control the angle of wires during stretch, turn *Lock Angles* on to preserve the angle of existing objects during stretch. See [Metal Density Options Form](#) for more information. Set the *Snap Mode* to *Orthogonal* or *Diagonal*, to control the permitted angles of wire objects being added.

Using the Stretch Command

The *Stretch* command can be used to stretch diagonal edges, intersecting wires, and pathSegs or paths by changing the vertices XY locations. When an isolated fully selected via, pathSeg, or path is stretched, the object is moved to a new location similar to using the *Move* command.

Note: PathSegs are supported by advanced features. Advanced features such as *Push* are activated when using pathSegs. Path objects are not supported by advanced features.

To stretch a wire, follow these steps:

1. Choose *Edit – Stretch*.

The layout window prompts you to select a figure to be stretched. Press F3 to open the *Stretch* form.

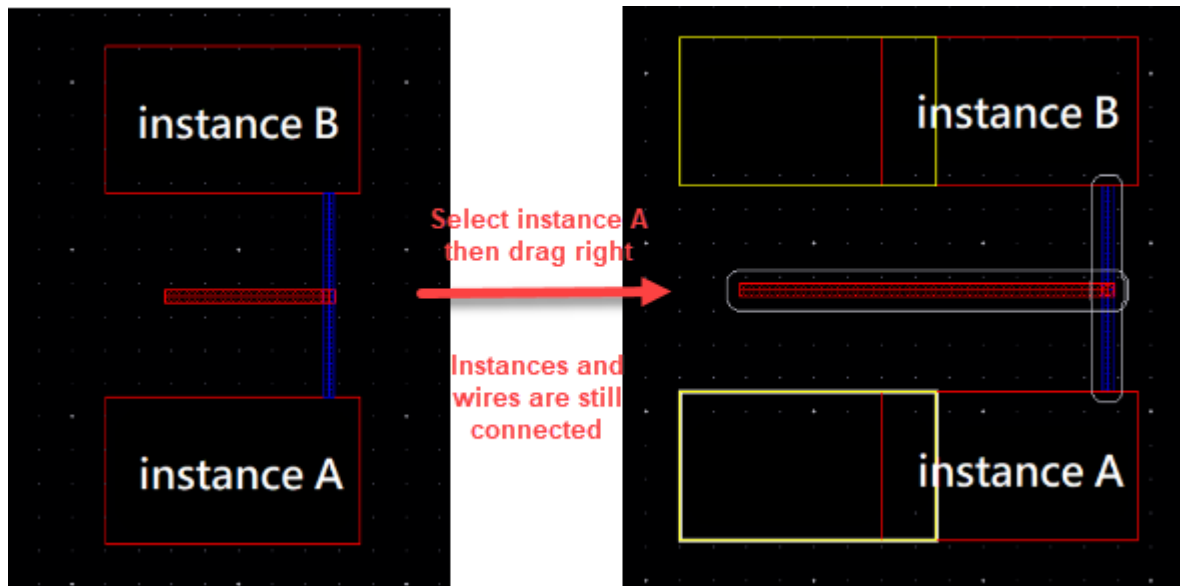
2. Select the desired *Snap Mode*.

Snap Mode will enforce restrictions on the movement of the pointer. For example, if one end of a diagonal segment in a wire is stretched when the snap mode is horizontal, that end will move only horizontally.

3. Select *Move Connected Instances and Wires*. It helps to maintain connectivity of the instances and pathSegs to the selected object while moving or stretching the instances or pathSegs. To maintain the connection from selected objects:

- ☐ The selected objects move and stretch in both x and y directions.

- ❑ The connected objects stretch or move to keep them connected to the selected objects.



4. (Optional) Select *Keep Wires Connected to*.

The *Keep Wires Connected* is selected to maintain the pathSeg-to-pathSeg and pathSeg-to-via connections during stretch. You can select one of the following options.

- ❑ **Spine wires only** maintains connections to spine wires but not to connecting vias when stretching a wire. This option is selected by default.
- ❑ **All wires and vias** maintains pathSeg-to-pathSeg and pathSeg-to-via connections during stretch. When this option is selected, the *Lock Unselected Vias* option is enabled. Selecting the *Lock Unselected Vias* option, locks the unselected vias and does not allow you to change the position of the via.
- ❑ **No wires and vias** does not maintain pathSeg-to-pathSeg and pathSeg-to-via connections during stretch.

5. (Optional) Turn *Lock Angles* on to preserve the current angles of pathSegs and path elements.

Turn *Lock Angles* off to allow the angles to change. Path elements may be oriented to any angle, but pathSegs can only be orthogonal or diagonal.

6. Click or area select the wire or portion of wire you want to stretch.

For more information about selection, see [Wire Objects](#).

7. Move the pointer and click to place the wire element.

The *Stretch* command does not keep overlapping wires connected if they are on different nets. In VLS L, overlapping wires on different nets are kept connected when stretched if their endpoints meet precisely.

If DRD is enabled, the placement of stretch wires is verified that the resulting geometry will follow the technology rules. This applies to via stacking as well. If DRD is not enabled, the *Stretch* command will allow vias (as well as other objects) to be moved to any location.

You can stretch a bus or multiple wires whose bit spacing is less than minSpacing, but minSpacing markers are generated while dragging the pointer, even if DRD is not enabled.

In the Layout L editor, logical connectivity (nets) is not taken into account when stretching wires. When DRD is on, DRD can use net information to flag or avoid shorts.

Preserving Paths

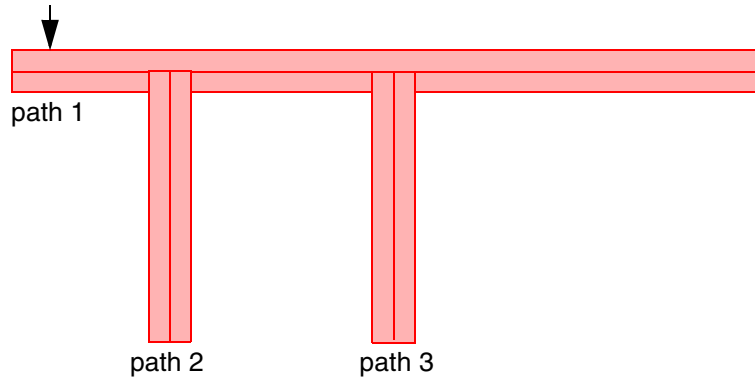
When wires are created using paths, the stretch command will preserve the path objects and not split the paths into separate paths when ever possible. However, there are cases when it

Virtuoso Interactive and Assisted Routing User Guide

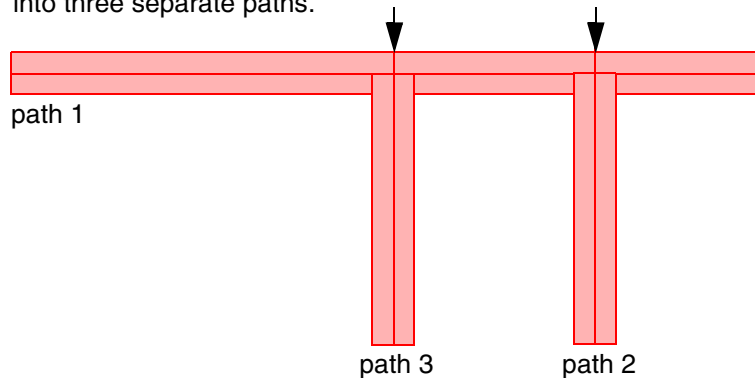
Editing Routed Connections

is necessary to split path objects. The following examples show situations where paths will be split due to stretching of the paths.

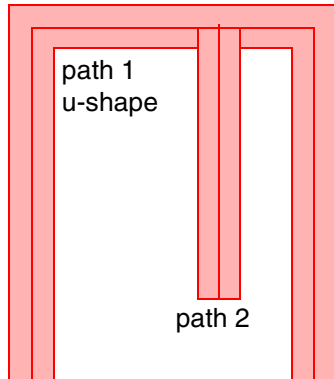
One continuous path.



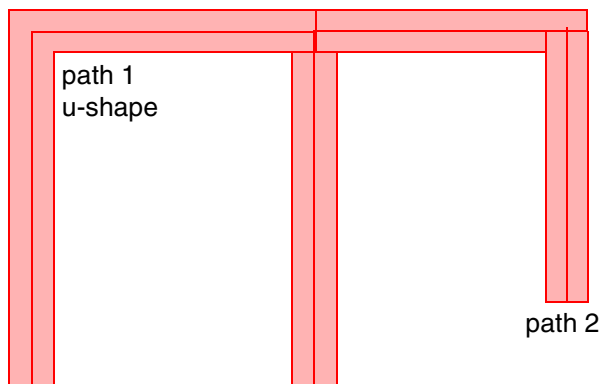
When path 2 is stretched to the right, the top path (path 1) is split into three separate paths.



Path 1 is one continuous u-shaped path.



When path 2 is stretched outside of the u-shape, path 1 is split into separate paths.



Stretching Wires Connected to Pins

When a wire connects to a pin, the end of the wire is considered to be fixed. To disconnect a wire from a pin, turn *Stretch – Keep Connected* off. This will release the end of the wire and allow you to disconnect from the pin while stretching the wire.

When a segment that is attached to a non-wire object such as a pin, and the pin is not also being stretched, a segment is added from the end of the stretched segment back to the original connection point in the pin only if it is necessary to maintain connectivity.

Stretching Wires Connected to Vias

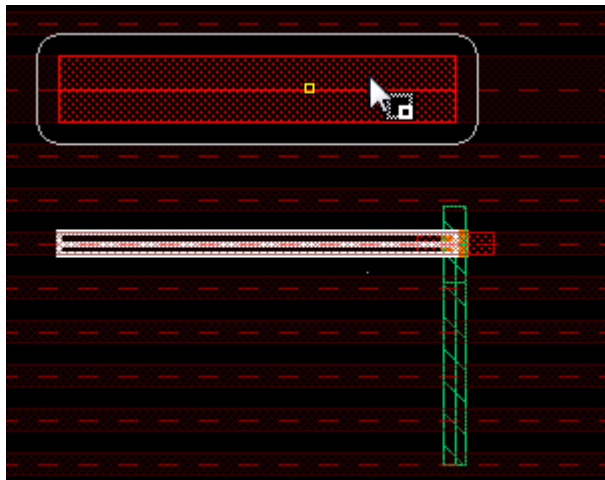
Wires will remain connected to vias when the *Stretch – Keep Connected* option is on, whether it is a single via or a stack of vias. To move individual vias in a stack you must use the *Move* command.

When stretching a via attached to a pathSeg, the pathSeg is also stretched and the endStyle of the pathSeg at the overlap with via is recomputed. This may result in the end styles of pathSegs being changed.

Stretching Wires Across Tracks

The *Stretch* command has been enhanced to support width-based track patterns as well as width spacing patterns.

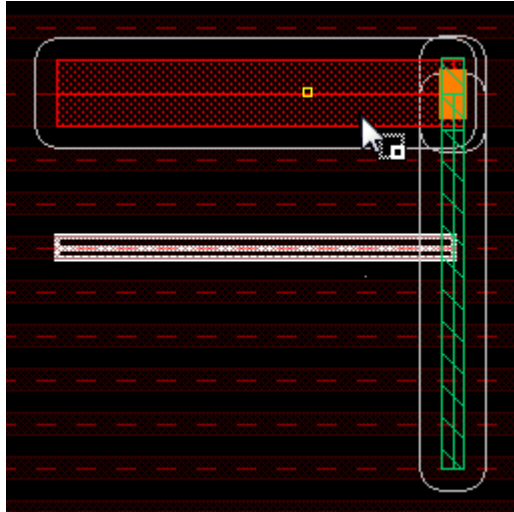
When you stretch a wire across tracks, the width of the wire is adjusted to the width of the track pattern to which it is stretched. For example, the global region has `stdCell` pattern, which has different width on different tracks. The figure below shows that when a wire is stretched, the width of the wire is adjusted to the width of the track pattern to which it is stretched.



In this case, the wire is stretched without maintaining the pathSeg-to-pathSeg and pathSeg-to-via connections. This is because the *No wires and vias* option is selected from the *Keep Wires Connected To* drop-down list in the Stretch form. The Stretch form is displayed when you start the *Stretch* command and press `F3`.

You can also stretch a wire by maintaining its pathSeg-to-pathSeg and pathSeg-to-via connections. To stretch a wire while maintaining its connectivity, select the *All wires and vias* option from the *Keep Wires Connected To* drop-down list in the Stretch form.

The following figure shows that when a wire is stretched while maintaining its connectivity, the width of the wire is adjusted to the width of the track pattern to which it is stretched. Additionally, the connectivity of the wire, such as the via placed on a wire, is also accordingly adjusted.



Moving Wires

Fully selected wires or wire elements can be moved by the *Move* command. Wires can also be rotated, reflected, or changed to a different layer by using *Move* form. For more information, see [Move Form](#).

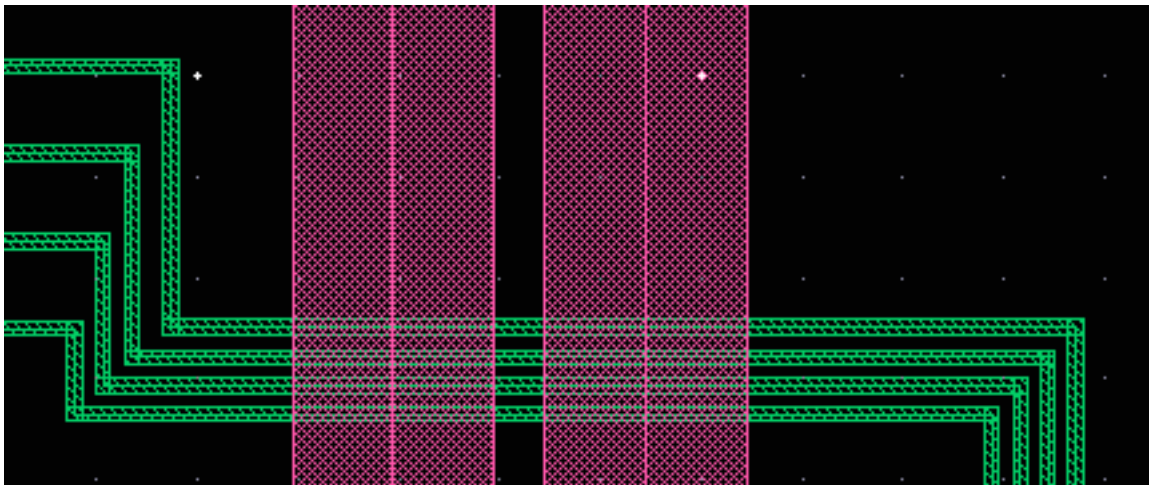
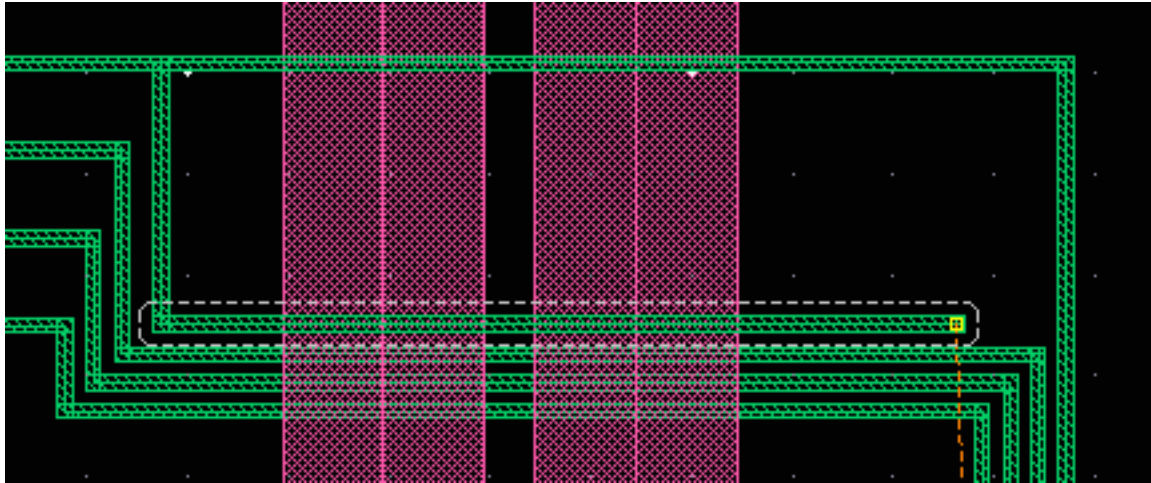
Copying Wires

Fully selected wires or wire elements can be copied by the *Copy* command. Wires can also be rotated, reflected, or changed to a different layer by using *Copy* form.

Currently the net information is discarded when copying wires.

Re-routing a Wire

A re-route is a type of wire editing where a new wire is started from a point along an existing wire, and eventually reconnects to another portion of the existing wire. The portion of the original wire made redundant by the new wire is removed.



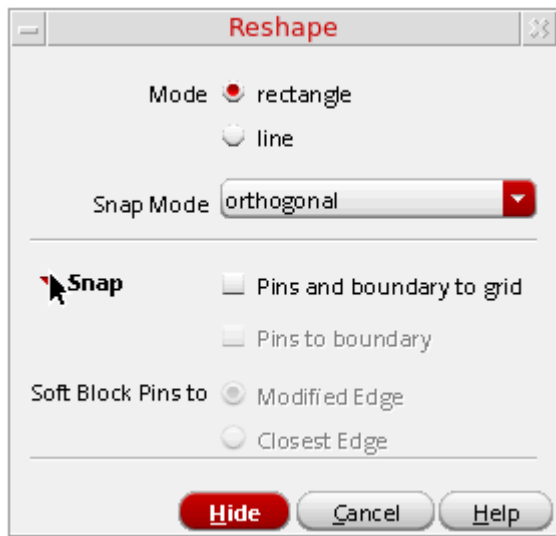
Reshaping Wires

Using the Reshape command, you can reshape the list of selected pathSegs and vias. You can also choose the selected part of a wire to be reshaped.

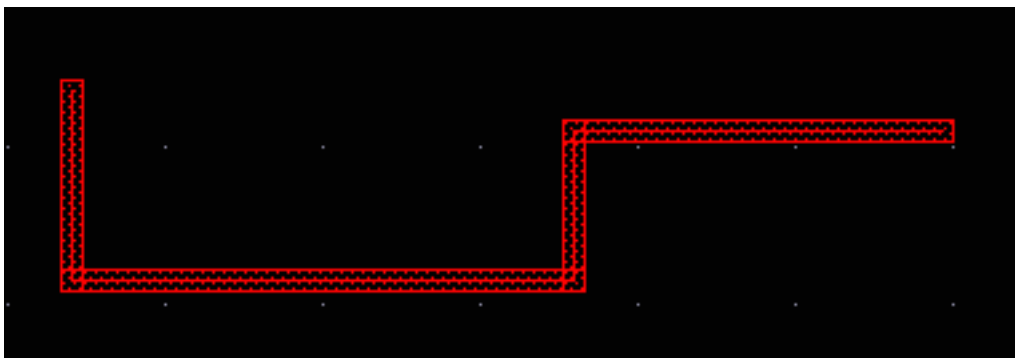
To reshape a wire, follow these steps:

1. Choose the type of *Routing Object Granularity* from the *Options – Selection* form.

2. Select the wire to be reshaped.
3. Select *Edit – Advanced – Reshape*. The Reshape form is displayed. By default, *rectangle* mode is selected.

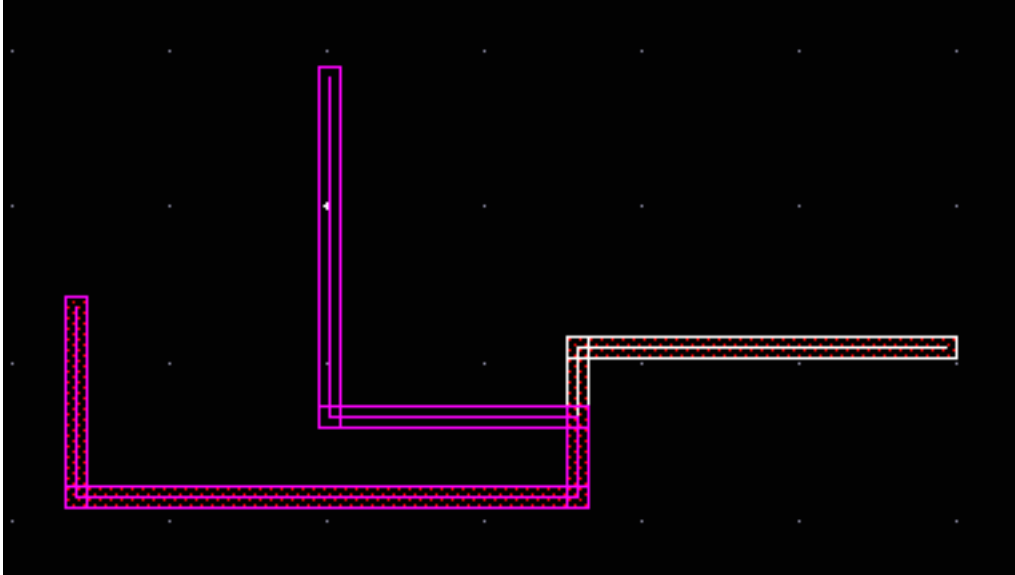


4. To reshape a wire in *line* mode, perform the following steps.
 - a. Click *Mode: line*.
 - b. Click a segment of an existing wire to select the wire.
 - c. The existing wire is automatically tapped for layer information and attributes.
 - d. The initial starting point is the first point of the wire to be reshaped. The initial point is snapped to the begin/end points, center-line of a segment, or to the origin of a via:

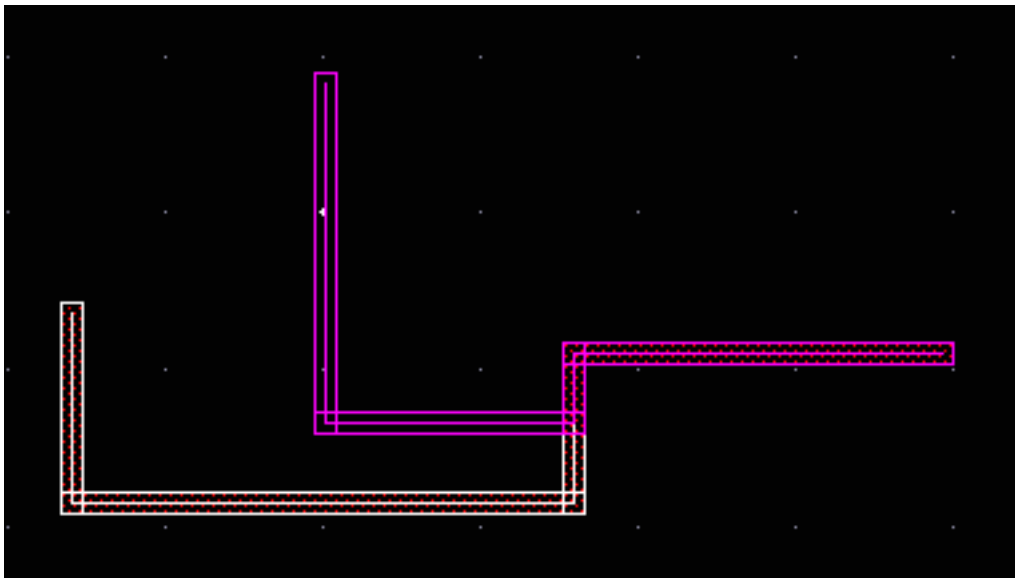


- e. When you select the *Reshape* command, all the other pathSegs or wires in the same route as the selected pathSegs or wire are also selected.
- f. Digitize points where you want to create a wire and double click.

- g.** This will highlight the option of the new wiring.



- h.** Middle click toggles between wire reshape options.

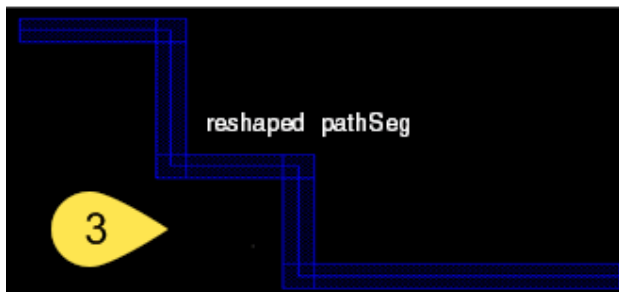
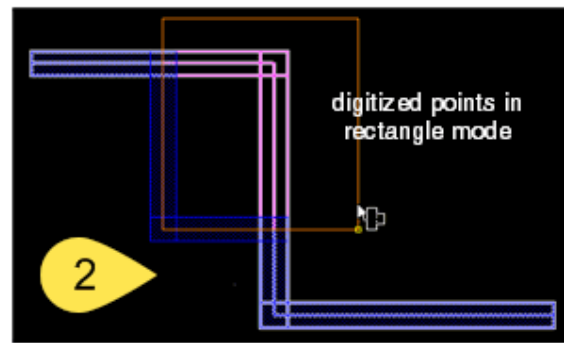
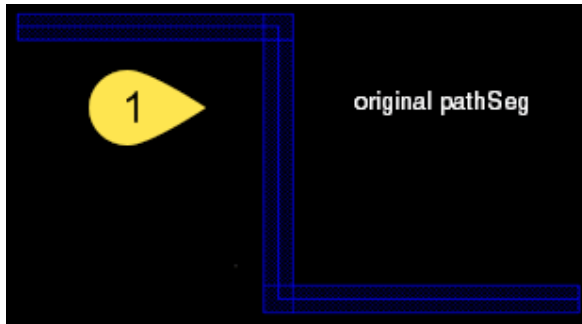


- i.** Press *Return* or double click to finish the desired reshape of the wire.
- 5.** To reshape a pathSeg in the *rectangle* mode, perform the following steps.
- j.** Select the wire to be reshaped.
- k.** Select *Edit – Advanced – Reshape*. The *rectangle* mode is selected by default.

Virtuoso Interactive and Assisted Routing User Guide

Editing Routed Connections

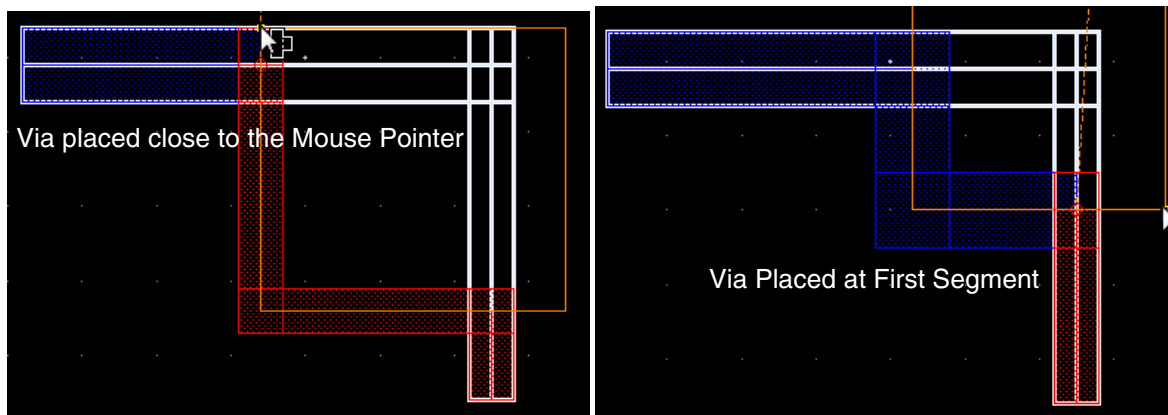
- l.** Click a segment of an existing wire to select the wire.
- m.** Digitize points where you want to create a wire and double click.



- n.** Press *Return* or double click to finish the desired reshape of the wire.

In addition, while reshaping a wire in a rectangle mode, if the two intersection points of selected wires are on different layers then reshape command will keep the connectivity by repositioning the vias in between them. The vias of the reshaped

wire are either placed close to the mouse pointer or at the first segment as shown in the following figure.



Removing Loops From Wires

The option *Allow Loops* is supported in Virtuoso Layout Suite.

By default, loops are removed from wires created as pathSegs. When creating or editing a wire, a preview of the intended wire is shown. Once the intended wire segment attempts to produce a loop, the resulting loop is removed. When editing a wire which contains a loop, the entire wire may be removed due to loop removal.

Note: All *Create* and *Edit* commands remove loops from wires created as single path objects.

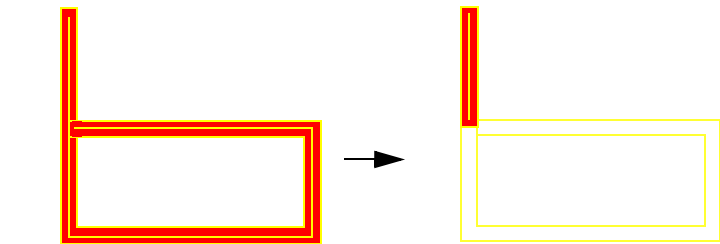
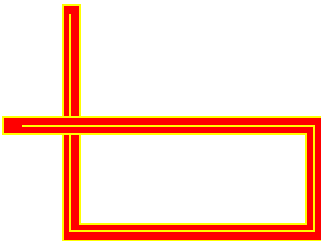
You can allow loops in pathSegs by turning on *Allow Loops* in the *Layout Editor Options* form. The following commands and options support loop removal.

- *Create – Wiring – Wire*
- *Edit – Stretch*

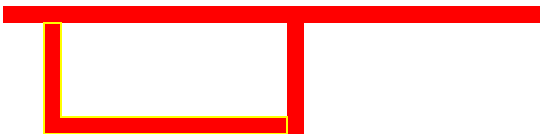
Virtuoso Interactive and Assisted Routing User Guide

Editing Routed Connections

The following graphics show the loop removal when creating wire using pathSegs.

With <i>Allow Loops</i> off, the loop is not allowed.	
If the intended wire segment attempts to produce a loop, when <i>Allow Loops</i> on, the loop is allowed.	

With *Allow Loops* on, loops are allowed in the following circumstances.

Behavior	Loops are allowed
Interconnecting wires: Creating a wire on interconnecting layers	
Branching wires: Reshaping an existing wire that contains a branch. In the case of a branching wire or wires, the loop is not removed because it can not be determined which side of the loop to remove, or removing the loop can disconnect the wire from an intended connection to pins or same net wires.	

Merging Wires

The following types of conditions are supported by the *Merge* command. Only wires on the same layer can be merged. Merging at segment ends which have a via is not allowed.

Collinear Abutment or Overlap of PathSegs

Collinear abutment of pathSegs is when two pathSegs meet end to end in a straight line at the same end point. Collinear overlap of pathSegs is when two pathSegs overlap in a straight line.

When two collinear pathSegs are merged, the elements are converted one continuous pathSeg. The resultant pathSeg will have the same end styles as the respective ends before merging.

Coincident Abutment or Overlap

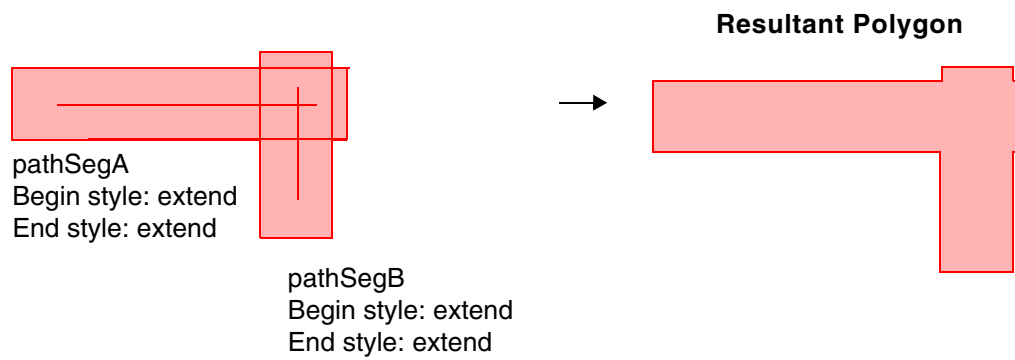
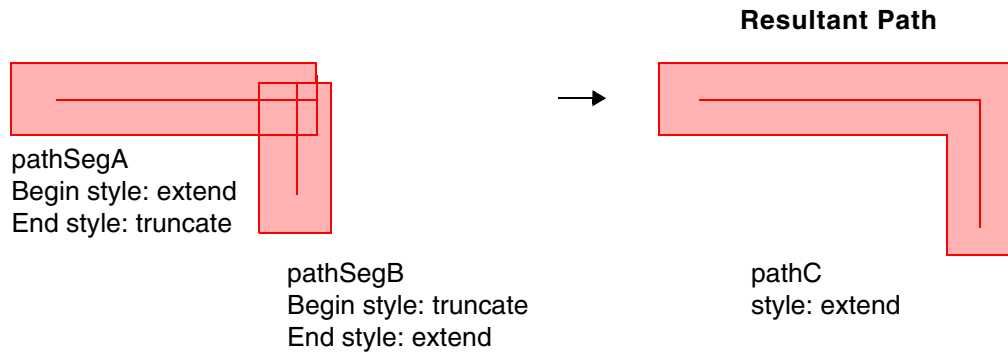
Coincident abutment of pathSegs is when two pathSegs meet at an angle, or in at a T junction. Coincident overlap of pathSegs is when two pathSegs overlap at an angle.

Elements are converted to a path or polygon depending on overlap and end styles of pathSegs.

The following table lists the combinations of end styles possible by merging pathSegs that abut or overlap at an angle. Also stated is the situations in which merging the pathSegs results in a polygon.

End Style of the First PathSeg	End Style of the Second PathSeg				
		Truncate	Variable	Extend	Custom
	Truncate	Truncate	Variable	Variable	Polygon
	Variable	Variable	Variable	Variable	Polygon
	Extend	Variable	Variable	Extend	Polygon
	Custom	Polygon	Polygon	Polygon	Polygon

Examples of Coincident Overlap



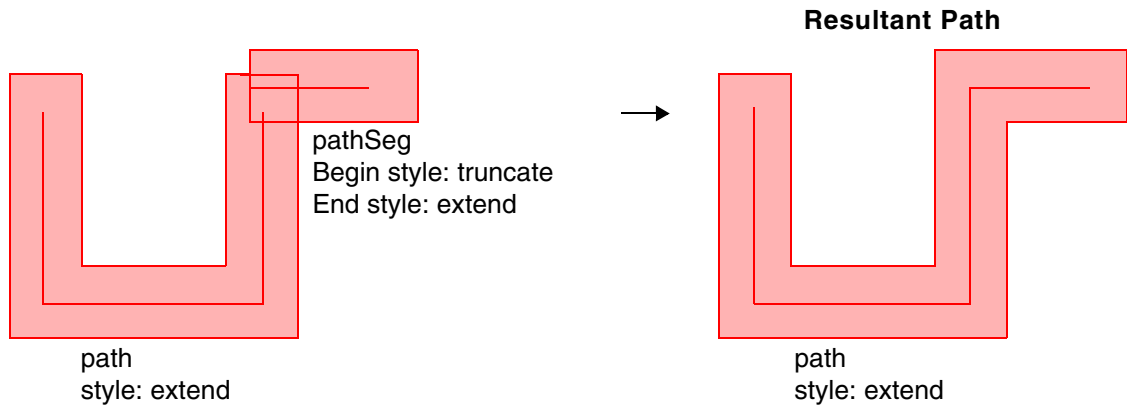
Merging a Pathseg with a Path

Merging a pathSeg with a path may result in either a path or a polygon.

The following table describes the combinations of path styles possible by merging a pathSeg and a path. Also described is the situations in which merging a path and pathSeg result in a polygon.

End Style of the Pathseg	Style of the Path				
		Truncate	Variable	Extend	Round
	Truncate	Truncate	Variable	Variable	Polygon
	Variable	Variable	Variable	Variable	Polygon
	Extend	Variable	Variable	Extend	Polygon
	Custom	Polygon	Polygon	Polygon	Polygon

Example of Merging Paths and PathSegs



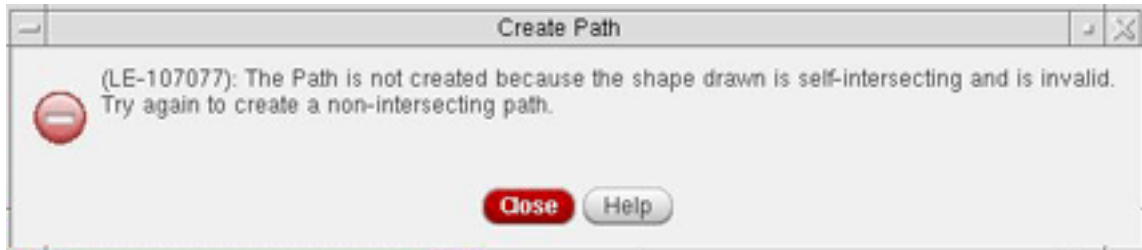
Self-intersecting Objects

A single database object is not allowed to intersect itself. Paths are shaped based objects represented by a point array and are not allowed to become self-intersecting during creation or editing.

Virtuoso Interactive and Assisted Routing User Guide

Editing Routed Connections

When an object is self-intersecting, an error message dialog box appears and the object is not allowed to be drawn.

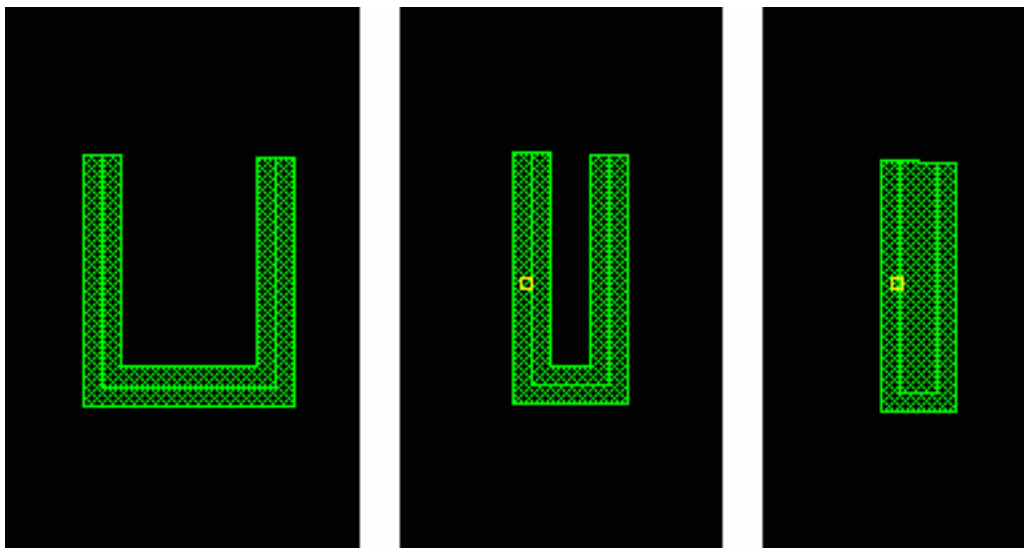


A single pathSeg consist of a two-point segment and cannot be drawn in a manner that would allow self-intersection.

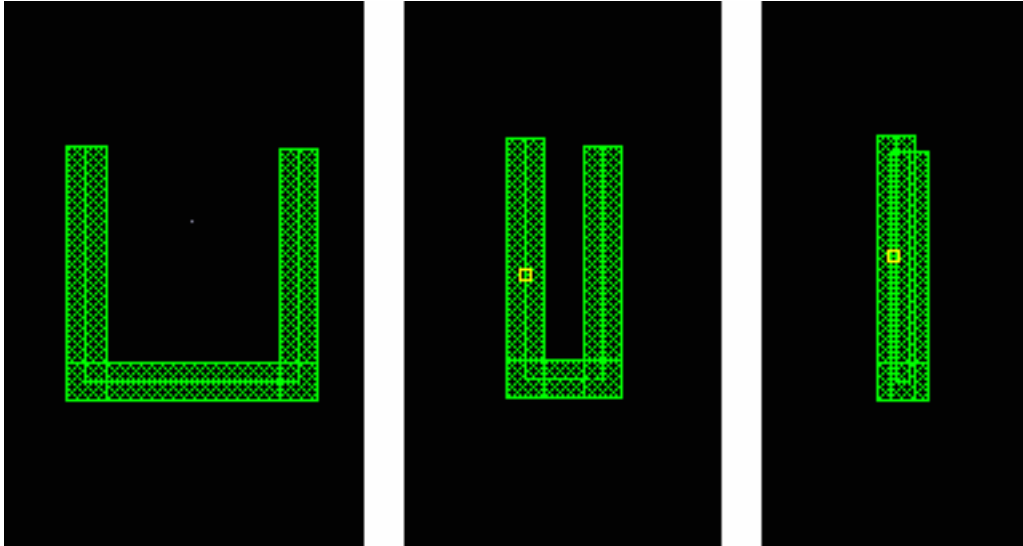
When creating or editing wires using paths, the edges of a path can be coincident, but not overlap. When creating or editing wires using pathSegs, a series of pathSegs can be stretched to cross over other pathSegs or overlap.

By default, in the cases where the pathSegs would create a loop, the loops are removed. See [Removing Loops From Wires](#) for more information.

The following image shows how a u-shaped path is not allowed to be stretched over itself and self-intersect.



When the same u-shape wiring is created with multiple pathSegs, the segments of the pathSegs are allowed to be stretched to cross over each other or overlap.



Deleting Wires

The *Delete* command can be used to delete an entire wire when the wire is fully selected.

Wires that are created on the same layer using path elements are created as a continuous path, without intersecting vertices at corners. When deleting wires created with path elements, sections of the wire on the same layer can be deleted without breaks or gaps being created.

Wires that are created on the same layer using pathSegs as wire elements are created with intersecting vertices at corners. When deleting wires created with pathSegs as wire elements, selected sections can be removed, breaking the wire into multiple wires if interior pathSegs are removed.

Copy Route Command

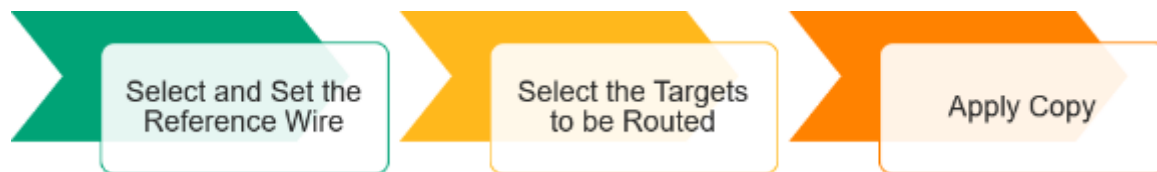
The copy route feature lets you work incrementally and generate thousands of bus wires. It:

- eases the routing of a large bus over a long distance by using a single bit as a topology reference.
- supports reference wires that are connected to pins as well as in empty spaces in the design.
- makes it easier to add bits to an already routed bus.
- customizes the spacing and the width of the copies.

Copy Route supports not only oaRoute objects but also non-oaRoute objects. Supporting non-oaRoute objects means that copy route is able to copy a trunk, which is an example of a non-oaRoute object. A trunk is a pathSeg but is not an oaRoute object.

The copies created using the copy route feature are DRC correct and can be customized to adjust the spacing and width. These routes can start from an empty space in the canvas or from pins and can be reordered if required to complete the connection.

The following figure illustrates the copy route flow.



You can copy the following routes.

- Pin-to-pin routes
- Pin-to-wire routes or Wire-to-wire routes

Related Topics

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

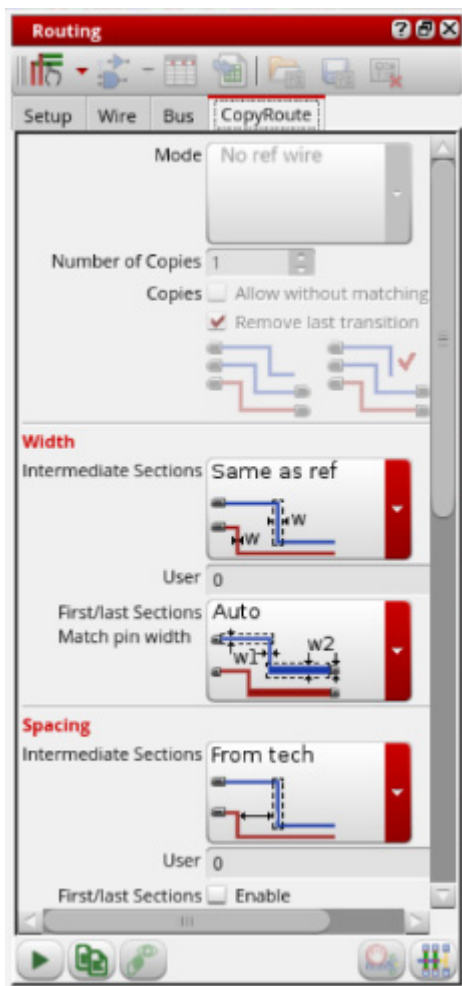
[Copying Pin-to-Wire Routes](#)

Setting Copy Route Options

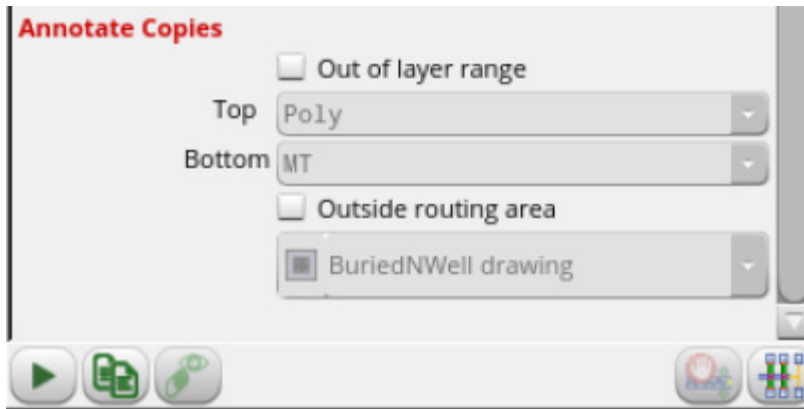
The *Copy Route* command lets you work incrementally and generate thousands of bus wires. When there are pins or instance pins for the copies, the width and the spacing of the copy route is based on them. Otherwise, you can control the width and spacing settings for the intermediate transitions from the form.

The *Copy Route* tab in the Routing Assistant provides a number of options for customizing the behavior of copying routes. To set Copy Route options:

1. Choose *Windows – Assistant – Routing*. The Routing assistant displays.
2. Click the *Change routing mode* button on the Routing assistant toolbar and select *Interactive* as the routing mode.
3. Click the *CopyRoute* tab.



4. Set the Copy Route options you require in the different sections of the assistant. For more information on these options. see [CopyRoute](#).
5. Click the *Start Copy Route* button at the bottom of the tab.



You can also set the copy route options in the Copy Route form. To open the Copy Route form, choose *Edit – Wiring – Copy Route* and press F3. For more information, see [Copy Route Form](#).

Related Topics

[Copy Route Command](#)

[Copying Pin-to-Pin Routes](#)

[CopyRoute](#)

[Setting Copy Route Options](#)

[Variables to Customize the Settings for Copy Route](#)

Copying Pin-to-Pin Routes

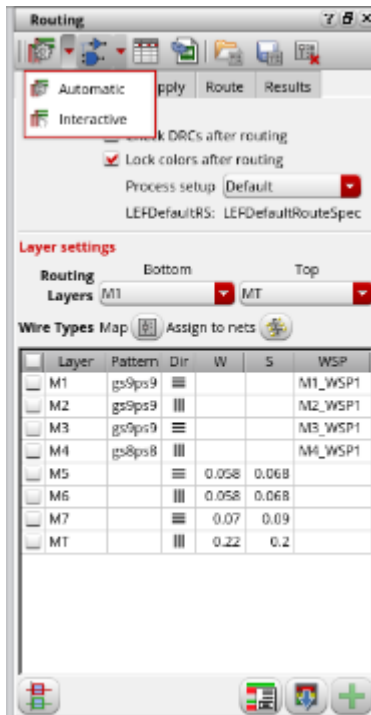
Copy Route lets you create copies from a reference wire. While creating copies, you can control the width and spacing as well as the layers for the intermediate transition of the copies. To create copies:

1. Choose *Windows – Assistant – Routing*. The Routing assistant displays.

Virtuoso Interactive and Assisted Routing User Guide

Editing Routed Connections

2. Click the *Change routing mode* button on the Routing assistant toolbar and select *Interactive* as the routing mode.

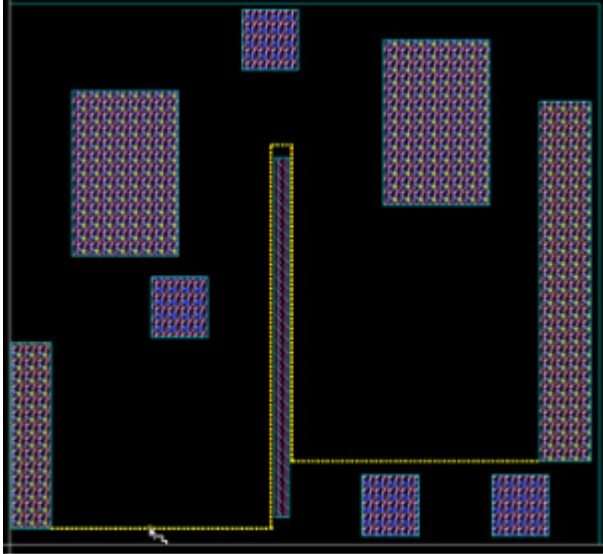


3. Click the *CopyRoute* tab.
4. Choose *Edit – Wiring – Copy Route*. Alternatively, click the *Start Copy Route* button at the bottom of the *CopyRoute* tab in the Routing assistant.

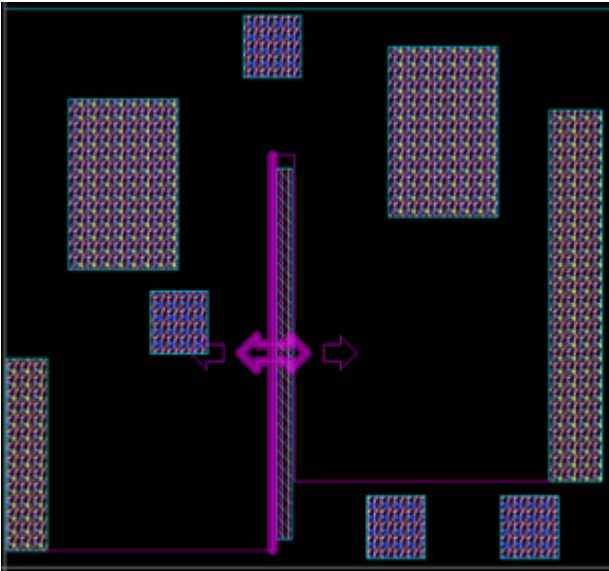
Virtuoso Interactive and Assisted Routing User Guide

Editing Routed Connections

5. Click to select the reference wire.



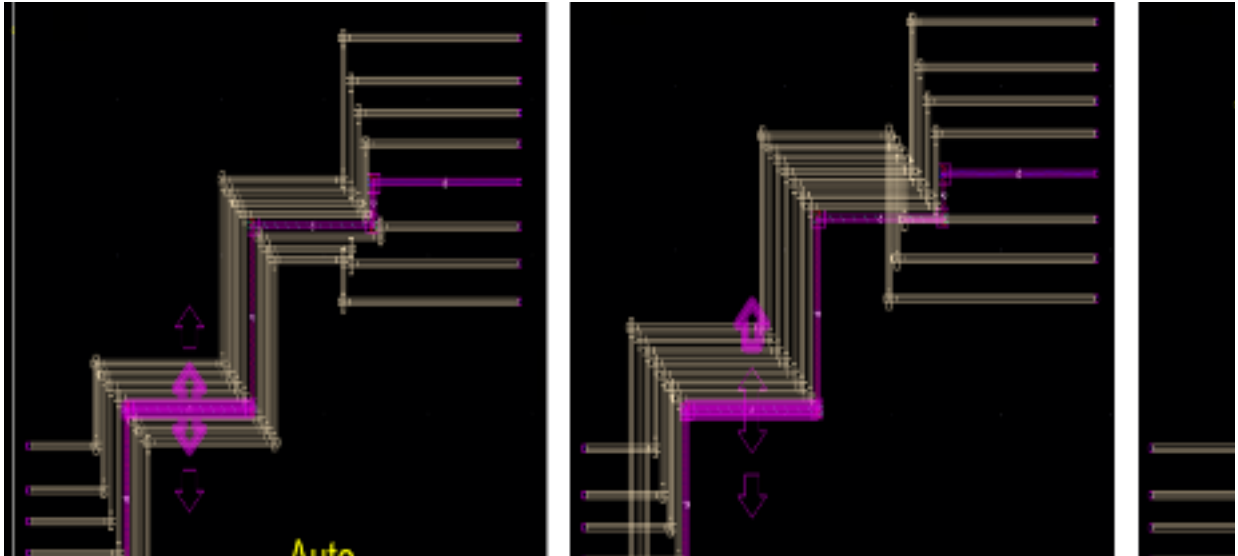
6. Press space bar to set the reference wire. After the reference wire is set, it is highlighted in the layer color and the arrows are displayed.



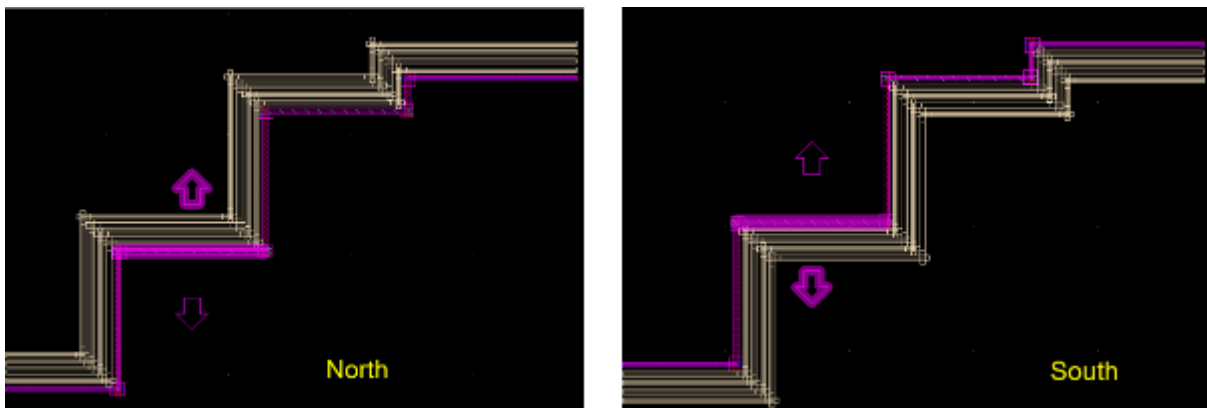
Virtuoso Interactive and Assisted Routing User Guide

Editing Routed Connections

If the mode is selected as *Connected to pins*, three arrows are displayed. These control the intermediate section and let you choose in which direction to copy the route. The first and last section must be connected to pins.



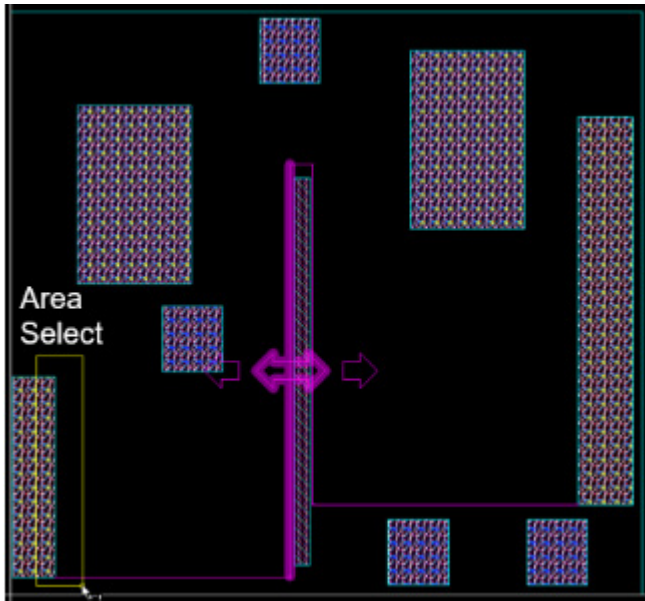
In the *Space* mode, two arrows are displayed. These arrows let you choose in which direction to copy the route.



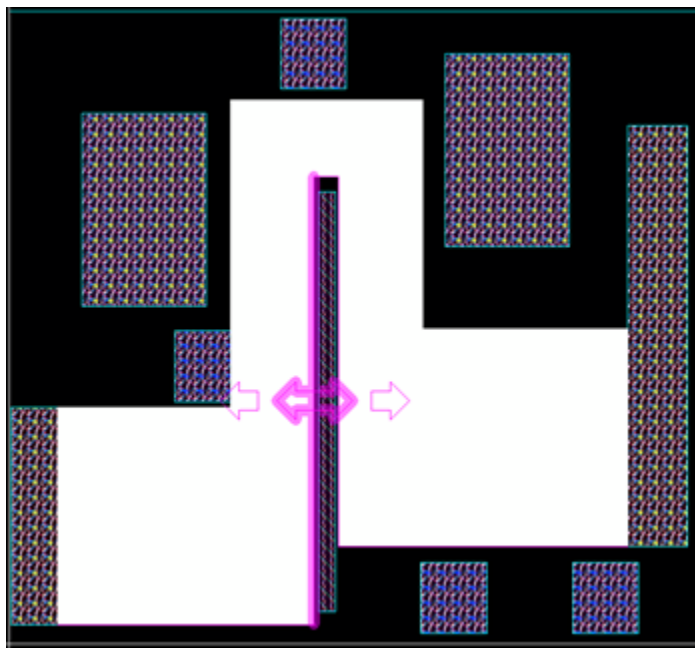
Virtuoso Interactive and Assisted Routing User Guide

Editing Routed Connections

7. Click and drag the mouse pointer to area select the instance pins that are to be routed. You can use the `Ctrl` or `Shift` keys to remove or add instance pins later.



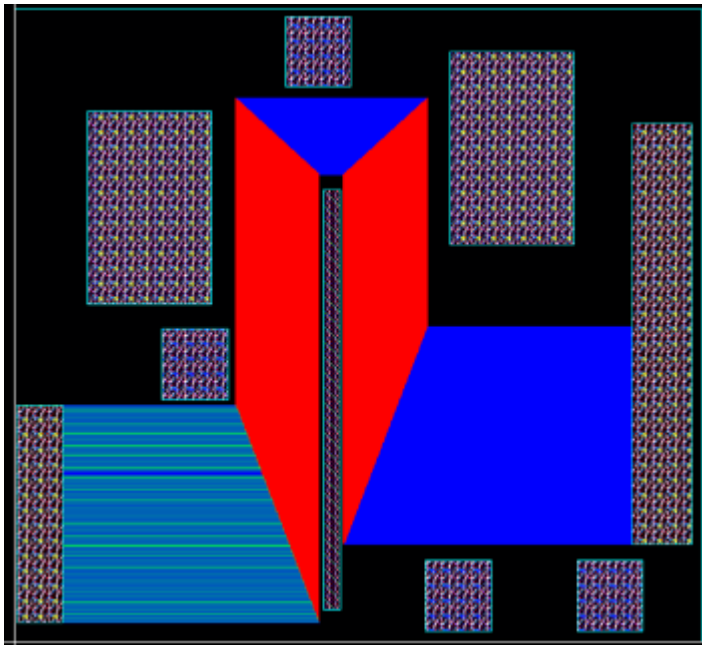
Once the instance pins are selected, click *Show preview* to see a preview of how the copies are created. If *Show preview* is unselected, flightlines are displayed.



8. Edit your design as required to avoid any shorts that might be created.

9. Choose how the target layer is determined. The available options are: *Same as reference*, *Same as reference Except pins*, *Based on pins*, *Based on pattern*.
10. Press `Enter`. Alternatively, click the *Run Copy Route* button at the bottom of the tab.

The copy route is run and the copies are created based on the settings and options. The following figure shows how the layers used for the copies are different if you set target layers as *Based on pins*.



Related Topics

[Setting Copy Route Options](#)

[Copy Route Command](#)

[Copy Route Form](#)

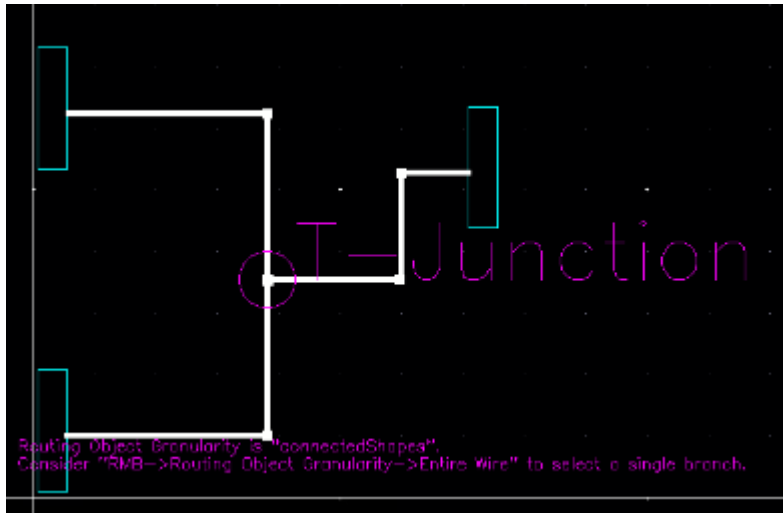
[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

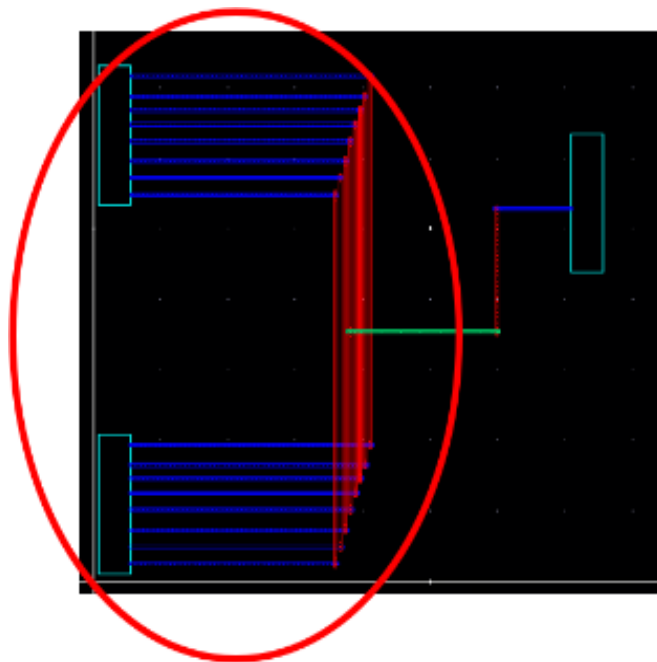
[Copying Pin-to-Wire Routes](#)

Copying Pin-to-Wire Routes

By default, the routing object granularity is set to *Connected Shapes*. In such a case, you are unable to select a reference wire with T-junctions. When there is a T-junction in the reference wire, a hint appears to suggest that you switch to the *Entire Wire* mode.



To copy a wire with T-junctions, you must perform Copy Route in multiple steps. You first need to copy the pin-to-pin routes and then copy the pin-to-wire or wire-to-wire routes. The following figure displays a scenario of copying pin-to-wire routes.

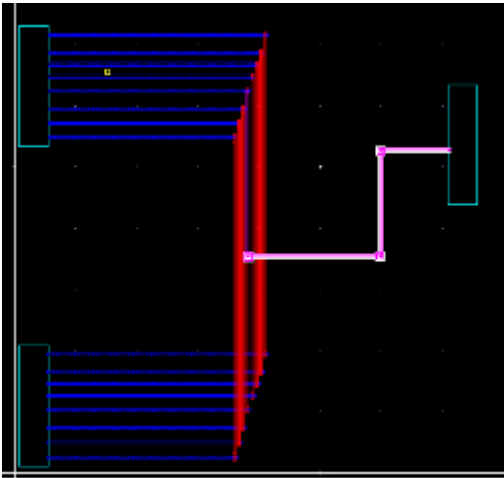


Virtuoso Interactive and Assisted Routing User Guide

Editing Routed Connections

To copy a route from pin-to-wire:

1. Choose *Windows – Assistant – Routing*. The Routing assistant displays.
2. Click the *Change routing mode* button on the Routing assistant toolbar and select *Interactive* as the routing mode.
3. Click the *CopyRoute* tab.
4. Choose *Edit – Wiring – Copy Route*. Alternatively, click the *Start Copy Route* button at the bottom of the *CopyRoute* tab in the Routing assistant.
5. Click to select a reference wire that is from pin to wire. You can also copy routes that are from wire to wire.

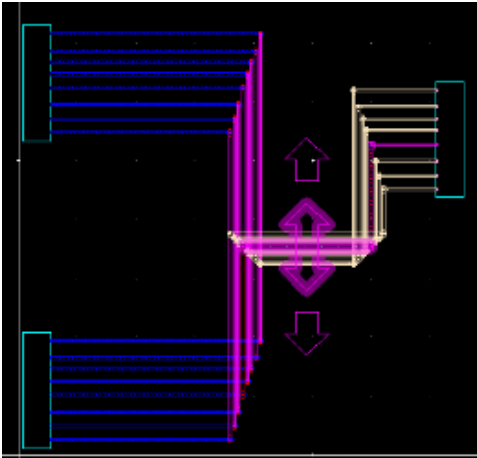


6. Press space bar to set the reference wire.

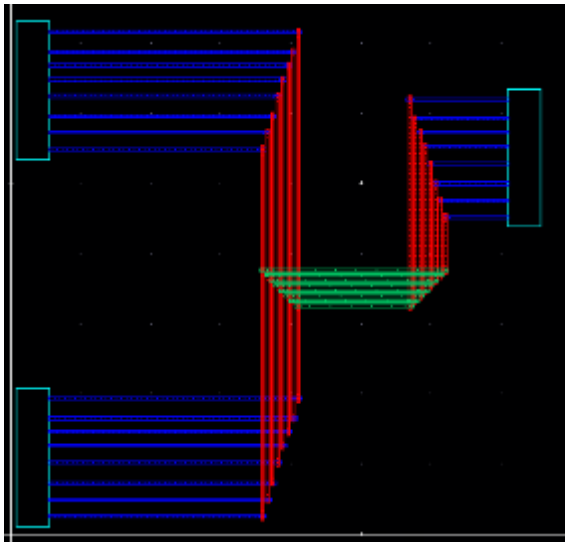
Virtuoso Interactive and Assisted Routing User Guide

Editing Routed Connections

7. Click and drag the mouse pointer to area select the instance pins or wires (pathSegs) that are to be routed. You can use the `Ctrl` or `Shift` keys to remove or add instance pins later.



8. Press `Enter` and see the result of pin-to-wire routes.



Related Topics

[Copying Pin-to-Pin Routes](#)

[Setting Copy Route Options](#)

[Copy Route Command](#)

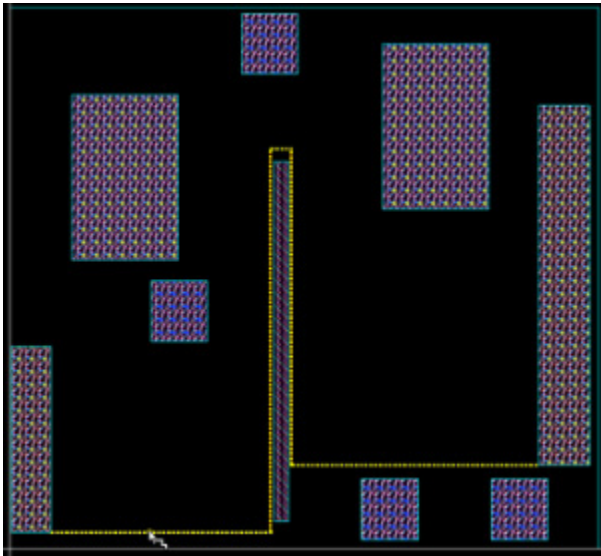
Copy Route Form

Variables to Customize the Settings for Copy Route

Handling Wire Copies for EM Analysis

You can control the SDR checker mode directly from the *CopyRoute* tab in the Routing assistant. This lets you adjust the wire width of copies to avoid EM violations. Copy Route eases the routing of a large bus based on a reference wire topology. To handle routing wires for EM analysis:

1. Choose *Windows – Assistant – Routing*. The Routing assistant displays.
2. Click the *Change routing mode* button on the Routing assistant toolbar and select *Interactive* as the routing mode.
3. Click the *CopyRoute* tab.
4. Click the *Start copy route* button to enable the command.
5. Select the *Enable Preview* option in the *Preview* section of the *Copy Route* tab.
6. Click to select the reference wire.



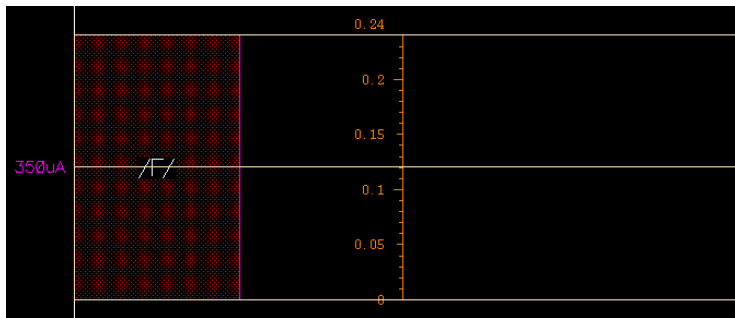
7. Press the space bar to set the reference wire. After the reference wire is set, it is highlighted in the layer color and the arrows are displayed.
8. Click and drag the mouse pointer to area select the instance pins that are to be routed.

Virtuoso Interactive and Assisted Routing User Guide

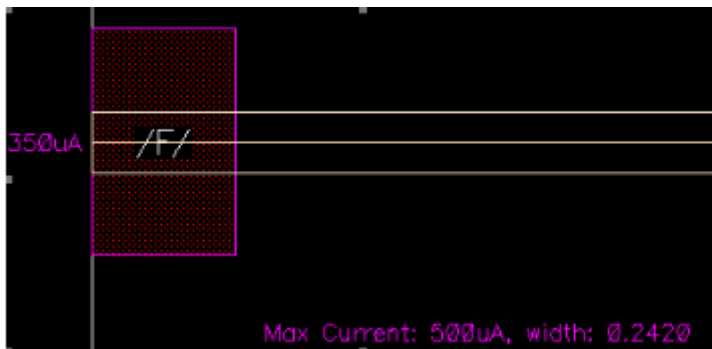
Editing Routed Connections

9. Click *SDR Checker Mode*  to adjust the width of the selected wires.

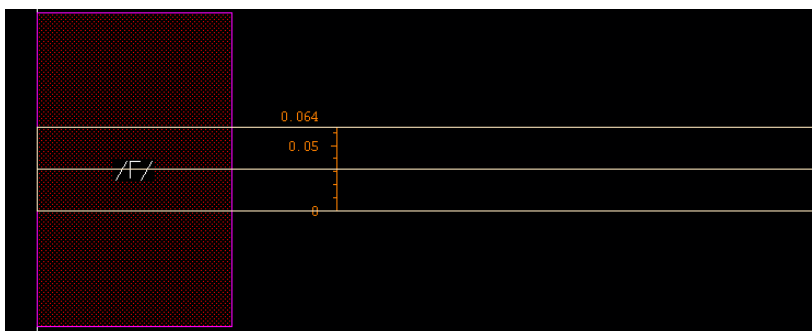
When the *SDR Checker Mode* is set to *Enforce*, the widths of the previewed wires are computed automatically according to the EM analysis to optimize their size based on the current estimation.



When the SDR Checker Mode is set to *Notify*, Copy Route indicates the optimal width that should be set on the wires to satisfy the EM analysis for all selected nets. The widths of the previewed wires are computed according to the width specified in the Routing assistant and a label is displayed in the layout canvas giving the maximum current for the copies.



When the *SDR Checker Mode* is deselected , the widths of the previewed wires are computed according to the width specified in the Routing assistant.



Related Topics

[Setting Copy Route Options](#)

[Copy Route Command](#)

[Copy Route Form](#)

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

Managing Metal Density

The metal density in a layout design needs to be managed to ensure that there is uniform metal distribution across the chip for manufacturing. In order to address and manage metal density issues, such as density checks, thin or local analysis, and fixing density errors through slotting, use the *Metal Density* toolbar. The *Metal Density* toolbar provides an interactive and user-friendly approach to address and manage the metal density issues. This toolbar is available in Layout XL and higher tiers. Depending on the layout tier, specific number of tokens are required to access the slotting feature. For more information, see [Table 1-6 Tokens per Feature \(VLS GXL\)](#) in the Virtuoso Software and Licensing User Guide.



The Metal Density toolbar currently addresses only maximum density issues.

This chapter covers the following topics:

- [Metal Density Toolbar](#)
- [Performing Metal Density Checks for Shapes](#)
- [Specification Cells](#)
- [Fillet Creation \(Virtuoso RF Solution Only\)](#)
- [Support for Slot Shapes in DRD and EAD](#)

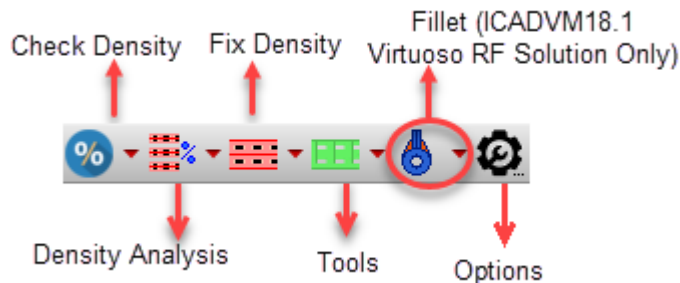
Metal Density Toolbar

To maintain the integrity of the original shape for as long as possible, slotting lets you add slot shapes to the original shape, such as wires and polygons. For more information on how to convert the original shapes covered by the slot shapes into slotted polygons, see [lxConvertSlotToPolygon](#). Also, the slotted shapes can be edited to solve electromigration violations. In addition, the slot shapes are created as a figGroup to prevent any incorrect manipulation.

Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

Using the *Metal Density* toolbar, you can perform DRC correct slotting, `maxDensity` checks, and analyze any kind of shapes. The different scopes provide a smooth and user-friendly experience of the entire design, a specific region, one or more layers, the current window, or by selection. In addition, the toolbar provides an options form for `maxDensity` checks and local slotting statistic reports. The *Metal Density* toolbar has the following buttons for metal density checks and slotting.



■ *Check Density*

Checks the maximum metal density rules according to the foundry constraints under different scopes. As a prerequisite for the density checks, the maximum density rules need to be defined in the technology library.

Command

Check Density All 

Check Density By Area 

Check Density Current Window 

Lets you

Check the maximum density rule in the entire design.

Check the maximum density rule in the area defined by the two specified points in the design.

Check the maximum density rule in the current window (layout canvas limit).

■ *Density Analysis*

Reports the local density of each slotted shape, which means that for each slotted shape, its total slotted surface over its associated shape surface is reported.

Command

Density Analysis All 

Lets you

Report the local density of each slotted shape in the entire design.

Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

Density Analysis By Area



Report the local density of each slotted shape in the area defined by the two specified points in the design.

Density Analysis Current Window



Report the local density of each slotted shape in the current window (layout canvas limit).

Density Analysis By Selection



Report the local density for all selected shapes in the design.

If both nets and shapes are selected in the design, then when *Density Analysis By Selection* is selected, it filters out the nets. If *Density Analysis By Net* is selected, it filters out the shapes.

Density Analysis By Net



Report the local density for each slotted shape of the selected nets in the design.

☐ *Check Slot Consistency*

Checks the consistency of the slot shapes in the design. If conditions do not match, an error marker is displayed in the Annotation Browser.

Command

Lets you

Check Slot Consistency All



Check the consistency of all slot shapes in the design.

Check Slot Consistency By Area



Check the consistency of created slot shapes in the area defined by the two specified points in the design.

Check Slot Consistency Current Window



Check the consistency of created slot shapes in the current window (layout canvas limit).

☒ *Fix Density*



Adds slot shapes using the metal layer-purpose pair `slot` on any kind of objects, such as wires, paths, rectangles, or polygons. The geometries of the slot shapes are default. These geometries can also be customized using the Metal Density Options form.

Command

Lets you

Slotting All



Add slot shapes on objects in the entire design.

Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

Slotting By Area



Add slot shapes on objects that are fully or partially enclosed in the area defined by the two specified points in the design.

Slotting Current Window



Add slot shapes on objects in the current window (layout canvas limit).

Slotting By Selection



Add slot shapes on objects of the selected shapes.

Slotting By Selection in Area



Add slot shapes on the selected objects, only in the section delimited by the two points of the area.

Slotting By Net



Add slot shapes on objects of nets selected in the Navigator assistant.

■ *Tools*



Lets you delete slots, or convert and reconstruct a polygon to slots.

☐ *Delete Slots*



Deletes the slot shapes on objects according to the scope of the deletion.

Command

Delete Slots

Lets you

Delete slot shapes on objects in the entire design.

Delete Slots Current Window

Delete slot shapes on objects in the current window (layout canvas limit).

Delete Slots By Area

Delete slot shapes from objects that are fully or partially enclosed in the area defined by the two specified points in the design.

Delete Slots By Selection

Delete slot shapes on objects of the selected shapes.

☐ *Convert Slots*



Converts the original shapes covered by the slot shapes into slotted polygons.

Command

Convert Slots

Lets you

Convert the original shapes covered by the slot shapes into slotted polygons.

Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

<i>Convert Slots Current Window</i>	Convert the original shapes covered by the slots in the current window (layout canvas limit) into slotted polygons.
<i>Convert Slots By Area</i>	Convert the original shapes covered by the slots defined by an area into slotted polygons.
<i>Convert Slots By Selection</i>	Convert the selected shapes covered by the slots into slotted polygons.

Environment Variable: [hideConvertToPolygon](#)

☐ Reconstruct Slots

Reconstructs a slotted polygon to a regular polygon that has no holes and also covers the slot shapes.

Command

Lets you

Reconstruct Slots

Reconstruct a slotted polygon to a regular polygon that has no holes and also covers the slot shapes in the entire design.

Reconstruct Slots Current Window

Reconstruct a slotted polygon to a regular polygon that has no holes and also covers the slot shapes in the current window (layout canvas limit).

Reconstruct Slots By Area

Reconstruct a slotted polygon to a regular polygon that has no holes and also covers the slot shapes, which is fully or partially enclosed in the area defined by the two specified points in the design.

Reconstruct Slots By Selection

Reconstruct a slotted polygon, which is selected, to a regular polygon that has no holes and also covers the slot shapes.

Environment Variable: [hideReconstruct](#)

☒ Create Fillet (Virtuoso RF Solution Only)

Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

Creates fillets. A fillets is an extra etch added to prevent acid traps on a design, which in turn improves manufacturability. In Virtuoso, fillets are created as curved polygons..

Command

Create Fillet All 

Create Fillet By Area 

Create Fillet Current Window 

Create Fillet by Net 

Delete Fillet All 

Delete Fillet By Area 

Delete Fillet Current Window 

Delete Fillet by Net 

Lets you

Create fillets across the entire design.

Create fillets in a specified area of the design.

Insert fillets in the area defined by the size of the current window.

Insert fillets on selected nets.

Delete fillets across the entire design.

Delete fillets from the specified area of the design.

Delete fillets in the area defined by the size of the current window.

Delete fillets on selected nets.

■ Options

Opens the Metal Density Options form, which contains all the geometric and style settings for the generation of the slot shapes. In addition, it contains valid layers that can be considered by each shape for checking, analysis, and fixing metal density.

Related Topics

[Metal Density Options Form](#)

[IxShapeSlotting](#)

[IxConvertSlotToPolygon](#)

[Variables to Customize the Settings for Slotting](#)

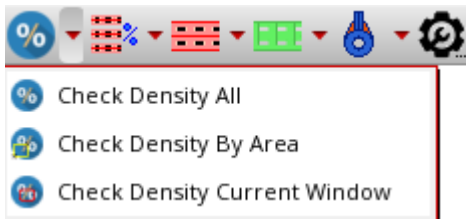
[Fillet Creation Between Curved Path and Other Objects](#)

[Fillet Creation \(Virtuoso RF Solution Only\)](#)

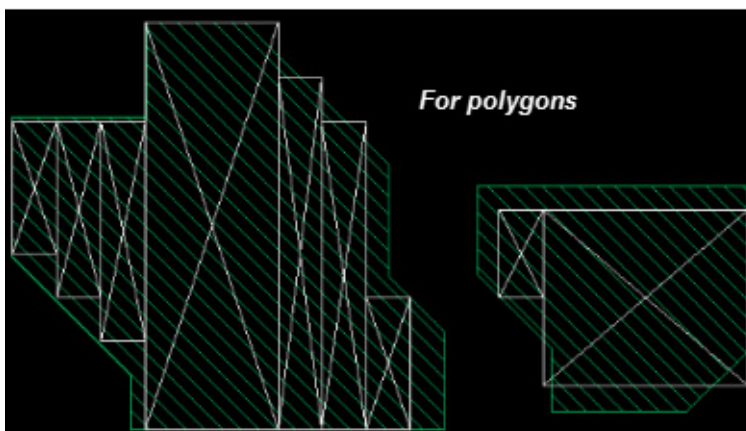
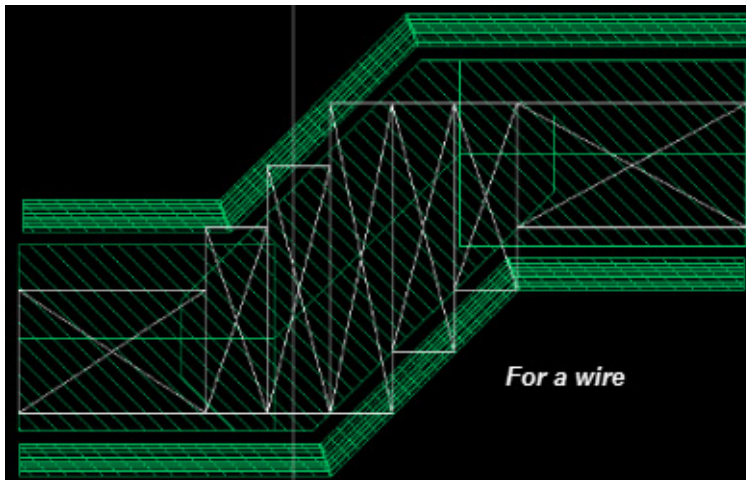
Performing Metal Density Checks for Shapes

You can check, analyze, and fix metal density for various routing shapes, such as for wires and polygons. To perform metal density checks for shapes, perform the following steps:

1. Click the arrow next to the *Check Density* icon on the *Metal Density* toolbar. A drop-down menu appears.



2. Choose the scope that you want from the drop-down list. The following figure shows the result of the metal density check when the *Check Density All* option is selected.



3. Click the *Options* button on the *Metal Density* toolbar.

The Metal Density Options form appears.

The screenshot shows the 'Metal Density Options' dialog box. It has four tabs: 'Layers' (selected), 'Slot Geometry', 'Slotting Style', and 'Slot Consistency'. In the 'Layers' tab, there is a 'Use Palette Layers' checkbox which is unchecked. Below it is a 'Material' dropdown menu with 'metal' selected. To the right of the material dropdown is a list box containing the following items: M1, M2, M3 (highlighted), M4, M5, M6, M7, CMT, and MT. At the bottom left of the dialog is a green circular button labeled 'Slotting'. At the bottom right are two buttons: 'Close' and 'Help'.

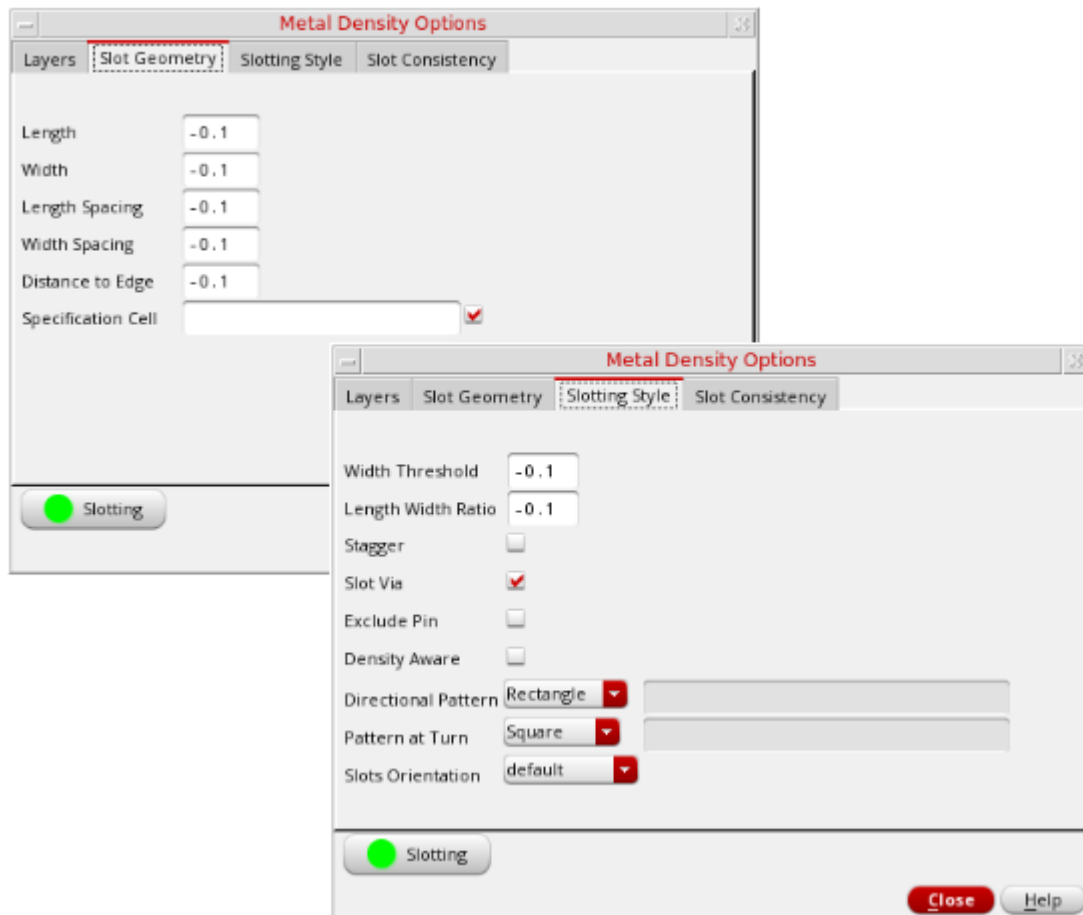
4. Select a material to be processed from the *Material* drop-down list. By default, *metal* is selected.
5. From the *Layers* list box, select a layer, for example `Metal3`. The selected layers are used for checking, analyzing, and slotting.

The fields in the other tabs of the form are pre-populated with the values defined in the environment variables in the `.cdsenv` file. You can edit the options needed to define the

Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

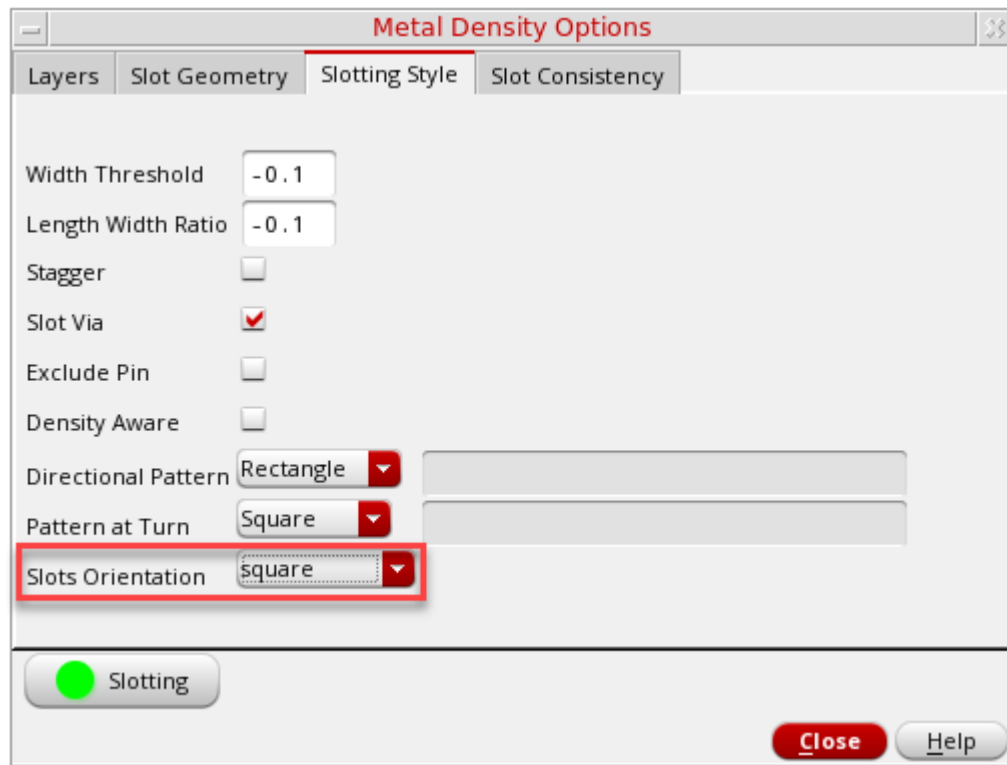
size and spacing of the slot shapes in the *Slot Geometry* tab of the Metal Density Options form. Also, you can edit the advanced slotting settings in the *Slotting Style* tab.



Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

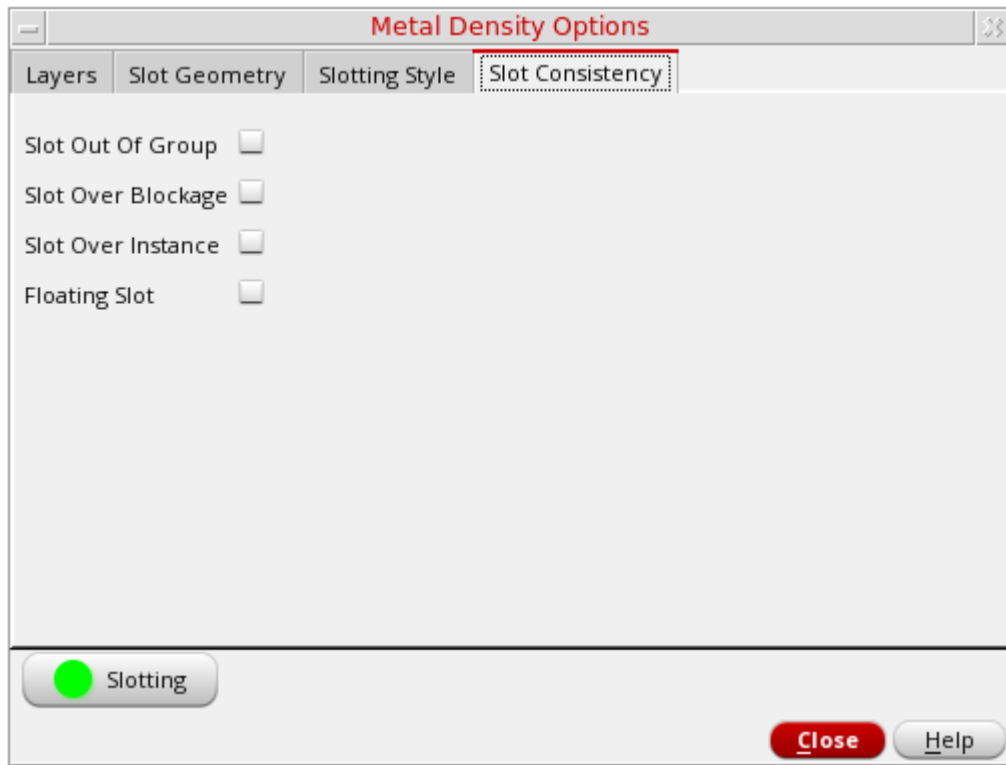
- For a polygon shape, select *square* from the *Slots Orientation* drop-down list in the *Slotting Style* page.



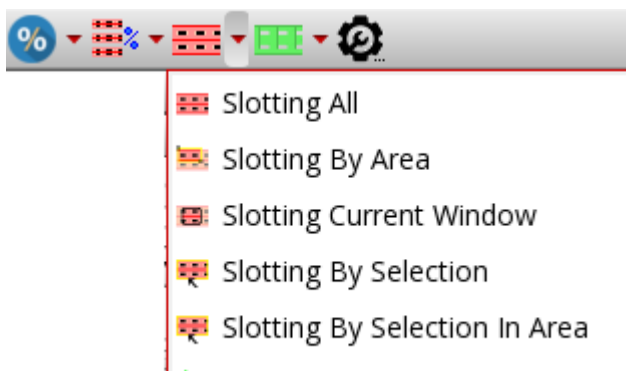
Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

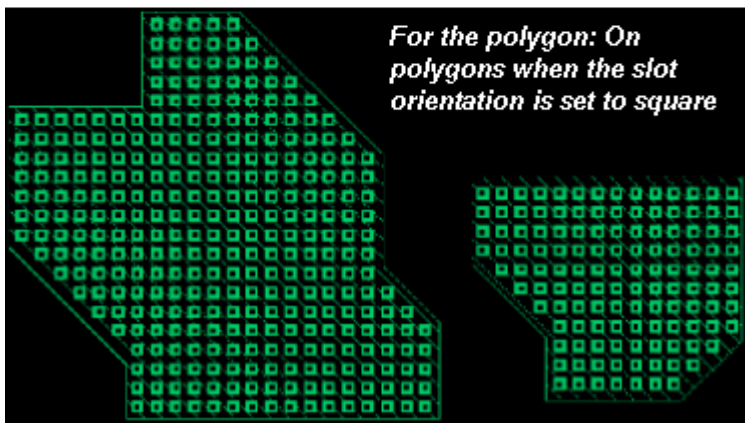
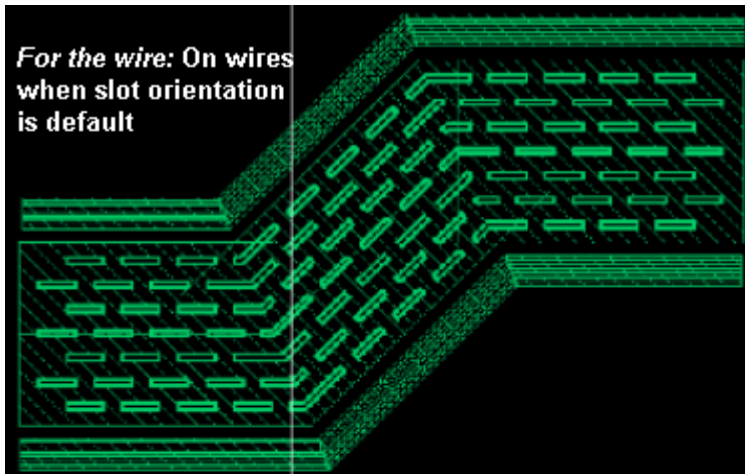
7. Select the consistency checks that you want to run for the slot shapes in the *Slot Consistency* page.



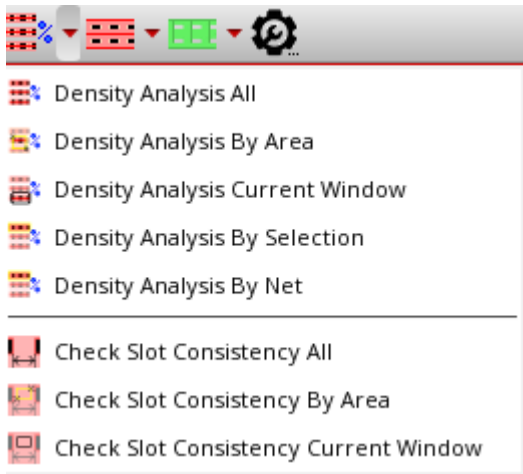
8. Click *Close* to close the form.
9. Click the arrow next to the *Fix Density* icon on the *Metal Density* toolbar. A drop-down menu appears.



10. Choose the scope that you want from the drop-down list to perform slotting. The following figure shows the result when the *Slotting All* option is selected to slot the shapes and fix metal density.



11. Click the arrow next to the *Density Analysis* icon on the *Metal Density* toolbar. A drop-down menu appears. You can either perform density analysis of each slotted shape or check for the correctness and consistency of the slot shapes.



- a. To report the local density of each slotted shape, choose the scope that you want from the drop-down list. The following figure shows the report of the local density check when the *Density Analysis All* option is selected.

Analyzing Density for ("Metal3") layers:

```

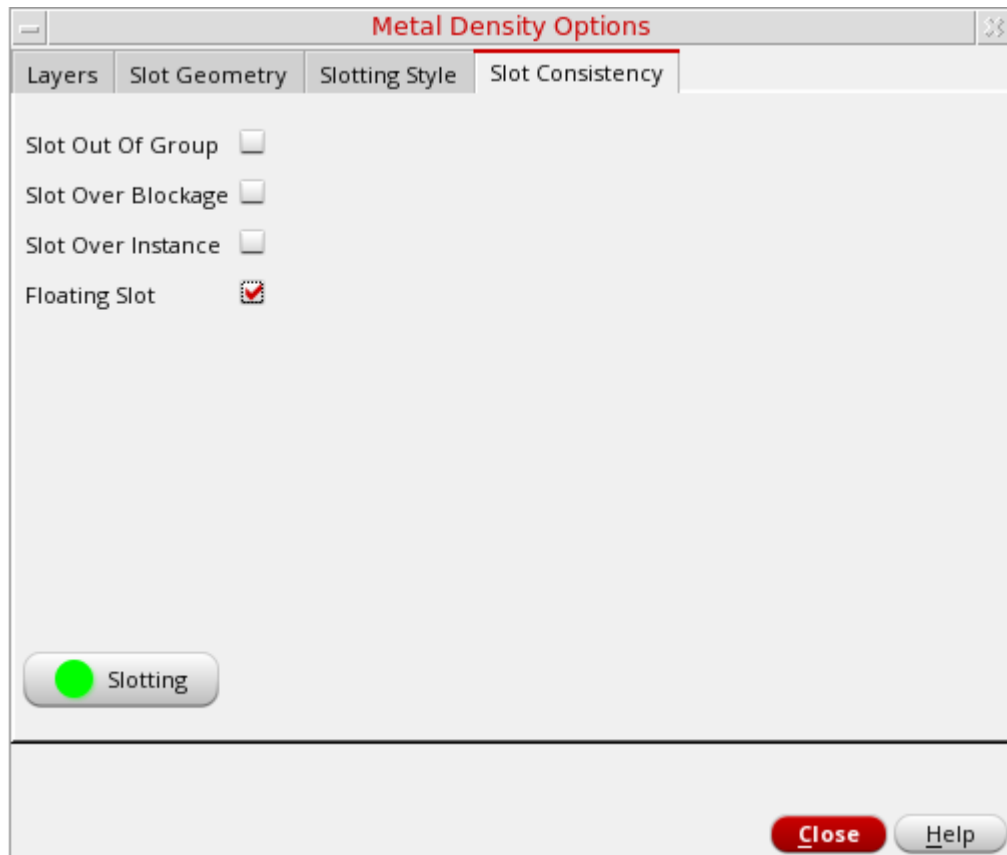
## =====
## The "Metal3" wire width=30.0um from (-44.10 32.85) to (-3.05 32.85) has a local Density of 81.01%
## =====
## The "Metal3" wire width=30.0um from (-3.05 32.85) to (26.75 62.65) has a local Density of 68.84%
## =====
## The "Metal3" wire width=30.0um from (26.75 62.65) to (72.70 62.65) has a local Density of 84.02%
## =====
## 3 shapes have been slotted within the specified area or among the selection set.
t

## =====
## The "Metal3" polygon located in bbox ((107.4 12.3) (168.5 63.6)) has a local Density of 81.06%
## =====
## The "Metal3" polygon located in bbox ((2.3 8.2) (99.8 100.8)) has a local Density of 80.14%
## =====
## 2 shapes have been slotted within the specified area or among the selection set.
t
  
```

Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

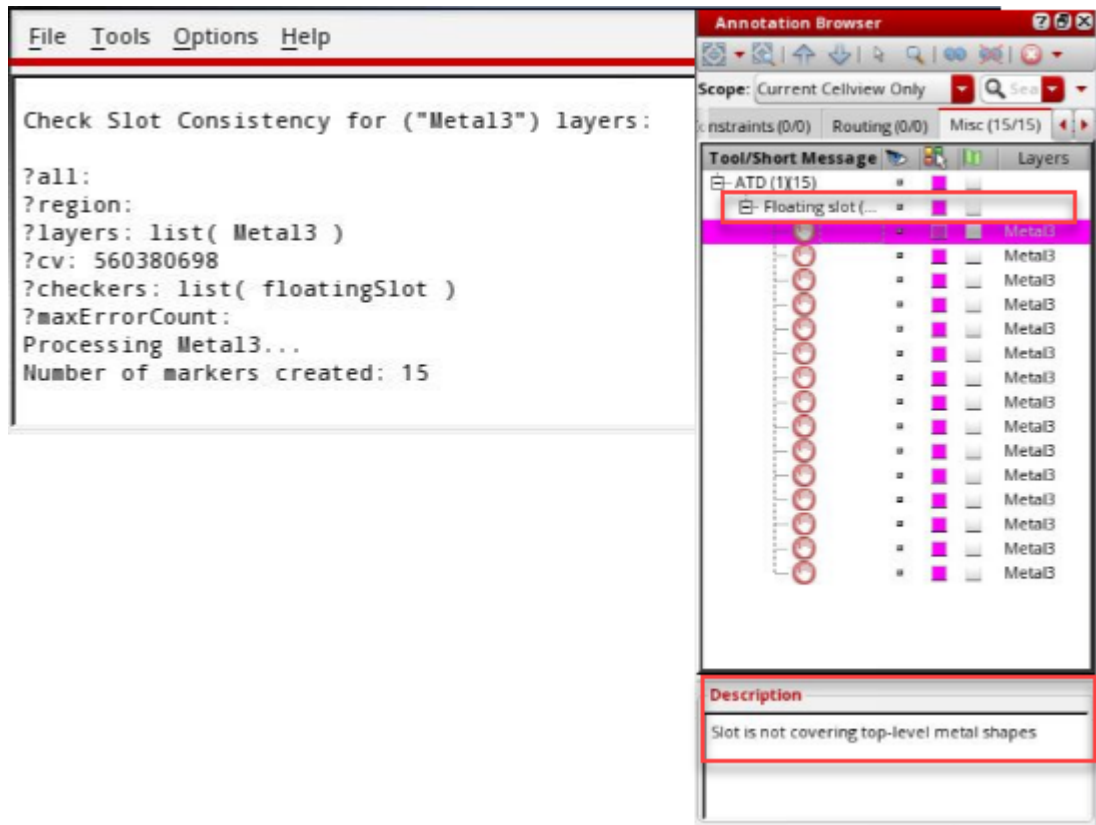
- b. To check the consistency and correctness of the slot shapes, select the required option in the *Slot Consistency* tab of the Metal Density Options form. For example, to check for a floating slot shape, select the *Floating Slot* option.



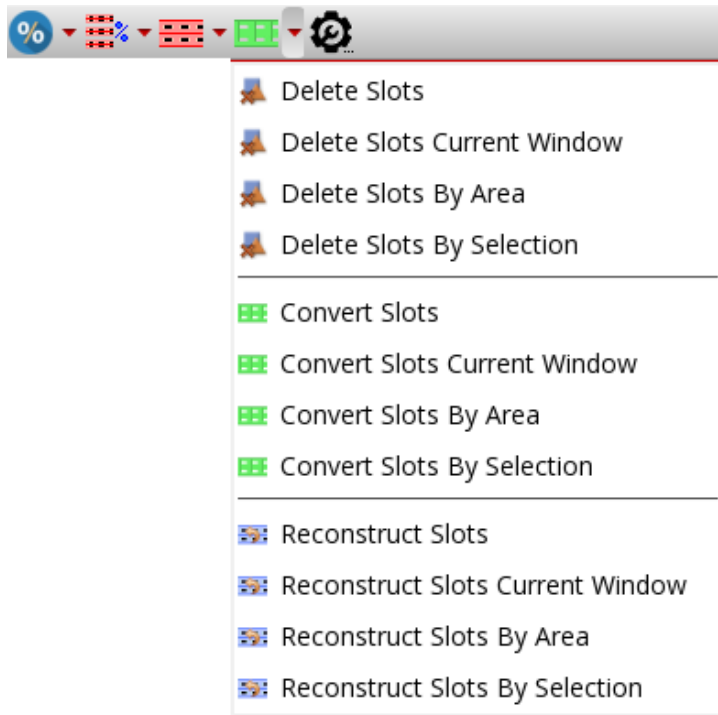
Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

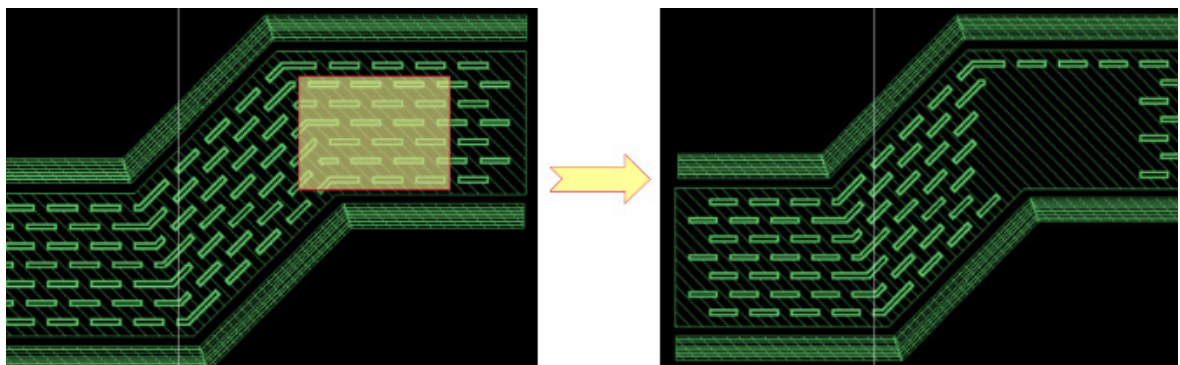
The following figure shows the report of the consistency check along with the markers being displayed in the Annotation Browser, when the *Check Slot Consistency All* option is selected.



12. Click the arrow next to the *Tools* icon on the *Metal Density* toolbar. A drop-down menu appears. You can either delete the slots on the polygon or convert and reconstruct the slots on the polygons.

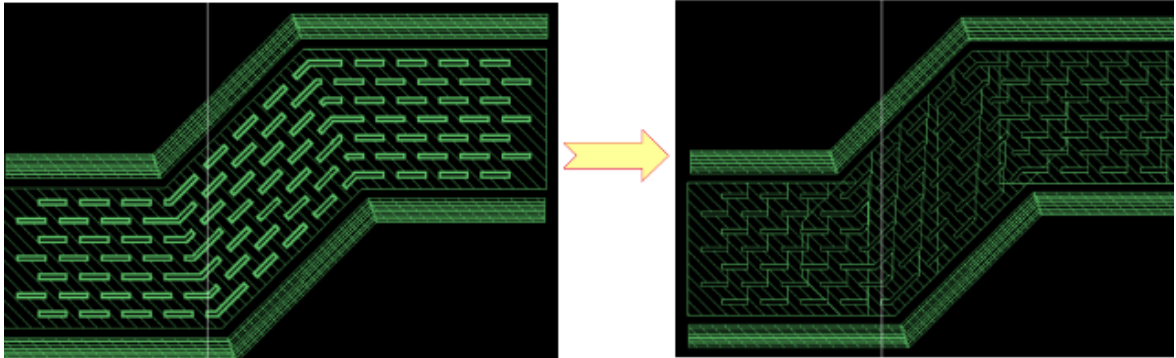


- c. To delete a slot shape on a polygon, choose the scope that you want from the drop-down list. The following figure shows the result when the *Delete Slots by Area* option is selected to delete the slots by area.

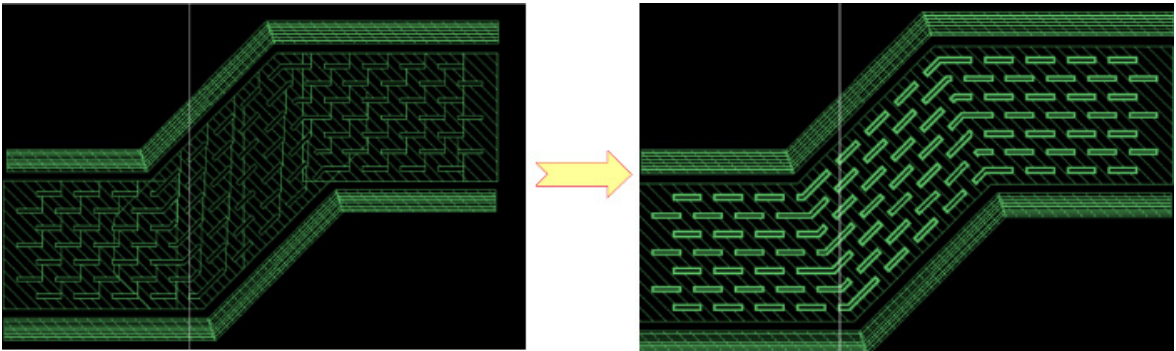


- d. To convert the original shape covered by the slot shapes as slotted polygons, choose the scope that you want from the drop-down list. The following figure shows

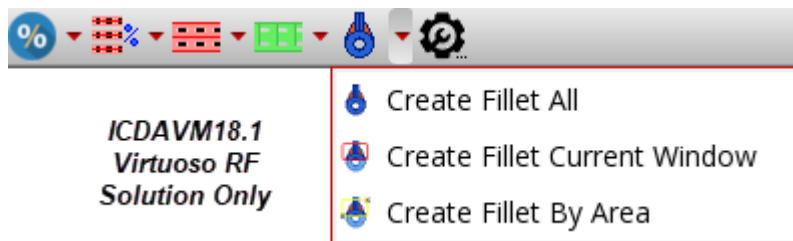
the result when the *Convert Slots* option is selected to convert a polygon to slot shapes.



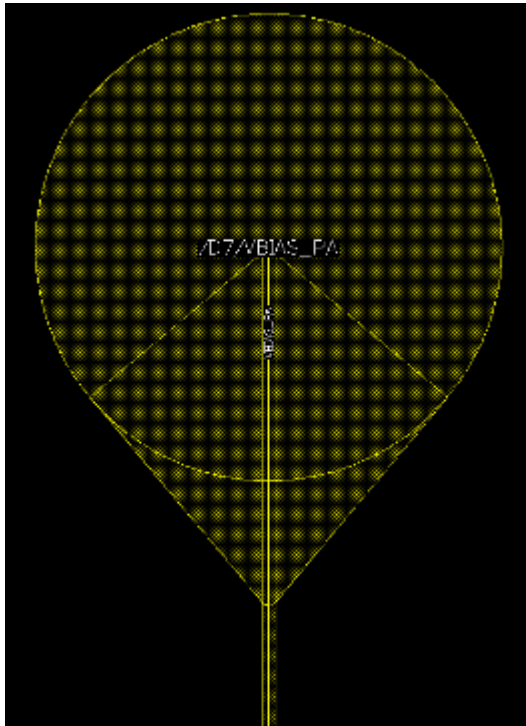
- e. To reconstruct a slotted polygon as a regular polygon that has no holes and also covers the slot shapes, choose the scope that you want from the drop-down list. The following figure shows the result when the *Reconstruct Slots* option is selected to reconstruct a slotted polygon shape into a regular polygon.



13. (Virtuoso RF Solution Only) Click the arrow next to the *Fillet* icon on the *Metal Density* toolbar. A drop-down menu displays.



Choose the scope you want to create fillets. The following figure shows how the fillet is created when the *Create Fillet All* option is selected.



For more information on fillet creation, see [Fillet Creation \(Virtuoso RF Solution Only\)](#).

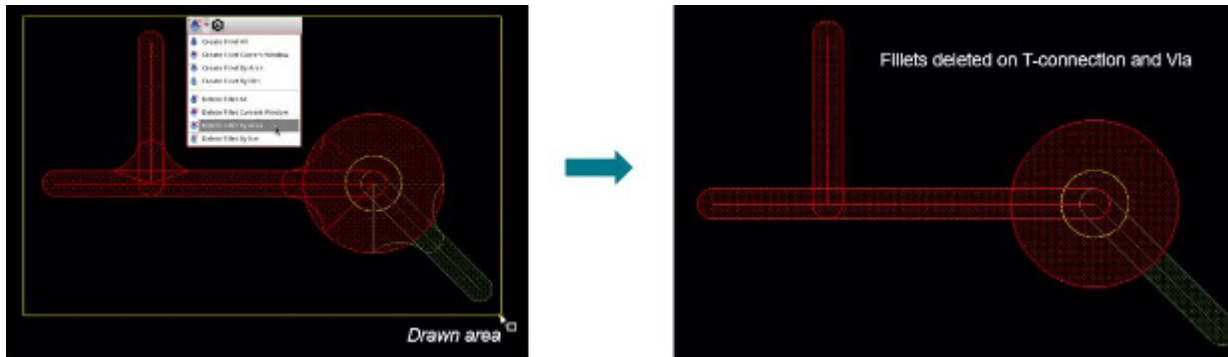
14. Click *Slotting* on the Metal Density Options form. For more information, see [Create Packaging Bus Form](#).

Deleting Fillets

To delete a fillet:

1. Click the arrow next to the *Fillet* icon on the *Metal Density* toolbar. A drop-down menu displays.
2. To delete a fillet on a curved shape, choose the scope you want, from the drop-down list.

The following figure shows the result when the *Delete Fillet by Area* option is selected.



Related Topics

[Metal Density Toolbar](#)

[Metal Density Options Form](#)

[IxShapeSlotting](#)

[IxConvertSlotToPolygon](#)

[Variables to Customize the Settings for Slotting](#)

[Specification Cells](#)

[Fillet Creation \(Virtuoso RF Solution Only\)](#)

Specification Cells

To handle contextual situations and to apply directives per layer, you can specify a specification cell. The specification cell is a cellview that contains, for each specified layer, the representation of the directional slot shapes and the turn slot shapes. The check box next to

Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

the *Specification Cell* field is used to enable or disable the field. The syntax to specify it in the Metal Density Options form is `libName/cellName/ViewName`.

The screenshot shows the 'Metal Density Options' dialog box with the 'Slot Geometry' tab selected. The 'Specification Cell' field is highlighted with a red box and contains a checkmark icon, indicating it is enabled. The 'Slotting' button is green, and the 'Close' and 'Help' buttons are at the bottom right.

When a specification cell is specified, all the geometric information for each layer is derived from the specified cell. For each layer, the following information is derived.

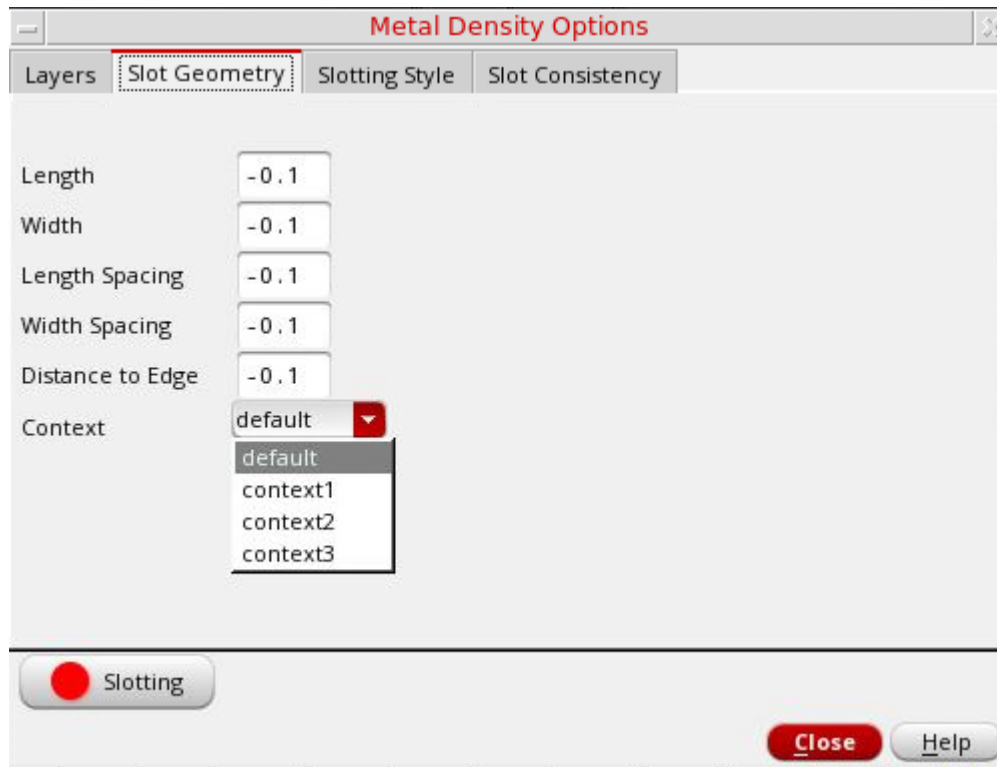
- A physical shape that represents the directional slot and the turn slot (optional) geometries. It implicitly defines the following slot parameters by its bounding box:
 - ❑ `slotLength`
 - ❑ `slotWidth`
- Optionally, some of the parameters in the specification cell are defined using:
 - ❑ The cellview properties
 - `slotStaggered`
 - `excludePins`
 - ❑ The directional slot shape properties
 - `slotToEdge`

Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

- lSpacing
- wSpacing
- widthThreshold
- lengthWidthThreshold

In case a list of contexts is defined using the `sltSetContextSpecificationCells` SKILL function, the *Context* field is available in the *Slot Geometry* tab of the Metal Density Options form. Instead of the string field for the *Specification Cell*, a drop-down list is available, which lists the defined contexts.

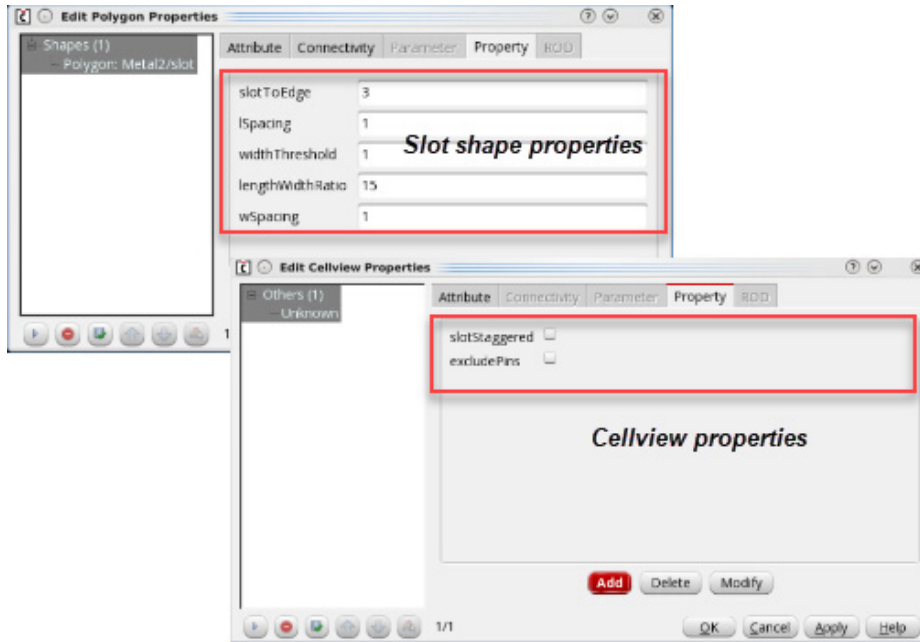


When a context is selected from the *Context* drop-down list, the slot geometry parameters are disabled. All the disabled parameters must be defined in the specification cell corresponding to the selected context. Else, the slotting cannot be completed and the status of the *Slotting* button is updated and the color is changed to Red.

Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

The following figures shows the cellview and the slot shape properties.

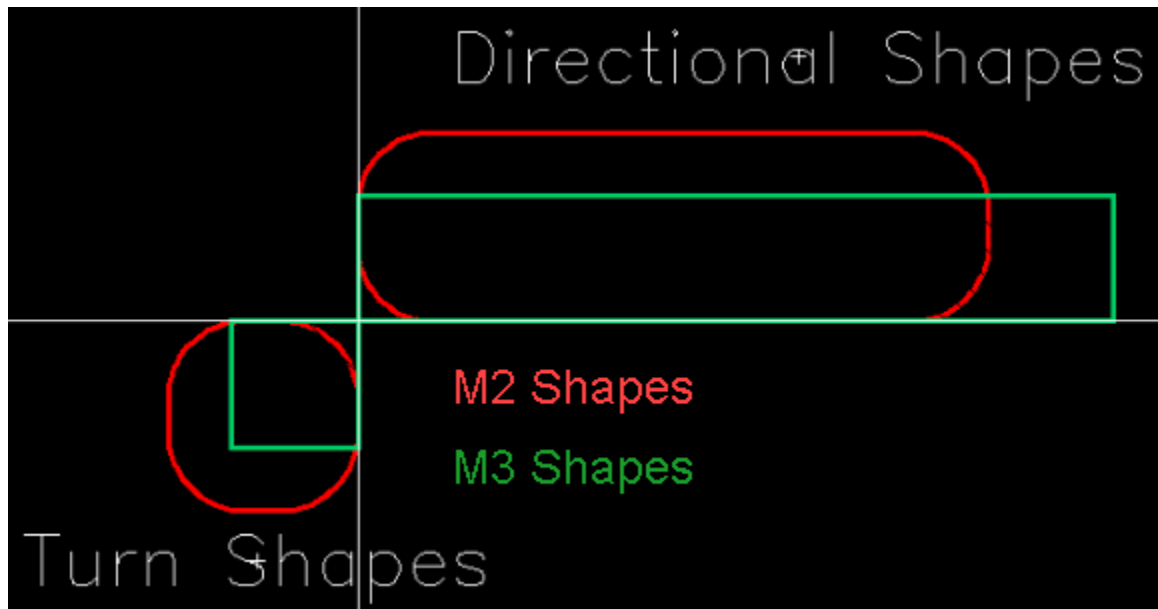


- The parameters derived from the specification cell take precedence and override the values that are available in the Metal Density Options form. If one of the above properties is not defined for the specification cell, then the values from the Metal Density Options form is used.
- The lower-left corner should be (0,0) for the directional slot shape and the upper-right corner should be (0,0) for the turn slot shape.
- The slot shape should be drawn in the direction of the current (from left to right).
- If the turn slot is not defined, then the width of the direction slot is used to adjust the size of the shape at turn (square).
- The purpose slot should be used.

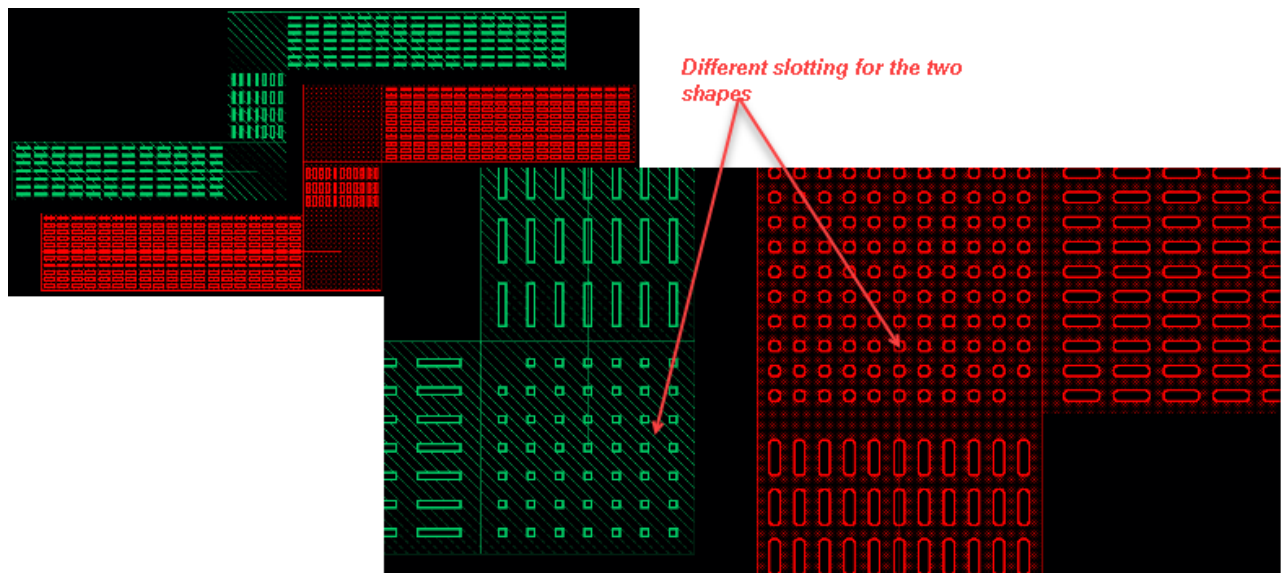
Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

The following figure shows an example of a specification cell.



The following figure shows the result of slotting when the *Slotting All* option is selected. As you can see, slotting is different on the two shapes.



Related Topics

[Metal Density Toolbar](#)

[Metal Density Options Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

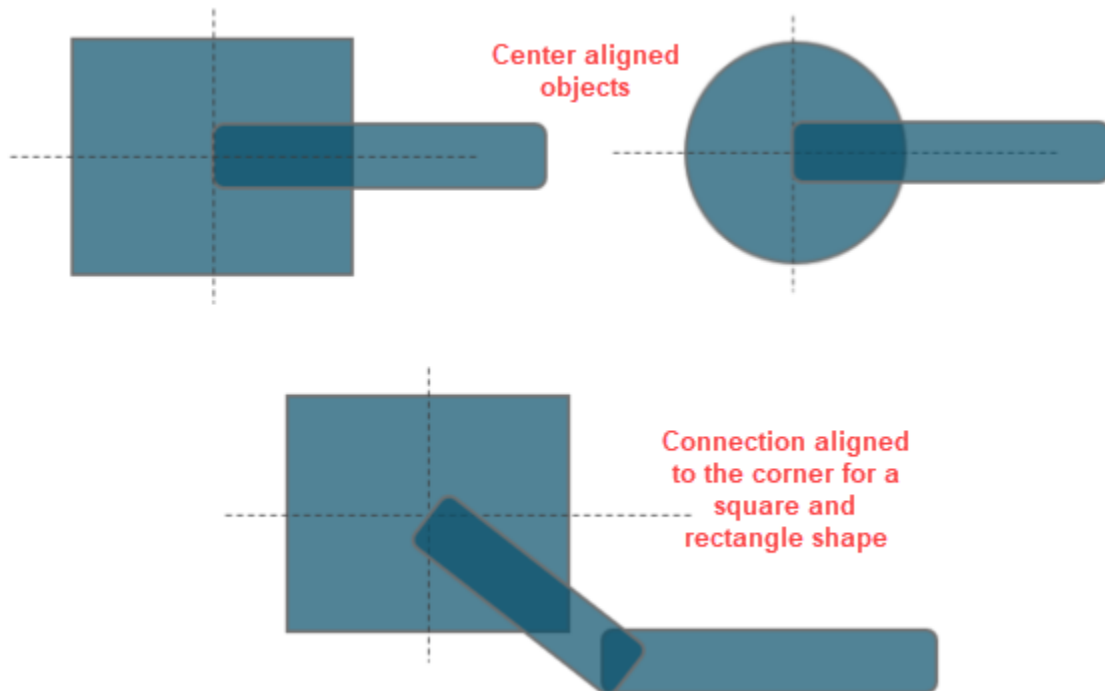
[Variables to Customize the Settings for Slotting](#)

Fillet Creation (Virtuoso RF Solution Only)

Fillets are an extra etch added to prevent acid traps on a design, which in turn improves manufacturability. In Virtuoso, fillets are created as curved polygons.

Some considerations for fillet creation are as follows.

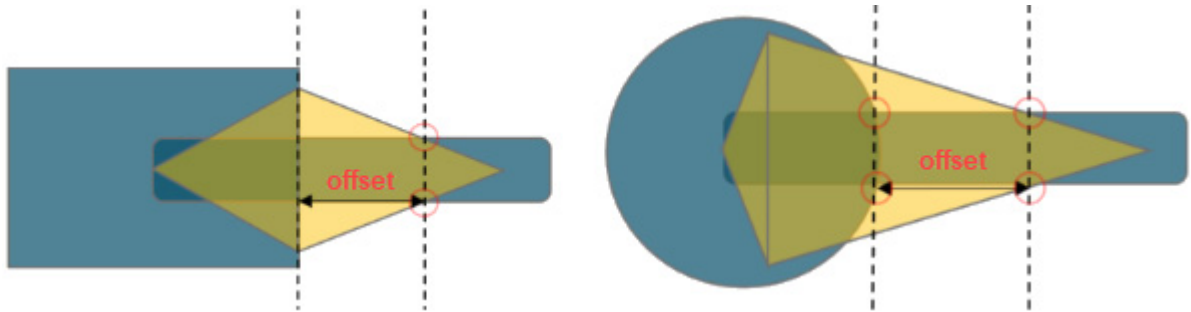
- Consider that all wires are always center aligned to the target objects. Also, for a square or rectangle target, the connection can be aligned to the corner.



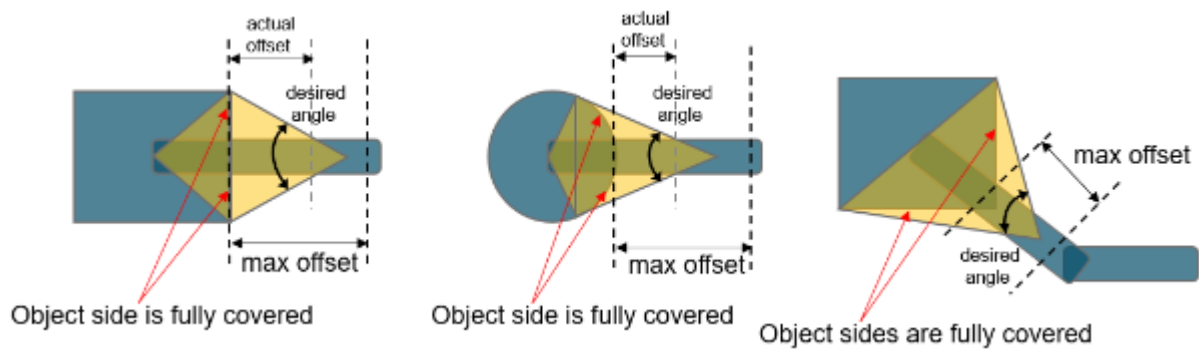
Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

- Offset distance is computed from the object side and the intersection between the wire edges and the fillet shape, as shown in the following figure. Also, fillet shapes are symmetric.



- When inserting a fillet, the object side is completely covered by the fillet shape. In addition, the desired angle is respected as long as the actual offset is less than the specified maximum offset.

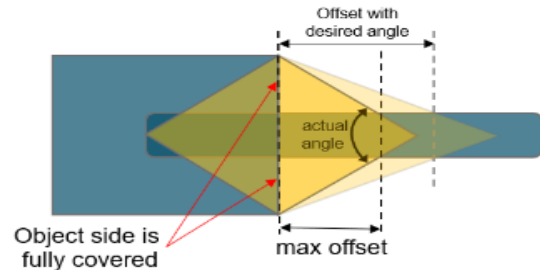
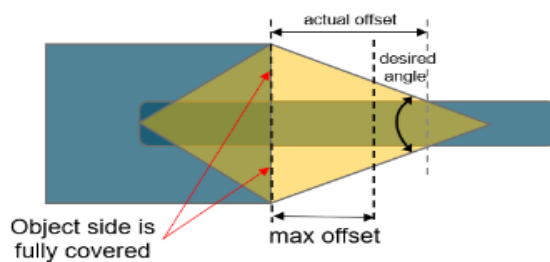


In cases where the maximum offset is not respected while trying to get the desired angle to fully cover the side object, there are two possible options:

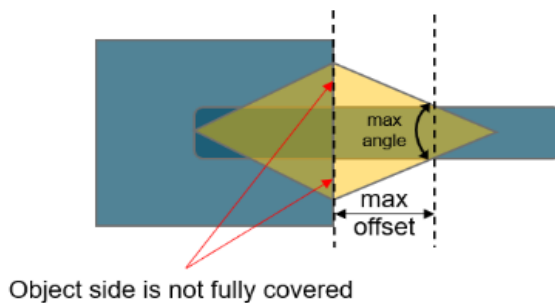
Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

- ❑ Object side can still be completely covered. This implies that the actual angle is greater than the desired angle.



- ❑ Object side cannot be completely covered. In this case, maximum offset and maximum angle are used.



Specifying the Fillet Options

You can specify the following in the Fillet tab of the Metal Density Options form.

- The unassigned shapes (without any connectivity) to be filtered out.
- The type of fillet that is to be created, linear or curved.
- The parameters used to create the fillet shape.
- The type of objects that a fillet can be created for: *Paths*, *Bond Fingers*, *Vias*, *Pins*, *Pad without drills*, or *T connections*.
- The type of fillet shape to be created (along with the maximum size to control the shape creation): *Circle*, *Square*, *Rectangle*, or *Oblong*.

Related Topics

[Fillet Tab](#)

[Fillet Creation Between Curved Path and Other Objects](#)

[Creating Fillets](#)

Support for Slot Shapes in DRD and EAD

The added slot shapes are supported both by the DRD checker and EAD. For DRD, the maximum density checks support the slot shapes.

In EAD, the R extraction as well as the EM analysis consider the slot shapes using the [layoutEAD.pex doSlotting](#) environment variable.

Related Topics

[Metal Density Toolbar](#)

[Metal Density Options Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

[Variables to Customize the Settings for Slotting](#)

Virtuoso Interactive and Assisted Routing User Guide

Managing Metal Density

Interactive Colored Routing

As technology advances and allows smaller transistors, layout designers want to create more complex circuits with an increasing number of geometries on the same silicon die. As a result, double- or triple- (and soon n-tuple) patterning techniques that rely on coloring information are being widely used for processes below 20nm. These techniques allow a layout designer to place geometries as close as possible to each other. Interactive colored routing is an enhancement that Virtuoso offers in this direction.

This chapter describes how to set options for interactive colored routing.

- Interactive Coloring for Create Wire
 - ❑ Creating a Colored Wire
 - ❑ Inserting Via on Colored Wires
 - ❑ Tapping Colored Objects
- Interactive Coloring for Create Bus
 - ❑ Creating a Bus in an Empty Space
 - ❑ Creating a Bus by Tapping Pins
 - ❑ Creating a Bus While Working with Colored WSPs
- Interactive Coloring for Create Stranded Wire
 - ❑ Specifying the Color Mode in Stranded Wire
 - ❑ Changing the Color of Stranded Wire
 - ❑ Changing the Stranded Wire Color Mode
 - ❑ Creating a Stranded Wire by Tapping Pins
 - ❑ Creating a Stranded Wire While Working with Colored WSPs
 - ❑ Avoiding Shorts with Existing Blockages on Colored WSPs
- Supporting End-to-End Spacing for Colored Shapes

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

- [Interactive Coloring for the Stretch Command with Colored WSPs](#)
- [Color Locks While Editing a Wire](#)
- [Inserting TrimMetal Layers in Interactive Colored Routing](#)
 - [Supporting Colored TrimMetal Layers](#)
 - [Supporting Colored TrimMetal Layers for the Stretch Command](#)
 - [Supporting Colored TrimMetal Layers for Create Bus](#)
 - [Supporting Patch Metal Shapes](#)

Interactive Coloring for Create Wire

Interactive wire editing supports Palette MPT (Multi-Patterning Technology) feature and lets you create wires with layer, color, and lock state information.

Video

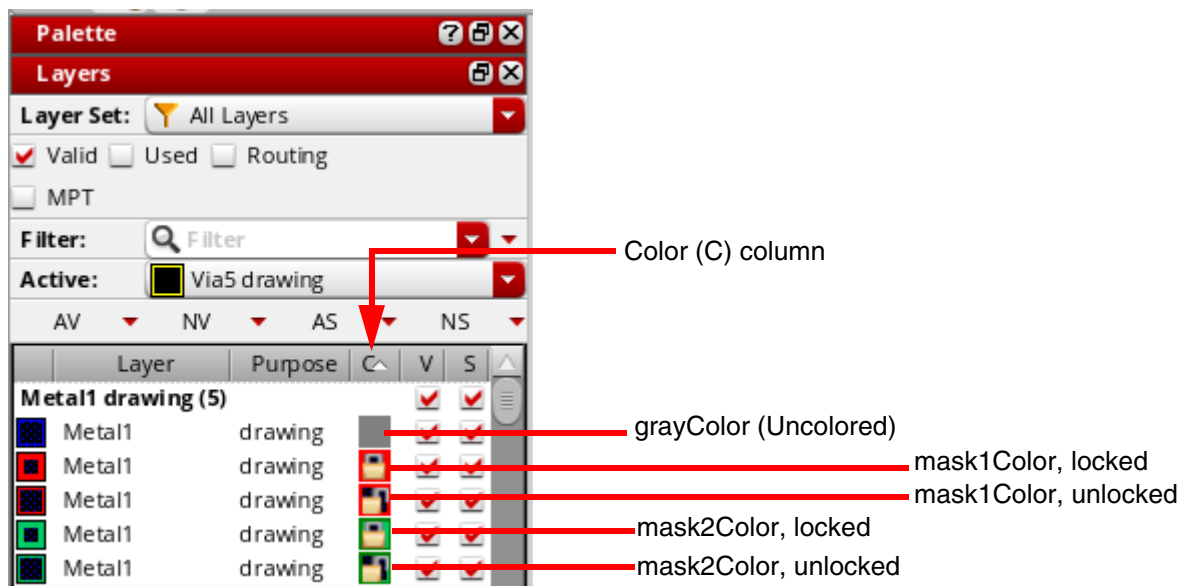
To view the demonstration on interactive coloring for create wire, see [Interactive Colored Routing - Part1 \(Using the Create Wire Command\)](#) video.

Creating a Colored Wire

The steps to create a colored wire are as follows:

1. Enable *MPT Support* in the Palette assistant. For more information, see [MPT Support](#) in the Virtuoso Layout Suite documentation.
2. (Optional) Select the *MPT* checkbox.
3. Set as active a layer-purpose pair with the required color and lock status.

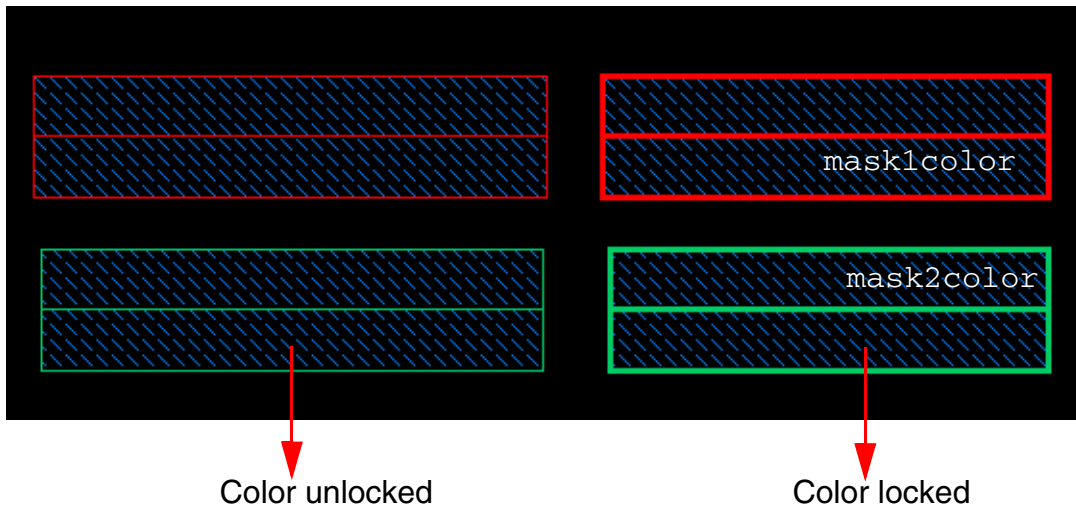
Notice that the icons listed in the Color (C) column for each layer-purpose pair provide color as well as lock information.



Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

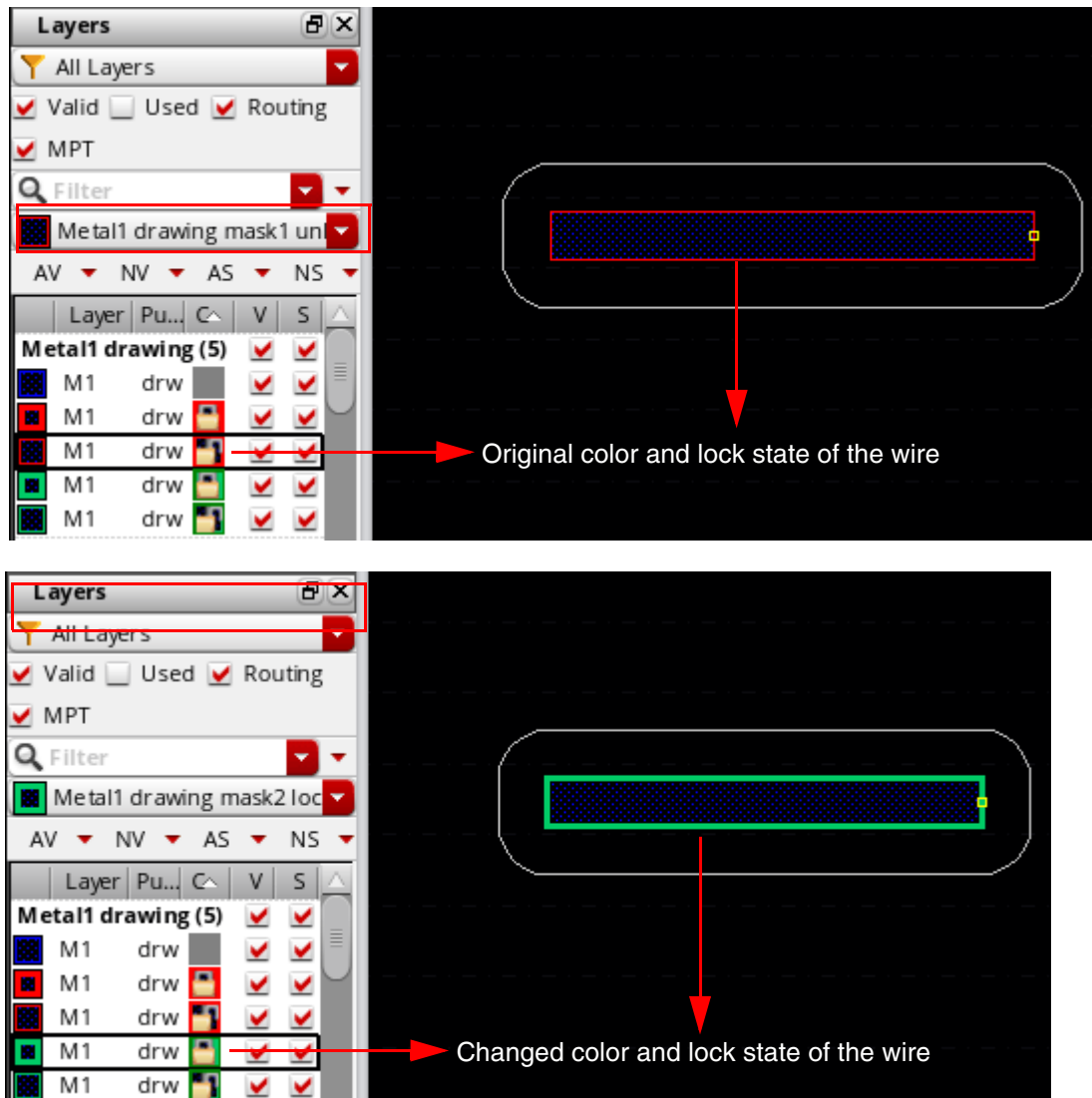
4. Create a wire. The color and lock state are automatically set for the shape, as shown in the following figure.



Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

While creating a wire, you can reset the color and lock state of the wire by selecting a layer-purpose pair with another maskColor and lock state from the Palette, as shown in the following figure.



When you create a wire with the shape color as gray, the default mask color defined for the layer is used for the created wire. To do this:

- Choose a layer with gray color from the Layers palette.
- Start creating a wire from an empty space. The newly created wire uses the layer default mask color defined for the layer.

You can change the color of the wire by selecting a layer-purpose pair with another `maskColor`. If you create a wire on a track, the color of the track and not the default color of the layer.

Inserting Via on Colored Wires

You can insert a via while creating a colored wire by selecting *Via up to <layer name>* from the *Create Single Wire* context-sensitive menu. By default, the via top and bottom shapes inherit the color information from the wire being drawn. Similarly, the via inherits the color lock status from the wire being drawn.

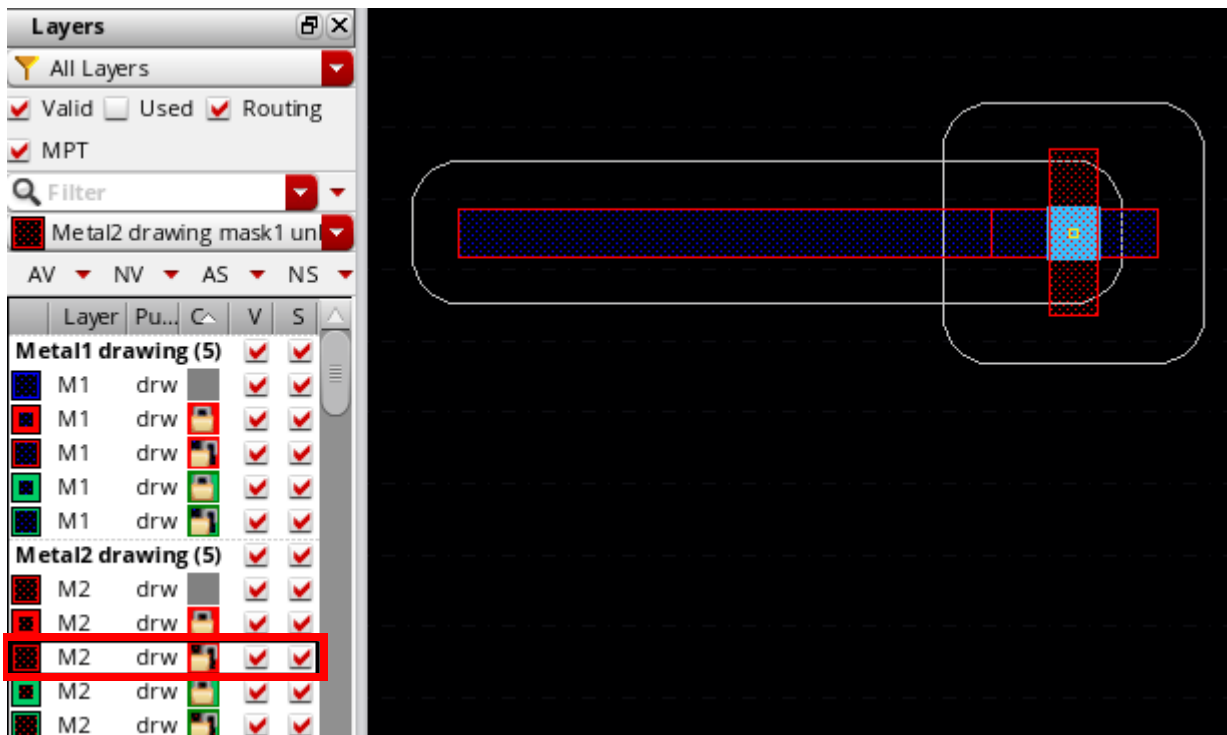
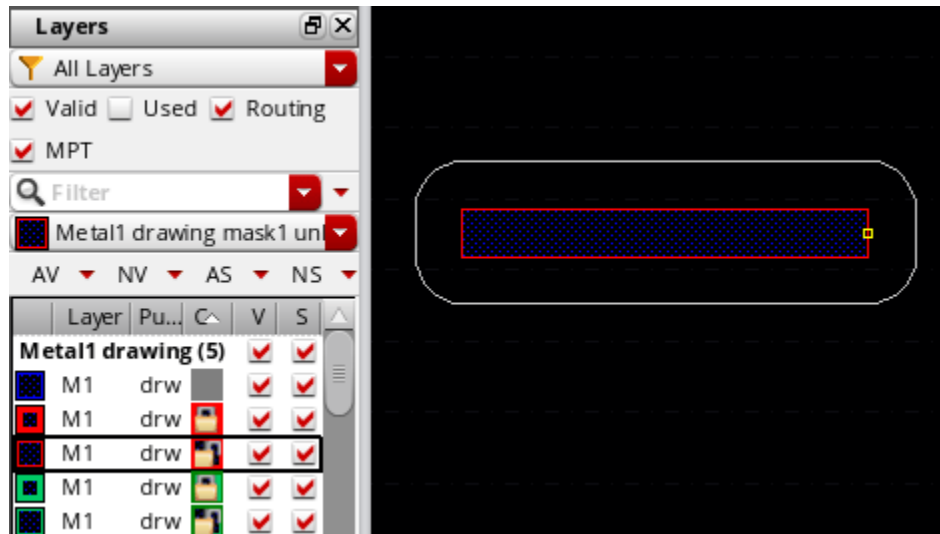
Note: The via cut shapes are kept uncolored (gray) and their color cannot be changed using the Palette MPT feature.

The following figure shows an example where you create a wire on M1 metal layer with `mask1Color`, and then insert an M1-M2 via by selecting *Via Up To M2*. The M2 via shape

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

and the wire on M2 both have `mask1Color` color set. The active layer in the Palette is reset to `Metal2` drawing, which is the output layer of the inserting via.

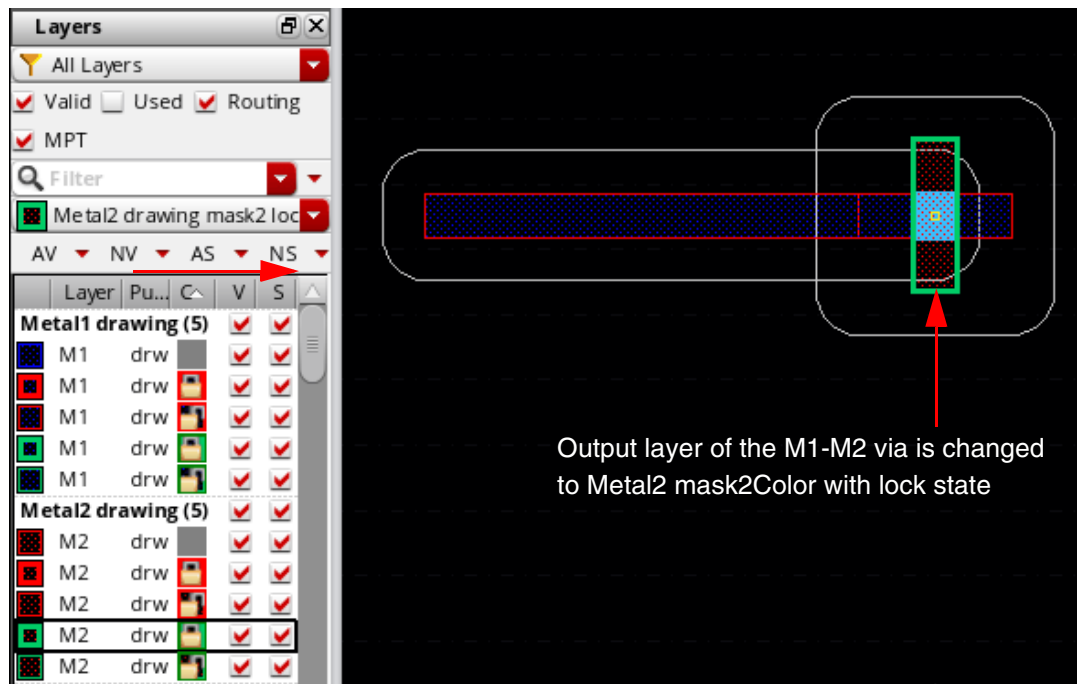
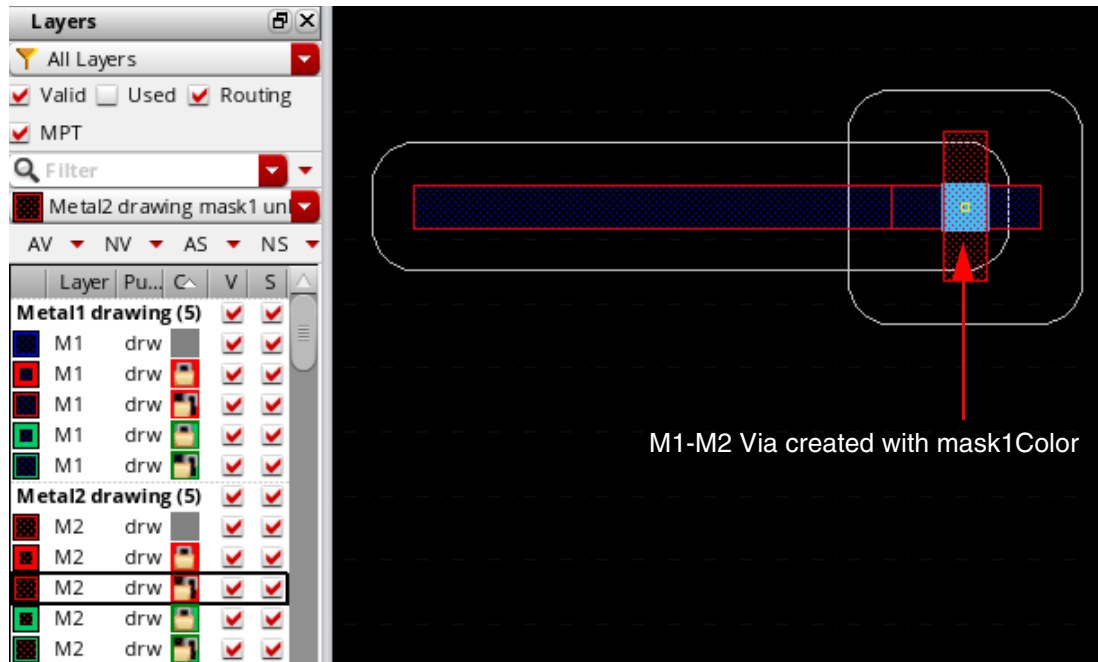


By using the Palette MPT options, you can change the color and lock status of a via before you place it. This changes the color and lock information of the output layer of the via enclosure. The following figures show that the active layer in the Palette is reset to `Metal2`

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

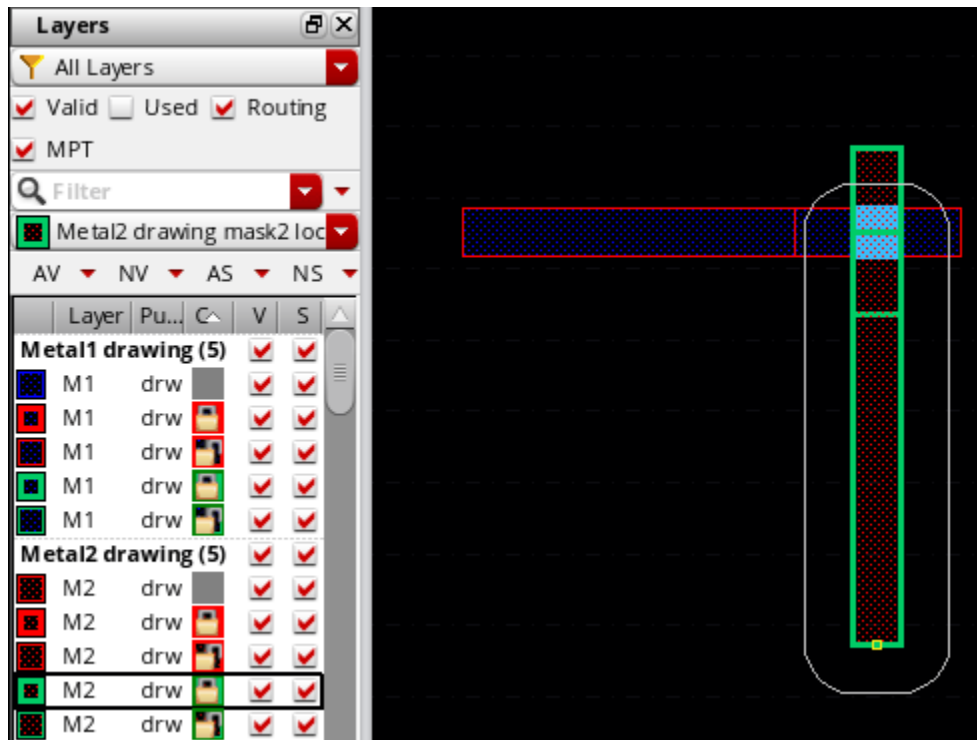
drawing, which is the output layer of the inserting via. It also shows a change in the via color and lock state from `mask1Color` `unlock` to `mask2Color` `lock`.



Virtuoso Interactive and Assisted Routing User Guide

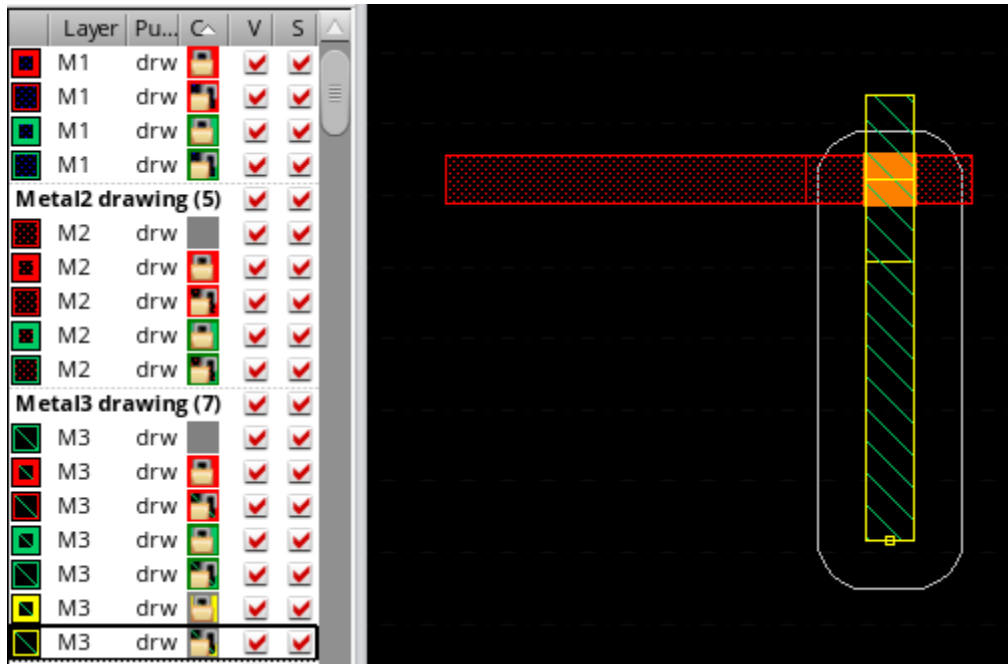
Interactive Colored Routing

After placing the via, you can continue to create a wire starting from the via on the changed metal layer. The wire inherits the color of the output layer of the via. You can see that the new wire is created in `mask2Color` with lock state in the following figure.



Similarly, after the via has been placed, you can change the color and lock information from the Palette for both the output layer via enclosure and the wire currently being drawn. The

following figure shows that the color information of the output layer via enclosure and the new pathSeg on the M3 layer has changed from `mask2Color` to `mask3Color`.



Tapping Colored Objects

The tapping behavior of an object is dependent on the *Auto Tap Wire* option in the Layout Editor Options form. If the *Auto Tap Wire* option is deselected, the wire inherits the color and lock status of the active layer in the palette, independent of the locked status of the starting shape. However, if the *Auto Tap Wire* option is selected, the wire inherits the color and lock status from the starting shape. There is no precedence of the locked status from the palette anymore. In addition, if you want to change the color or the lock status of the wire that you have started from an existing object, you can do it after clicking the starting object and then choose another color or locked status from the palette.

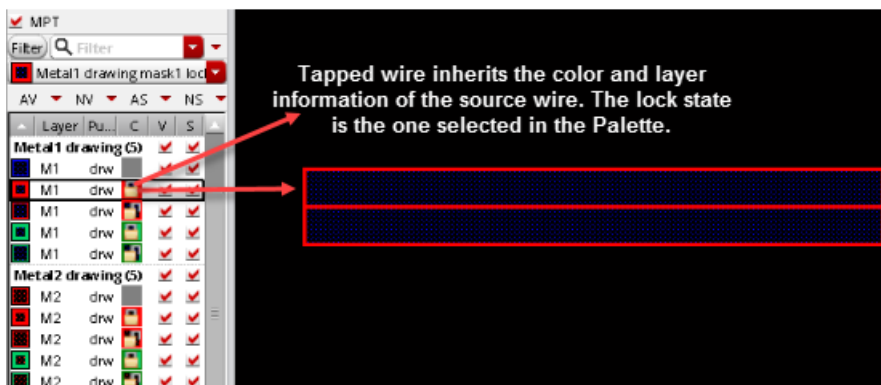
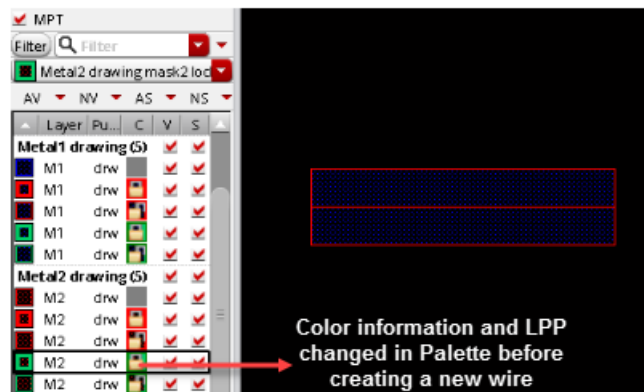
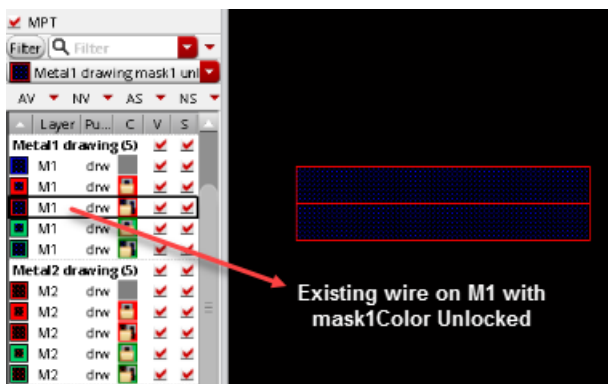
This section covers the tapping behavior of various colored objects.

- Tapping from Colored Wires
- Tapping from Colored Pins
- Tapping from Colored Width Spacing Patterns (WSPs)

Tapping from Colored Wires

Interactive wire editing supports automatic tapping from colored wires. When you create a wire from an existing wire, the new wire taps and inherits the width, color information, layer-purpose pair, and net information of the source wire. The lock status of the new wire is inherited from the existing wire if it is in the locked state.

The following figure displays an example where a new wire is created from an existing wire where the lock state of the existing wire is unlocked but the lock state in the Palette is locked. In this case, the tapped wire inherits the lock state from the Palette.



Tapping from Colored Pins

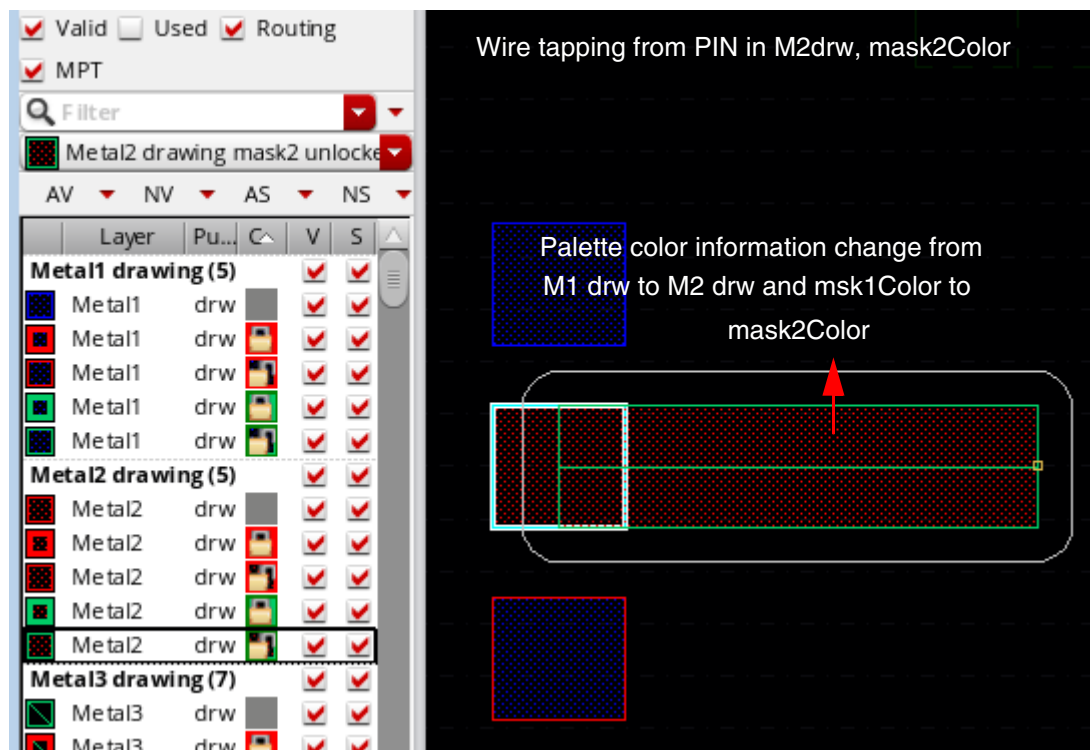
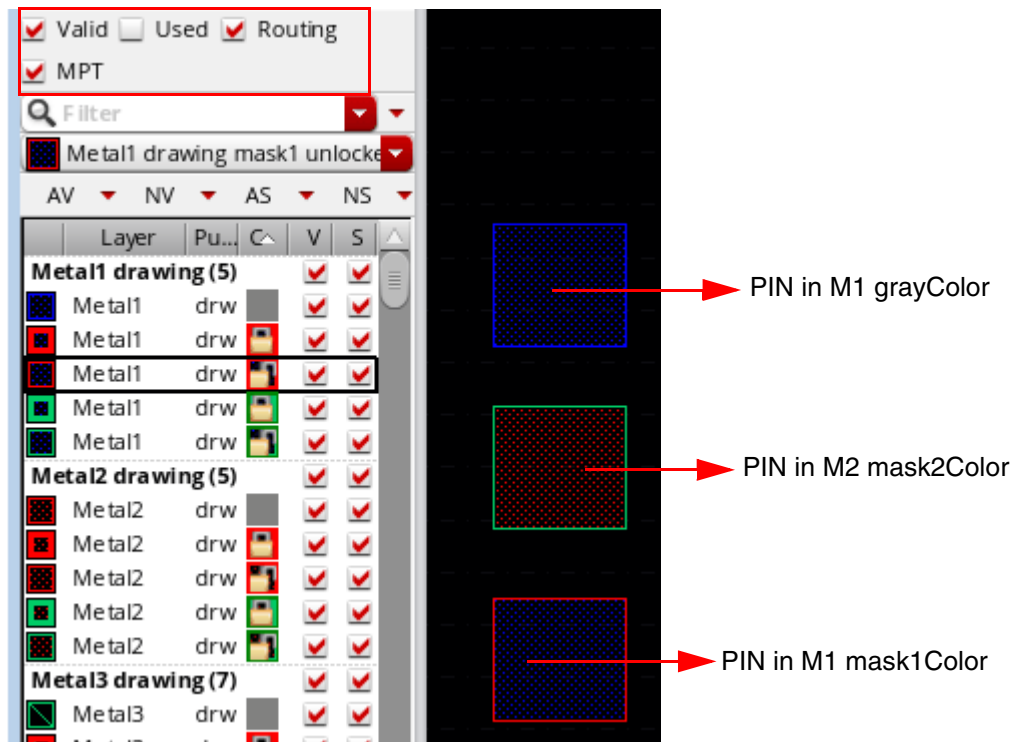
Interactive wire editing supports automatic tapping from colored pins. When you create a wire from an existing pin that has a color assigned to it, the wire automatically inherits the width, color information, layer-purpose pair, and net information of the source pin. The lock information that is selected in the Palette is considered.

The following figure displays an example where a wire is created by tapping an existing pin (`M2 drw, mask2Color`). In this case, the wire is created on `M2 drw` and has

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

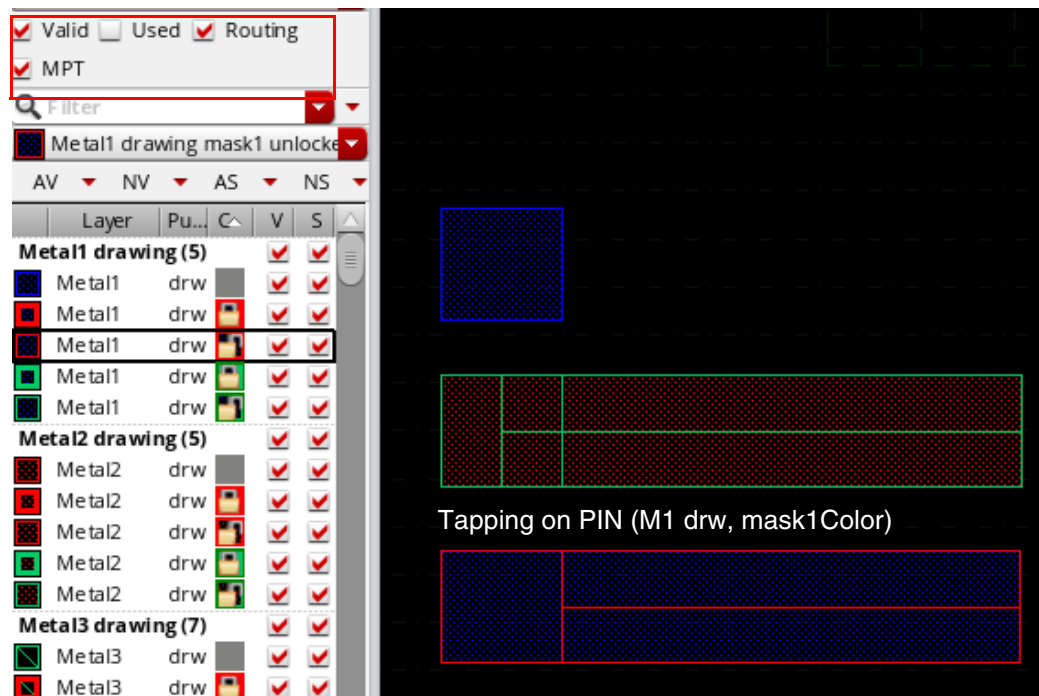
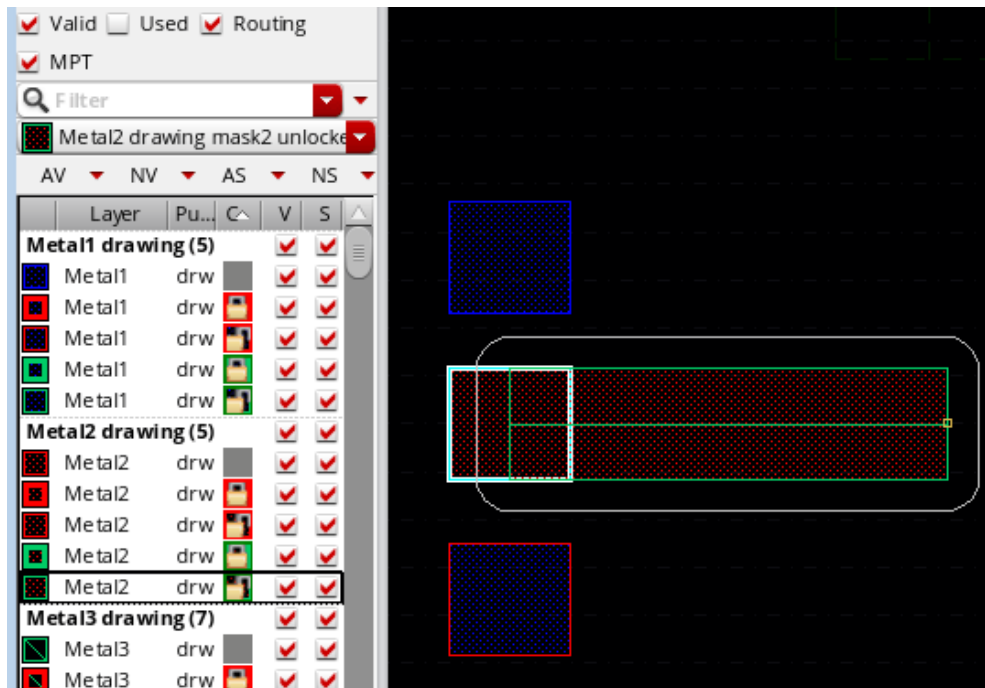
mask2Color. You can see that the Palette layer information has changed from M1 drw to M2 drw and the color has changed from mask1Color to mask2Color.



Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

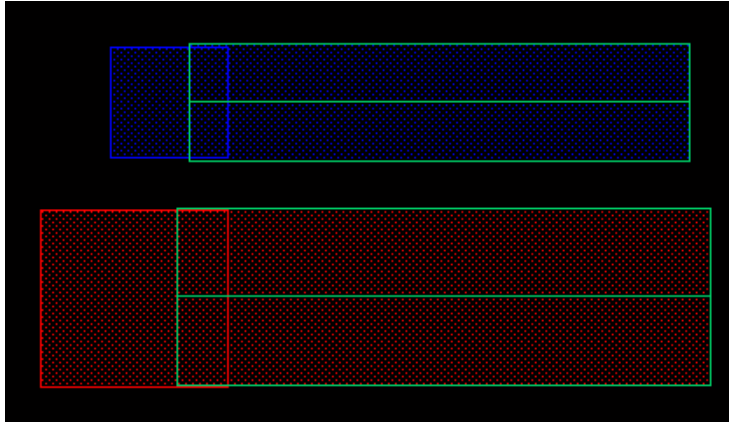
If you now create a wire from a pin on M1 drw with `mask1Color`, the wire automatically inherits the layer and color information from the pin and the color information in the Palette changes, as shown in the following figure.



Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

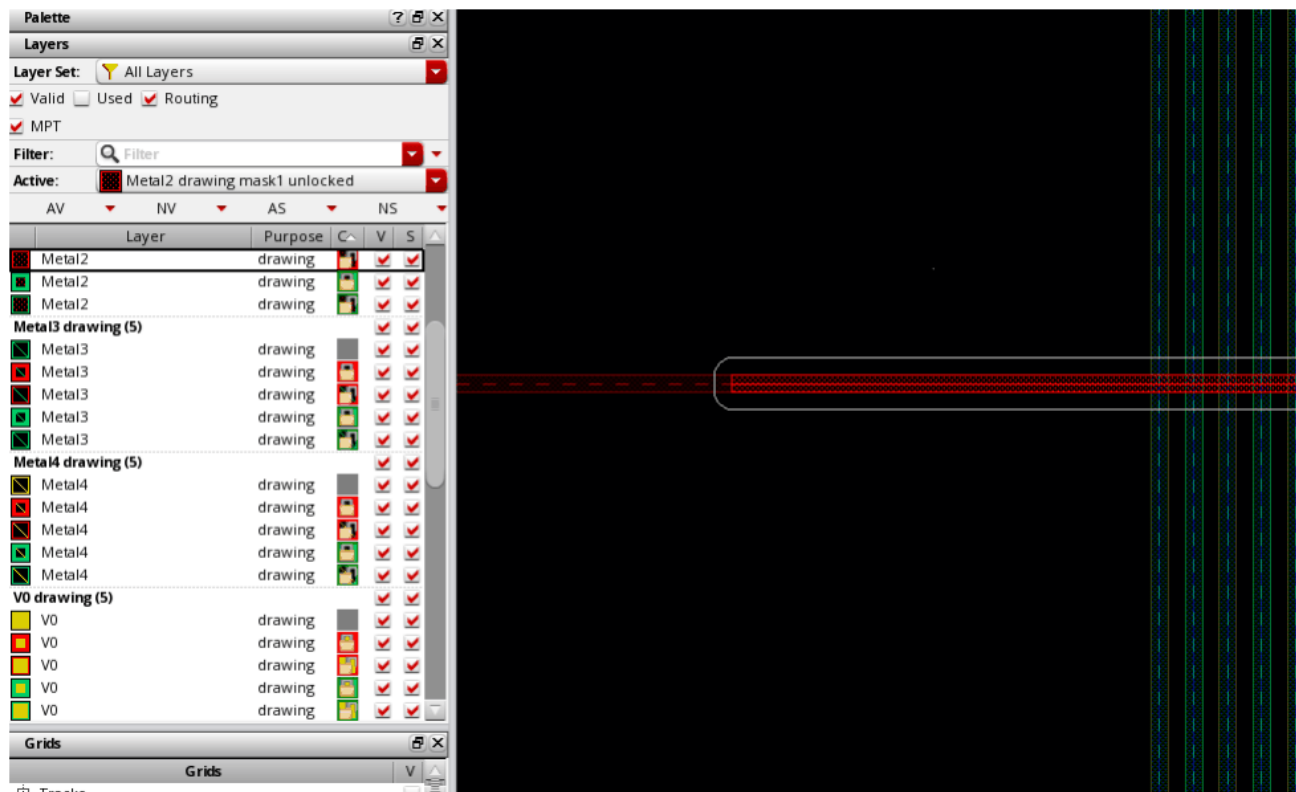
However, when you create a wire from a pin with color set to gray, it is created with the color that is currently set as active in the Palette, but on the same layer as the pin, that is, the wire inherits only the layer information from the pin. The following figure shows wires created from the two pins that are on different metal layers when the pin color is set to gray.



Color information from the Palette
and layer information by pin tapping

Tapping from Colored Width Spacing Patterns (WSPs)

When you create a wire from a WSP that has a color assigned to it, the wire taps and inherits the width, color information, and layer-purpose pair from the WSP, as shown in the following figure.

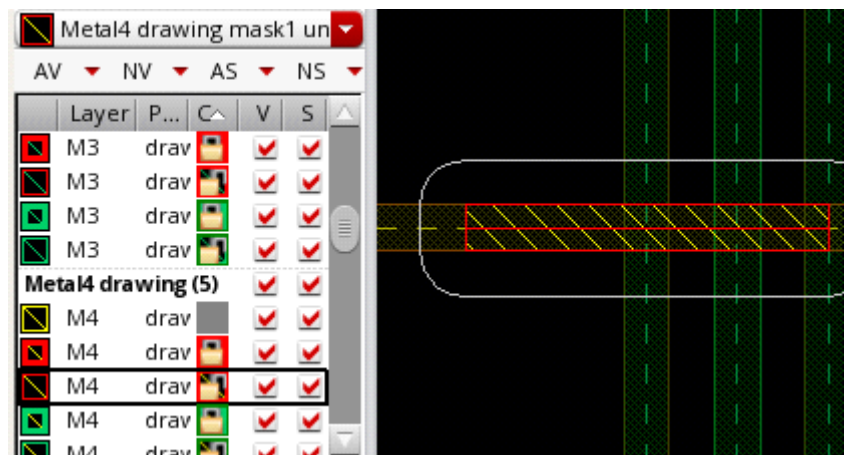
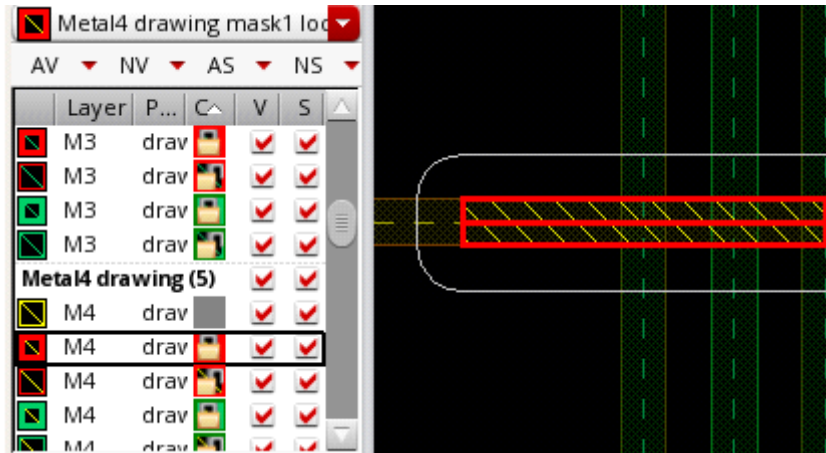


In this scenario, you cannot change the color of the wire from the Palette; the color of the WSP overrides the color that is currently set as active in the Palette. However, if the WSP does not have a color assigned to it, the wire is created with the color information from the Palette. In addition, the wire does not inherit the unlock state of the colored WSP. It inherits the color lock

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

status from the Palette. For example, if the color lock state is changed in the Palette, the wire automatically inherits the new lock state from the Palette, as shown in the following figure.



Interactive Coloring for Create Bus

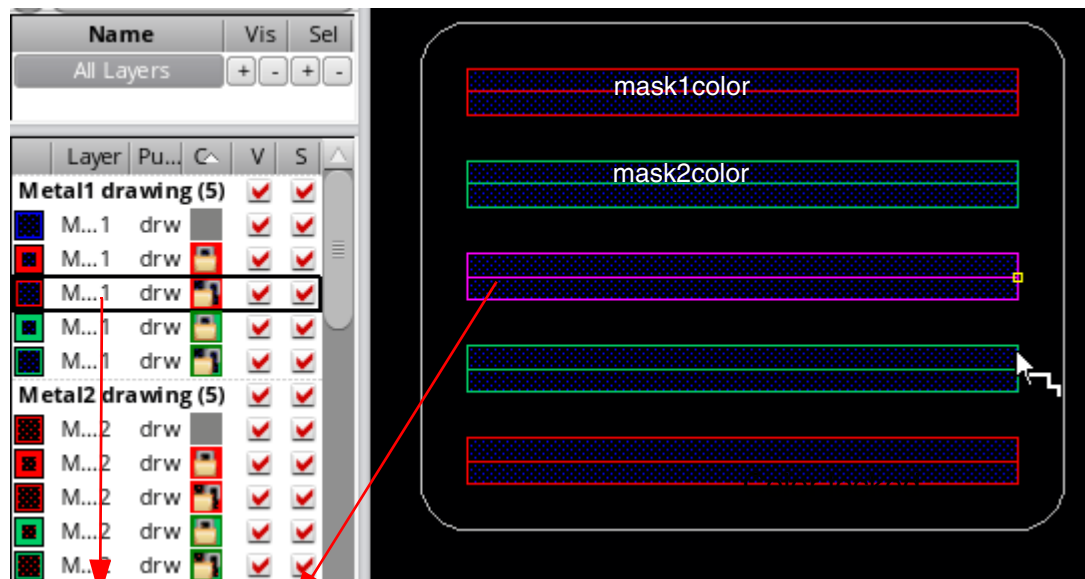
The *Create Bus* command supports interactive coloring and lets you create a single layer bus with color and lock state information. You can create a colored bus starting in an empty space or by tapping the objects, such as pins, pathSegs and buses, and colored WSPs.

- Creating a Bus in an Empty Space
 - Specifying the Bus Color Mode
- Creating a Bus by Tapping Pins
- Creating a Bus While Working with Colored WSPs

Creating a Bus in an Empty Space

To create a bus with color in an empty space in the layout:

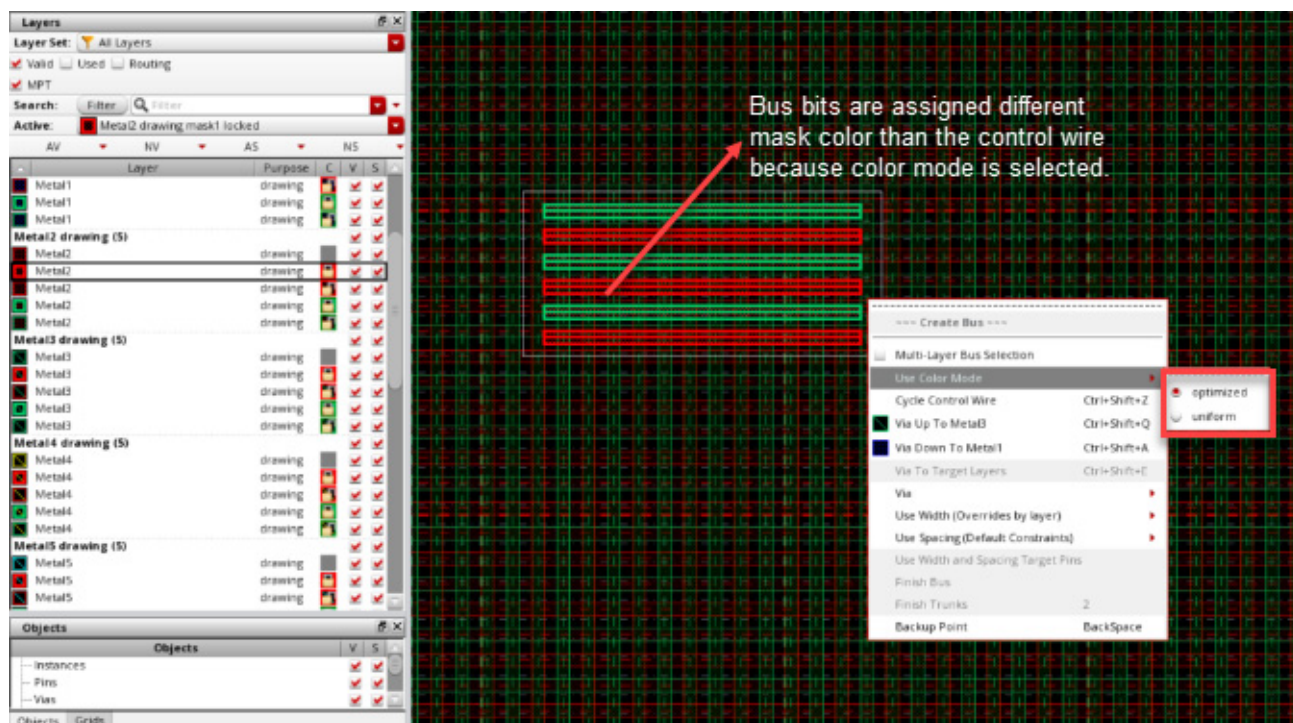
1. Enable *MPT Support* in the Palette assistant. For more information, see [MPT Support](#) in the Virtuoso Layout Suite documentation.
2. Set as active a layer-purpose pair with the required color and lock status.
3. Choose *Create – Wiring – Bus* to run the *Create Bus* command.
4. Click in an empty space in the canvas to start creating the bus. The control wire acquires the color and lock state from the active layer in the Palette. The color of the other bus bits depends on the bus color mode, which can be set to *uniform* or *optimized*. For more information about bus color mode, see [Specifying the Bus Color Mode](#). In the figure below, the bus is colored based on the *optimized* mode, starting from a control wire with mask1Color, Unlock.



Color of the control wire (mask1color, unlock)

Specifying the Bus Color Mode

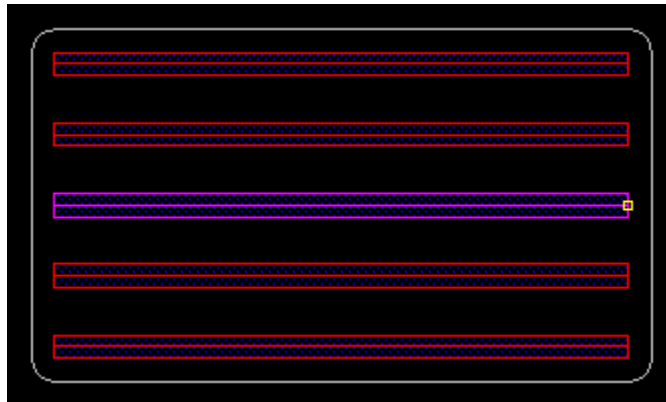
You can use the options available on the *Use Color Mode* menu to control the color mode of the bus. To access the color mode options, right-click the bus being created and point to *Use Color Mode*. You can either select the *optimized* or the *uniform* color mode.



Uniform Color Mode

When *uniform* is selected, all bus bits are assigned the same mask color as that of the control wire. The mask color assigned to the control wire and other bus bits is the same as the active layer in the Palette. The spacing between two adjacent bus bits on the same mask is

determined by `minSpacing(sameMask)`. The following figure shows the bus bits in the *uniform* color mode.



Optimized Color Mode

When you select *optimized* color mode, the control wire acquires the same mask color as that specified for the active layer in the Palette. The other bus bits acquire the following coloring pattern:

- The bus bits above the control wire acquire the mask color as `maskColor(controlWire)+1`, `maskColor(controlWire)+2`, and so on. This pattern continues until the highest mask color available for the selected layer-purpose pair is assigned to a bus bit. The pattern is then repeated starting at the mask color of control wire.
- The bus bits below the control wire acquire the mask color as `maskColor(controlWire)-1`, `maskColor(controlWire)-2`, and so on. If the control wire has the lowest available mask color assigned to it, then the bus bits below the control wire are colored starting with the highest mask color available for the layer on which the bus is being placed. The pattern is then repeated for the remaining bus bits.

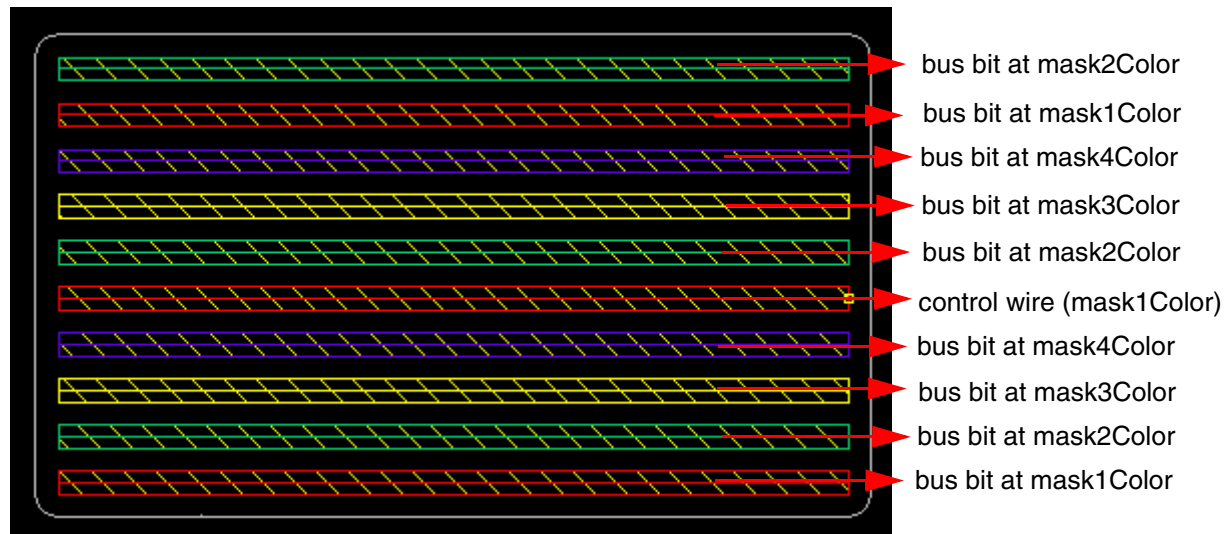
In *optimized* color mode, the spacing between two adjacent bus bits on different masks is determined by `minSpacing(diffMask)`. The *optimized* is the default color mode.

Let us consider an example where the active layer-purpose pair is Metal4 Drawing and the control wire is assigned `mask1Color`. The highest available mask color for Metal4 Drawing is `mask4Color`.

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

The following figure shows the color pattern that is assigned to the bus bits in a bus with 10 bits.

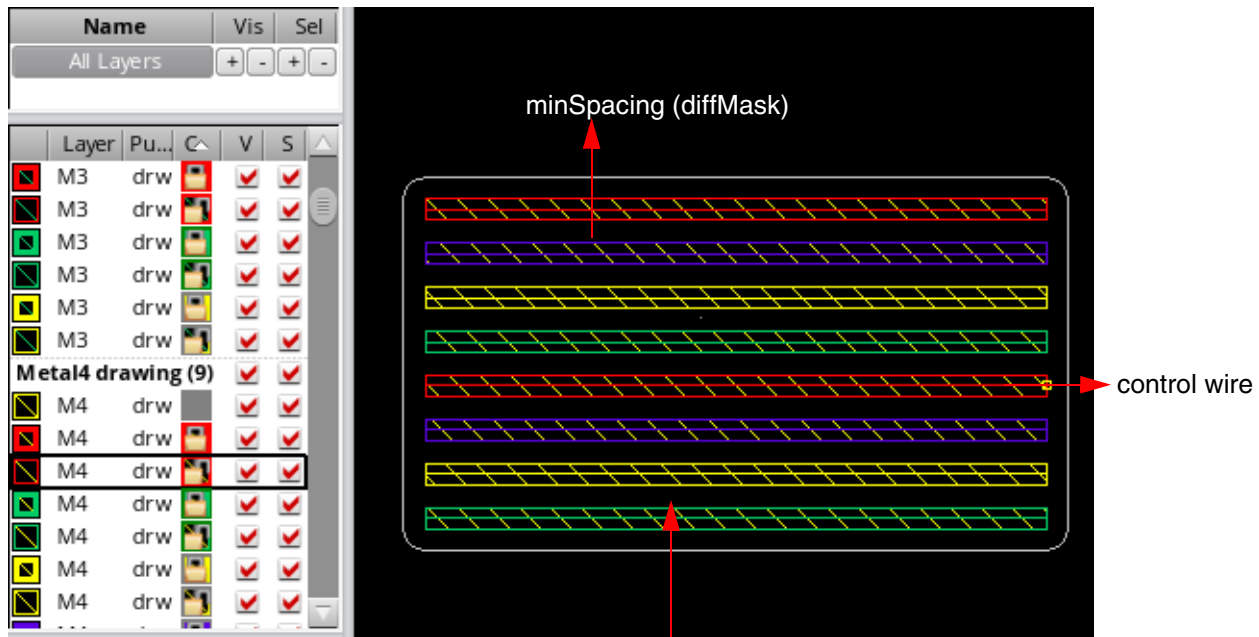


Changing Bus Color Mode

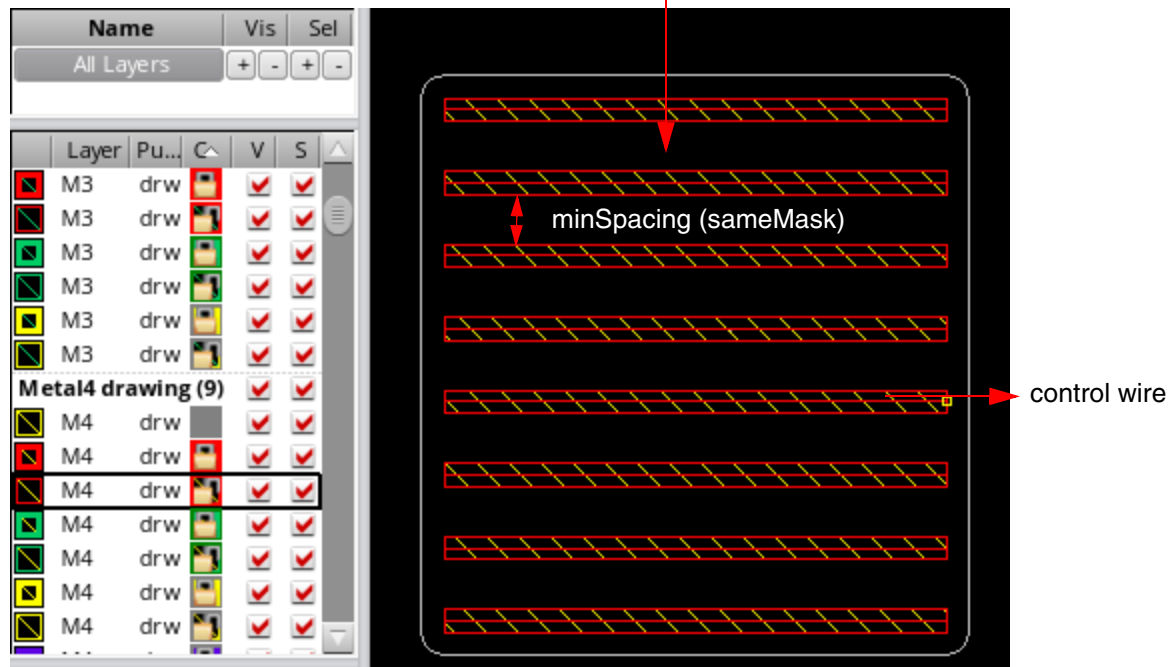
When the bus color mode is changed, the bus bits are recolored. The spacing between the bus bits is recomputed based on the bus color mode that is set. The following figure shows how the color and the spacing of bus bits changes when the color mode is changed from *optimized* to *uniform*.

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing



color mode changes from to uniform



Changing Color of Bus Bits

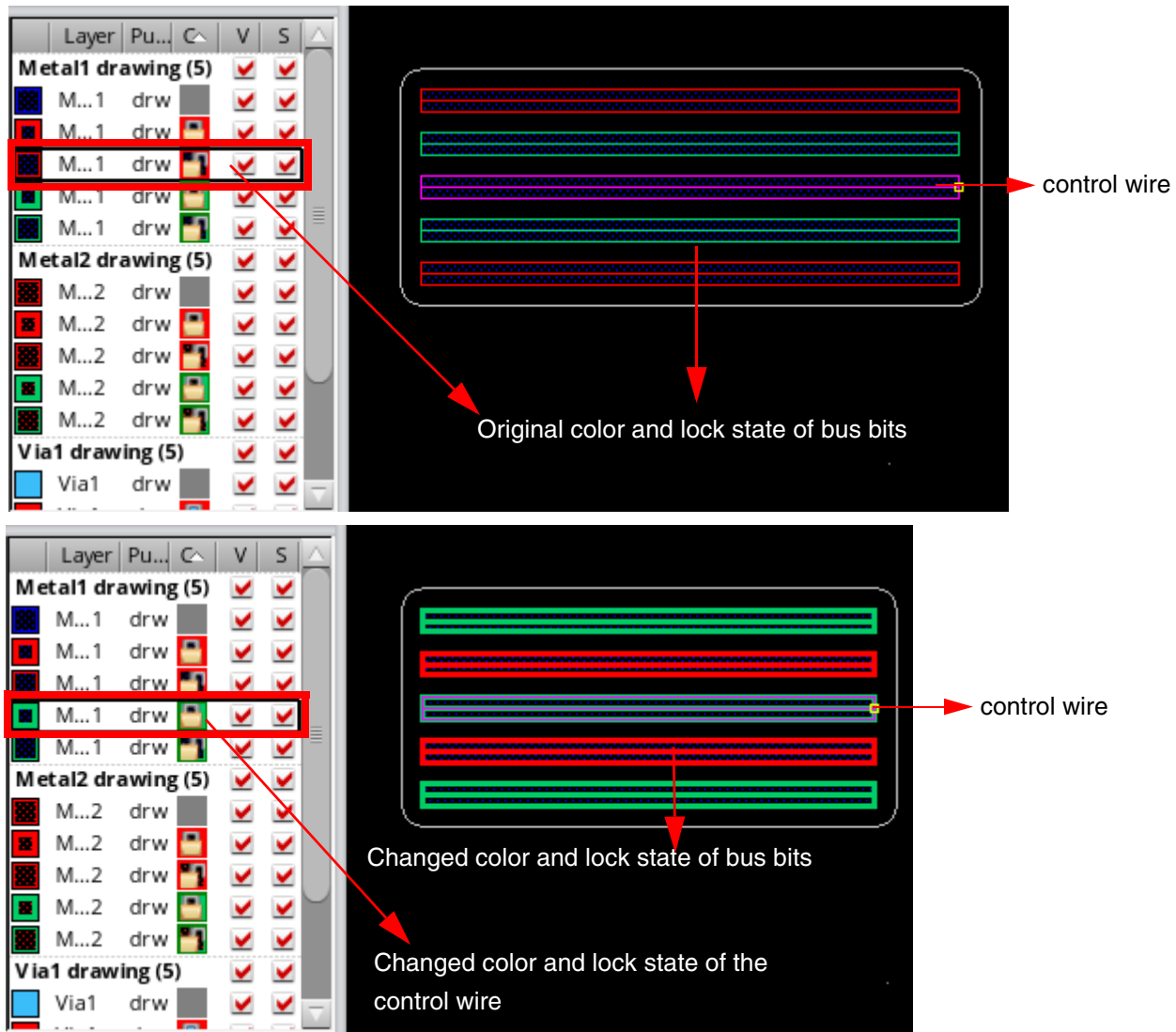
While creating a bus, you can apply a different mask color to the control wire, which, in turn, resets the color of the remaining bits in the bus. You can do this by resetting the active layer-

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

purpose pair in the Palette to a different mask color, which resets the color and lock information of the control wire and also of the remaining bus bits. You can also reapply the color mode, if required.

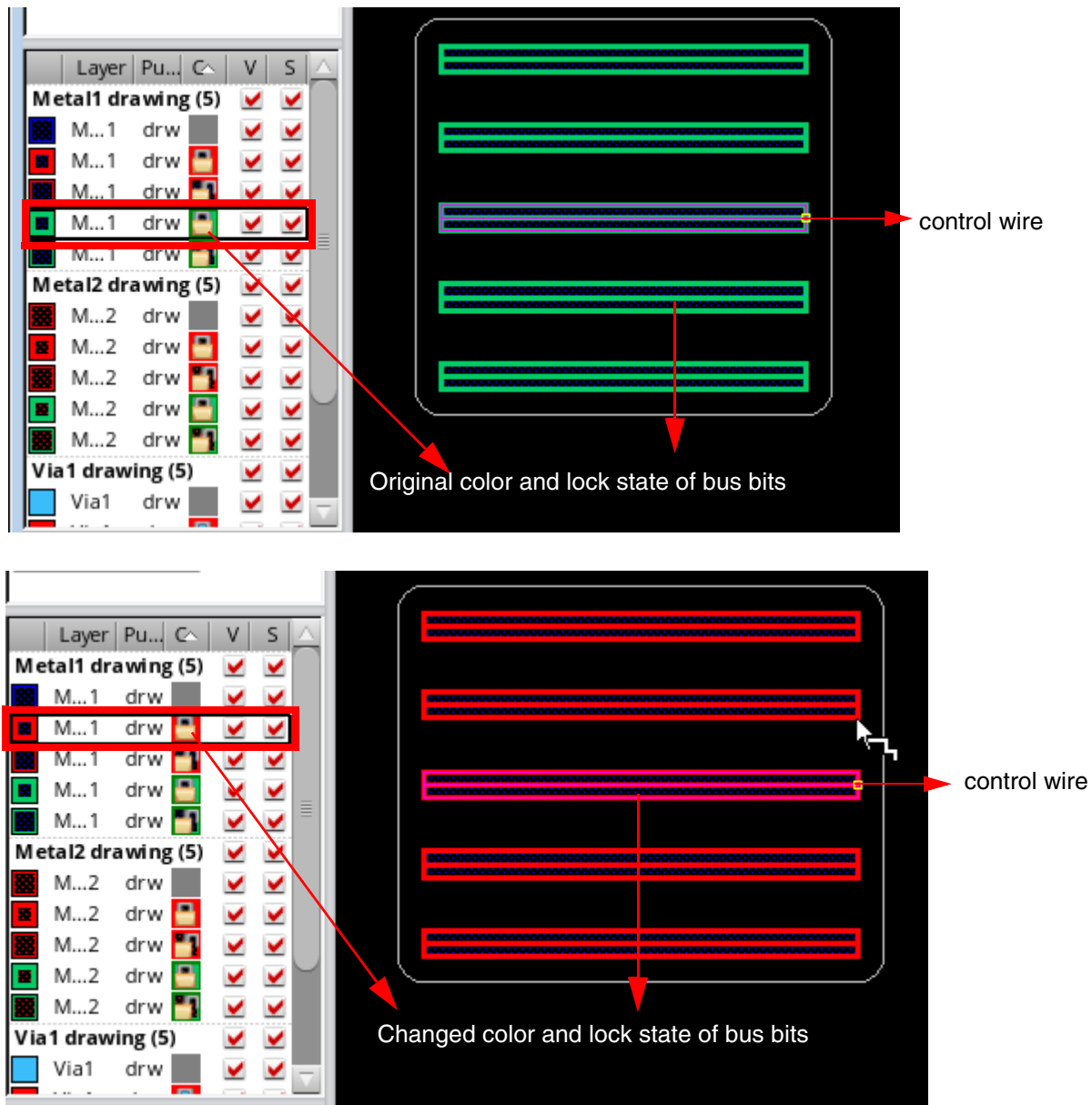
The following figure shows the change in color and lock state in the *optimized* bus color mode. You can see here that the control wire has acquired the color and lock state of the layer-purpose pair that is now set as active in the Palette.



Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

The following figure shows the change in color and lock state when *uniform* bus color mode is selected:



Creating a Bus by Tapping Pins

Interactive bus coloring supports automatic tapping when you start creating a bus from a set of existing pins. To create a bus starting from a set of pins:

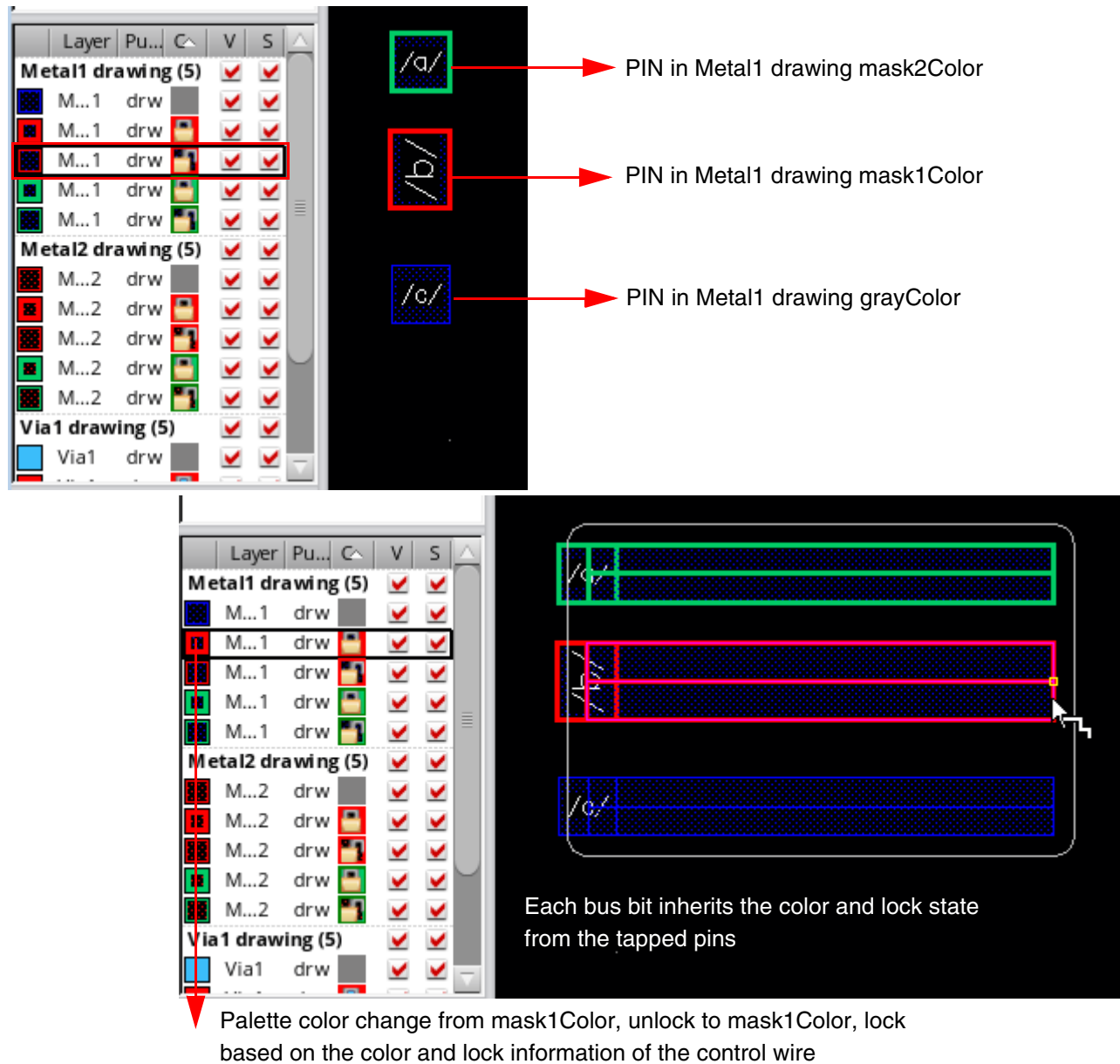
1. Choose *Create – Wiring – Bus*.
2. Select the required pins/pathSegs.

When you create a bus starting from an existing set of pins that have color assigned to them, each bus bit automatically inherits the width, color information, layer-purpose pair information, and net information from the tapped pins. As shown in the following example, the bus bits inherit the color and lock state information from the tapped pins and the spacing

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

between bus bits. The layer-purpose pair in the Palette is reset with the color and lock state information of the control wire.



Changing Color of Bus Bits by Tapping from Pins

When you start creating a bus by tapping existing colored pins, you cannot change the color of the control wire and other bus bits by selecting another layer-purpose pair with a different mask color. You can however switch between locked and unlocked states of the same mask color.

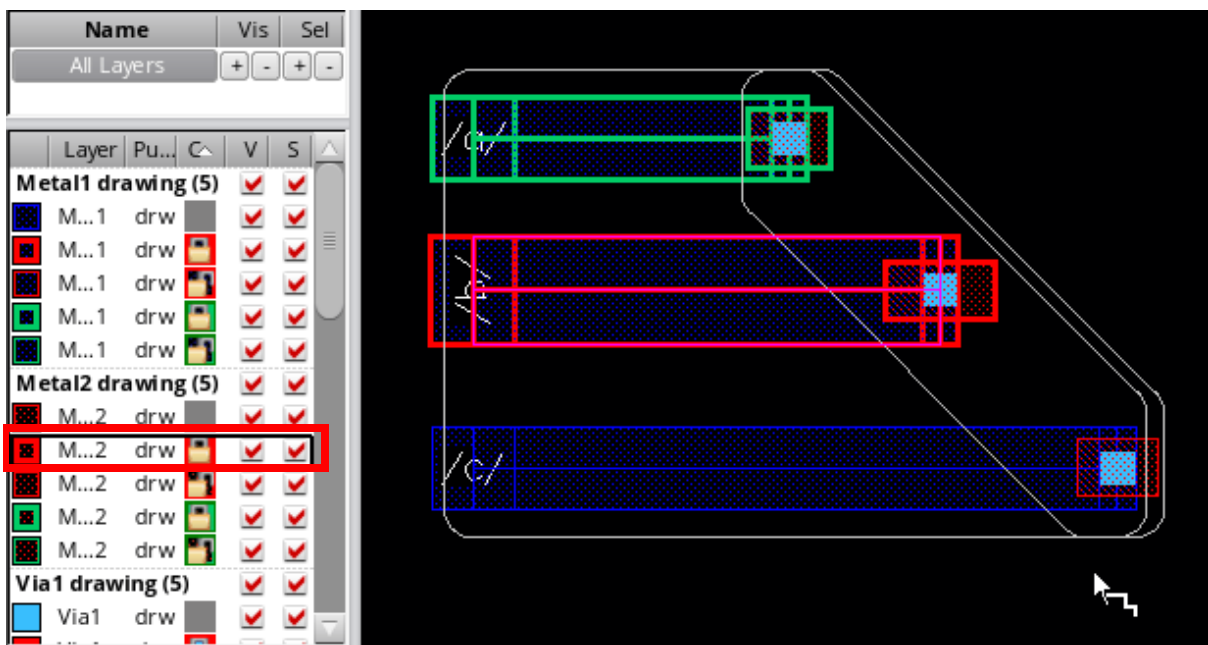
Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

To change the mask color of the control wire and other bus bits while creating a bus by tapping pins, you need to first insert a via. To change the mask color of the control wire and bus bits:

1. Insert a via by selecting *Via up to <layer name>* from the *Create Bus* context-sensitive menu.

By default, the via and the via top and bottom shapes inherit the color and lock information from the control wire of the bus being drawn. The active layer in the Palette is reset to `Metal2 drawing`, `lock` which is the output layer of the via that is inserted on the control wire, as shown in the following figure.



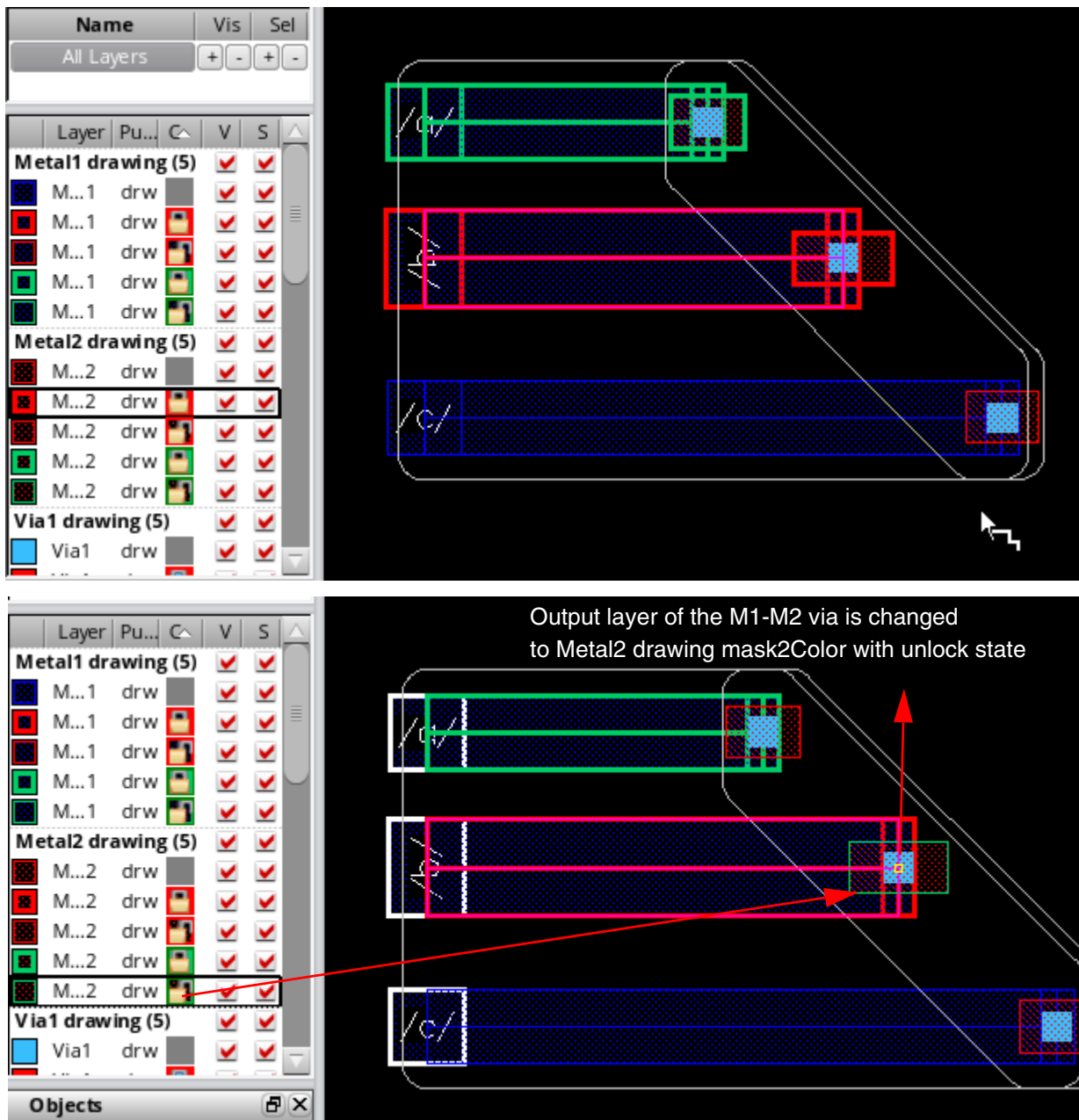
2. To change the mask color of the control wire and other bus bits, select another layer-purpose pair with a different mask color from the Palette.

If you change the mask color of the bus bits before placing a via, the color and lock information of the output layer of the via enclosure is accordingly updated. The following figures show that when you select `Metal2 drawing mask2Color` as the active layer from the Palette, the output layer of the via on the control wire is automatically reset to

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

mask2Color. It also shows a change in the via color and lock state from mask1Color, lock to mask2Color, unlock.

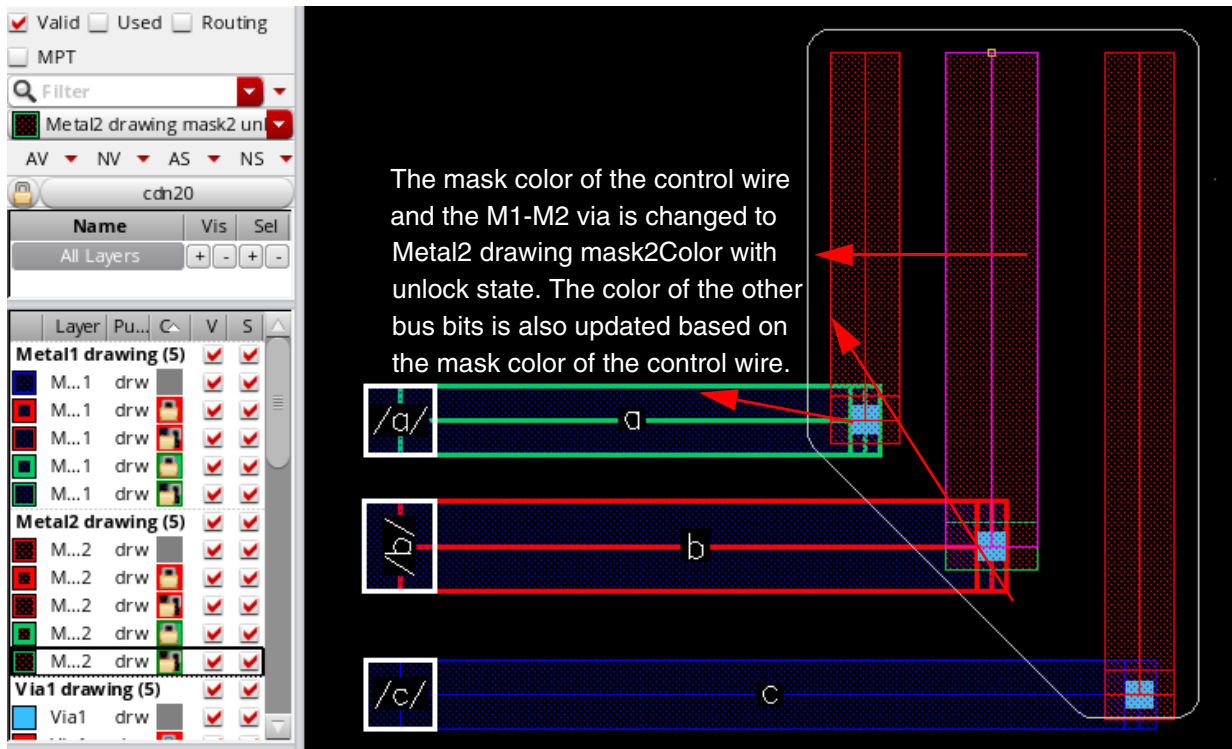


You can also change the mask color of the control wire and the other bus bits after placing the via by resetting another layer-purpose pair as active in the Palette. The following figure shows how the mask color of the control wire and the via placed on it is updated when you change the active layer-purpose pair to mask2Color, unlock. The

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

color of the other bus bits is also updated based on the mask color of the control wire and the color mode.



Changing Bus Color Mode from Tapped Pins

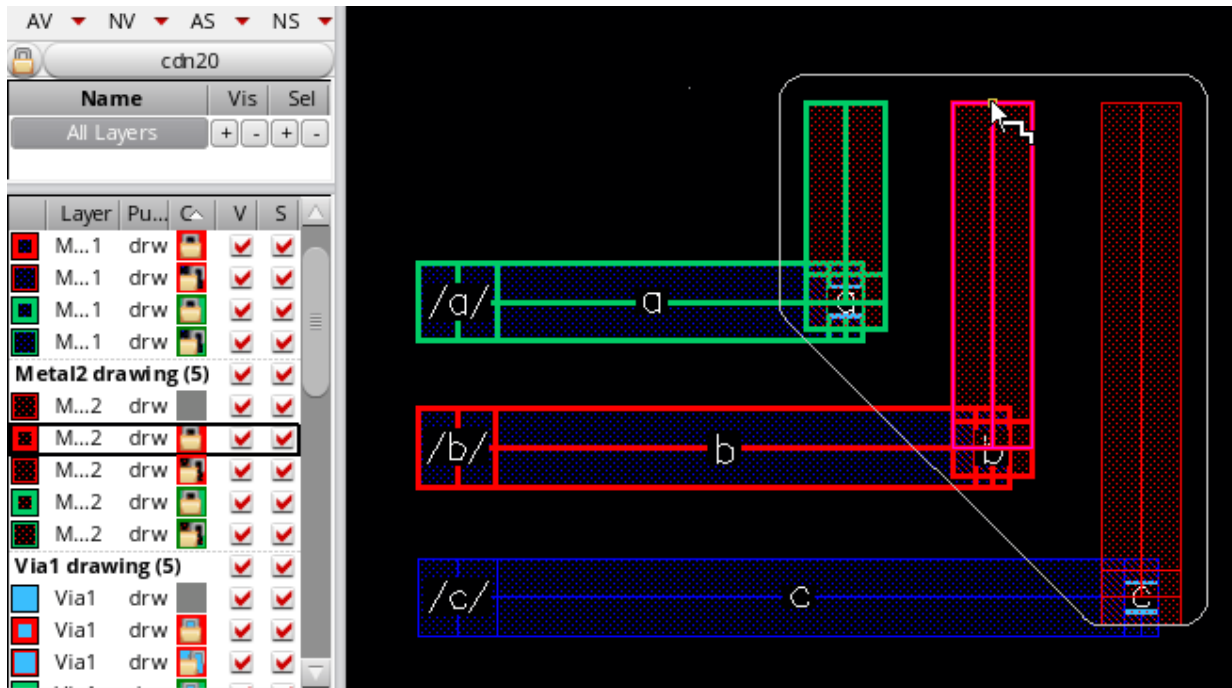
To change the color mode of the bus bits that are created from the tapped pins or objects, use the *Use Color Mode* option available on the *Create Bus* context-sensitive menu. The *Use Color Mode* option is disabled until the first via is inserted. The via insertion ensures a change in the layer-purpose pair and enables the *Use Color Mode* option. To change the bus color mode and color of the bus starting from the existing pins:

1. Choose *Create – Wiring – Bus*.
2. Select the required pins or objects.
3. Right-click the bus that is being created. The *Create Bus* context-sensitive menu is displayed.

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

4. Select *Via Up to <layer_name>* to insert the via.

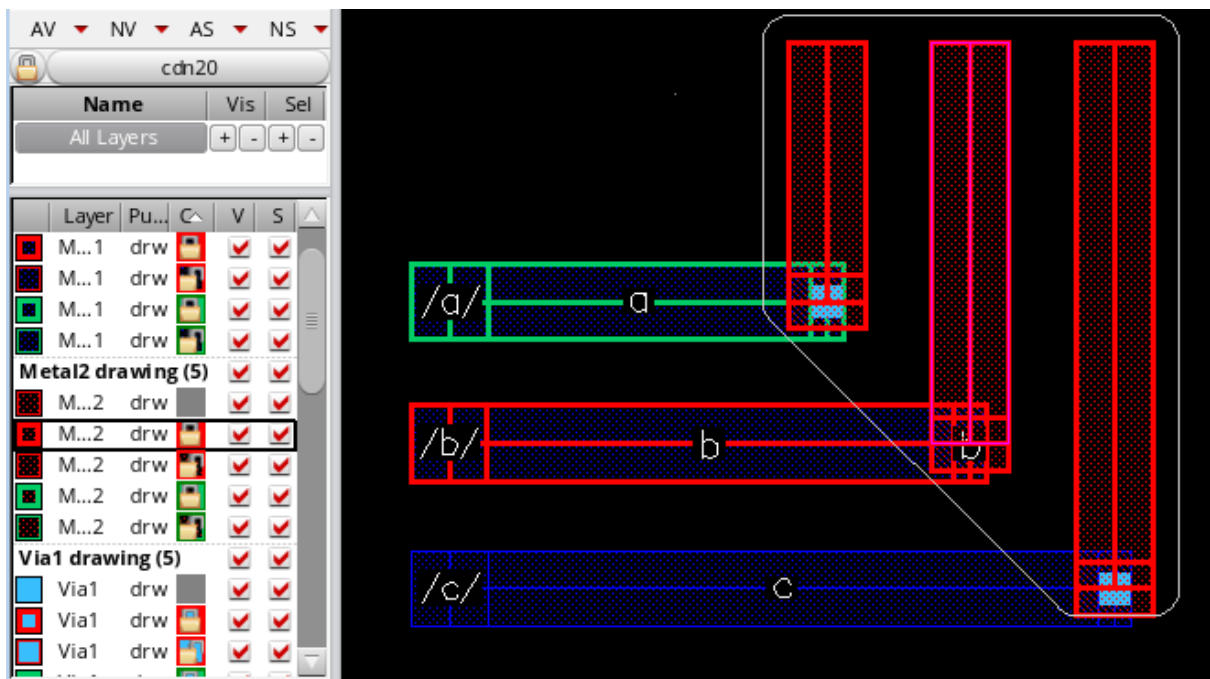


5. The *Use Color Mode* option is now enabled.
6. Right-click the bus being created and point to *Use Color Mode*. Then, select *optimized* or *uniform*. For more information on color mode, see [Specifying the Bus Color Mode](#).

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

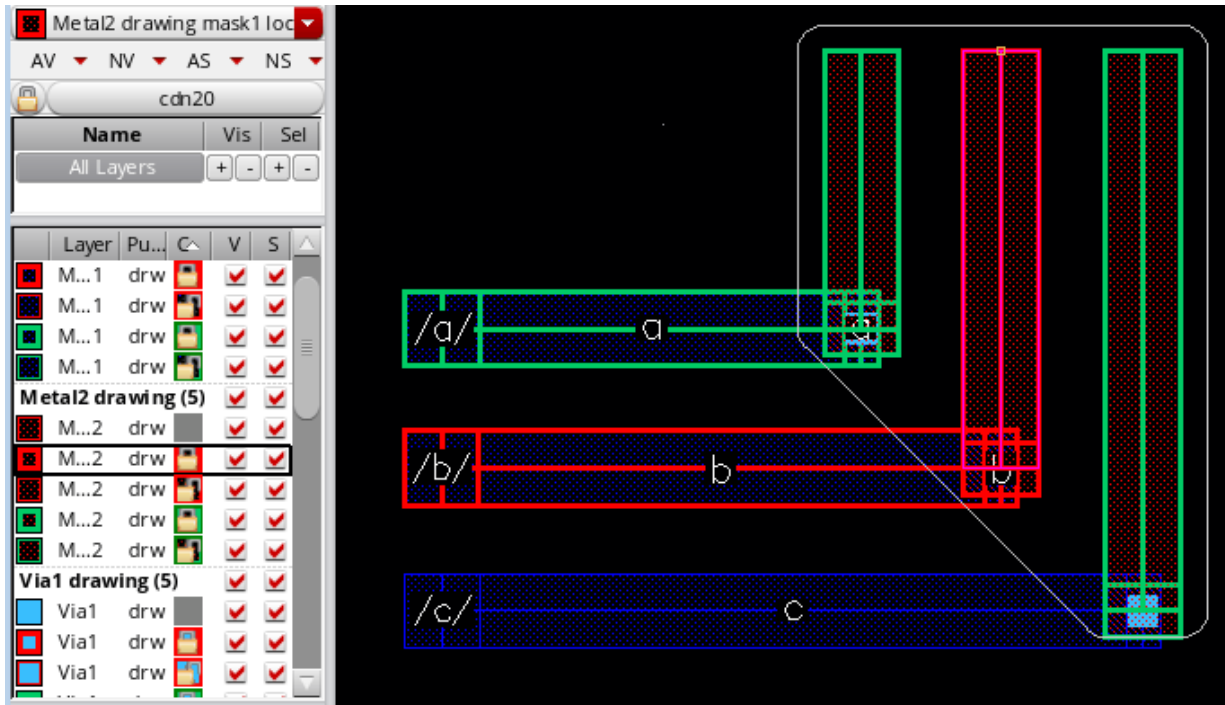
The following figure shows how the bus bits get displayed when the color mode is selected as *uniform*.



Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

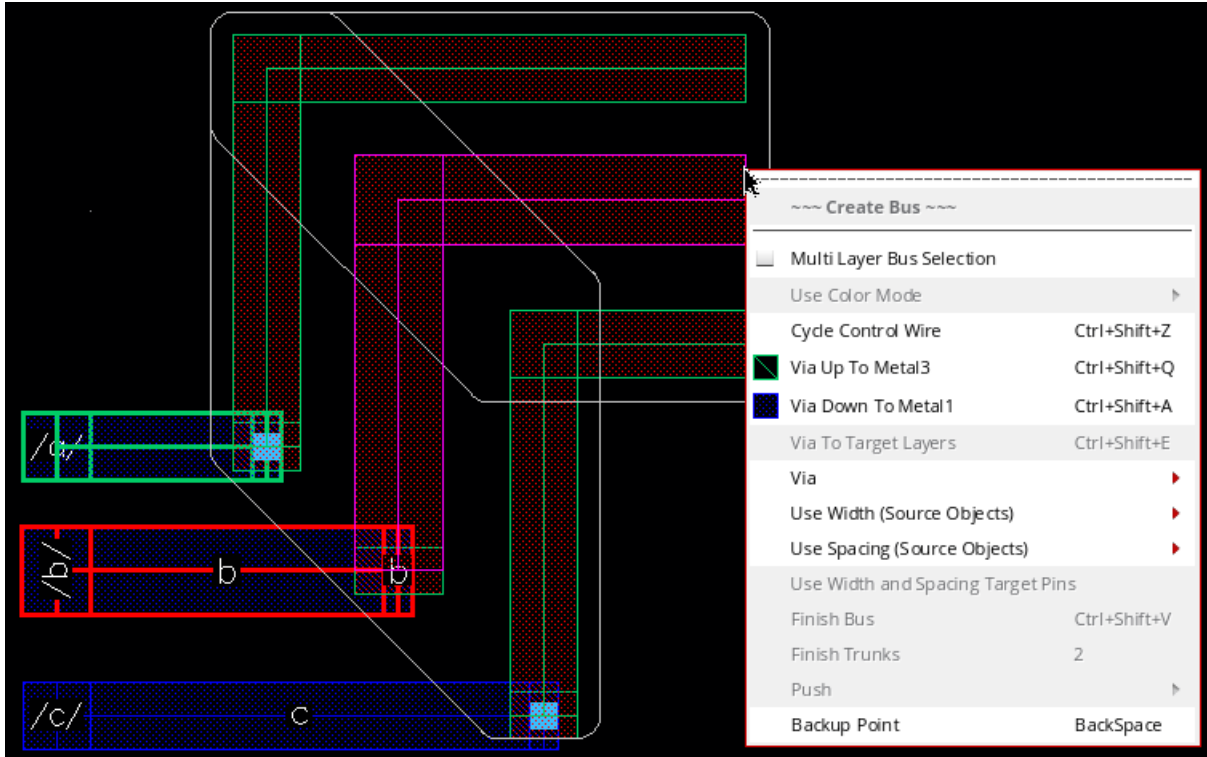
When you again change the bus color mode from *uniform* to *optimized*, the bus bits get displayed as shown in the following figure.



Virtuoso Interactive and Assisted Routing User Guide

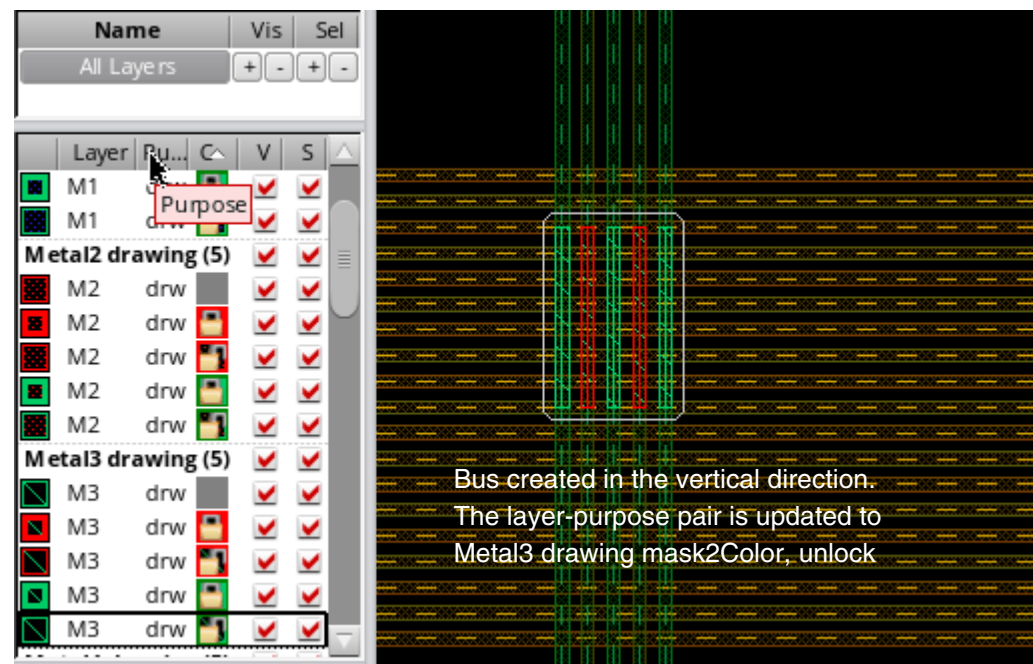
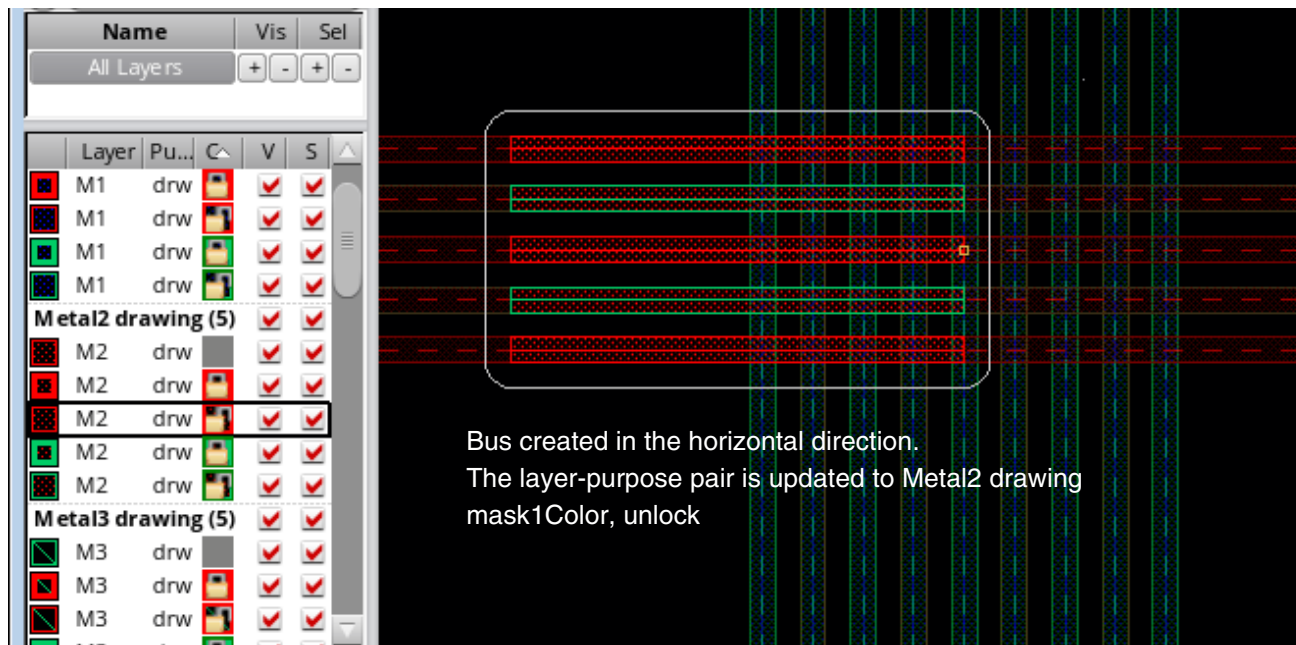
Interactive Colored Routing

The *Use Color Mode* option is disabled after a bend is created in the bus, as shown in the following figure.



Creating a Bus While Working with Colored WSPs

When you create a bus by tapping a Width Spacing Pattern (WSP) that has a color assigned to it, the bus bits inherit the color from the colored routing pattern, as shown in the following figure. The spacing between adjacent bus bits is also inherited from the routing pattern. The layer-purpose pair in the Palette is updated based on the layer and the direction in which the bus is created.

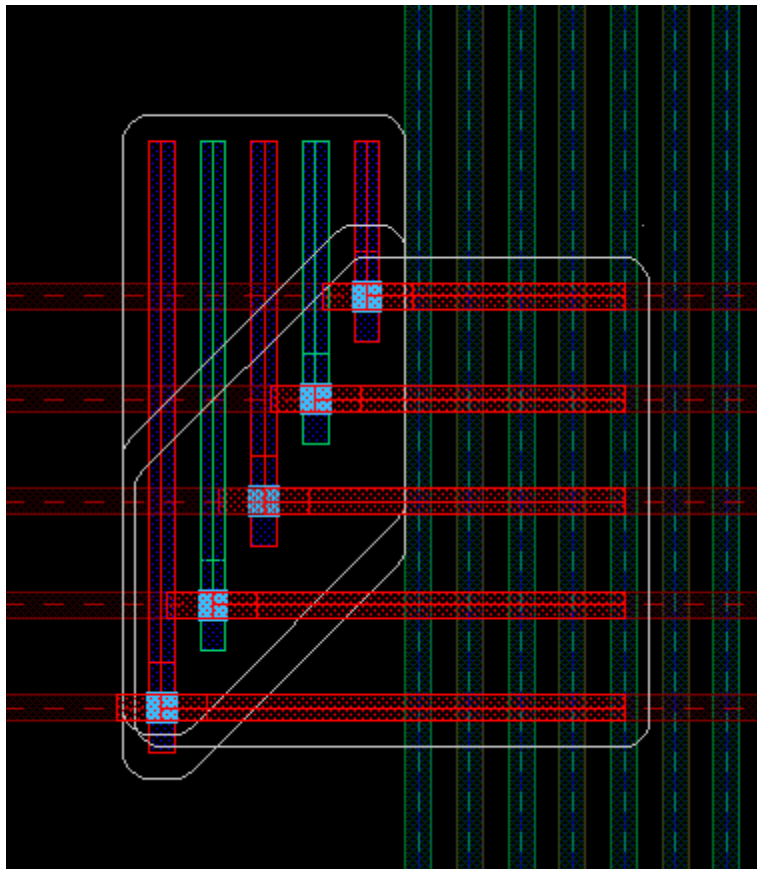


Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

When you start creating a bus from an existing routing pattern, the bus bits inherit the color of the colored track under them. As a result, you cannot change the color of the bus bits by selecting another layer with a different mask color from the Palette. You can however switch between locked and unlocked states of the same mask color. For more information on color change of bus bits, see [Changing Color of Bus Bits by Tapping from Pins](#).

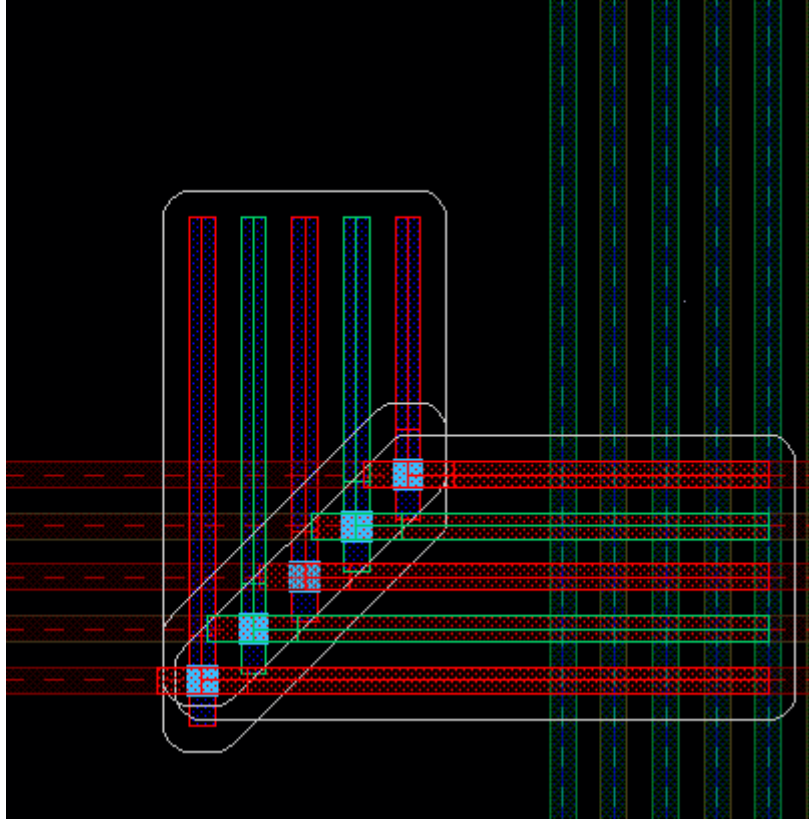
You can change the bus color mode of bus bits while creating a bus from an existing WSP. Changing the bus color mode allows bus bits to be created on the closest track according to their color. The following figure shows how the bus bits get displayed when the color mode is changed to *uniform*.



Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

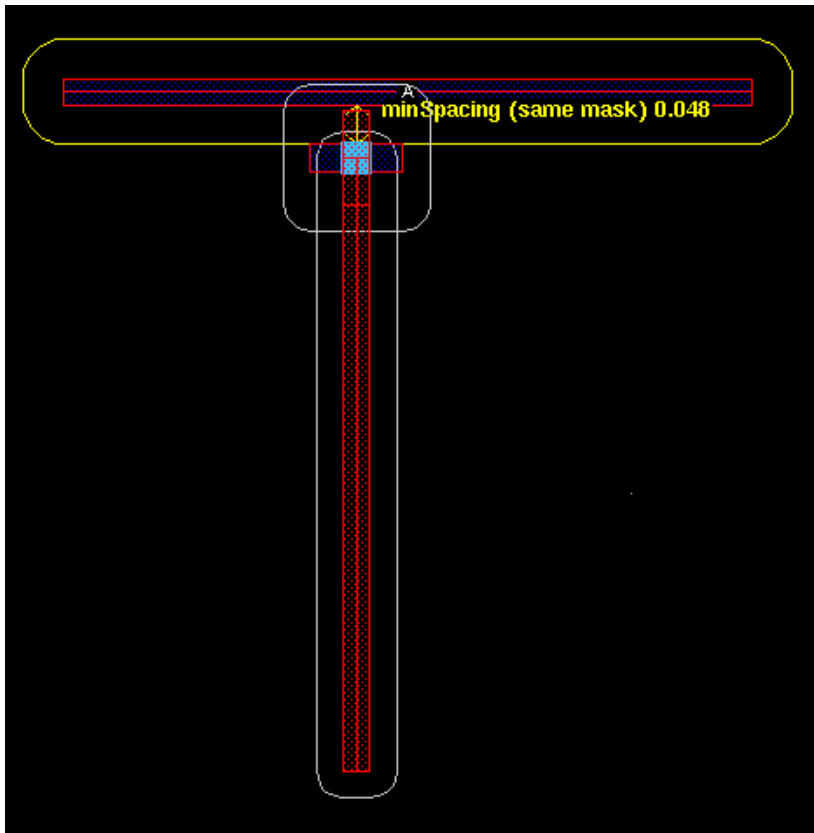
The following figure displays the bus bits when the color mode is changed to *optimized*.



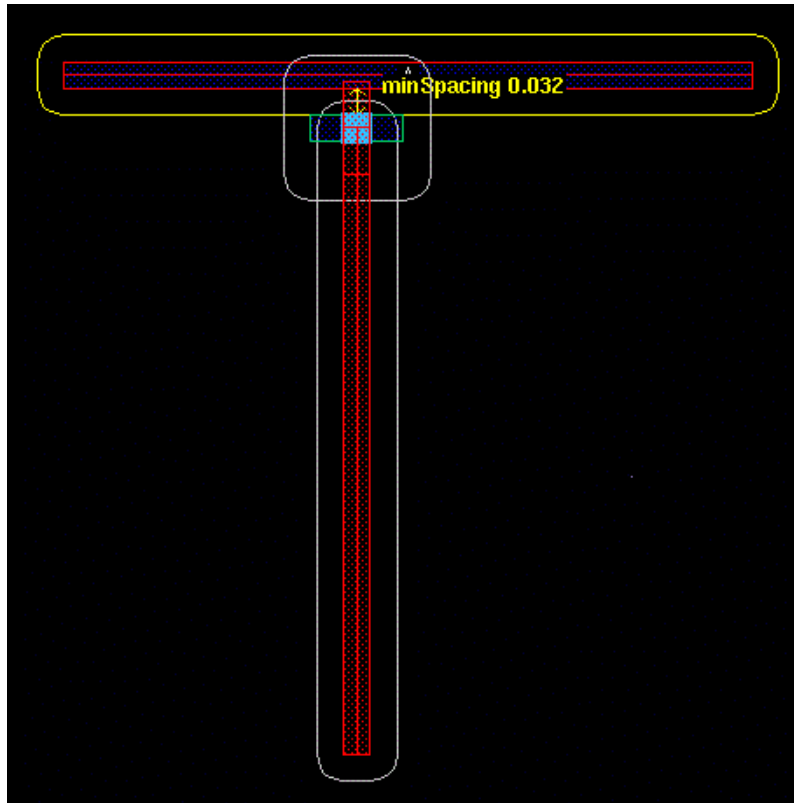
For more information about how to change the color mode of bus bits, see [Changing Bus Color Mode from Tapped Pins](#).

Performing DRC Colored Checks

Using DRD Notify and Enforce, you can perform interactive DRC colored checks on wires and that carry dynamic coloring information. This helps check for violations between two wires or via enclosures with same or different color information. For example, the following figure shows a `minSpacing (same mask)` violation between the top horizontal wire and the via enclosure, both of which have `mask1Color`.



Similarly, the following figure shows a `minSpacing (diff mask)` violation between the top horizontal wire with `mask1Color` and the via enclosure with `mask2Color`.



Interactive Coloring for Create Stranded Wire

The *Create Stranded Wire* command supports interactive coloring and lets you create a stranded wire with color and lock state information. The control wire acquires the color and lock state from the active layer in the Palette. The color of the other strands in the stranded wire depends on the color mode, which can be set to *uniform* or *optimized*.

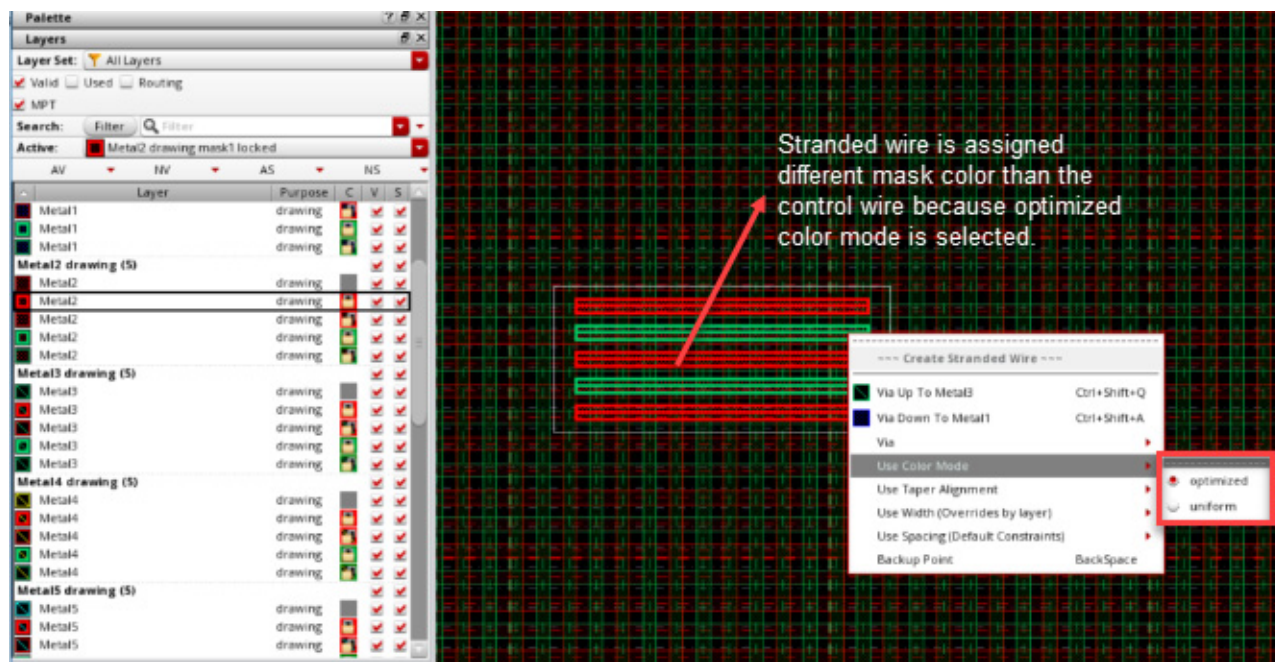
Specifying the Color Mode in Stranded Wire

You can use the options available on the *Use Color Mode* menu to control the color mode of the stranded wire. To access the color mode options, right-click the stranded wire being

Virtuoso Interactive and Assisted Routing User Guide

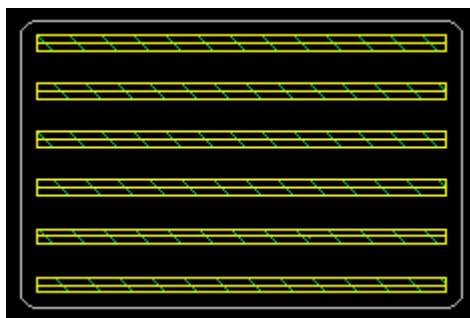
Interactive Colored Routing

created and point to *Use Color Mode*. You can either select the *optimized* or the *uniform* color mode.



Uniform Color Mode

When *uniform* is selected, all stranded wires are assigned the same mask color as that of the control wire. The mask color assigned to the control wire and other stranded wires is the same as the active layer in the Palette. The spacing between stranded wires on the same mask color is determined by `minSpacing (sameMask)`. The default color mode is *uniform*. The following figure shows the stranded wires in the *uniform* color mode with `minSpacing (sameMask)`.



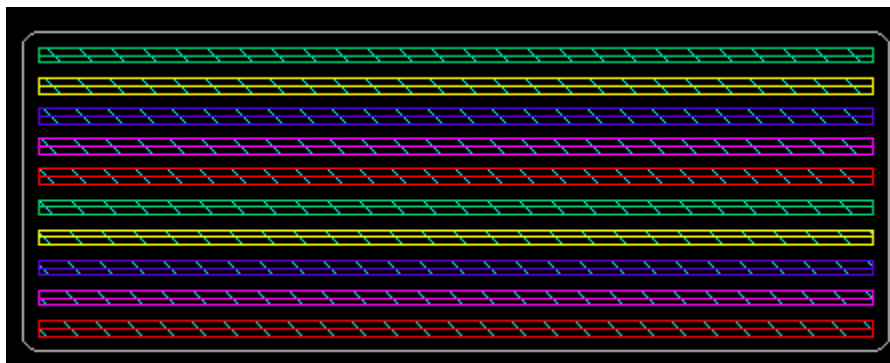
Optimized Color Mode

When you select *optimized* color mode, the control wire acquires the same mask color as that specified for the active layer in the Palette. The other stranded wires acquire the following coloring pattern:

- The stranded wires above the control wire acquire the mask color as `maskColor(controlWire)+1`, `maskColor(controlWire)+2`, and so on. This pattern continues until the highest mask color available for the selected layer-purpose pair is assigned to the stranded wire. The pattern is then repeated starting at the mask color of control wire.
- The stranded wires below the control wire acquire the mask color as `maskColor(controlWire)-1`, `maskColor(controlWire)-2`, and so on. If the control wire has the lowest available mask color assigned to it, then the stranded wires below the control wire are colored starting with the highest mask color available for the layer on which the stranded wire is being placed. The pattern is then repeated for the remaining stranded wires.

In *optimized* color mode, the spacing between two adjacent stranded wires on different masks is determined by `minSpacing(diffMask)`. This color mode is more compact than the *uniform* color mode as the `diffmask` spacing is always smaller than the `samemask` spacing.

Let us consider an example where the active layer-purpose pair is `Metal4 Drawing` and the control wire is assigned `mask1Color`. The highest available mask color for `Metal4 Drawing` is `mask4Color`.



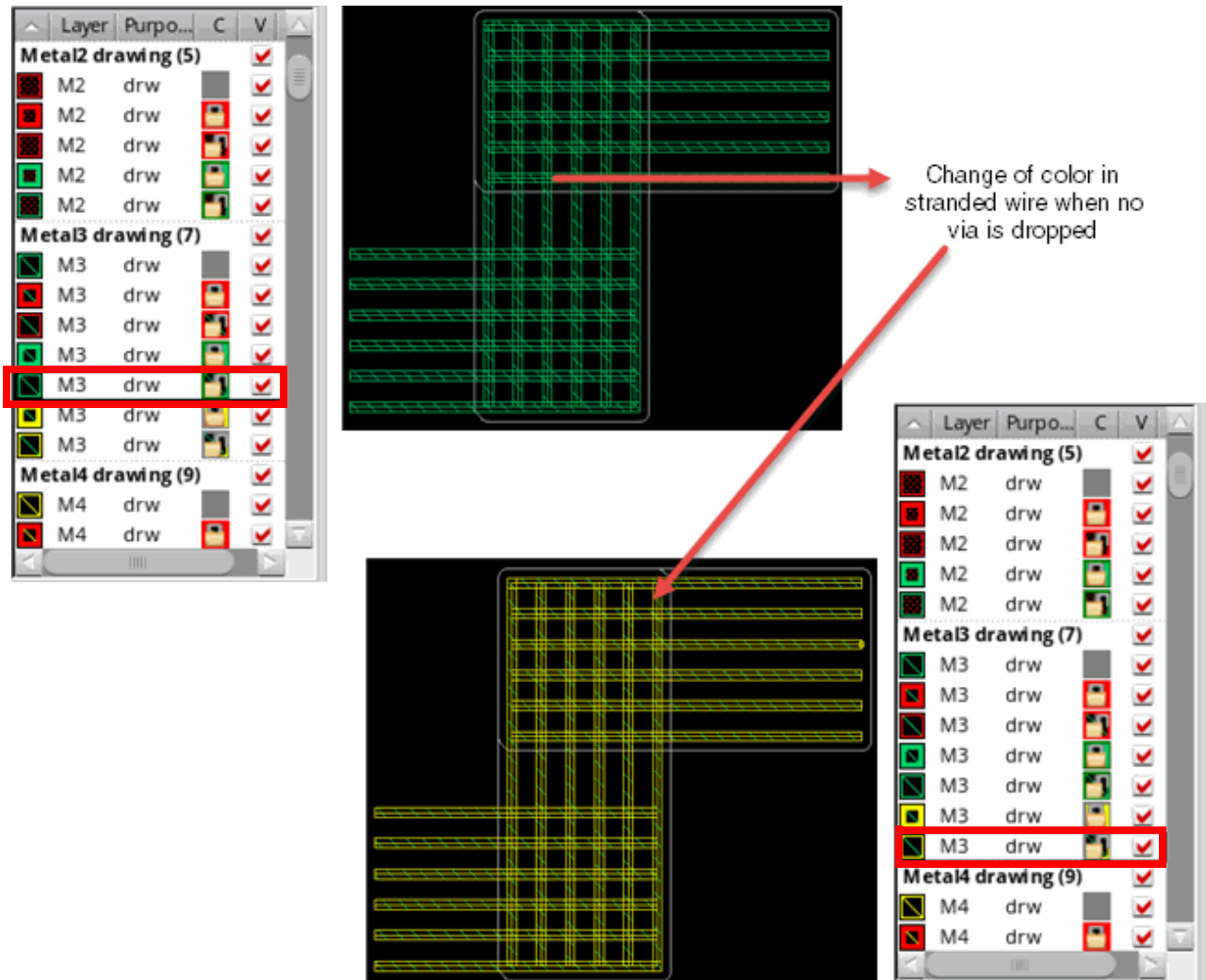
Changing the Color of Stranded Wire

While creating a stranded wire, you can apply a different mask color to the control wire, which, in turn, resets the color of the stranded wire. When you change the mask color of the stranded

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

wire before placing a via, the color and lock information of the complete stranded wire is updated as there is no layer change.

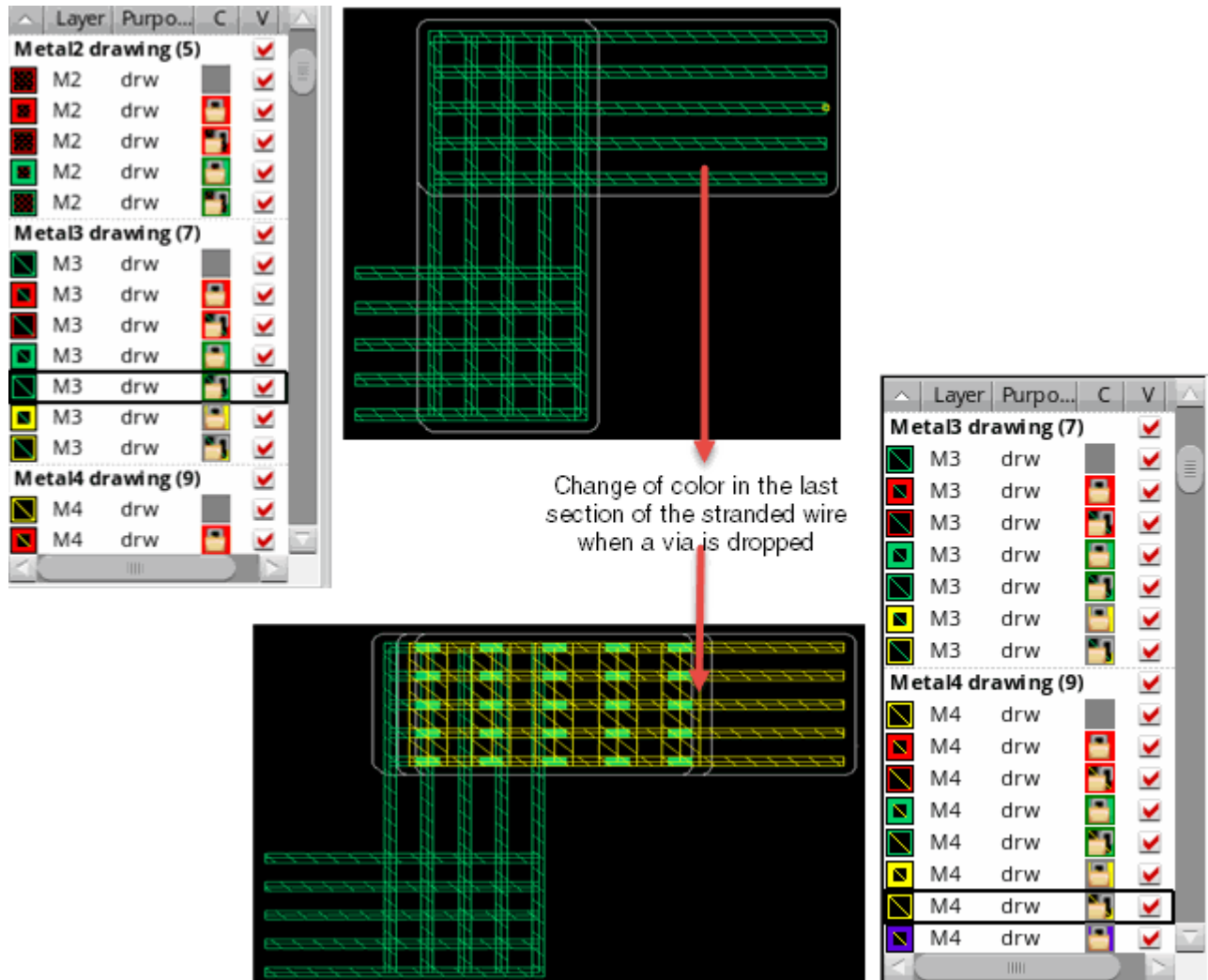


You can also change the mask color of the control wire and the stranded wire after placing the via by resetting another layer-purpose pair as active in the Palette. In this case the change of the active colored layer is applied only in the last section of stranded wire on the same layer.

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

The following figure shows how the mask color of the control wire and the last section of the stranded wire is updated when you change the active layer-purpose pair to `mask3Color`, `lock`.



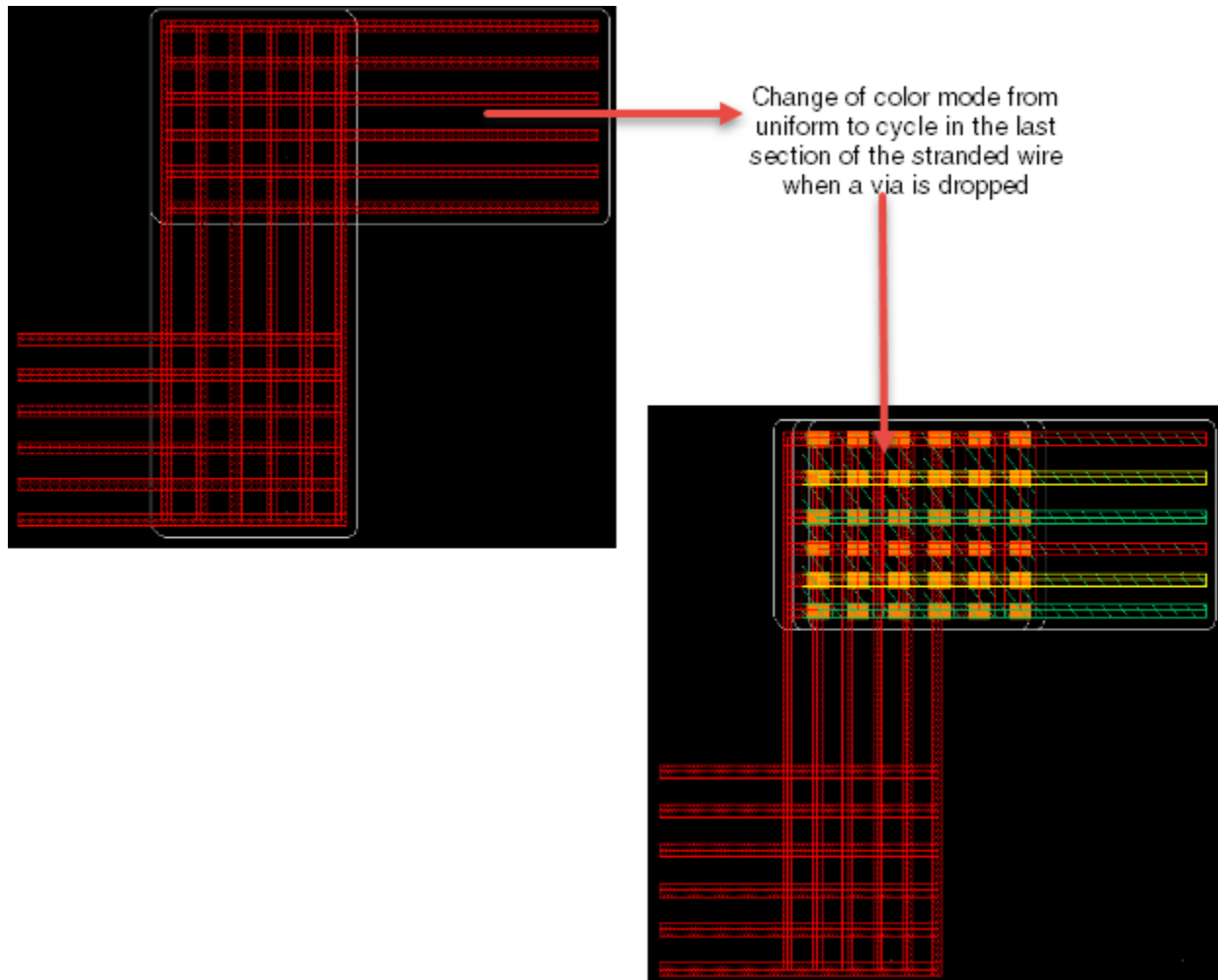
Changing the Stranded Wire Color Mode

When the color mode is changed, the stranded wire is recolored. The spacing between the stranded wire is recomputed based on the color mode that is set. In stranded wire, the color mode can only be changed only when a via is dropped. The change of the color mode is

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

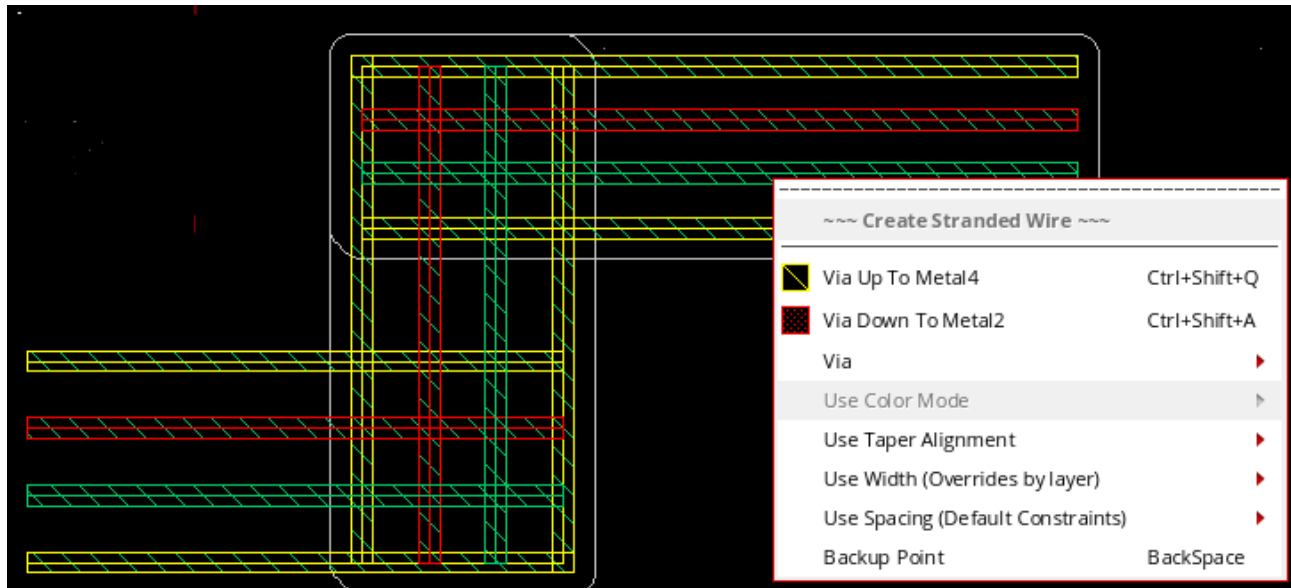
applied only in the last section of stranded wires (parallels pathSegs between a bend), as shown in the following figure.



Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

You cannot change the color mode in a stranded wire after a bend if there is no layer change, as shown in the following figure.



Creating a Stranded Wire by Tapping Pins

Interactive coloring supports automatic tapping when you start creating a stranded wire from an existing pin. To create a stranded wire starting from a pin:

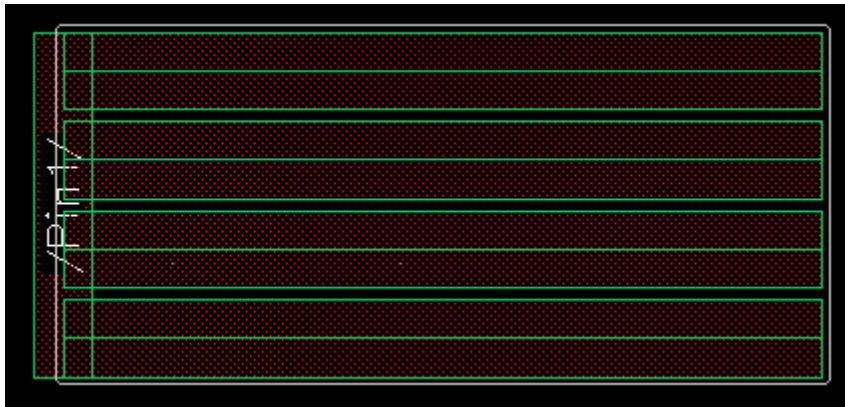
1. Choose *Create – Wiring – Stranded Wire*.
2. Select the required pin.

When you create a stranded wire starting from an existing pin that has color assigned to it, each stranded wire automatically inherits the width, color information, layer-purpose pair

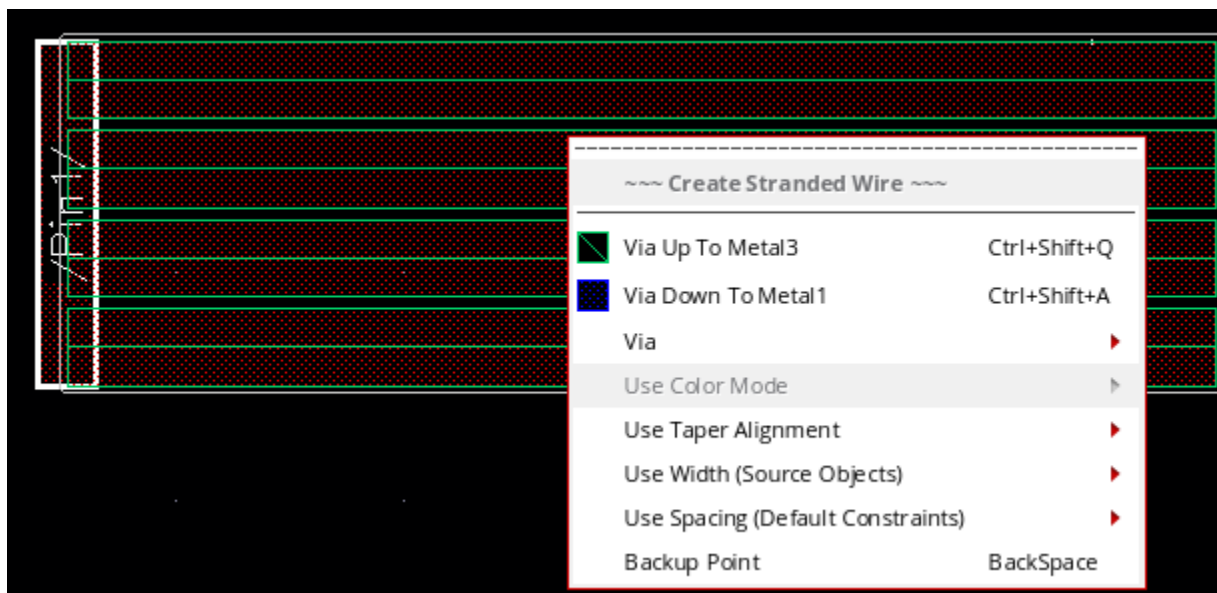
Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

information, and net information from the tapped pin. As shown in the following example, each stranded wire inherits the color and lock state information from the tapped pin.



To change the color mode of the stranded wire that is created from the tapped pin, insert a via and then use the *Use Color Mode* option available on the *Create Stranded Wire* context-sensitive menu. The *Use Color Mode* option is disabled until the first via is inserted, as shown in the following figure.

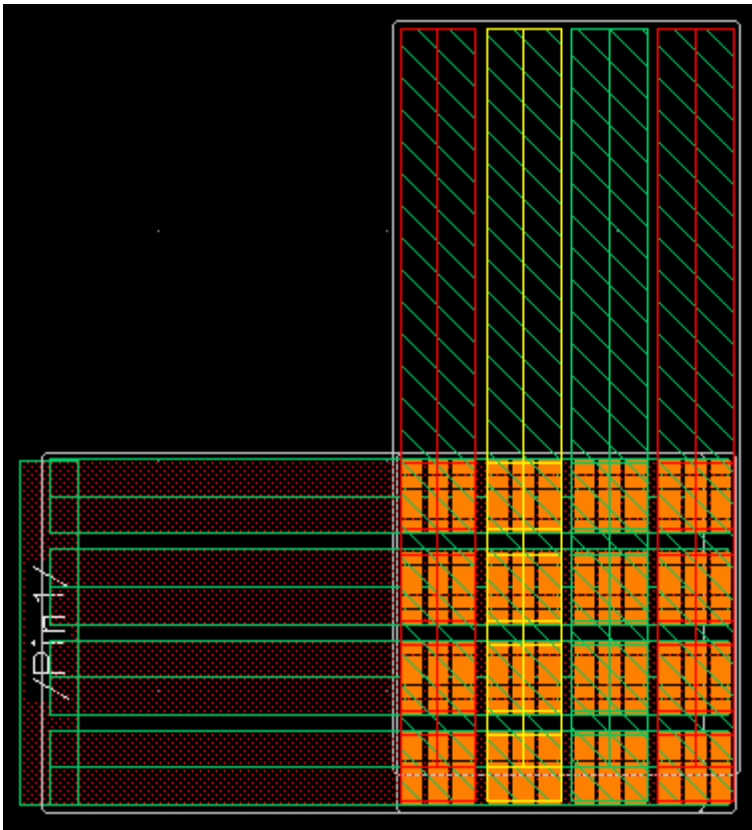


The via insertion ensures a change in the layer-purpose pair and enables the *Use Color Mode* option. To change the color mode and color of the stranded wire starting from the existing pin:

1. Choose *Create – Wiring – Stranded Wire*.
2. Select the required pins or objects.

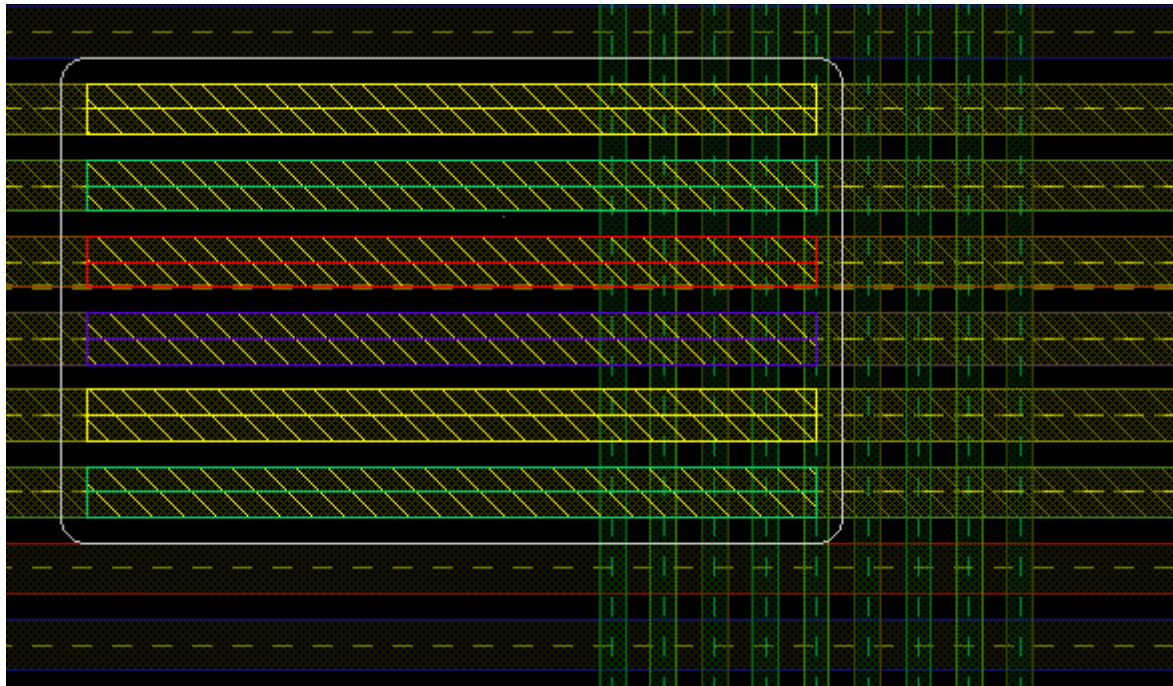
3. Right-click the stranded wire that is being created. The *Create Stranded Wire* context-sensitive menu is displayed.
4. Select *Via Up/Down to <layer_name>* to insert the via. The *Use Color Mode* option is now enabled.
5. Place the mouse pointer on *Use Color Mode*. Then, select *optimized* or *uniform*. For more information on color mode, see [Specifying the Color Mode in Stranded Wire](#).

The following figure shows how the stranded wire get displayed when the color mode is set to *optimized*. The layer-purpose pair in the Palette is reset with the color and lock state information of the control wire.



Creating a Stranded Wire While Working with Colored WSPs

When you create a stranded wire by tapping a Width Spacing Pattern (WSP) that has a color assigned to it, the stranded wire inherits the color from the colored routing pattern, as shown in the following figure. The spacing between each strand is also inherited from the routing pattern. The layer-purpose pair in the Palette is updated based on the layer and the direction in which the stranded wire is created.



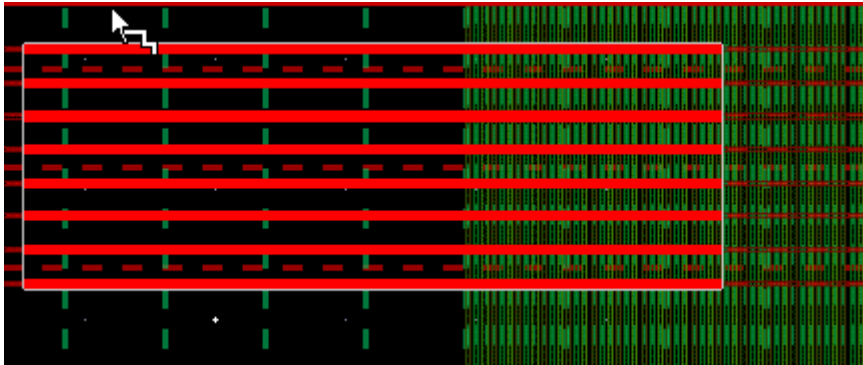
When you start creating a stranded wire from an existing routing pattern, each stranded wire inherits the color of the colored track on which it sits. As a result, you cannot change the color of the stranded wire by selecting another layer with a different mask color from the Palette. You can, however, switch between locked and unlocked states of the same mask color. For more information on changing the color of a stranded wire, see [Creating a Stranded Wire by Tapping Pins](#).

You can change the color mode of the stranded wire while creating a stranded wire from an existing WSP. Changing the color mode allows the stranded wire to be created on the closest

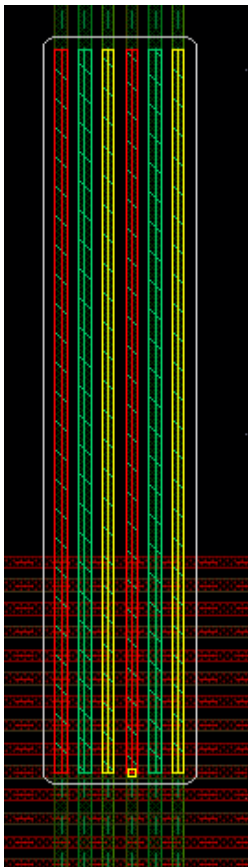
Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

track based on color. The following figure shows how the stranded wire is displayed when the color mode is changed to *uniform*.



The following figure displays the stranded wire when the color mode is changed to *optimized*. The stranded wire inherits the nearest track spacing.



For more information about how to change the color mode of stranded wire, see [Creating a Stranded Wire by Tapping Pins](#).

Avoiding Shorts with Existing Blockages on Colored WSPs

The stranded routing skips tracks to avoid creating a short with an existing blockage on colored WSPs.

Example Scenarios for Avoiding Shorts with Existing Blockages on Colored WSPs

This section illustrates some of the possible scenarios, with different requirements, on how a stranded wire skips the track on which a blockage appears and accordingly adjusts to the next nearest track that does not have any blockage when MPT is enabled and stranded wires are colored.

Scenario 1

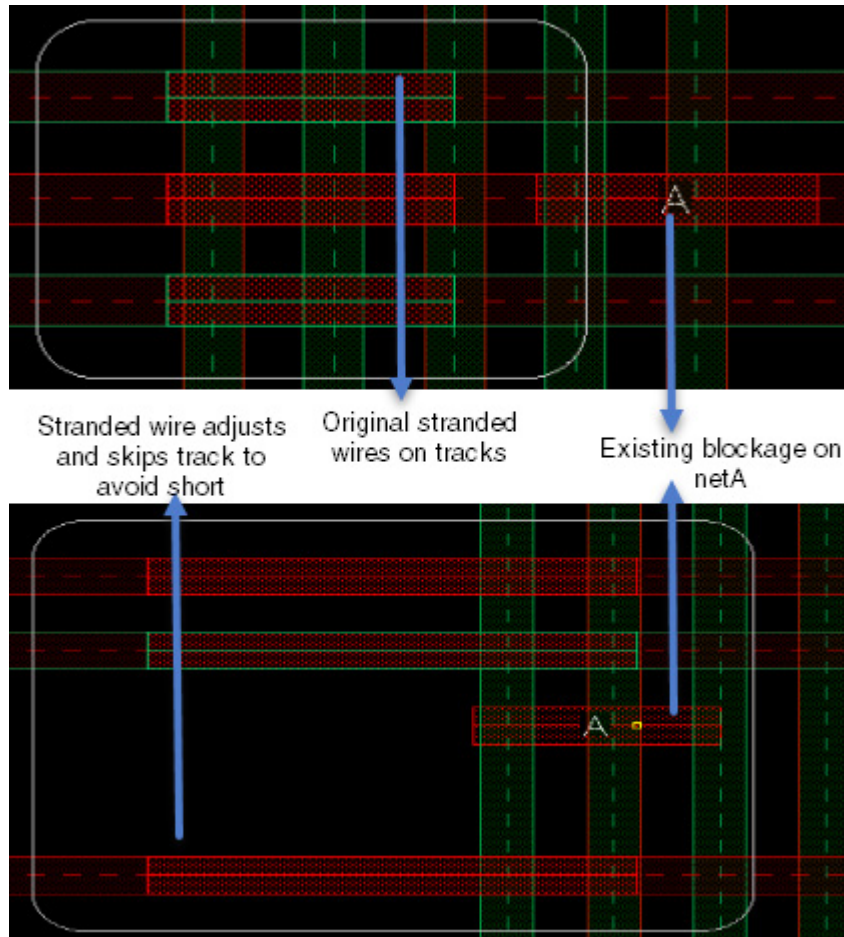
Blockage exists on layer `Metal2` with net name `A`, and the color mode is *optimized*.

1. Enable *MPT Support* in the Palette assistant. For more information, see [MPT Support](#) in the Virtuoso Layout Suite documentation.
2. (Optional) Select the *MPT* check box.
3. Set a layer-purpose pair with the required color and lock status as active.
4. Choose *Create – Wiring – Stranded Wire* and press `F3`.
5. Specify `B` in the *Net Name* field.
6. Use the options available on the *Use Color Mode* menu to control the color mode of the stranded wire. To access the color mode options, right-click the stranded wire being created and point to *Use Color Mode*.
7. Choose the *optimized* color mode.
8. Move the stranded wire being created in the direction in which a blockage with net name `A` appears.

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

The stranded wire with net name **B** skips the track that has blockage with net name **A** and accordingly adjusts to the next nearest colored track, which follows the *optimized* color mode.

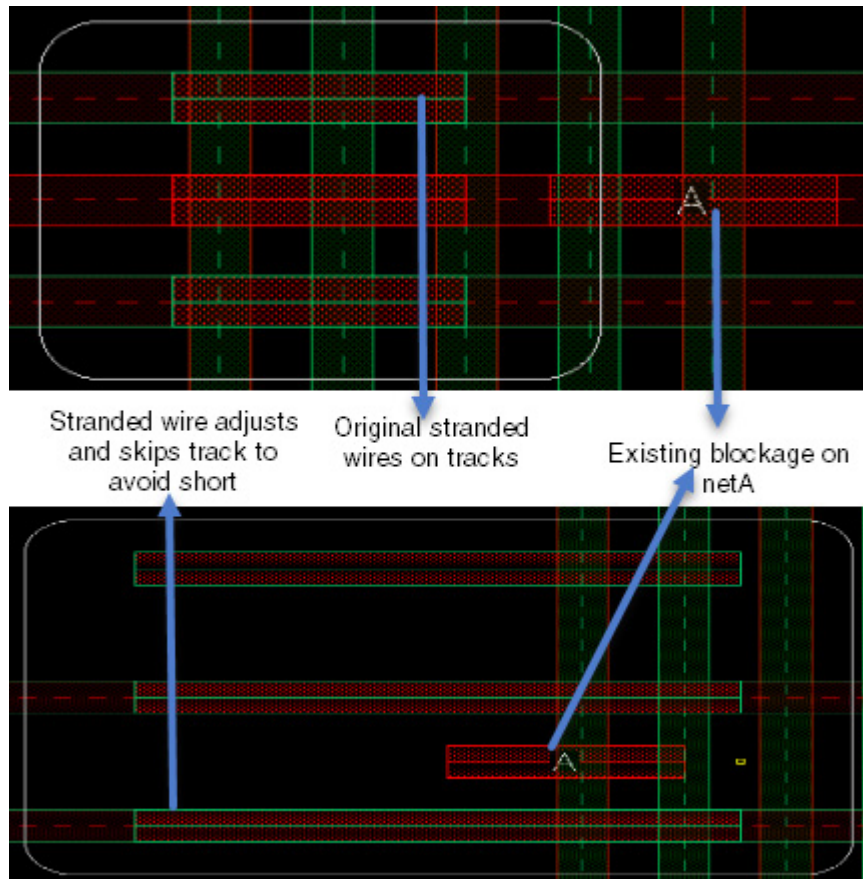


Scenario 2

Blockage exists on `Metal2` layer with net name **A**, and the color mode is changed from *optimized* to *uniform* or conversely, from *uniform* to *optimized*.

1. Perform steps 1 through 7 in [Scenario 1](#).
2. Select the *uniform* color mode.
3. Move the stranded wire being created in the direction in which a blockage with net name **A** appears.

In this case, the stranded wire with net name **B** skips the track that has blockage with net name **A** and accordingly adjusts to the next nearest colored track, which follows the *uniform* color mode.



Let us now see some scenarios when the colored stranded wires along with the vias are slided. The behavior of the stranded wire depends on the color mode that is selected.

Scenario 3

When the color mode is *optimized* and the blockage with net name **A** exists on the **Metal3** layer. The stranded wire is created from an empty space on the **Metal2** layer and changes the direction by inserting vias.

1. Perform steps 1 through 7 in [Scenario 1](#).
2. Choose the *Via Up* or *Via Down* option from the *Create Stranded Wire* context-sensitive menu to insert a via on the stranded wire.

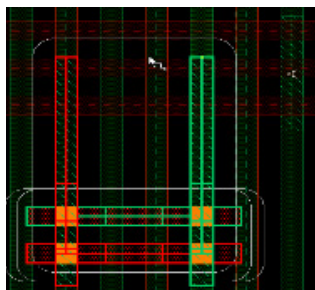
When the via is inserted, the direction of the stranded wire is changed.

Virtuoso Interactive and Assisted Routing User Guide

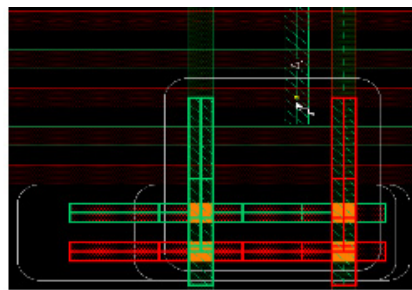
Interactive Colored Routing

3. Move the stranded wire in the direction in which `Meta13` blockage with net name `A` appears.
4. To slide the current section of the stranded wire, hold the `Ctrl+Shift` keys and move the mouse pointer to the right or left of the current position.

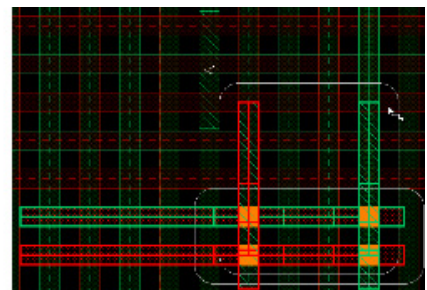
In this case, the stranded wire with net name `B` skips the track that has a blockage with net name `A` and accordingly adjusts to the next nearest colored track, which follows the *optimized* color mode.



Position of stranded wire when slid to left

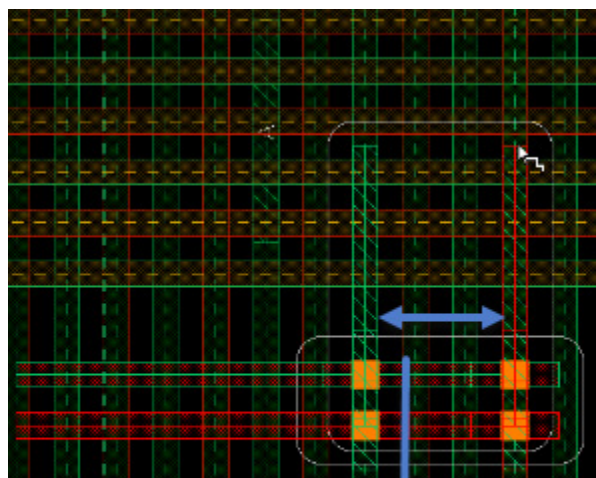


Original position of stranded wire

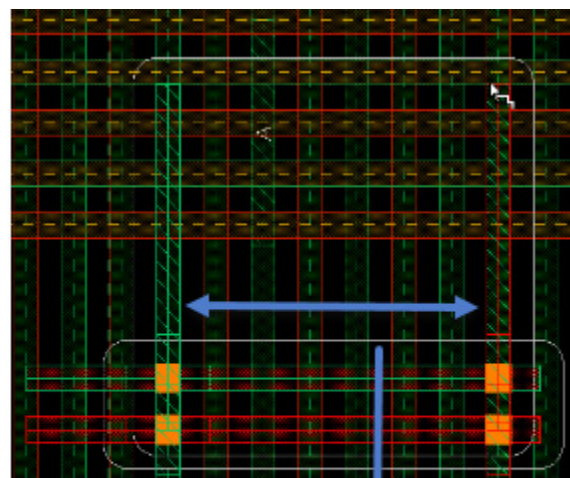


Position of stranded wire when slid to right

In the same scenario, you can also increase the spacing between the stranded wires and slide the wires. Use the `Ctrl+3` bindkey to increase the spacing between the stranded wires. The stranded wire with net name `B` skips the track that has a blockage with net name `A` and accordingly adjusts to the next nearest colored track, which follows the *optimized* color mode.



Original spacing between stranded wires



Increased spacing between strands. The stranded wire moves to the

Similarly, use the `Ctrl+4` bindkey to decrease the spacing between the stranded wires. The decreased spacing snaps the stranded wires to adjacent WSP track. However, after decreasing the spacing between stranded wires, the stranded wires create shorts with other shapes. To avoid short violations and the blockage, again increase the spacing between the stranded wires. This can be done by resetting the spacing value in the Create Stranded Wire form and then use the slider to enhance results of the stranded wires.

Scenario 4

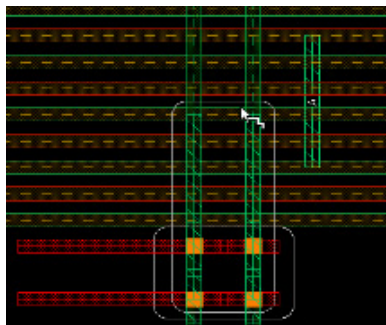
When the color mode changes from *optimized* to *uniform* and the blockage with net name `A` exists on the `Metal3` layer. The stranded wire is created from an empty space on `Metal2` layer and changes the direction by inserting vias.

1. Perform steps 1 through 6 in [Scenario 1](#).
2. Select the *uniform* color mode.
3. Choose the *Via Up* or *Via Down* option from the *Create Stranded Wire* context-sensitive menu to insert a via on the stranded wire.

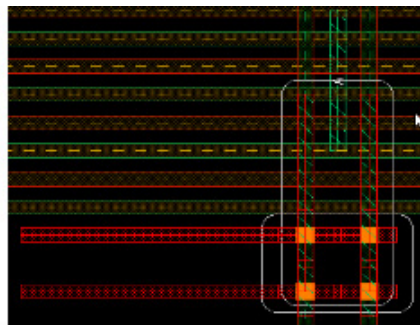
When the via is inserted, the direction of the stranded wire is changed.

4. Move the stranded wire in the direction in which `Metal3` blockage with net name `A` appears.
5. To slide the current section of the stranded wire, hold the `Ctrl+Shift` keys and move the mouse pointer to the right or left of the current position.

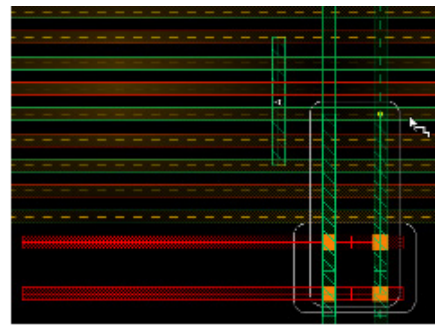
In this case, the stranded wire with net name `B` skips the track that has a blockage with net name `A` and accordingly adjusts to the next nearest colored track, which follows the *uniform* color mode.



Position of stranded wire when slid to left

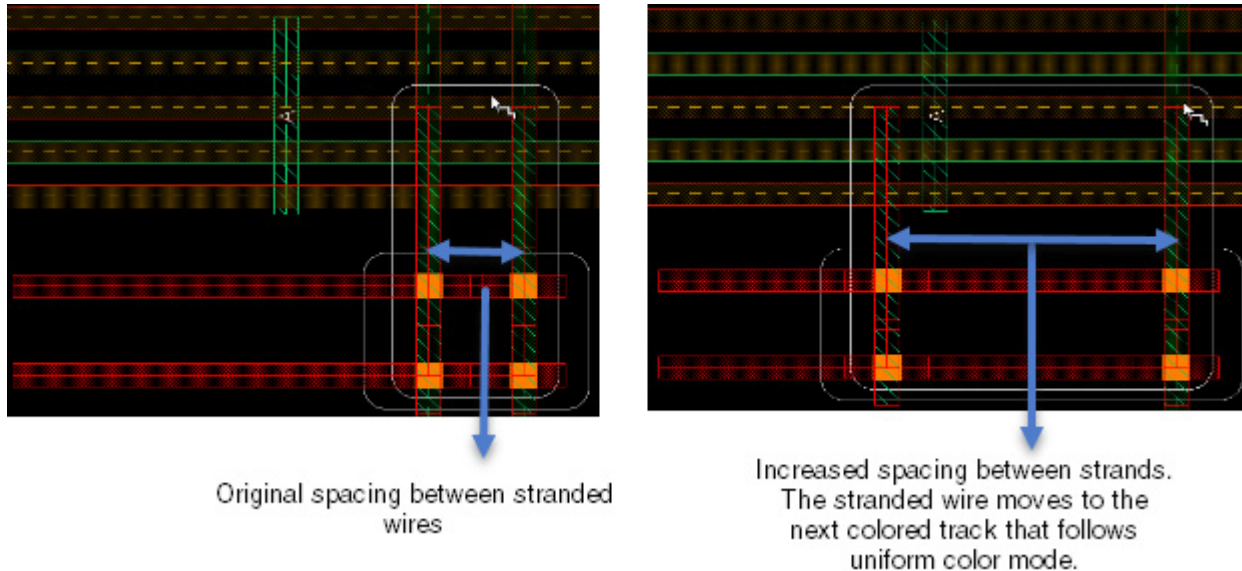


Original position of stranded wire



Position of stranded wire when slid to right

In the same scenario, you can also increase the spacing between the stranded wires and slide the wires. Use the `Ctrl+3` bindkey to increase the spacing between the stranded wires. The stranded wire with net name B skips the track that has a blockage with net name A and accordingly adjusts to the next nearest colored track, which follows the *uniform* color mode.

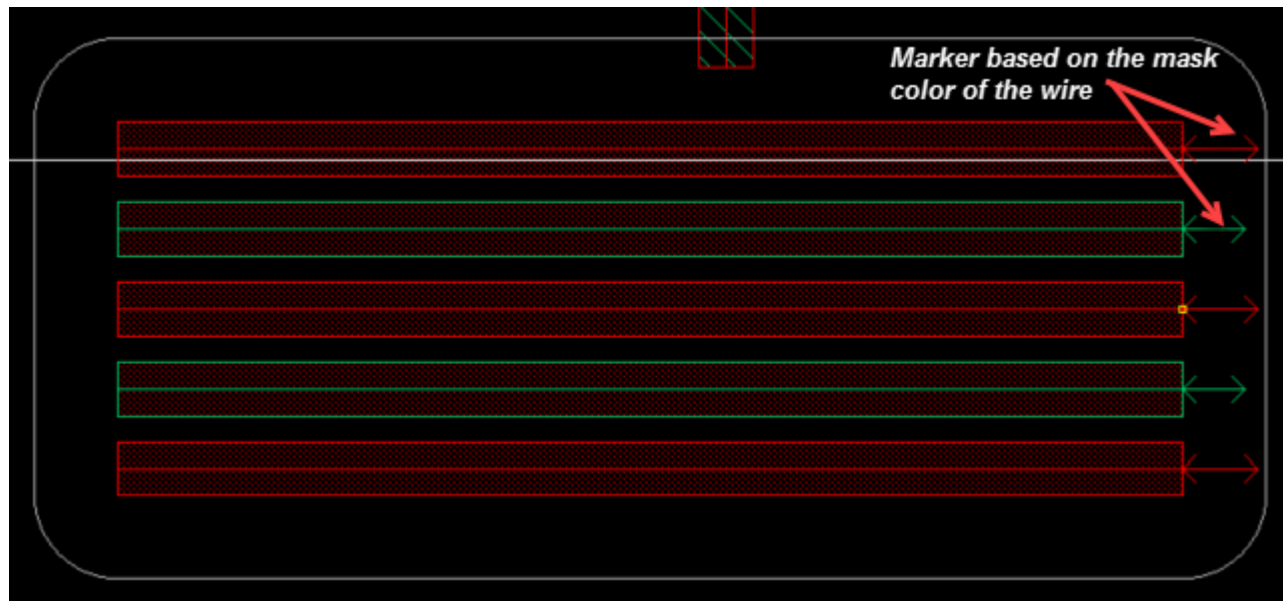


Supporting End-to-End Spacing for Colored Shapes

The `minEndOfLineSpacing` constraint provides the capability to define the end-of-line spacing for `sameMask` and `diffMask`. However, a color-specific `sameMask` spacing is not defined in the `minEndOfLineSpacing` constraint. To describe the color-specific `sameMask` spacing, the `minSideSpacing` constraint is considered and for this spacing to correspond to `minEndOfLineSpacing`, use the `mask1`, `mask2`, and `mask3` parameters with `sameMask` in the `minSideSpacing` constraint.

When the *Create Wire*, *Create Stranded Wire*, or the *Create Bus* command is run, a white halo surrounding the active wires displays. This halo determines `minSpacing`, `minEndOfLineSpacing`, and some other violations. In order to show the `sameMask` spacing for `mask1`, `mask2`, and `mask3` separately, additional markers appear according to the mask color of the wire. If the wire is of `mask1` color, a `mask1` arrow appears at the end of active wire. Similarly, if the wire is of `mask2` or `mask3` color, a `mask2` or a `mask3` arrow appear at the end of the active wire. The existing white halo determines the spacing for

different mask colors. The following figure shows an example when a bus is created on the *Metal2* layer, which is in the horizontal direction.



Interactive Coloring for the Stretch Command with Colored WSPs

The *Stretch* command supports interactive coloring and lets you edit a wire and via with color and lock state information that has been earlier created by tapping objects, such as pins, pathSegs and buses, or colored WSPs. As the wire or via is dragged interactively, its color is inherited from the colored track the stretch command snaps the wire or via to. However, the lock state of the wire or via remains unchanged.

Color Locks While Editing a Wire

While editing an existing wire by either create wire or moving or stretching, the system lock is automatically set to a user lock.

Note: The system lock is set using the *Lock Color* icon on the *Multiple Patterning* toolbar, whereas a user lock is set using the *MPT* feature of the Palette Assistant.

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

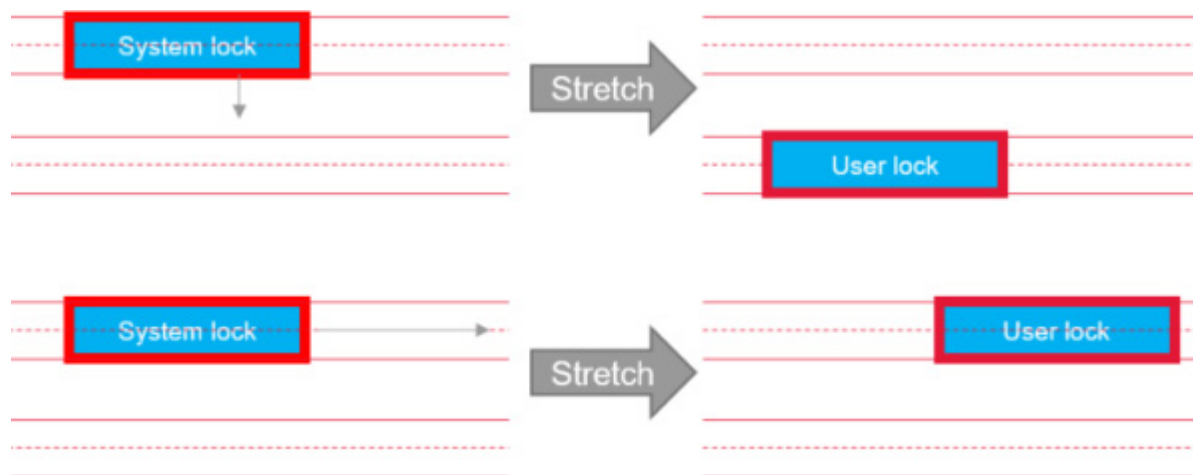
When merging an existing wire, which has system lock, by either create wire or by stretching, merges with a wire having user lock, the resultant merged wire is considered as a new wire and sets the system lock to user lock.



Similarly, editing an existing wire by create wire or stretch, which has system lock, results in a wire having user lock.

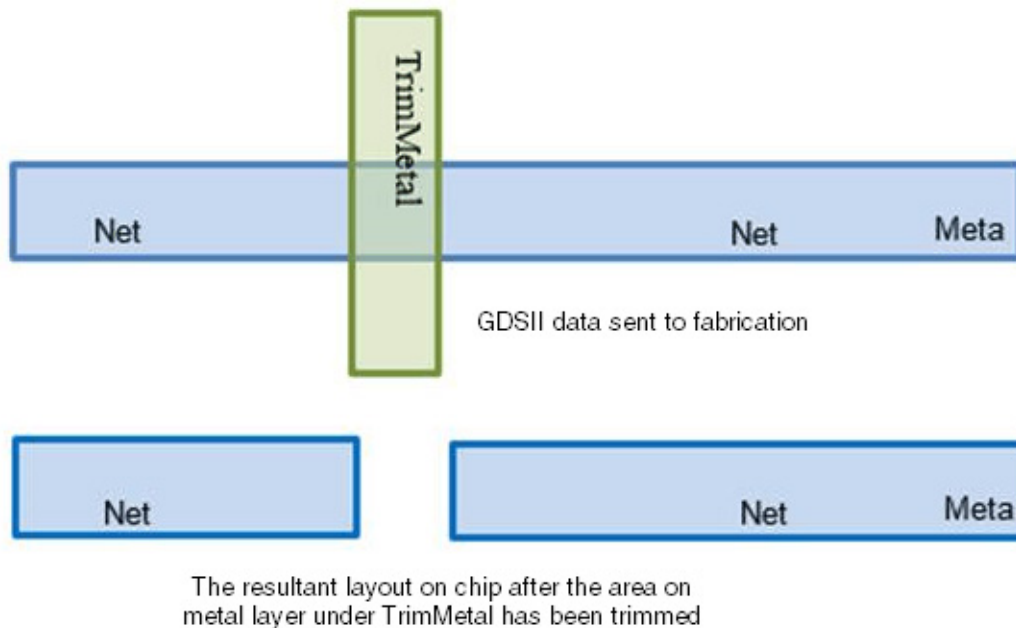


Again, stretching or moving a whole pathSeg that has system lock results in a pathSeg with user lock.



Inserting TrimMetal Layers in Interactive Colored Routing

TrimMetal is a physical layer where trim shapes are inserted. In some process technologies, these metal layers are used to trim portions of a metal layer. This enables routing with much smaller end-of-line spacing that would otherwise be impossible. This is denoted in the GDSII data that is sent to the fabrication and checked by DRC, as shown in the following figure.



Interactive colored routing supports automatic insertion of trim metal layers when the `trimShape` constraint is present in the technology file. In such case, DRD checks the `trimShape` constraint and transmits the trim shape information to the Wire Editor. In addition, interactive routing commands supports trim insertion only when *MPT Support* in the Palette assistant is enabled. For more information, see [trimShape](#) in Virtuoso® Technology Data Constraint Reference.

The Wire Editor then creates and modifies the wires and inserts the trim shapes, as follows:

- At the end of wires when two wires are precisely aligned.
- On a layer-purpose pair, which depends on the layer and color of the wire for advanced nodes.
- Only if the distance between the two wires is less than or equal to the `exceptSpacing` value given in the `trimShape` constraint defined in the technology file.

Virtuoso Interactive and Assisted Routing User Guide

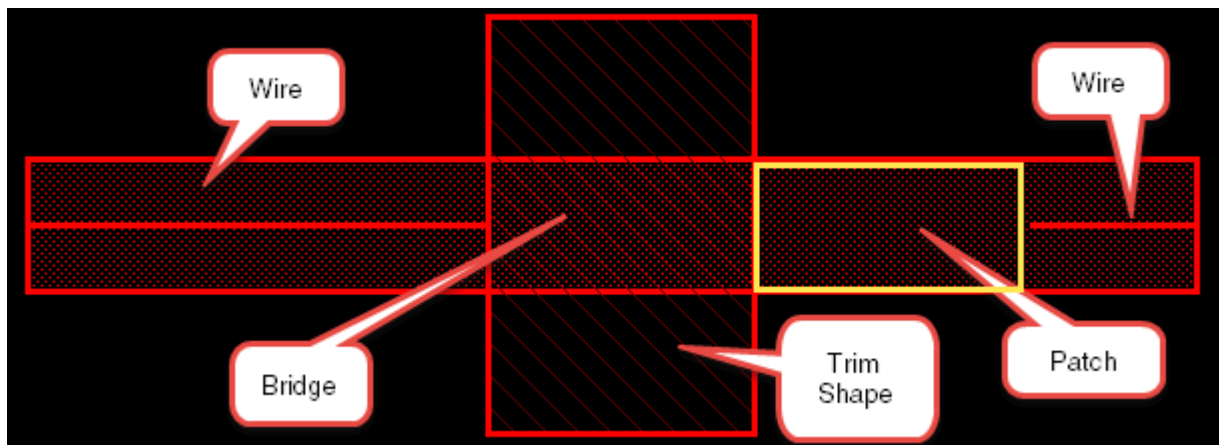
Interactive Colored Routing

In addition, prBoundary also impacts trim creation and insertion. If there is a prBoundary at the top-level, the trim shapes are inserted between pathSegs if the pathSegs are within the PR boundary. No trim shape is inserted between pathSeg out of of the prBoundary, When the prBoundary is deleted, the *Stretch* and *Create Wire* commands create trim shapes between the pathSegs. However, if there is no prBoundary at the top-level, there is no restriction for trim creation at the top-level.

Note: The *Create Stranded Wire* command does not support interactive trimMetal layer insertion.

The trim, bridge, and patch shapes are inserted interactively when the wire is created. Under the trim shape, there is a bridge shape on the metal layer. The width of the bridge shape is the same as that of the wire. There is also a patch shape on the metal layer that extends to already existing wires to the trim shape. The patch shape is inserted between the trim shape and the far end of the already existing wire.

The following figure shows an already existing wire and the new wire being created. After the new wire is placed, it results in inserting trim, bridge, and patch shapes. The patch shapes are represented by extensions on wires.

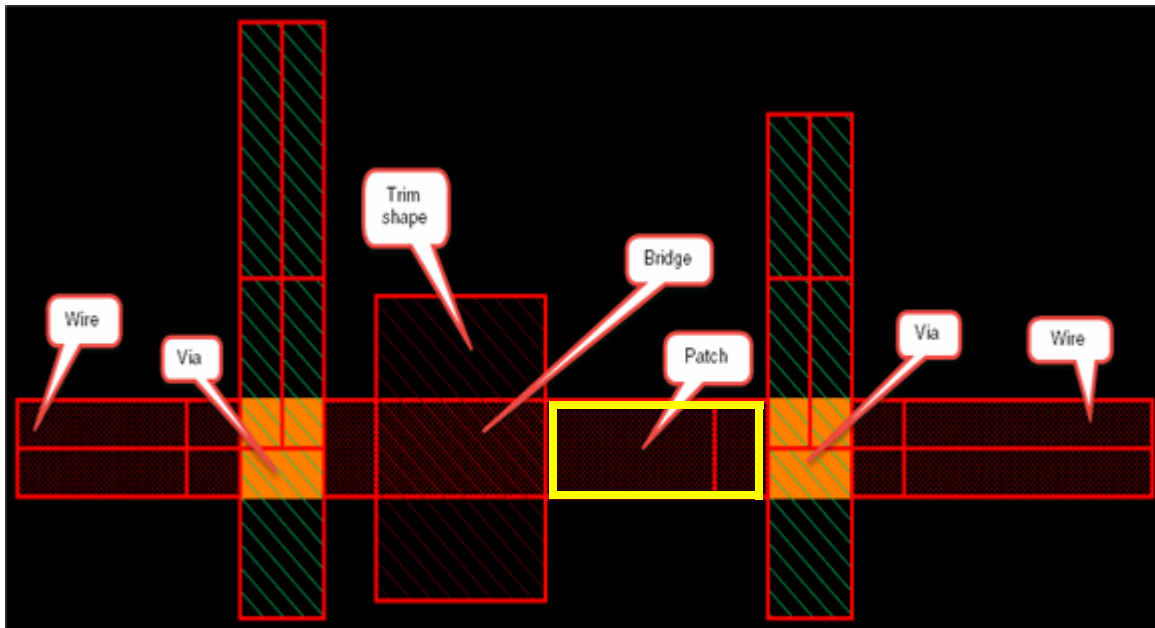


Similarly, the following figure shows an example where a wire with inserted vias already exists and a new wire with inserted vias is being created. The trim, bridge, and patch shapes get

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

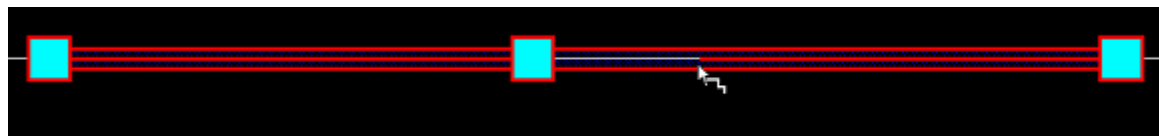
inserted after the new wire along with the inserted vias is placed, as shown in the following figure.



Another example, which is a special case, is when the `exceptSpacing` value is set to 0, then trim and bridge shapes are inserted at all wire ends, as shown in the following figure.



If you create another wire next to the existing wire with trim shapes at the wire end, the middle trim shape is merged and we get three trim shapes for the two wires that are created next to each other.



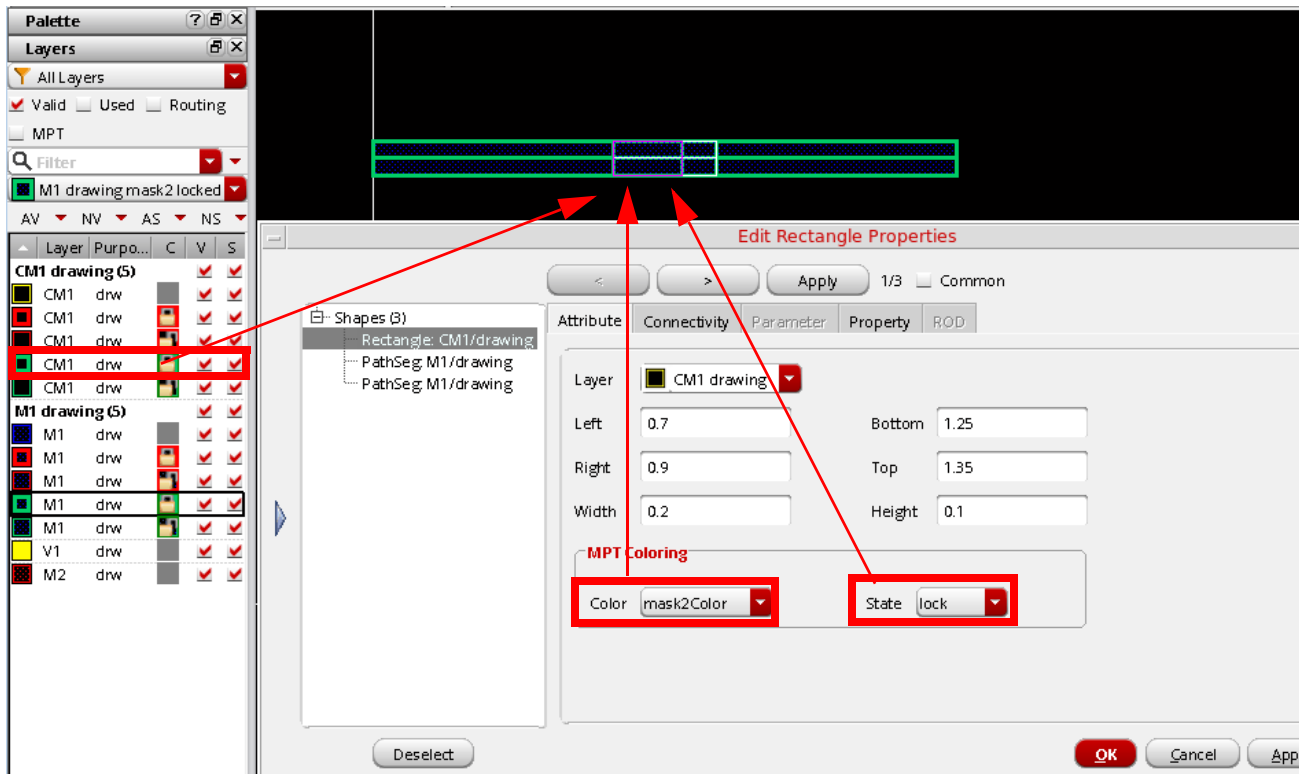
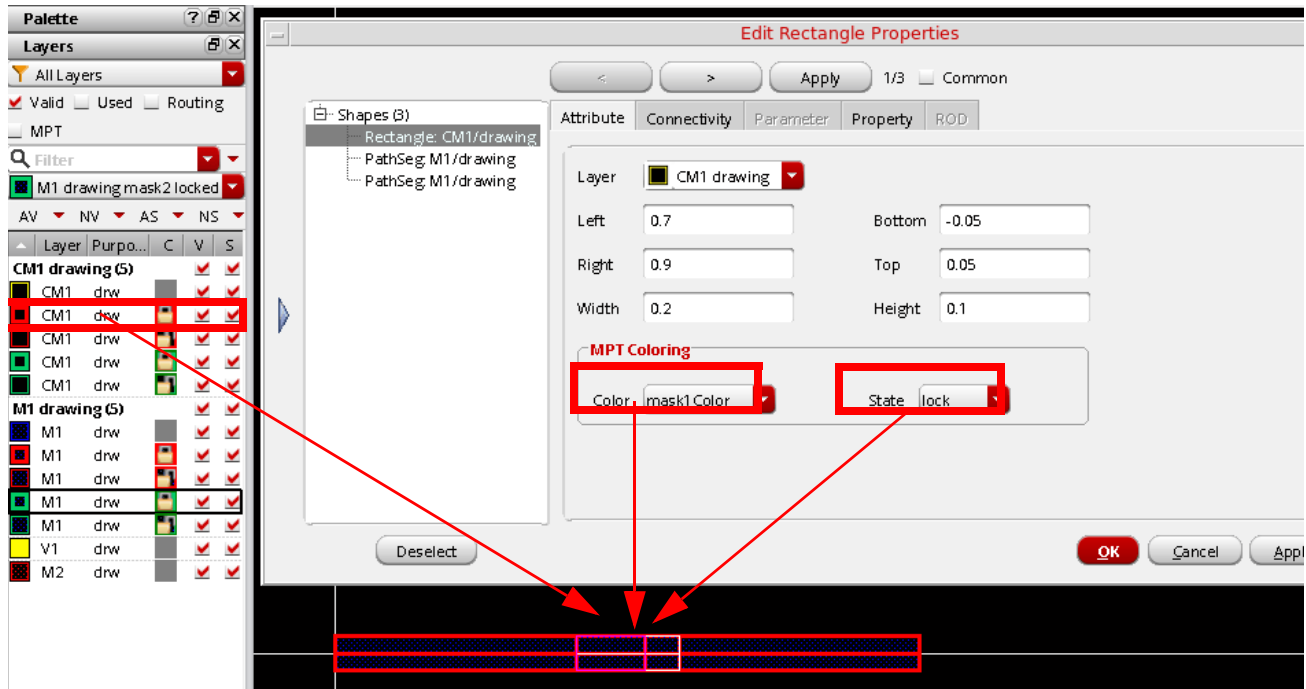
Supporting Colored TrimMetal Layers

The trim layer can be defined in the technology file. You can assign color to a trim layer and specify that a trim layer with a given color should cut a routing layer with that color. As a result, trim shapes are automatically assigned the correct color when they are added during interactive routing. For more information, see [trimShape](#).

Virtuoso Interactive and Assisted Routing User Guide

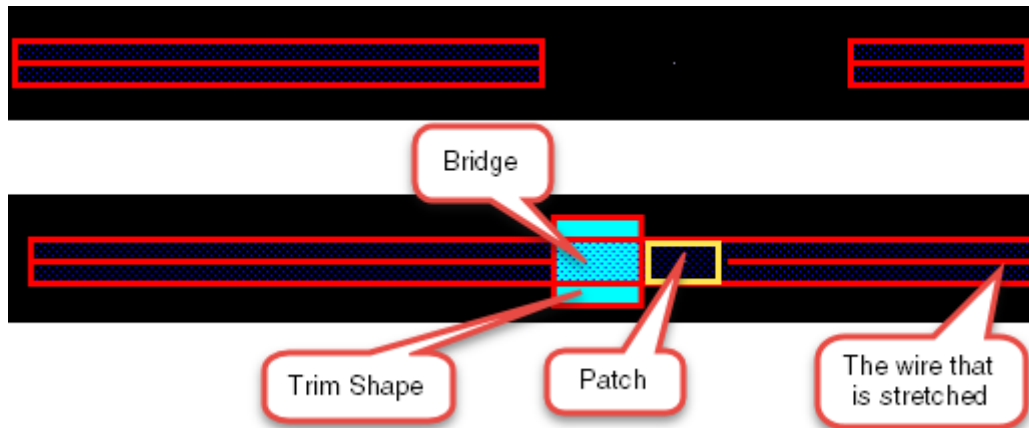
Interactive Colored Routing

For example, CM1 layer with mask1 color cuts Metal1 with mask1 color. Similarly, CM1 layer with mask2 color cuts Metal1 with mask2 color, as shown in the following figures.



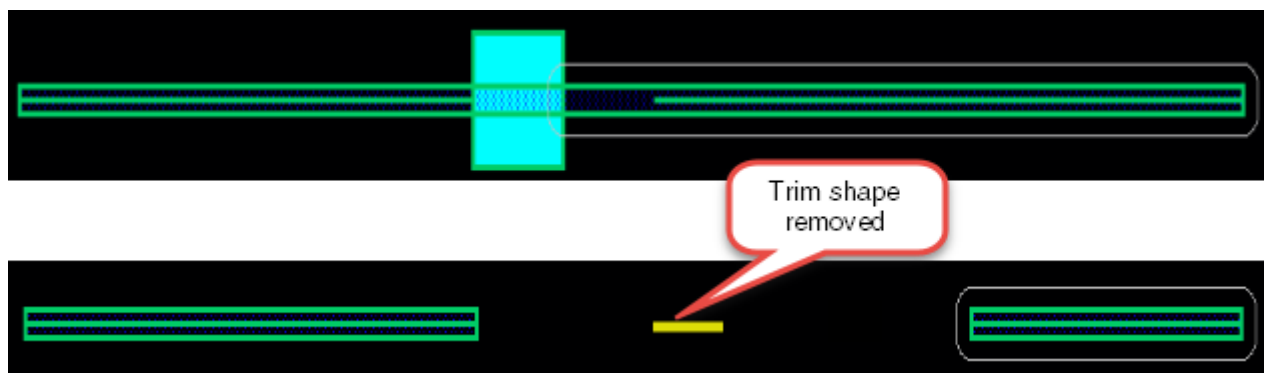
Supporting Colored TrimMetal Layers for the Stretch Command

The *Stretch* command supports automatic insertion of trim metal layers between two existing wires when one of them is being stretched. After the new wire is stretched, it results in insertion of trim, bridge, and patch shapes. The patch shapes are represented by extensions on wires, as shown in the following figure.



For more information, see [Inserting TrimMetal Layers in Interactive Colored Routing](#).

When you retract the stretched wire, the trim shape is removed, as shown in the following figure



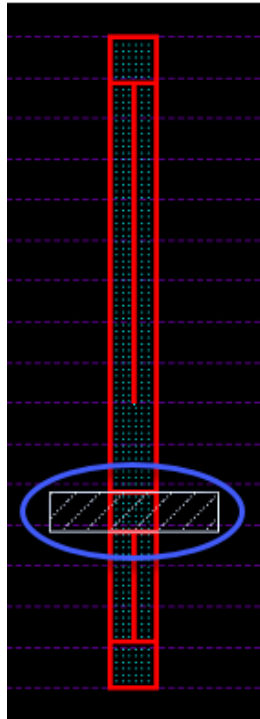
Selecting and Stretching a Trim Shape

When a trim shape is fully selected and there is only one selected trim shape, the trim insertion is deactivated to allow the *Stretch* command to move the trim shape and keep the abutted pathSegs connected. This is possible only if the `weTrimInsertion` environment

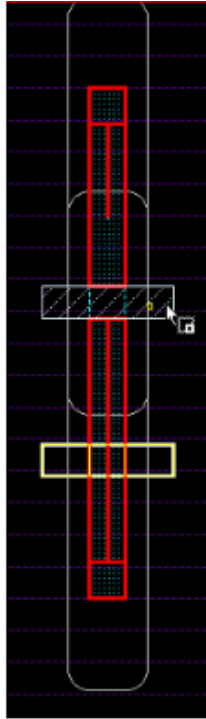
Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

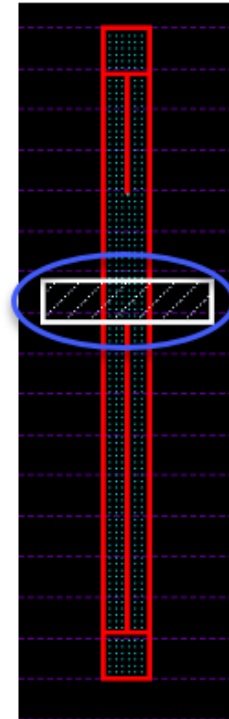
variable is enabled. The following figure displays how the trim shape keeps the pathSeg connected when the trim shape is selected and stretched.



Original trim Shape
on pathSegs



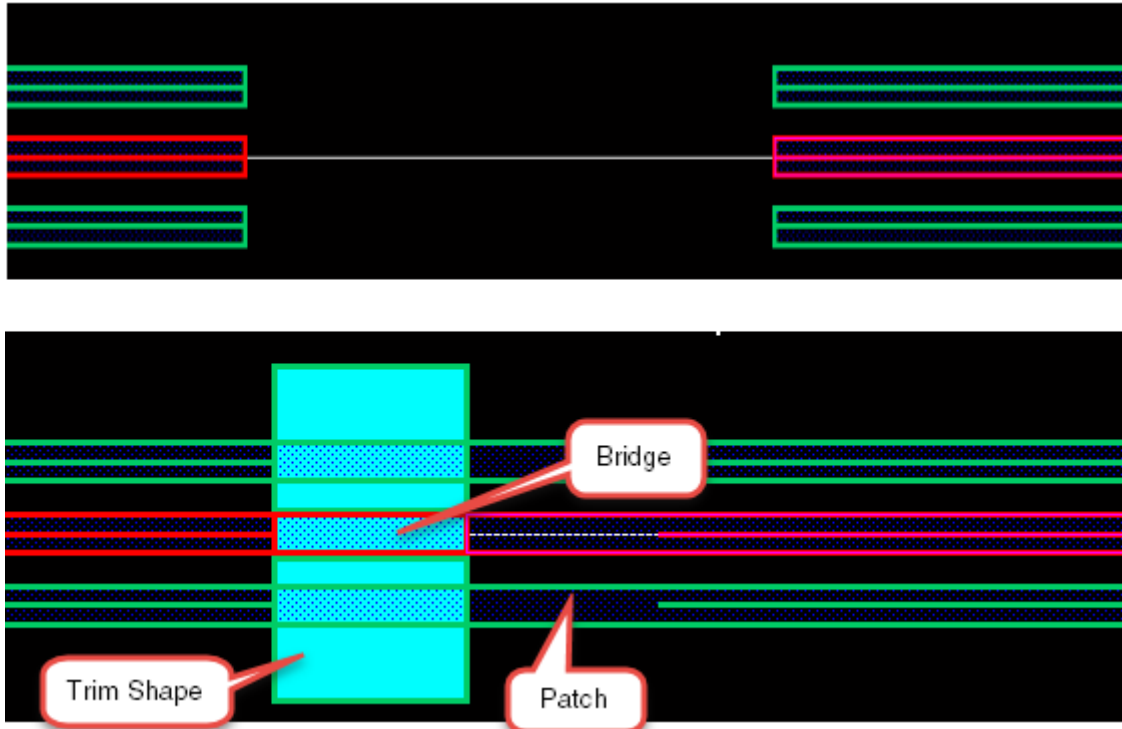
Selected trim Shape
being stretched



Trim Shape is moved keeping
the pathSegs connected

Supporting Colored TrimMetal Layers for Create Bus

The *Create Bus* command supports automatic insertion of trim metal layers between an already existing bus and the new bus being created. After the new bus is placed, it results in insertion of trim, bridge, and patch shapes. The patch shapes are represented by extensions on buses, as shown in the following figure.



For more information, see [Inserting TrimMetal Layers in Interactive Colored Routing](#).

Supporting Patch Metal Shapes

In Virtuoso, interactive routing represents the patch as an extension to the pathSeg that abuts the trim shape. The created and modified pathSegs, which are abutted to the inserted trims are flagged. This allows Innovus to quickly translate the oaDataBase to its internal database. However, If the abutted pathseg to a trim is not correctly flagged, Innovus does not generate the trim in its internal database and the trim does not appear in Innovus window.

For a given pathSeg, a patch is represented by two end styles (begin and end), both, none, or by any one of the ends. The patch type can be any of the following: `notPatch`, `beginStyle`, `endStyle`, `bothStyle`, `beginFull`, `endFull`, `bothFull`.

Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

The following scenarios illustrate the different patch types that can be inserted on a pathSeg.

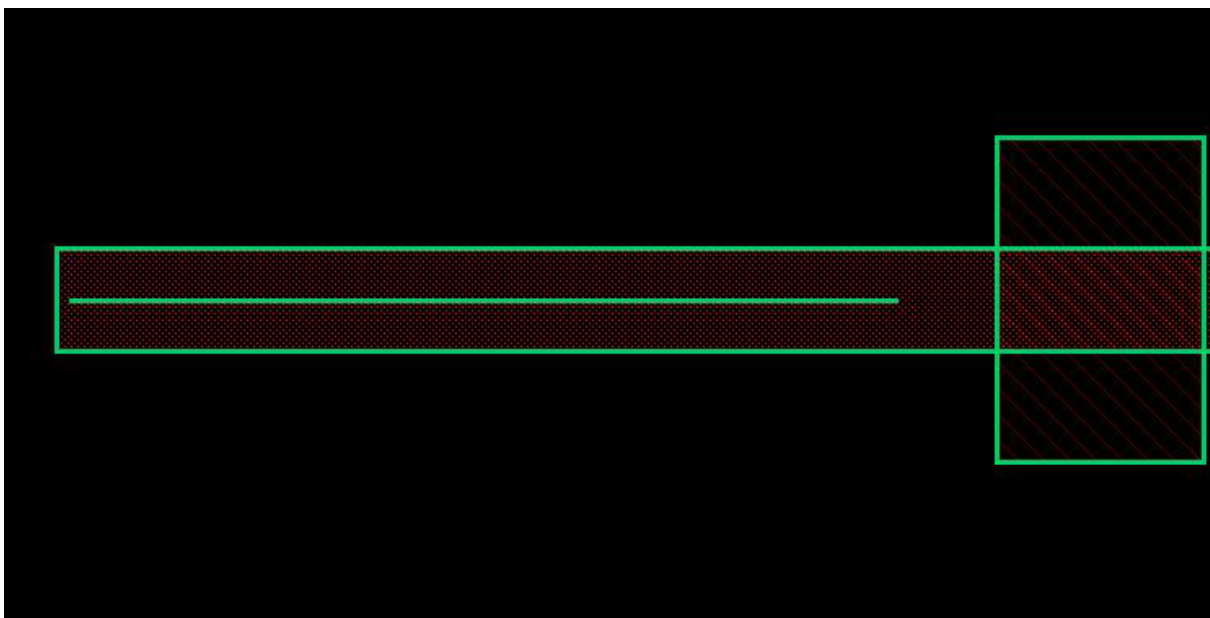
- The patch type is `beginStyle`.

In this case, the trim shape exists at the beginning with a variable extension of pathSeg.



- The patch type is `endStyle`.

In this case, the trim shape exists at the end with a variable extension of pathSeg.

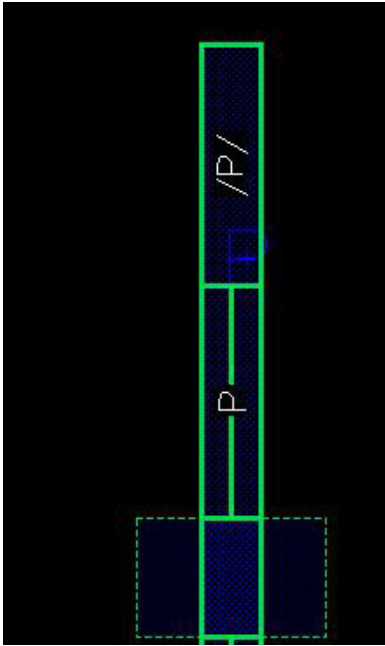


Virtuoso Interactive and Assisted Routing User Guide

Interactive Colored Routing

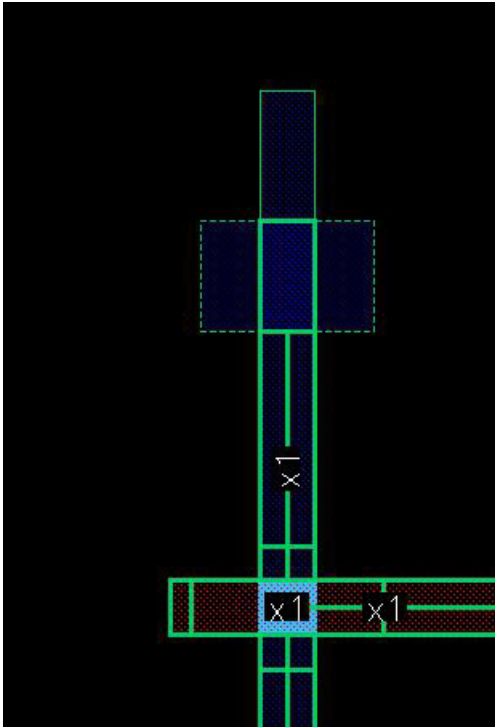
- The patch type is `beginFull`.

In this case, the trim shape exists at the beginning of the pathSeg with no variable extension (truncate style). The space between a trim shape on one end and the pathSeg in Figure A is filled with a patch shape (pathSeg2), as shown in Figure B below.



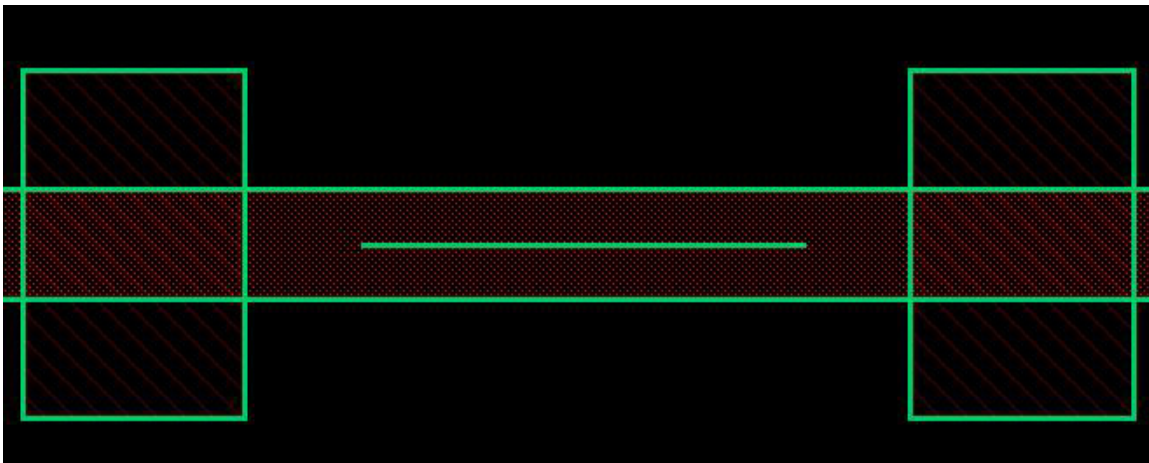
- The patch type is `endFull`.

In this case, the trim shape exists at the end of the pathSeg with no variable extension (truncate style). The space between a trim shape on one end and the pathSeg in Figure A is filled with a patch shape (pathSeg2), as shown in Figure B below.



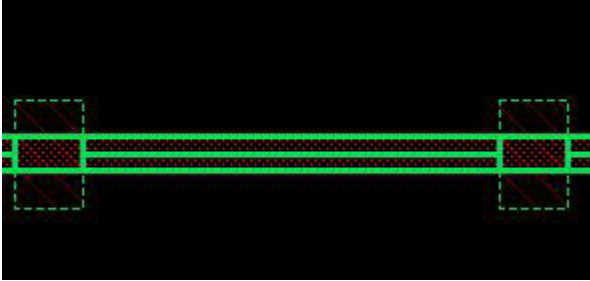
- The patch type is `bothStyle`.

In this case, the trim shape exists both at the begin and at the end of the pathSeg. The space between the trim shape at the begin and end of the pathSeg in Figure A is filled with patch shapes (pathSeg2 and pathSeg3), as shown in Figure B. The patch shapes are added with variable extension.



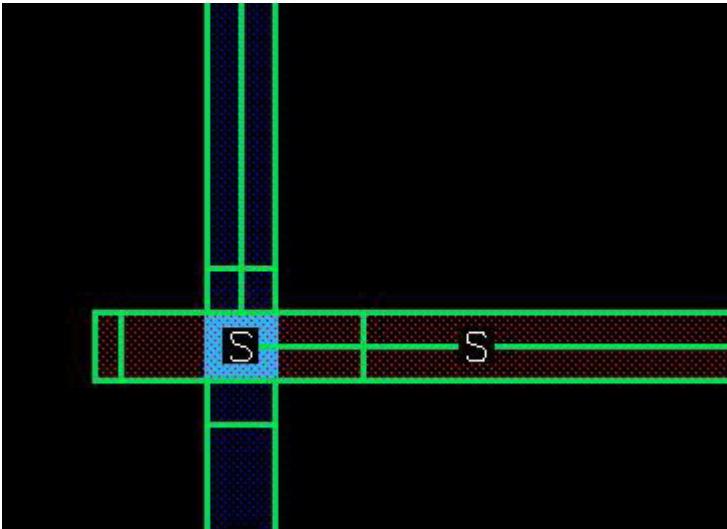
- The patch type is `bothFull`.

In this case, the trim shape exists both at the begin and at the end of the pathSeg. The space between the trim shape at the begin and end of the pathSeg in Figure A is filled with patch shapes (pathSeg2 and pathSeg3), as shown in Figure B. The patch shapes are added with truncate style at both ends.



- When the patch type is `notPatch`

In this case, the pathSeg line ends and does not have trim shapes. The following figure shows the `noPatch` patch type.



Supporting Width Spacing Patterns

Note: Requires Virtuoso Advanced Node Option for Layout (95511) license.

Wire editing has been enhanced to support snap pattern definitions called [widthSpacingSnapPatternDefs](#). The width spacing pattern (WSP) definitions used with `widthSpacingSnapPatternDef` are stored either in the technology database or the design database. When a width spacing snap pattern definition is enabled as a global grid through a `snapGridVertical` or `snapGridHorizontal` constraint, global WSP-based editing is supported in wire editor. In addition, wire editing allows individual regions in a cellview to use different WSPs instead of the global grid. To create a region, see [Creating Width Spacing Pattern Regions](#). The `widthSpacingSnapPatternDef` defined in a region need not be used as a global grid in the cellview.

Furthermore, wire editing allows multiple width spacing snap pattern definitions to share the same LPP. This means that an LPP of a region does not directly identify a unique width spacing snap pattern definition. Therefore, a new string attribute can be associated with a region to determine the width spacing snap pattern definition with which it is associated. To set the `widthSpacingSnapPatternDef` to be used for a region, see [dbSetPatternRegionWidthSpacingSnapPatternDefName](#).

~~This chapter describes the following width spacing pattern features.~~

- ~~[Width Spacing Pattern Display in Wire Editing](#)~~
 - ❑ ~~[Dynamic Display](#)~~
 - ❑ ~~[Alpha Blending](#)~~
- ~~[allowedWireTypes Support](#)~~
- ~~[Snapping Behavior in WSP During Wire Editing](#)~~
 - ❑ ~~[Enabling Wire Snapping](#)~~
 - ❑ ~~[Snapping of Wire in Width Spacing Patterns](#)~~
 - ❑ ~~[Automatic Via Insertion during Wire Snapping](#)~~
 - ❑ ~~[Snapping on WSP for Assisted Routing Commands](#)~~

- [Working with Width Spacing Patterns During Wire Editing](#)
 - ❑ [Switching Patterns Dynamically](#)
 - ❑ [Crossing Region Boundaries](#)
 - ❑ [Wire Editor Gravity](#)
- [Automatic Switching of the Active Pattern Across Regions](#)
- [Working with Selected Sets of Wires](#)
- [Nested WSP Regions](#)
- [Supporting pathSegs with Different WSPs across Regions](#)
- [Wire Editor Hierarchy Support](#)
- [Pattern Flipping Support](#)

Width Spacing Pattern Display in Wire Editing

When using width spacing patterns, it is essential that you are able to visualize the available tracks and their respective widths in the canvas. Therefore, the following display enhancements have been made available for WSP.

- [Dynamic Display](#)
- [Alpha Blending](#)

Dynamic Display

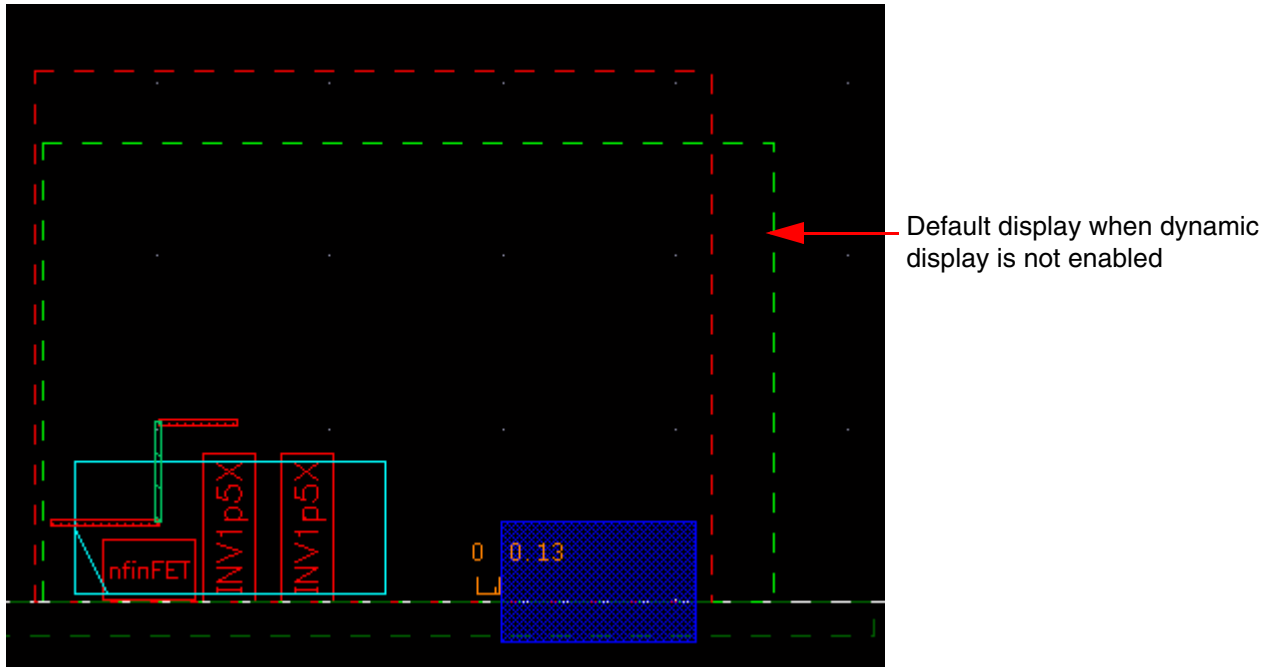
Dynamic display is visible in wire editing only when the *Snap Pattern* option is selected from the *Wires* drop-down list in the Layout Editor Options form and the reference object is snapable to the WSP grid. The dynamic display feature facilitates the display of accessible or valid tracks. This feature displays the tracks based on the current layer-purpose pair, active pattern, `minWidth` constraints, `wireType` constraints, and regions. The dynamic display also displays the valid tracks of upper and lower layers.

During wire creation and editing, the active pattern in the applicable WSP track is displayed in the design automatically as soon as the *Create Wire* command is invoked. While creating a wire, the corresponding WSP grid for the selected layer is displayed, even if the visibility of the grid is turned off in the *Grids* panel of the Palette assistant.

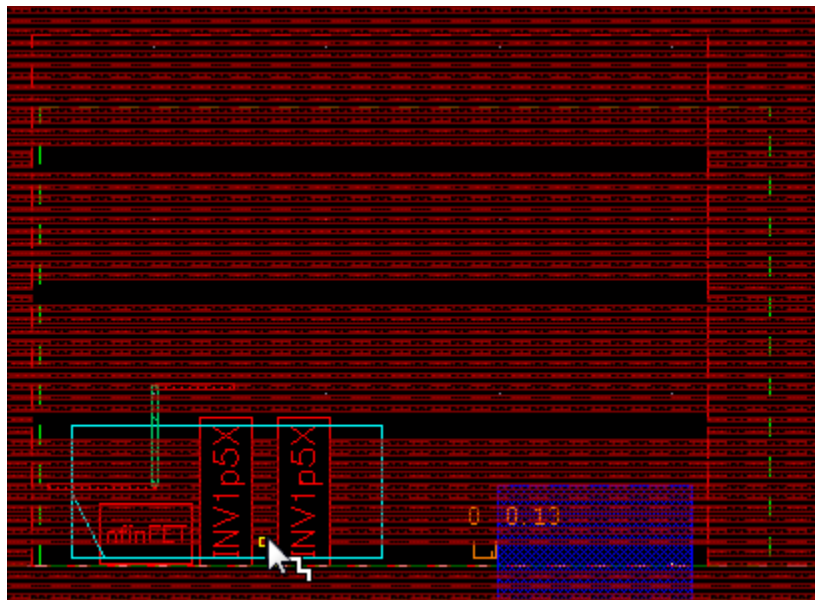
Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

The following figure shows the display of tracks when the *Create Wire* command is invoked.



Automatic track display when the *Create Wire* command is invoked



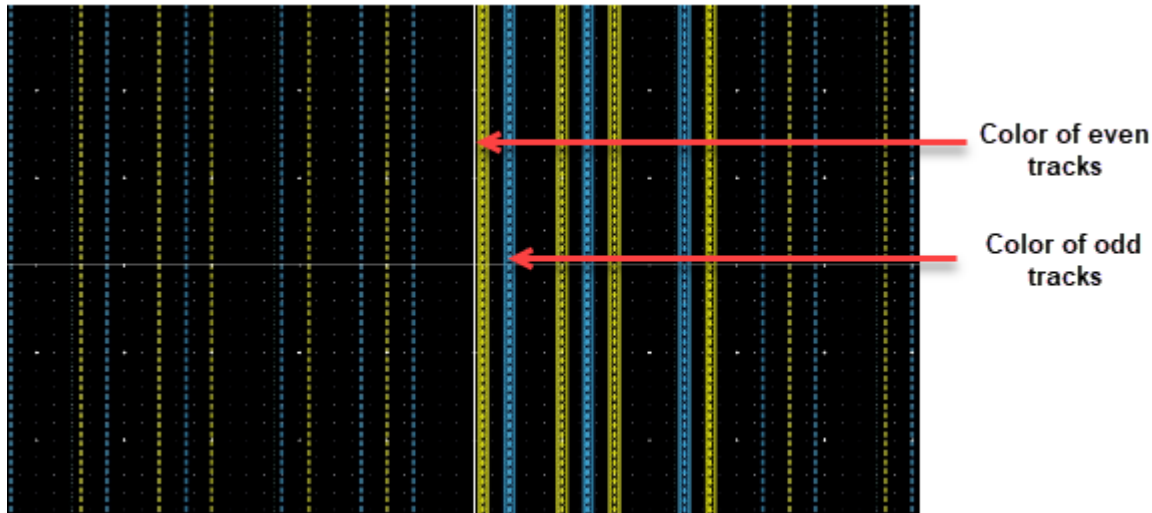
Dynamic Display for Colored snapPatternDef Tracks

During wire creation, for `snapPattern` the even and odd tracks are displayed in alternate colors based on the packets defined in the display resource (.drf) file. The track immediately

Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

to the right of the anchor point gets the regular packet. The track after that gets the override packet.



Important

This feature applies only to Snap Pattern Definition (SPDef) objects and not to Width Spacing Snap Pattern Definition (WSSPDef) objects.

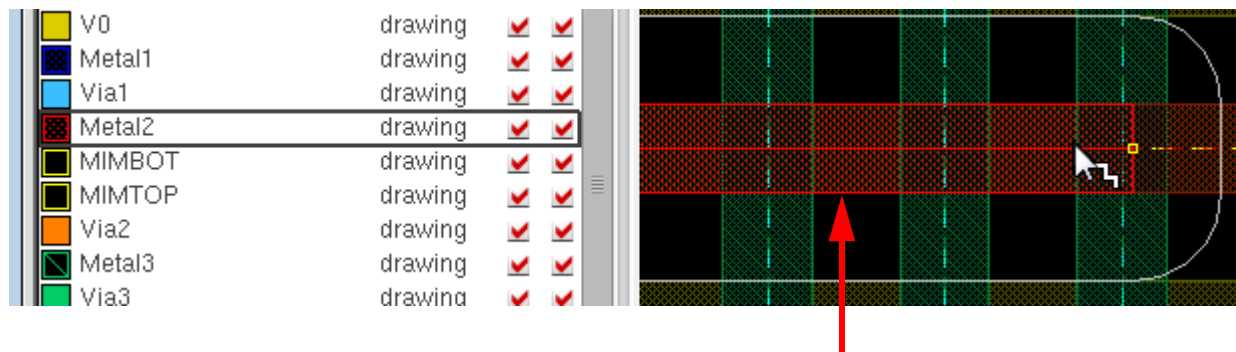
Alpha Blending

During wire creation and editing, there is a transparent display of the current track and upper and lower layer tracks in the design as soon as a layer is selected in the *Layers* panel of the Palette assistant and the *Create Wire* command is invoked. This is known as alpha blending. This feature helps you to visualize the layer on which a wire is being created. The following

Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

figure shows the alpha blending display when the `Meta12` layer is selected in the Layers panel of the Palette assistant and the `Create Wire` command is invoked.

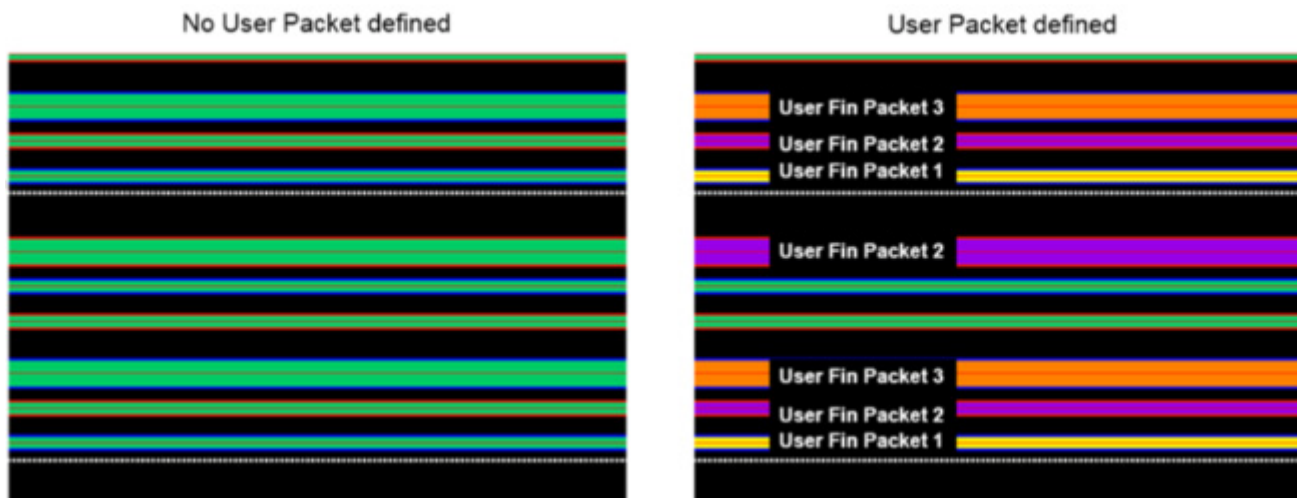


The alpha blending display when `Meta12` layer is selected and the `Create Wire` command is invoked.

The transparency at different levels for the track fill or the track outline, is controlled by several environment variables. For more information about the alpha blending environment variables, see [Support for Alpha Blending](#).

Customizing Display of WSP Tracks

To visually differentiate and easily identify some WSP tracks, you can now customize stipple patterns. You can define a display packet and attach it to a WSP track. The following figure shows WSPs when a display packet is defined for a WSP track and when it is undefined (and a default display packet is used).



For more information, see [widthSpacingPatterns](#) in the Virtuoso Technology Data ASCII Files Reference. Also, see [techCreateWidthSpacingPattern](#) and [techCreateWidthSpacingPatternWithColor](#), and [wspCreateWSP](#) SKILL APIs.

allowedWireTypes Support

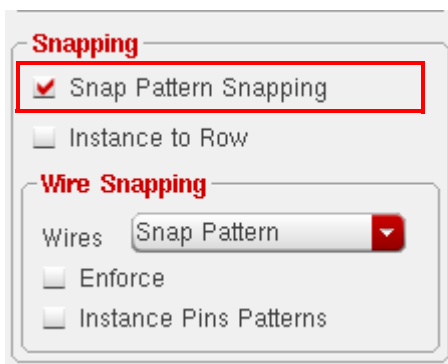
The wire editing supports the new `allowedWireTypes` constraint. When the `allowedWireTypes` constraint is set on a net in a design, then wire editing on that net conforms to the constraint rules and only shows and snaps to the tracks whose `wireType` matches the value of the constraint.

Snapping Behavior in WSP During Wire Editing

Enabling Wire Snapping

To enable snapping of wires, you can specify the *Wire Snapping* options in the Layout Editor Options form.

1. Choose *Options – Editor*.
2. In the Layout Editor Options form, the *Snap Pattern Snapping* option is selected by default. Select the *Snap Pattern* option from the *Wires* drop-down list in the *Wire Snapping* group box. When the *Snap Pattern* option is selected, the *Enforce* and *Instance Pins Patterns* check boxes are automatically enabled.



3. Select the *Enforce* check box if you want the wire to inherit the width and the direction of the active WSP pattern. When the *Enforce* check box is selected, the wire is snapped to the centerline of the track pattern. Also, the wire inherits the width of the active WSP pattern and the direction in which the wire is created is the same as the direction of the track pattern. By default, the *Enforce* check box is selected.

However, if the *Enforce* check box is deselected, while the wire is snapped to the centerline of the track pattern, the width of the wire is the default width and there is no restriction on the direction in which the wire is snapped.

Note: The *Enforce* check box is only enabled when the *Wires* option is either set to *Track Pattern* or *Snap Pattern*.

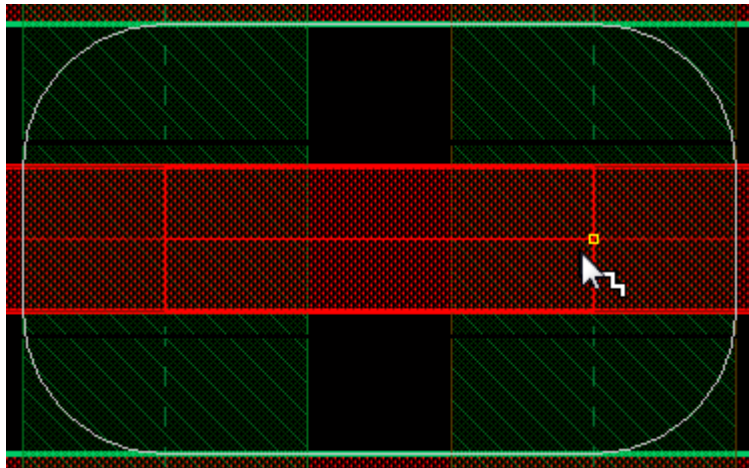
Also, the width associated with a WSP track can be set to 0. Wire Editor supports the track that has 0 width. The track with 0 width is similar to the trackPattern use model; where any wire width can be used for a track. In addition, you can press the `Ctrl + Shift` key and scroll the mouse wheel up or down to dynamically change the width of a wire.

4. To view the patterns that are available in an instance, select the *Instance Pin Patterns* check box. When this option is selected, you can view the active pattern of the instance. However, if this option is deselected, the active pattern of the top level is visible even in the instance.

Snapping of Wire in Width Spacing Patterns

During wire creation, the valid track patterns of a layer are displayed. Also, only the tracks corresponding to the current layer and a layer above or below it are shown.

When a wire is created on a metal layer, the track magnet automatically pulls the wire towards the track intersection and the wire is snapped to the nearest track of the active pattern. In addition, the created wire inherits the width and direction of the track to which it is snapped, as shown in the following figure.

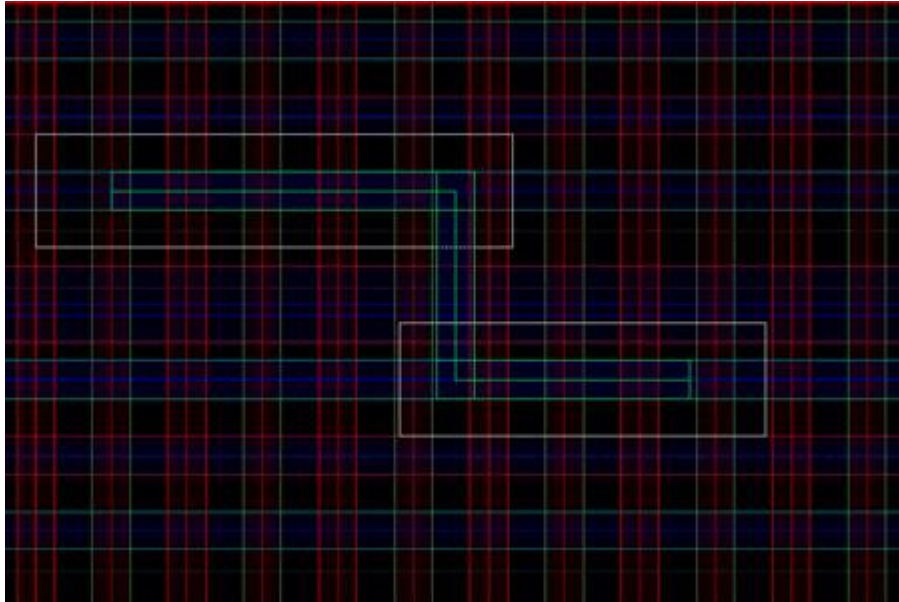


Also, while snapping, the Create Wire command follows the tracks and lets you create a jog by remaining in the same metal layer even if there are only horizontal tracks on a metal layer and vice versa. The wire is created as a Z-shape, where the first and last pathSeg are in the preferred routing direction and the middle pathSeg is in the non-preferred routing direction. This is only possible when the `weEnforceDirFromRoutingPattern` variable is set to `nil`. However, the automatic via insertion is disabled when the environment variable

Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

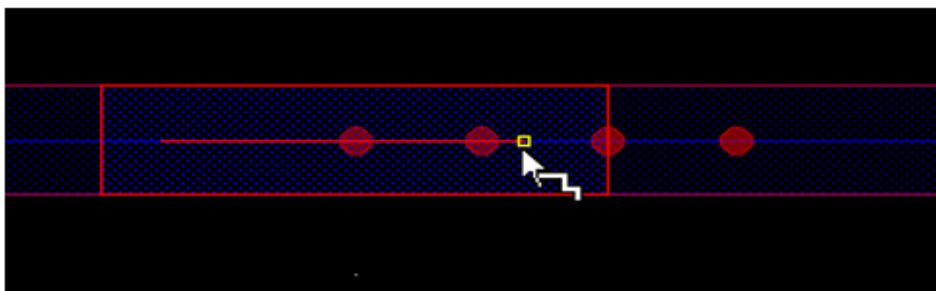
`weEnforceDirFromRoutingPattern` is set to `nil`. In addition, a via can be created only in the preferred direction and not in the non-preferred direction.



While creating a wire on a defined region, the wire is snapped to the closest strip lines on the routing direction. The high-level steps that are followed for snapping a wire to the width spacing patterns are listed below:

- Specify the width spacing pattern definition defined in the ASCII technology file.
- Specify the name of the layer, typically the *Active* layer, on which a wire will snap.
- Create a wire.

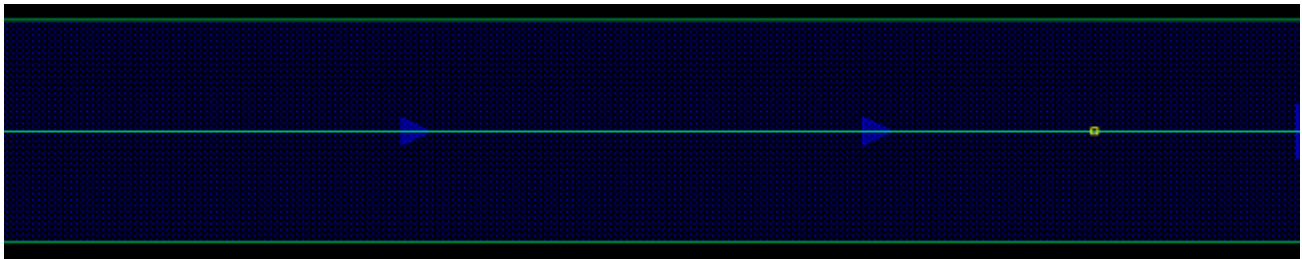
Also, while creating a wire, orthogonal WSP grid constraint defined in the technology file automatically adjusts the wire with a truncate extension value so that the begins and ends points of a wire are on the grids. The snapping points that are applicable to both upper and lower are visible as dots, as shown in the following figure.



Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

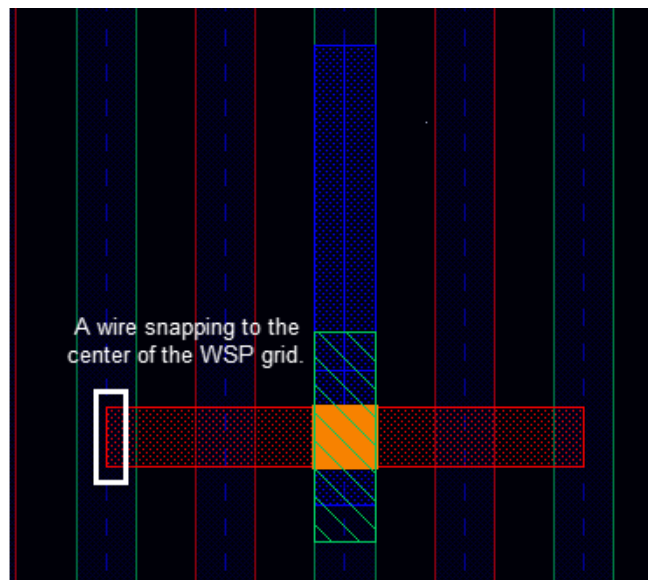
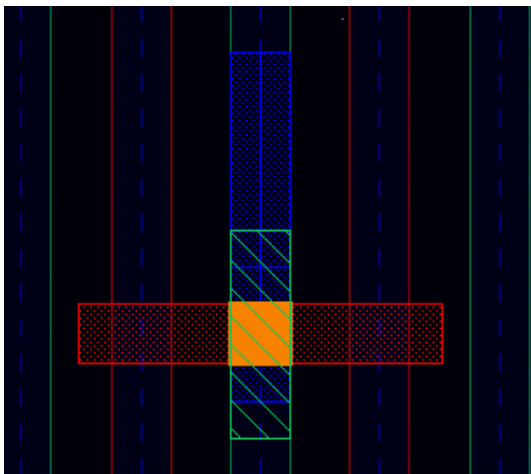
However, when the snapping points are either applicable to lower or upper, then the `orthogonalWSPGrid` is represented by a small triangle, as shown in the following figure,



In addition, the via enclosure of intermediate layers extend to align to the grid defined by `orthogonalWSPGrid`.

```
spacings (  
    (orthogonalWSPGrid "M2" "M1WSP")  
    ) ; spacing
```

In the above example, a wire is created On M1 layer and a stack via up to M3 is added. The via enclosure on M2 is extended to align the WSP grid on M1.



Video

For a short demonstration on how WSPs are used to control wire creation and routing, see [Create Wires on WSP](#).

Now, you can define `orthogonalWSPGrid` constraint in the local region which is different from the one that is defined in the global region.

Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

Note: When you are creating a local region with orthogonal `WSSPDef`, it requires that the primary layer has a standard routing `WSSPDef` selected as well.

The `orthogonalWSPGrid` constraint in the local region overrides the `orthogonalWSPGrid` constraint defined in the global region.

When the local end grid of a region differs from the local end grid of the global region, the wire snaps to the orthogonal grid near the boundary instead of the boundary of the region. This implies that the yellow cursor is not allowed to be placed near the boundary.



Support for snapPattern singleTrackCenter

To support the enhanced snapping behavior, a specific attribute `singleTrackCenter` is added to `snapPatternDefs` in the technology file.

```
layerRules(  
  snapPatternDefs(  
    (t_name (tx_layer tx_purpose)  
      ['snappingLayers  
        (  
          ...  
          'snappingMode "singleTrackCenter"  
          ...  
        )  
      ]  
    )  
  ) ;snapPatternDefs
```

Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

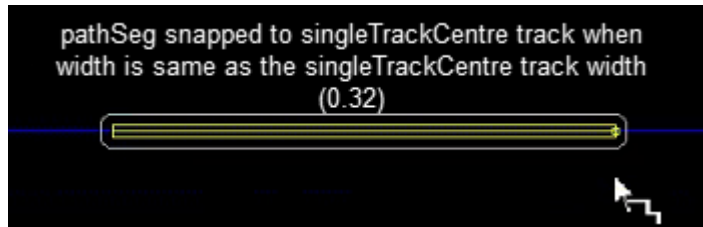
```
) ;layerRules
```

Some scenarios supported for this enhancement are as follows:

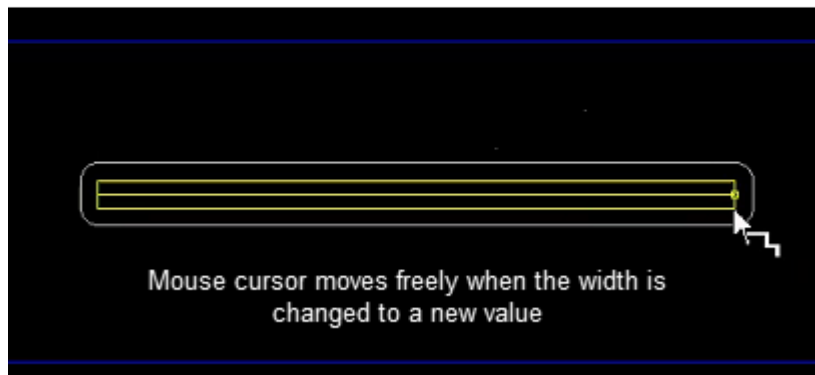
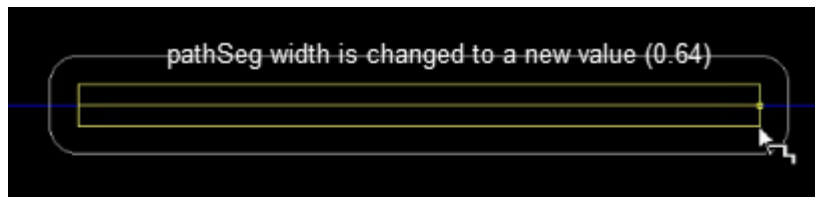
- Changing Width and Snapping
- Sliding a pathSeg

Changing Width and Snapping

When you start creating a pathSeg and the width of the pathSeg is same as the `singleTrackCenter` track width, the pathSeg is automatically snapped to the `singleTrackCenter` track and only `singleTrackCenter` track is displayed.



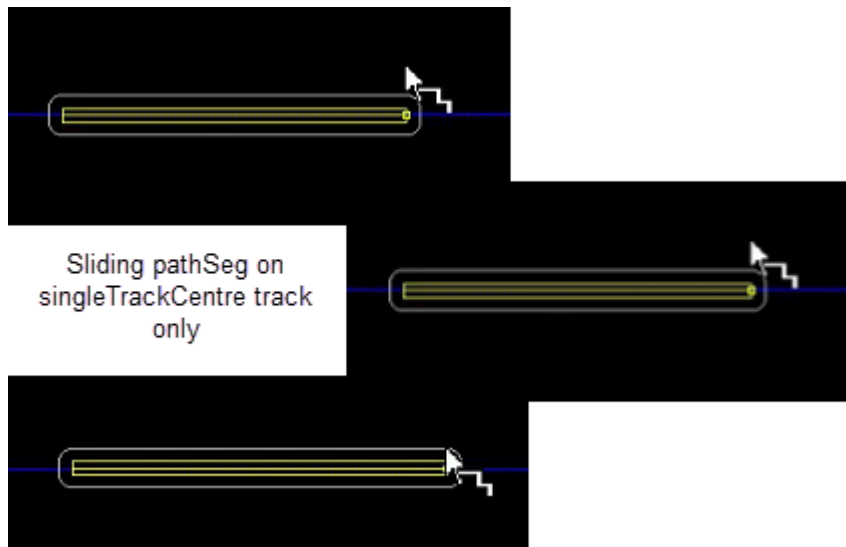
However, if the pathSeg is on a `singleTrackCenter` track and the width of the pathSeg is changed to a new value, the width of the pathSeg is changed and the mouse cursor moves freely in the design.



In case, the `pathSeg` is not on a `singleTrackCenter` track, the width of the `pathSeg` cannot be changed to a `singleTrackCenter` track width and a warning message is displayed in CIW.

Sliding a `pathSeg`

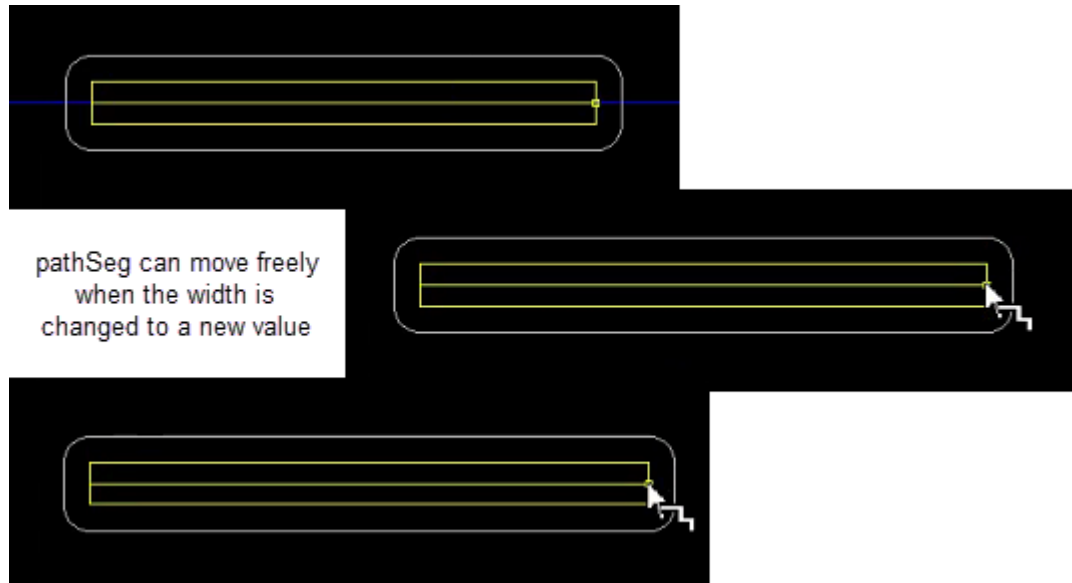
Use the width before sliding to determine if the wire needs to be snapped to the `singleTrackCenter` track. If the width before sliding is the same as the `singleTrackCenter` track width, the wire can only be moved to the `singleTrackCenter` track. Otherwise, it can be snapped to any track.



While sliding a `pathSeg` from a `singleTrackCenter` track to a non `singleTrackCenter` track, perform the following steps:

1. Change the width of the `pathSeg` to another value, which is not the same as the `minWidth`. (The width is not adjusted to the `singleTrackCenter` track width when it is on the `singleTrackCenter` track)

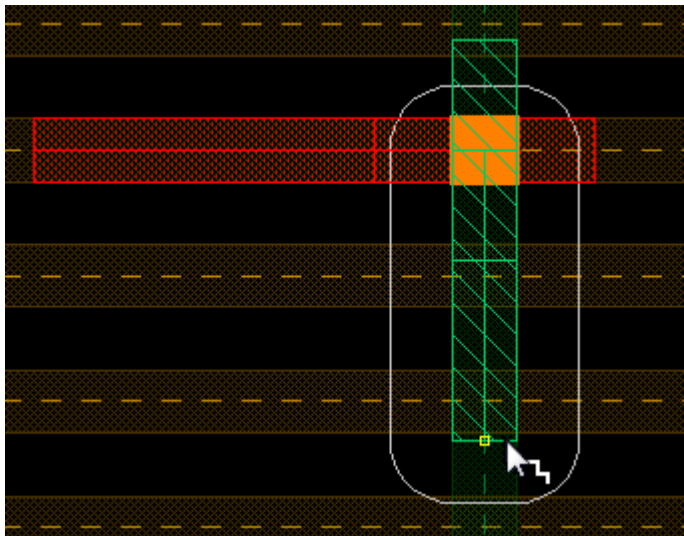
2. Press `Ctrl + Shift` to slide the pathSeg. The pathSeg can now move freely.



3. (Optional) Change the width of the pathSeg, if required.

Automatic Via Insertion during Wire Snapping

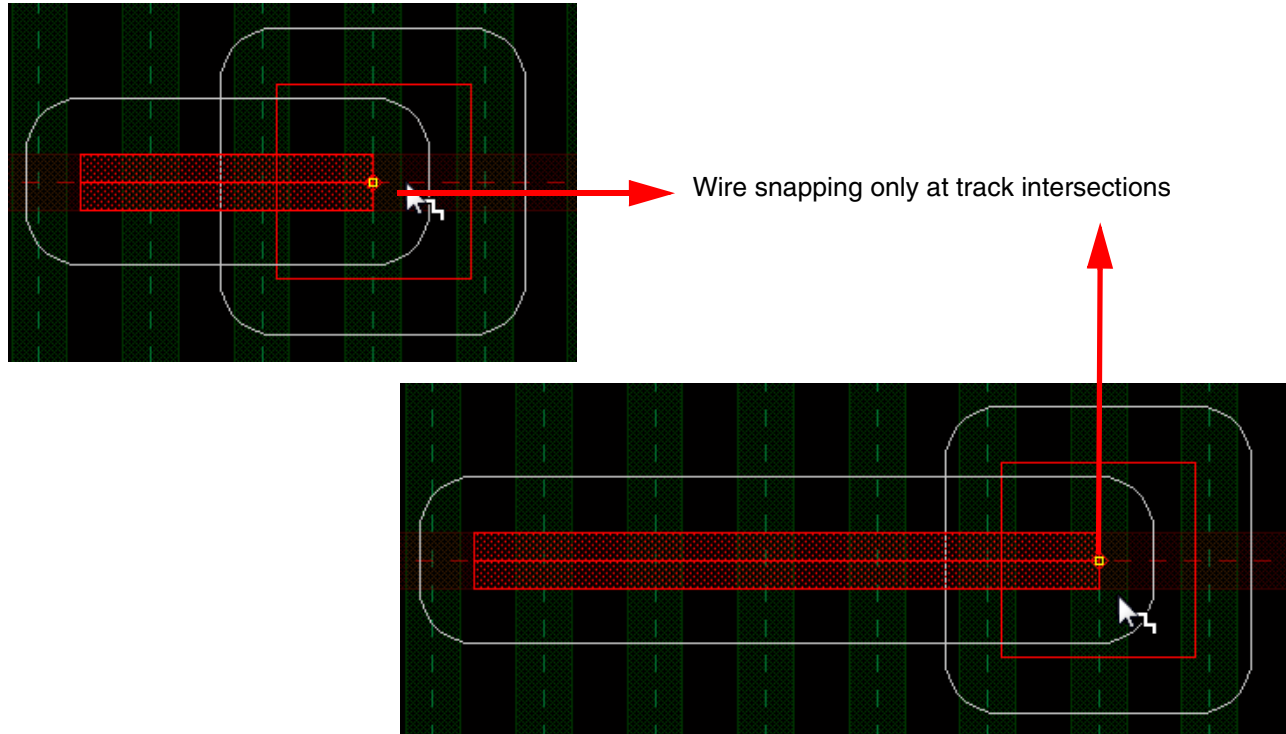
During wire creation, the track magnet snaps to the intersection of the upper or lower layer tracks. When you click on the track intersection, a via is automatically placed using the active width spacing pattern and the wire then changes direction and moves in the perpendicular direction, as shown in the following figure.



Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

You can also place a via before clicking on a track intersection by using the *Via up to <layer name>* and *Via Down to <layer name>* options in the *Create Single Wire* context-sensitive menu. In this case, the wire always snaps at the track intersection, as shown in the following figure.



After the via is automatically placed, one of the metal layers is selected by default if the tracks of the lower and upper layers coincide. However, you can switch between the valid tracks of the upper and lower layers by selecting the *Cycle Snapping Layer* option in the *Use Width Spacing Pattern* submenu.

In the following set of figures, Figure A shows that the wire is created on `Metal4`, and when the via is placed on the track intersection, `Metal5` is selected as the default layer.

Figure B displays the switch between the selected layers when the *Cycle Snapping Layer* option is clicked and `Meta13` layer is now selected.

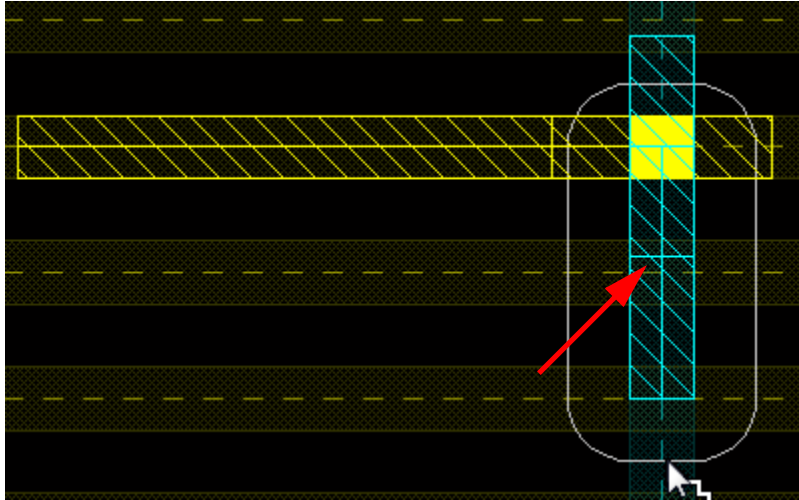


Figure A: Default layer selection (`Meta15`) for via insertion

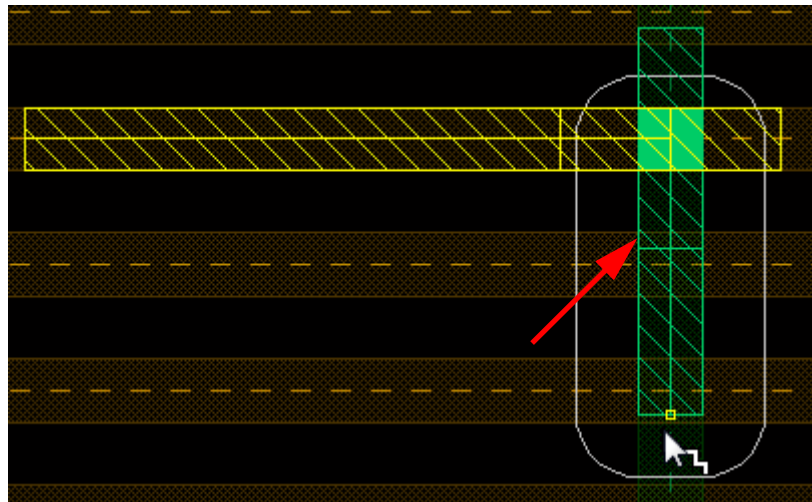


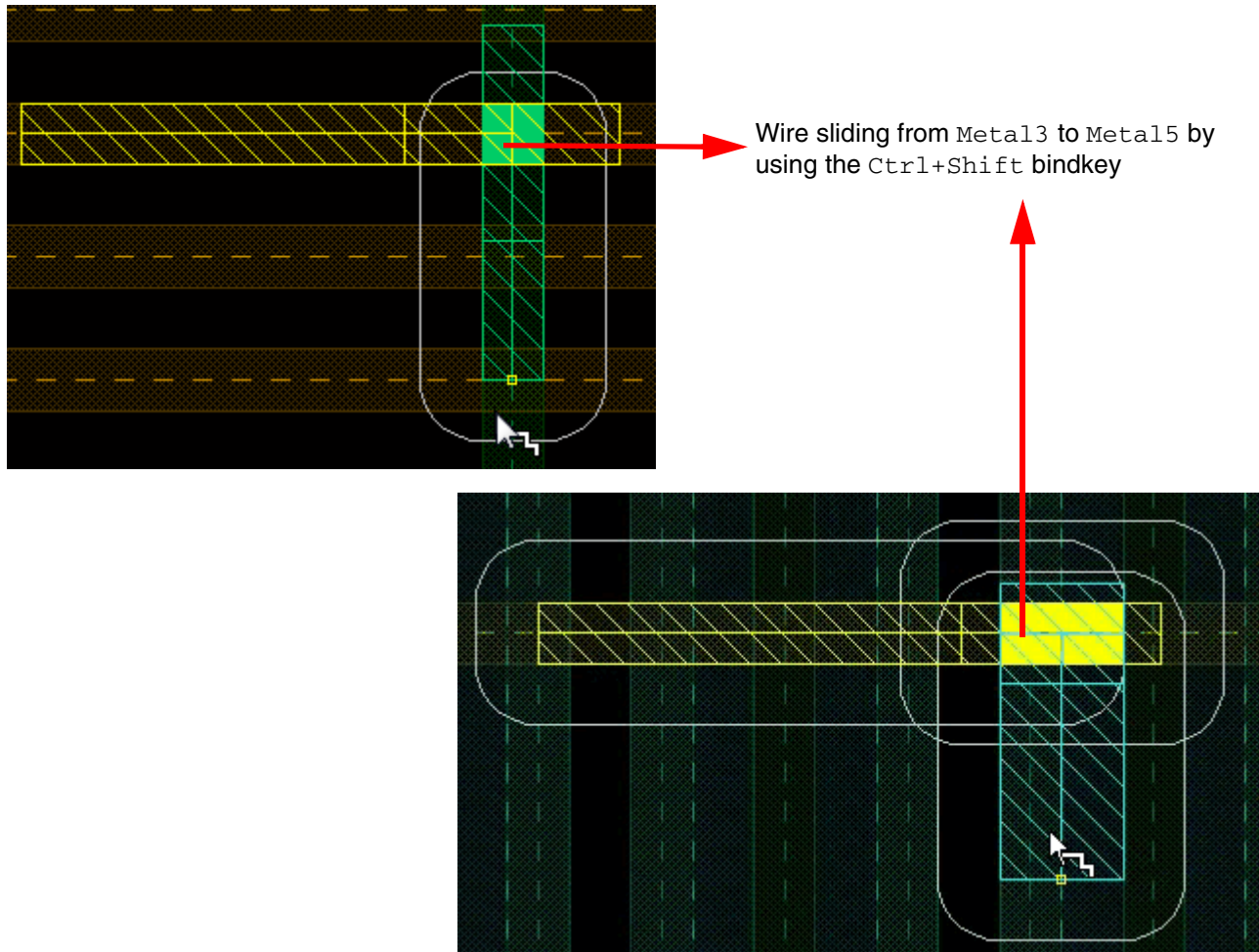
Figure B: Layer selection (`Meta13`) using Cycle Snapping Layer option

If the tracks of the lower and upper layers do not coincide, which means that the active pattern on the upper layer is different from that in the lower layer, then the layer at the track intersection is picked. However, you can slide the wire across the tracks to automatically change the layer and the corresponding via using the `Ctrl+Shift` bindkey. The following

Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

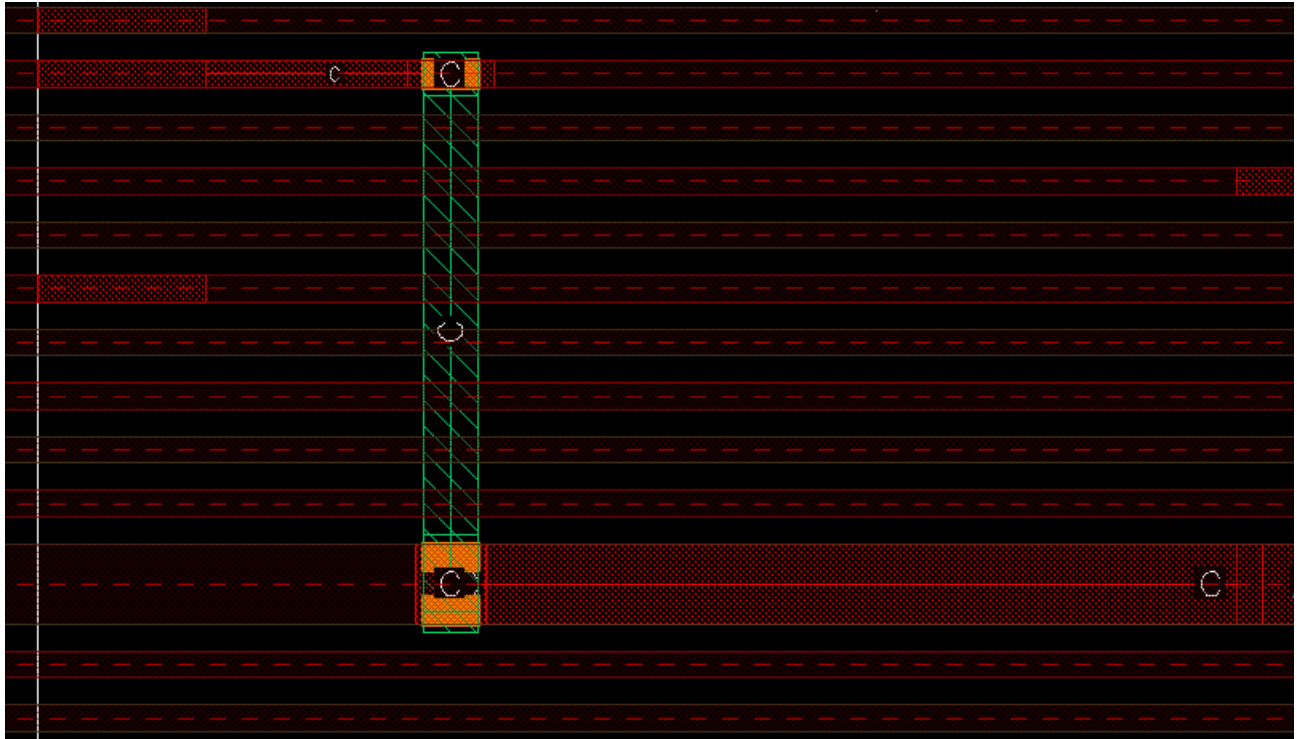
figure displays how the wire slides across the tracks and changes the layer and the corresponding via.



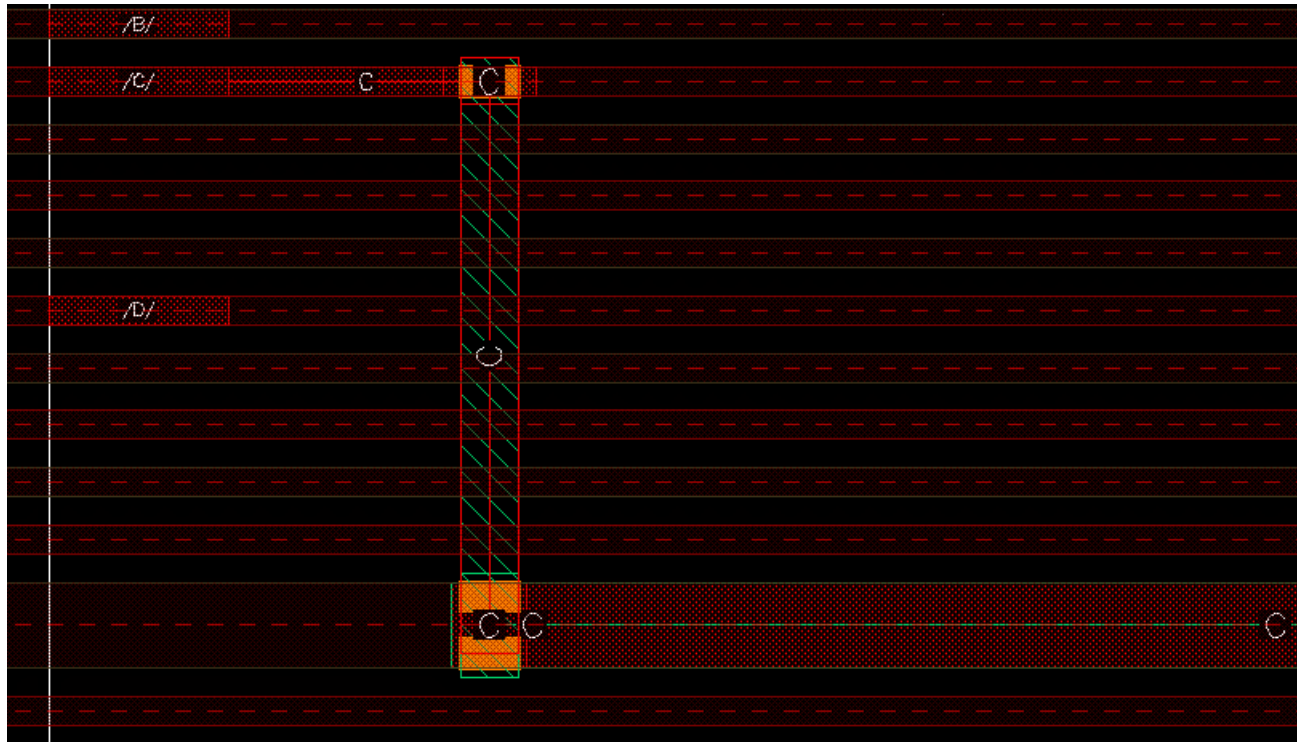
It also allows you to change the tracks on the same layer. The layer changes when the tracks are not aligned. If the tracks are aligned, the previous layer is preserved.

Snapping on WSP for Assisted Routing Commands

In the Layout Editor Options form, when you select the wire snapping option, *Snap Pattern*, the segments and vias are snapped to the active width spacing pattern track. The new segment inherits the width of the track it overlaps, as shown in the following figure.



In addition, when the MPT support is enabled in Palette, the wire segment also inherits the color and lock information of the active width spacing pattern track, as shown in the following figure.



Working with Width Spacing Patterns During Wire Editing

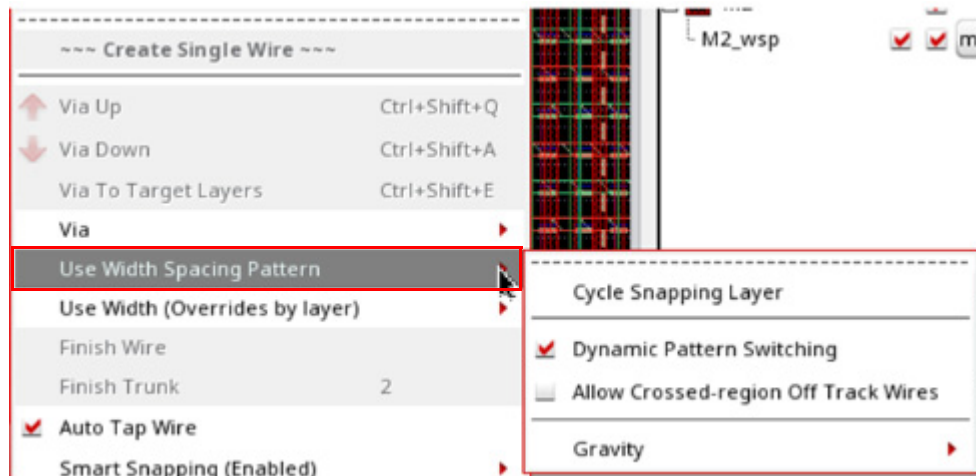
The *Use Width Spacing Pattern* option in the *Create Single Wire* context-sensitive menu lets you perform the following tasks.

- Switching Patterns Dynamically
- Crossing Region Boundaries
- Wire Editor Gravity

Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

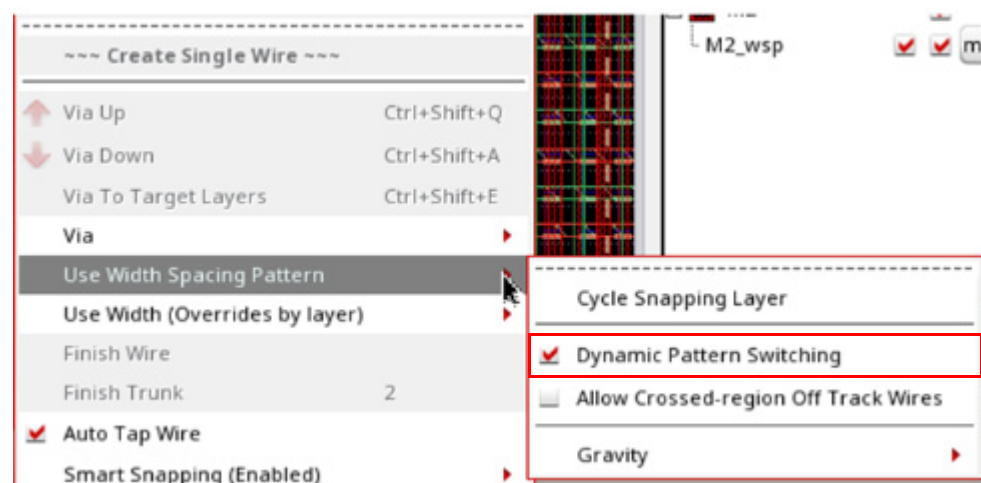
The following figure shows the *Use Width Spacing Pattern* option in the *Create Single Wire* context-sensitive menu.



Switching Patterns Dynamically

Pattern switching happens dynamically letting you cross region boundaries only if the WSP track on which you started creating the wire has allowed WSP tracks.

When *Dynamic Pattern Switching* is enabled, it checks to ensure that the pathSeg you started on a track in one region can cross a region boundary. The target region should have a track that is a member of the allowed WSP patterns in the other regions.



For example:

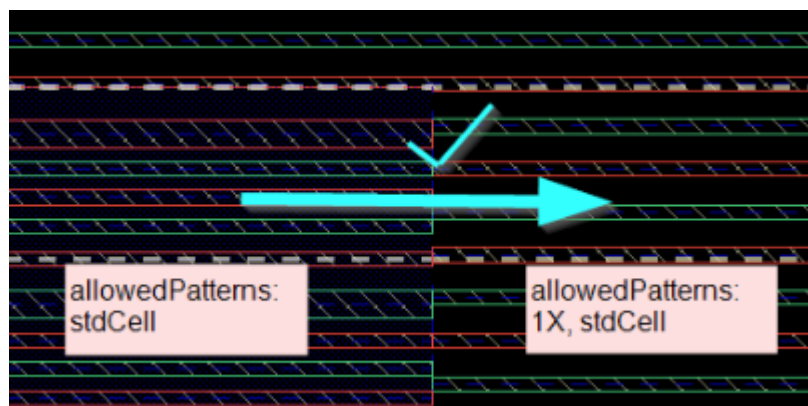
Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

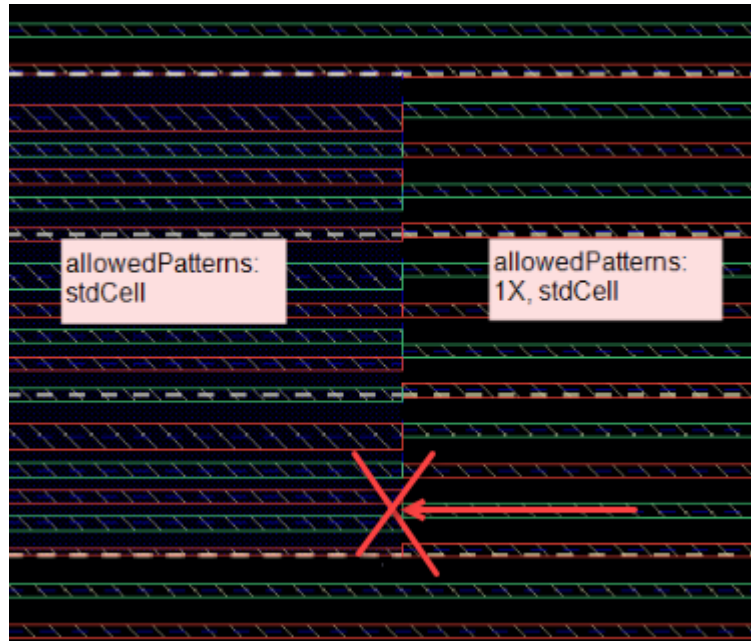
- consider that a region is defined with only a `stdCell` pattern. The `stdCell` pattern is the currently active pattern of the other region.
- consider that the global region has two patterns: `1X` and `stdCell`. In this region, `1X` is the currently active pattern.



Now, start a wire on a wide track of the `stdCell` pattern in the region. Without dynamic pattern switching, `1X` being the currently active pattern, the wire can be drawn only within the region. However, with dynamic pattern switching enabled, the wire can cross the region-to-global boundary. Even though `1X` is the currently active pattern, the wide track with the `stdCell` pattern in the global region lets the wire cross the region. Hence, the pattern is dynamically switched from `1X` to `stdCell` in the global region without changing its currently active pattern.



If you start a wire from a global region with 1x as the currently active pattern, then even with dynamic pattern switching enabled, the wire cannot cross the global-region boundary. This is because 1x is the currently active pattern in the global region, which is not the allowed WSP pattern in the other region. Therefore, the wire is unable to cross the global-region boundary.



Environment Variable: [enableDynamicPatternSwitching](#)

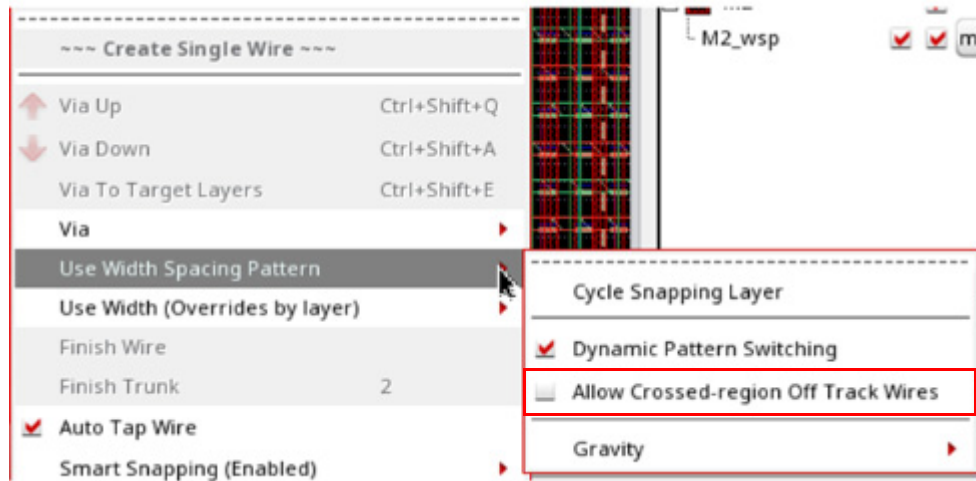
Crossing Region Boundaries

When a wire is created on global and local regions with non matching WSPs, the wire is not allowed to span the region and terminates at the region boundary.

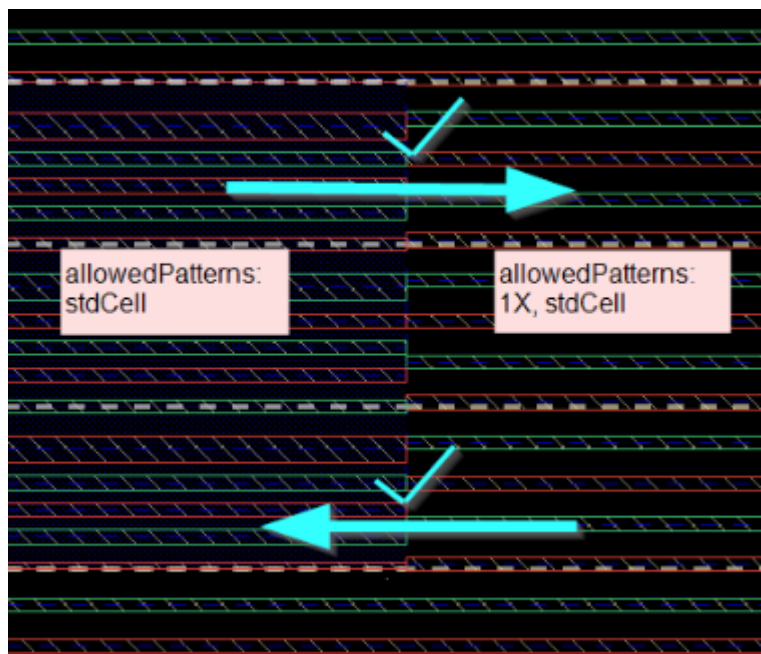
Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

Using the *Allow Crossed-region Off Track Wires* option in the *Use Width Spacing Pattern* menu in the *Create Single Wire* context-sensitive menu, you can ignore the region boundary and continue routing across regions.



When the *Allow Crossed-region Off Track Wires* option is selected, it lets a wire cross regions, regardless of it being on track or off track and the patterns for secondary regions. The end point of the wire can be across a region boundary such that it is off track or on track with a different color and width. When the option is unselected and the WSP tracks are not identical, the pathSeg stops at local region boundaries.



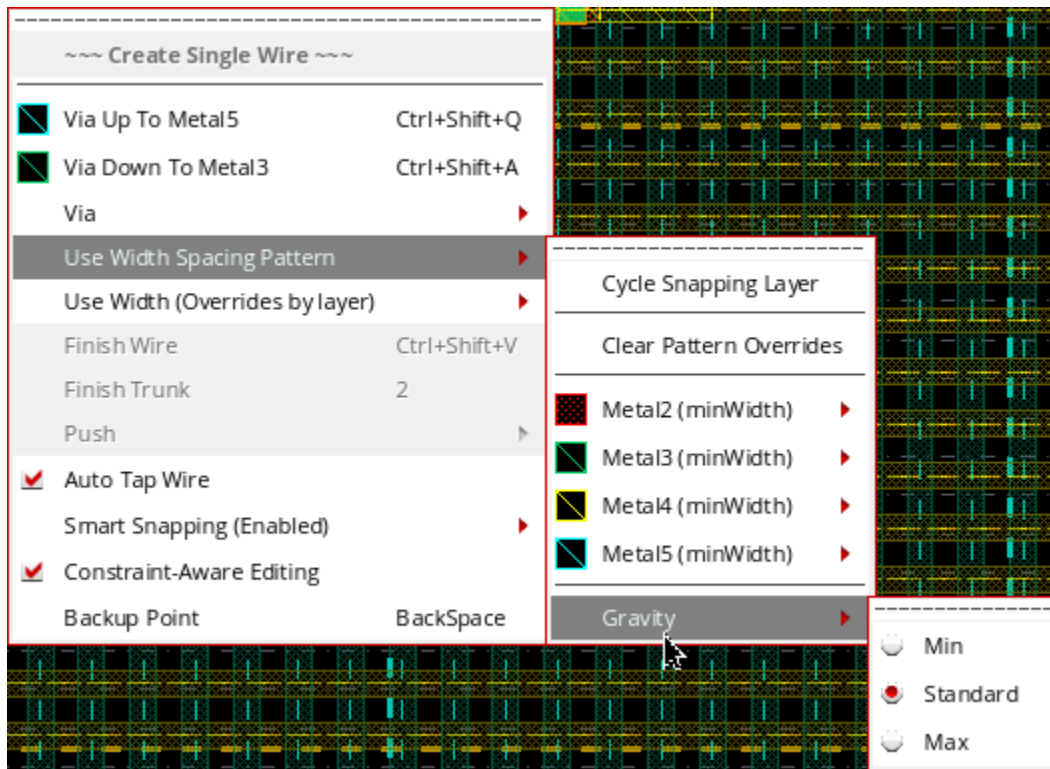
Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

Environment Variable: [adjustEditedViasParams](#)

Wire Editor Gravity

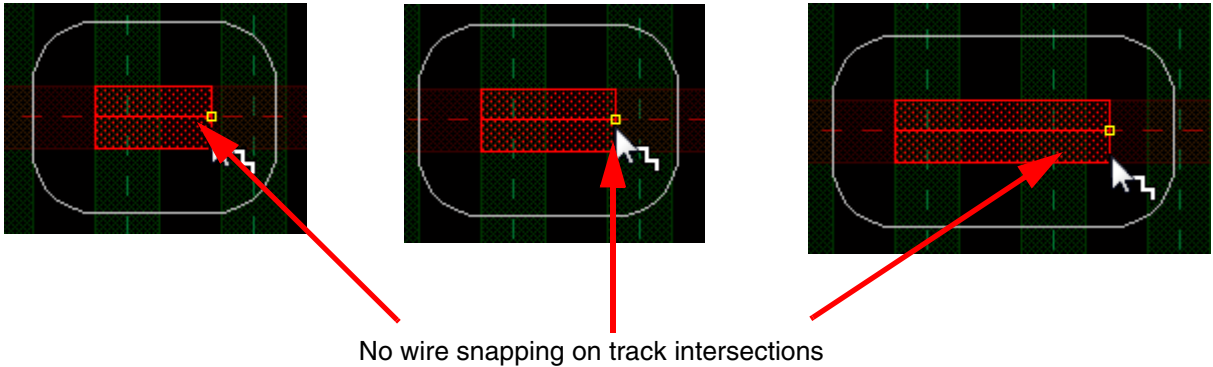
In order to ease the creation of local interconnect wires and to control the magnetic force to track intersections, use the *Gravity* option in the *Use Width Spacing Pattern* menu of the *Create Single Wire* context-sensitive menu. Using the *Gravity* options, you can control how the cursor gravitates towards track and snap pattern intersections.



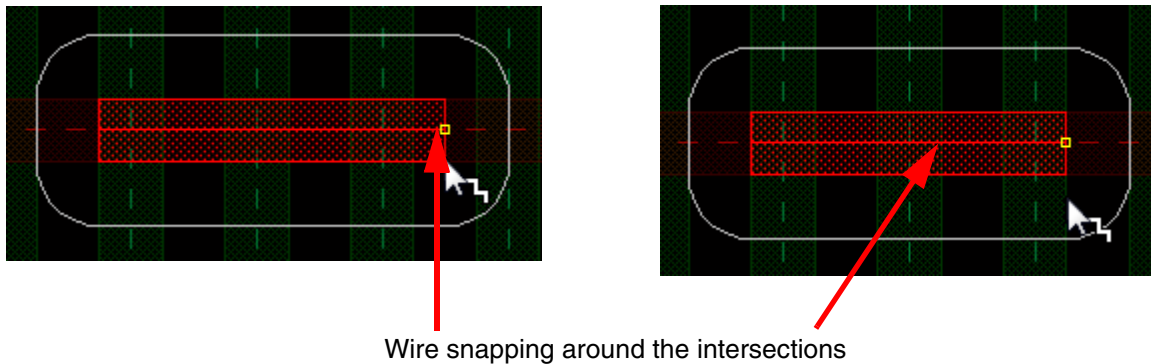
Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

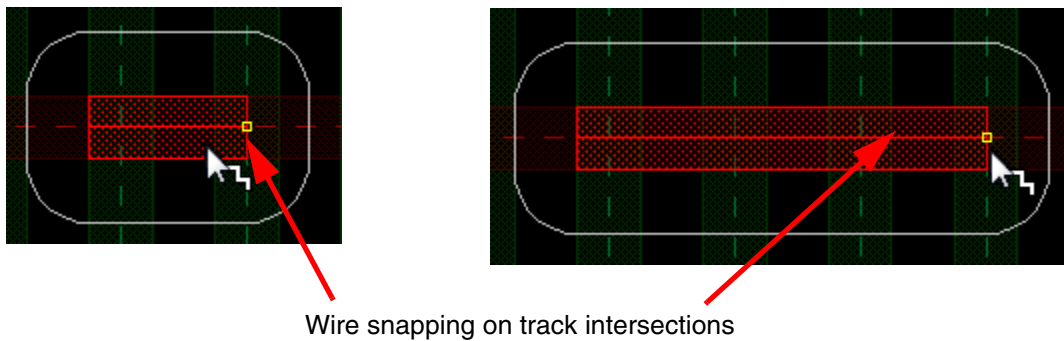
- When set to *Min*, there is no snapping on track and snap pattern intersections and the mouse pointer gravitates pixel by pixel, as shown in the figure below.



- When set to *Standard*, the wire snaps on the track and snap pattern intersections if the mouse pointer projection on the track axis is around the intersection.



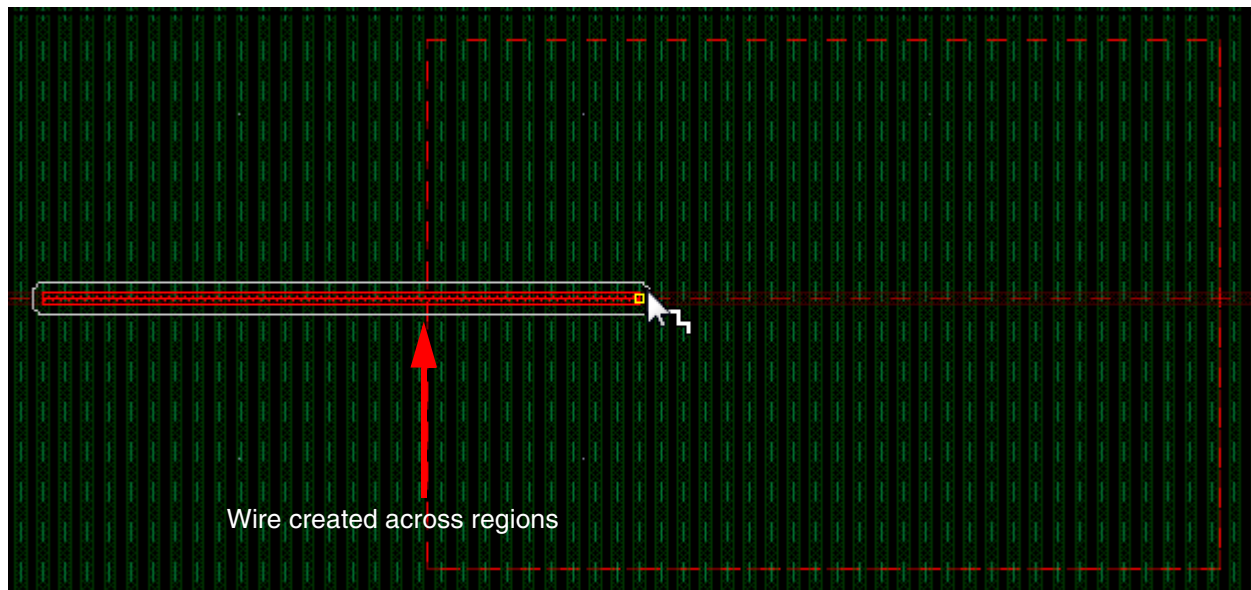
- When set to *Max*, the wire always snaps on the track and snap pattern intersections, as shown in the following figure.



Automatic Switching of the Active Pattern Across Regions

Wire editing provides you the flexibility of automatically switching patterns when crossing regions. You can switch an active pattern of a local region to the active pattern of the global region, even if the active pattern of the local region is not the default but is one of the allowed patterns.

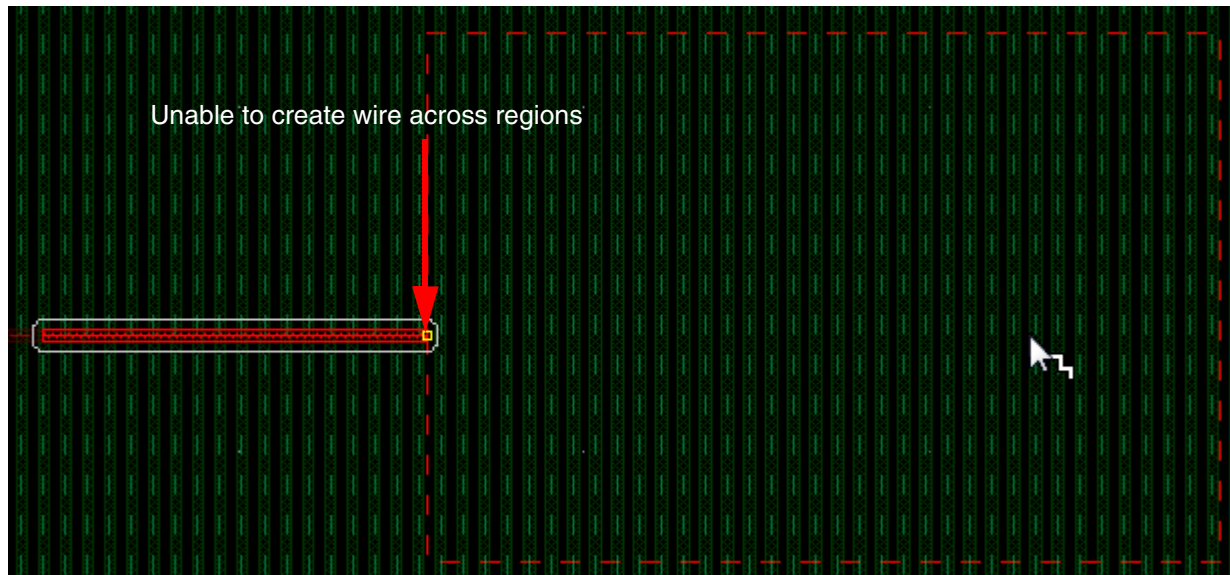
The following figure shows that `minWidth` is the active pattern on `Metal2` layer for the global region and `2Xwidth` is the active pattern of the local region. However, because `2Xwidth` is an allowed pattern in the local region, you are able to create a wire starting from the global region and continuing to extend it in the local region.



Virtuoso Interactive and Assisted Routing User Guide

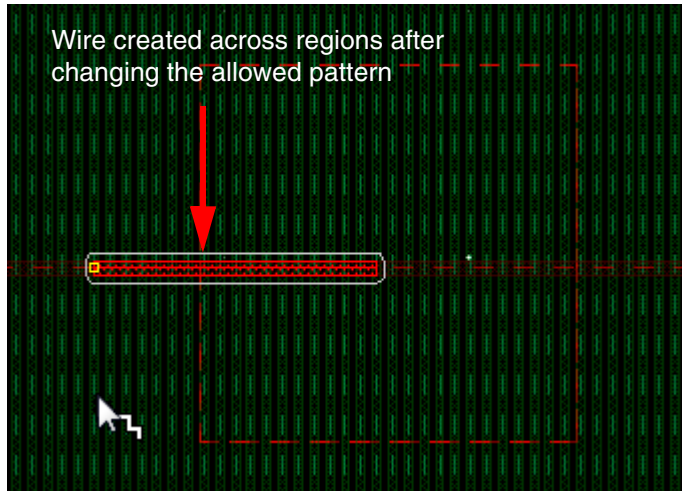
Supporting Width Spacing Patterns

However, if an active pattern of a local region is not an allowed pattern in the global region, then you cannot create a wire across the two regions. The wire is snapped at the boundary of the region. The following figure shows that a wire is created on `Metal2` layer in the global region for which the default active pattern is `2XWidth`. The `2XWidth` is not an allowed pattern in the local region. Therefore, the wire is not able to cross across regions and is snapped at the boundary of the local region.



If the local and global regions have different active and allowed patterns on a metal layer, you can change the active and allowed patterns of a region while creating a wire. For example, a local region has `2XWidth` as the active pattern and the global region has `minWidth` as the only allowed pattern on `Metal2` layer. Once you start creating a wire, you can change the active and allowed patterns of a region using the *Width Spacing Pattern* assistant. When you change the allowed patterns of a region and make the patterns available for the other region, you can traverse through and continue to create a wire in the other region as well.

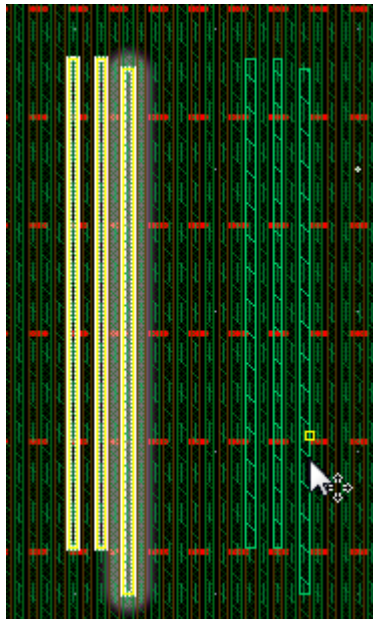
The following figure shows that you can create a wire starting from a local region, which has a `2xWidth` pattern, and continue to create the wire in the global region for which you have changed the allowed patterns.



Working with Selected Sets of Wires

In wire editing, when multiple wires are selected and are moved, copied, or stretched, all the selected wires move together as a group. The wire that is closest to the point clicked on the layout canvas is highlighted and becomes the leader as shown in the following figure. From

the selected set, only the leader is snapped to its corresponding grid. The remaining wires in the selected set move by the same distance but are not snapped to their corresponding grid.



Leader being snapped to grid

Nested WSP Regions

A WSP region can have a local `widthSpacingSnapPatternDef` and not just the global `widthSpacingSnapPatternDef`. The Wire Editor can display and use the patterns of WSP regions embedded inside other WSP regions. Several regions can be nested in one and within one another.

The following are special cases of partially and fully overlapping regions that you can come across while creating wires.

- Region R1 Partially Overlaps Another Region R2
- Two Regions Overlap With the Exact Same Boundary
- Region R1 Partially Overlaps Another Region R2 With the Same Width Spacing Pattern
- Region R1 is Fully Enclosed in Another Region R2

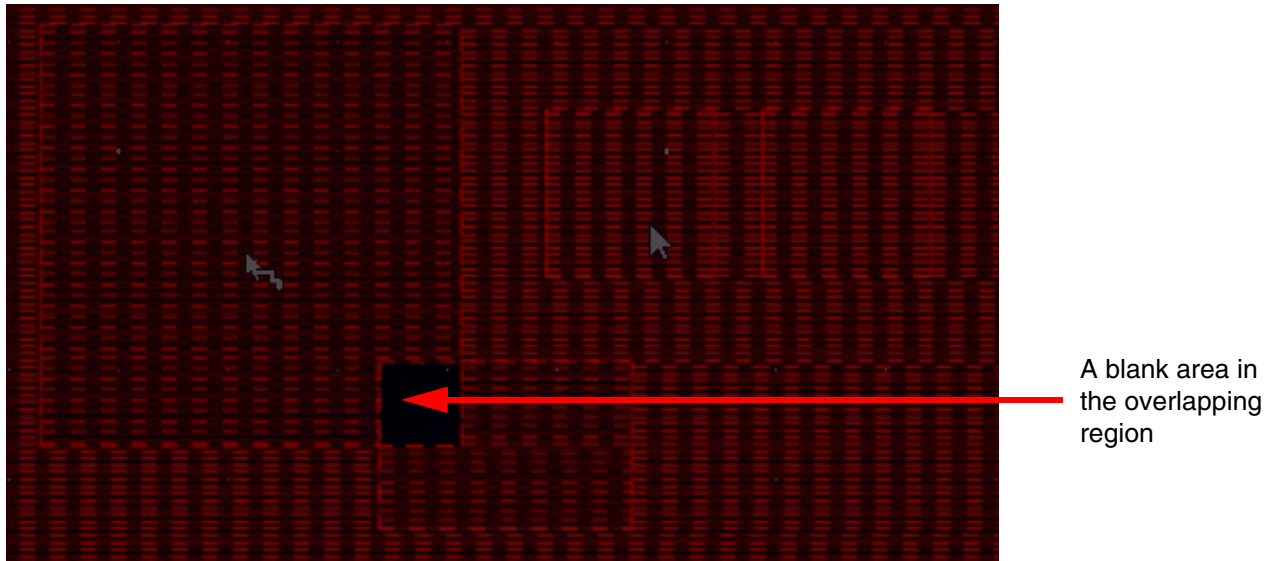
Region R1 Partially Overlaps Another Region R2

The region R1 has `WSP1` as the active pattern and the region R2 has `WSP2` as the active pattern. When the *Create Wire* command is invoked, the region R1 will have `WSP1` pattern

Virtuoso Interactive and Assisted Routing User Guide

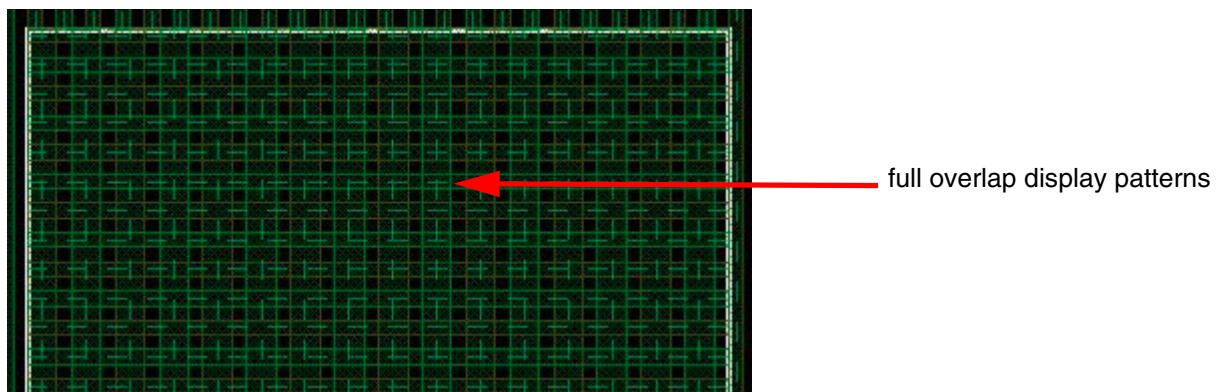
Supporting Width Spacing Patterns

and region R2 will have `WSP2` pattern. However, the overlap area of the two regions does not have any pattern available and displays a blank area. Also, a wire cannot be created and snapped inside the overlapping region.



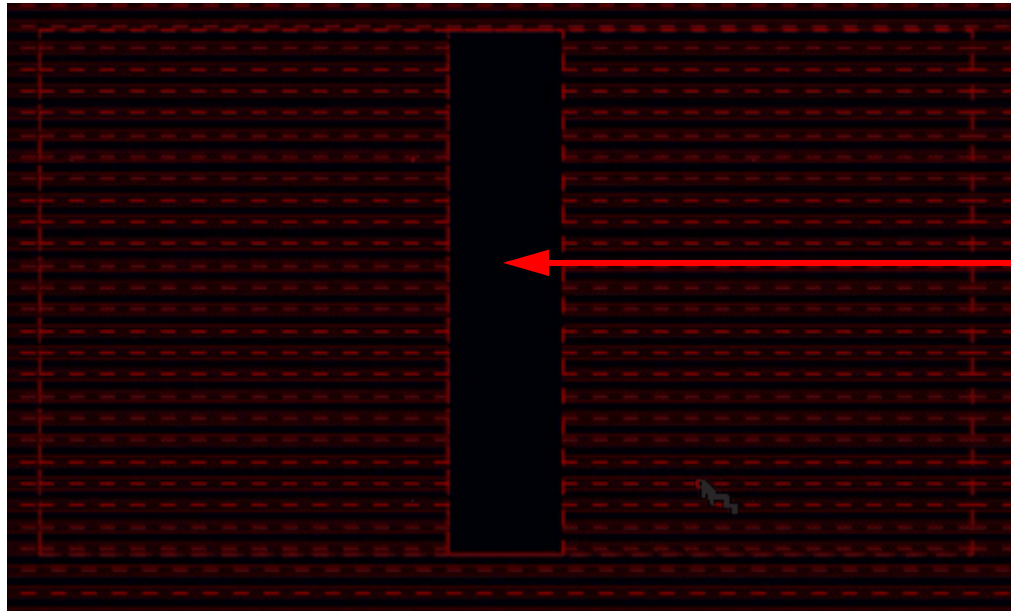
Two Regions Overlap With the Exact Same Boundary

The two regions R1 and R2 have the exact same boundary and completely overlap each other. When the *Create Wire* command is invoked, the pattern of each region is displayed in the overlapped region. You can now create a wire on any of the WSP pattern displayed in the two exactly coincident regions on the same layer. Therefore, snapping is possible to either region's WSP active pattern.



Region R1 Partially Overlaps Another Region R2 With the Same Width Spacing Pattern

When two regions R1 and R2 partially overlap and use the same active pattern, there is no pattern in the overlapping region. The active pattern is valid only where the two regions, R1 and R2, do not overlap each other.

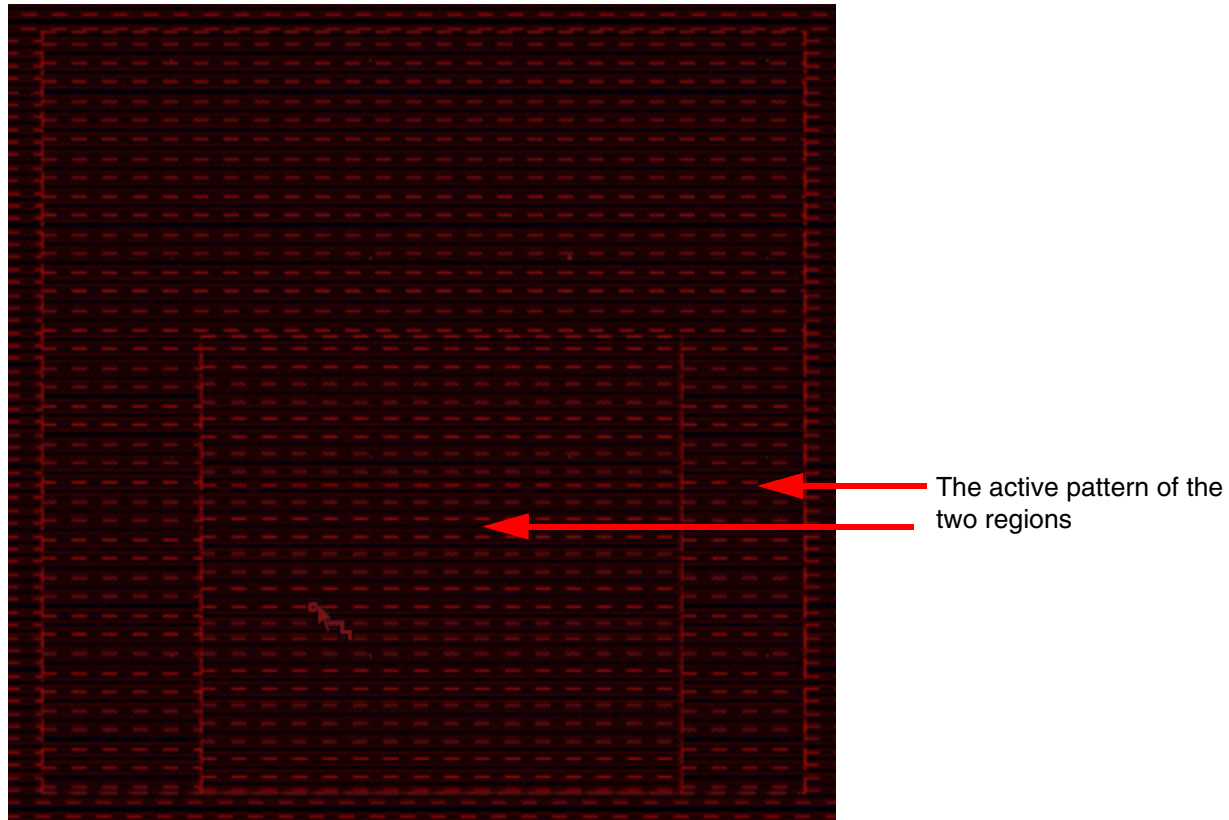


A blank area in the overlapping region when two regions have the same width spacing pattern

Region R1 is Fully Enclosed in Another Region R2

Region R1 has `2xWidth` as the active pattern and region R2 has `WSP1` as the active pattern. When region R2 is fully enclosed within region R1, on invoking the *Create Wire* command, the dynamic display shows the active pattern of each region as shown in the following figure.

In other words, `WSP1` is available inside region R2 and `2Xwidth` is available in region R1 where R2 is not overlapping (R1-R2).



When a wire is created inside the enclosed region (R2), snapping and wire creation is possible only inside the enclosed region. However, if a similar pattern is available in the enclosing region (R1), the wire can cross the region boundary and continue snapping in region R1.

Similarly, if a wire starts from the enclosing region (R1), it stops at the enclosed region (R2) boundary. If a similar pattern is available in the enclosed region (R2), the wire can cross the boundary and continue snapping in the enclosed region (R2).

Note: There is no limitation on the number of regions that can be nested.

Supporting pathSegs with Different WSPs across Regions

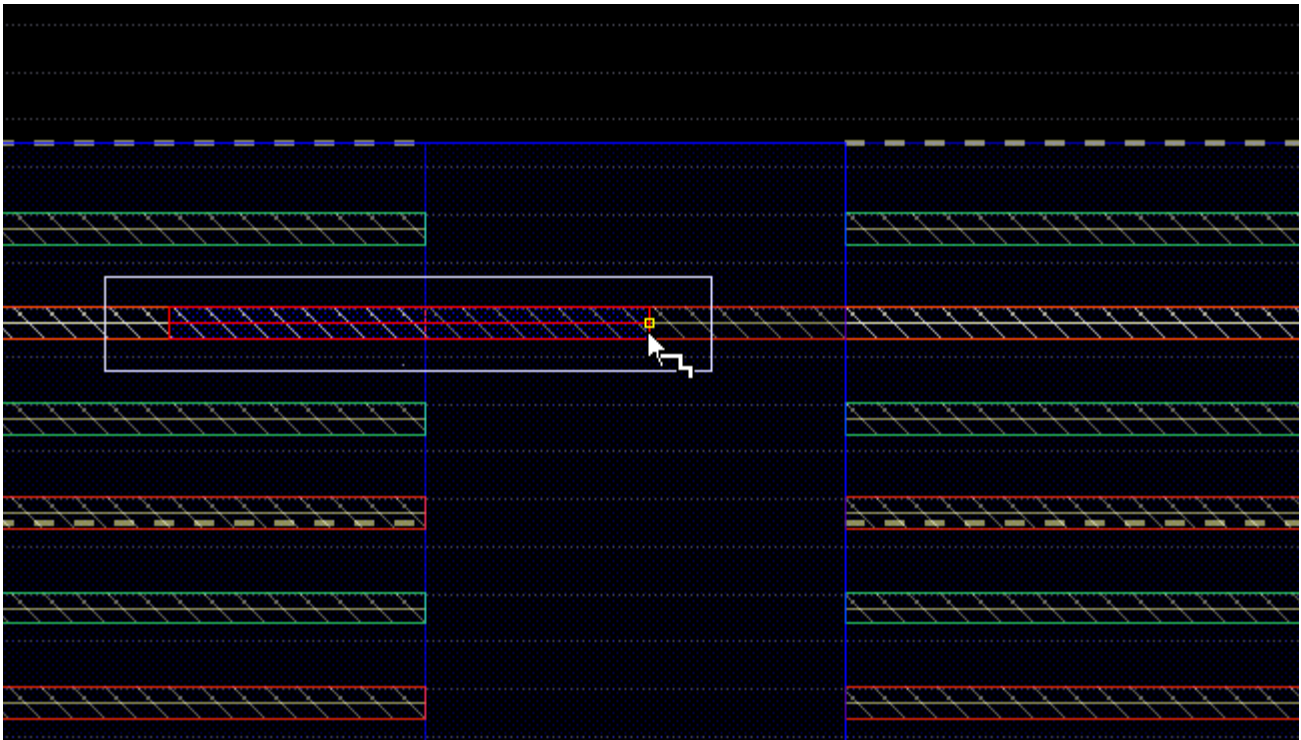
Wire editing provides an environment variable, `adjustEditedViasParams`, which when set to `t` provides the flexibility to cross the region boundaries even when the WSPs (and allowed

Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

WSPs) do not match. This means that the two regions have different WSPs, which is not supported when the switching of active patterns across regions is automatic.

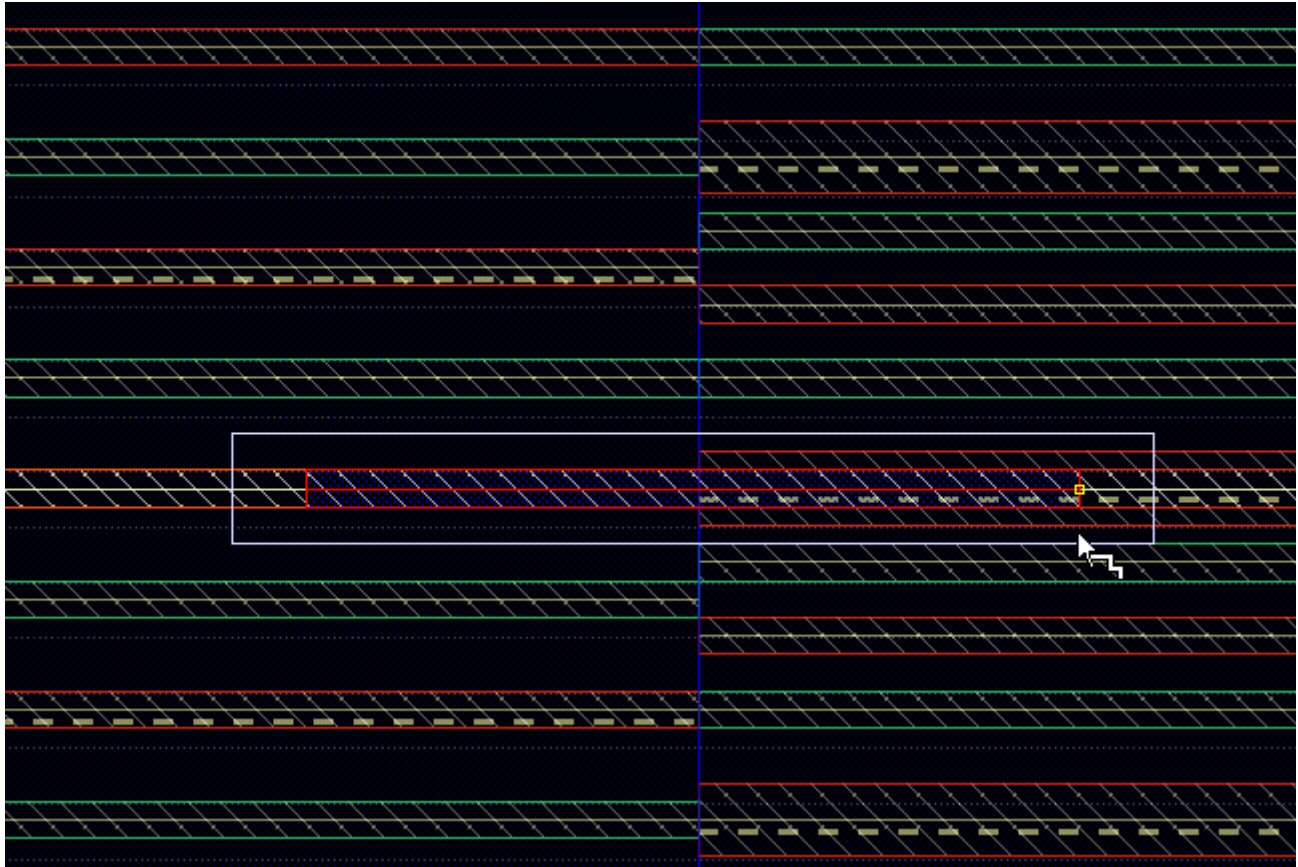
In such cases, the pathSeg acquires all properties from the start point and allows the end point to cross region boundaries. The following figure shows that the pathSeg crosses the region when the two regions having the same WSP pattern are overlapped and there is an empty track space in between.



Virtuoso Interactive and Assisted Routing User Guide

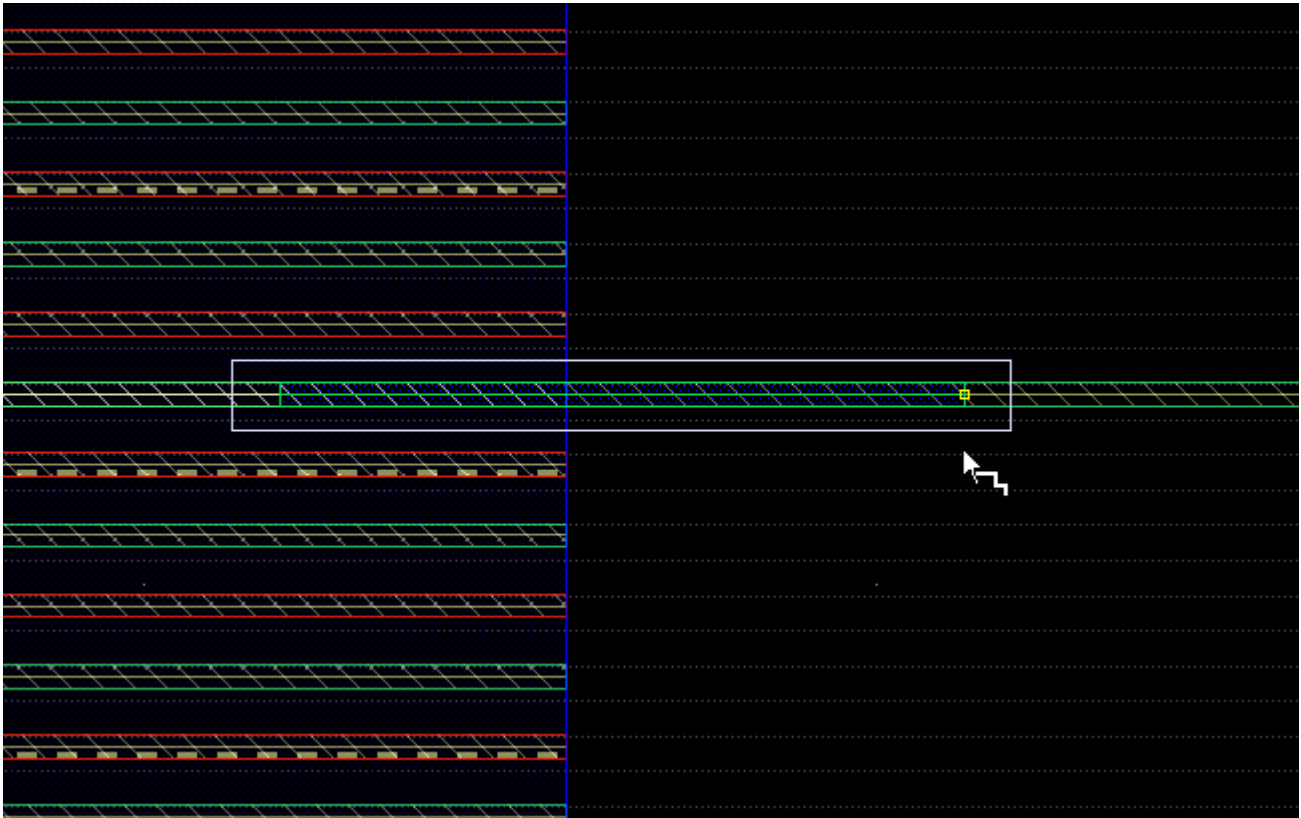
Supporting Width Spacing Patterns

Crossing the boundary region is also possible when there are two abutted regions with different WSPs. The following figure shows that the pathSeg crosses the regions for which the tracks do not match.



Another case in which wire editing lets the pathSeg cross the region boundaries is when one of the two regions is a global region, which implies that there is only one local region. The

following figure shows that the pathSeg crosses from a local region to a global region that does not have any defined WSP.



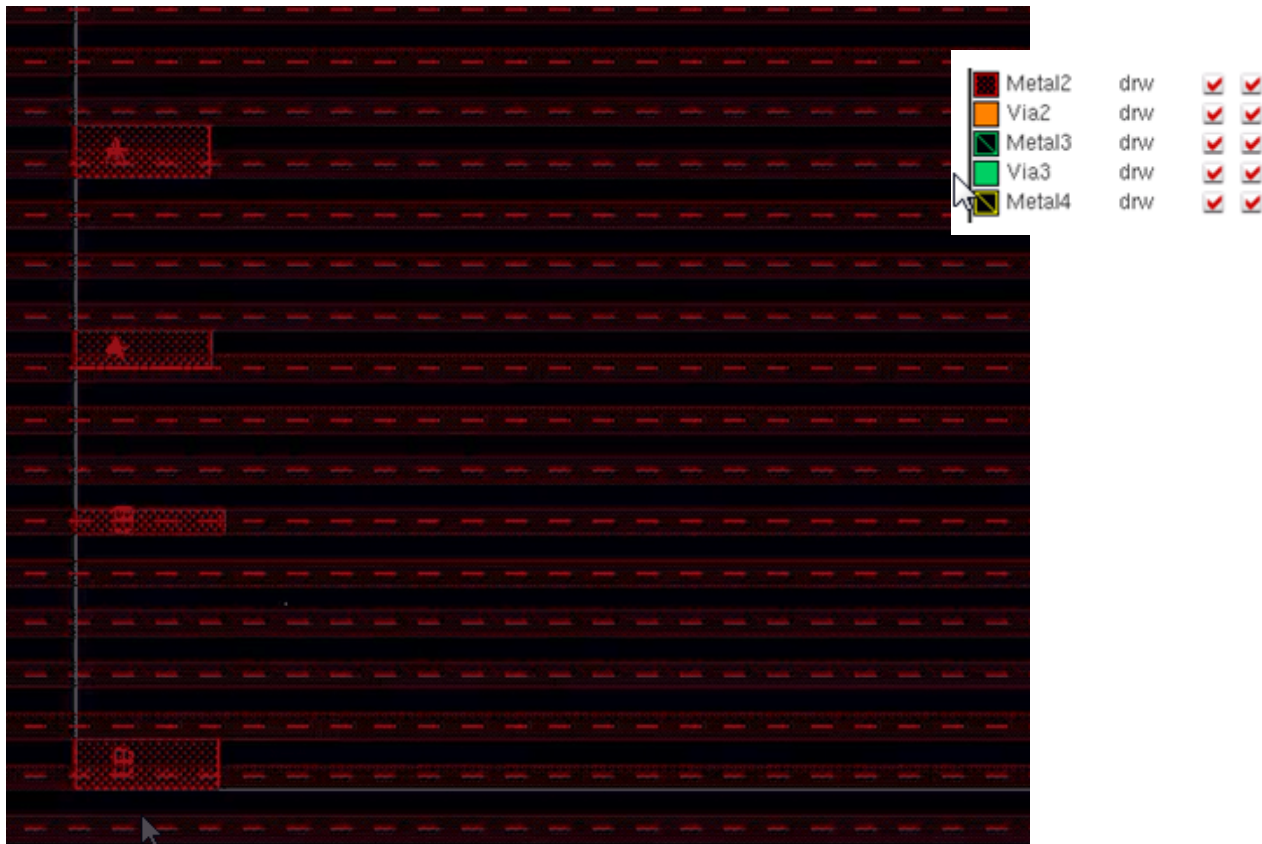
Wire Editor Hierarchy Support

By default, the width spacing patterns of pins are not available as instance pin patterns. Consequently, the global width spacing pattern is available in the instance on a metal layer.

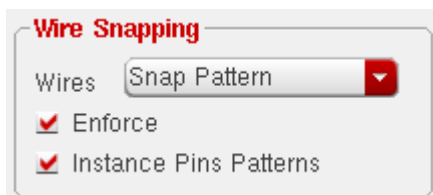
Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

For example, the following figure shows that the global width spacing pattern on Metal2 is being considered even though the instance pins have been created using different patterns.



To identify the pattern of the instance pins, select the *Instance Pins Patterns* check box in the *Wire Snapping* section of the Layout Editor Options form and then run the *Create Wire* command.



When enabled, the width spacing patterns of an instance are considered instead of the width spacing patterns of the enclosing region.

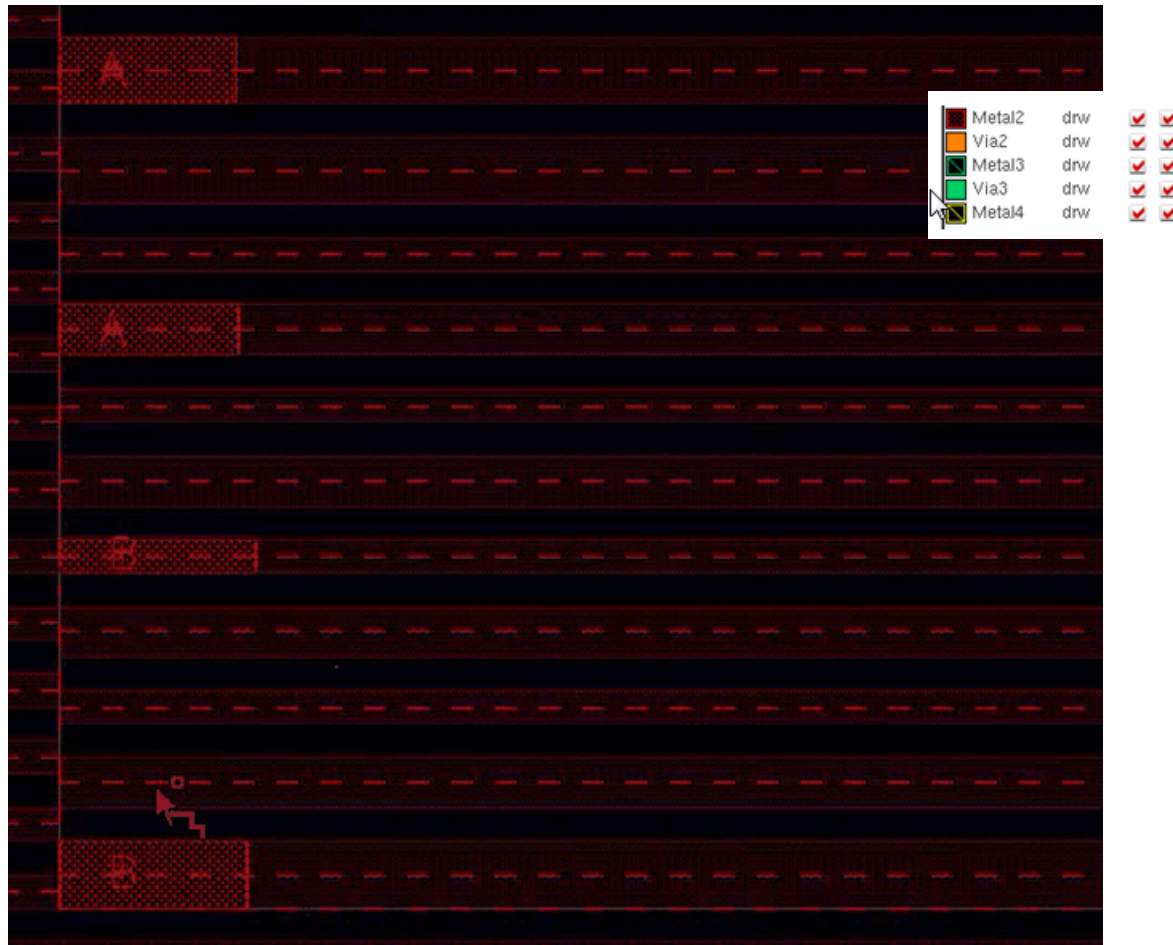
- The width spacing patterns on the layers below the lowest instance pins layer are not accessible.

Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

- On instance pins layers, only the patterns of the instance whose tracks match the instance pins are available within the instance bounding box.
- The width spacing patterns of the enclosing region are only available above the highest instance pins layer.

When the *Instance Pins Patterns* check box is selected, the global width spacing pattern on `Metal2` stops at the instance boundary and the pin width spacing patterns are used to make connections inside the instance. However, in `Metal3`, the global width spacing pattern is still available in the instance. The pin pattern is available only inside the instance boundary and is not exposed outside the instance boundary, as shown in the following figure. Also, the instance pin pattern allows to connect to the pattern of the upper layer, which is coming from the global region.



Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

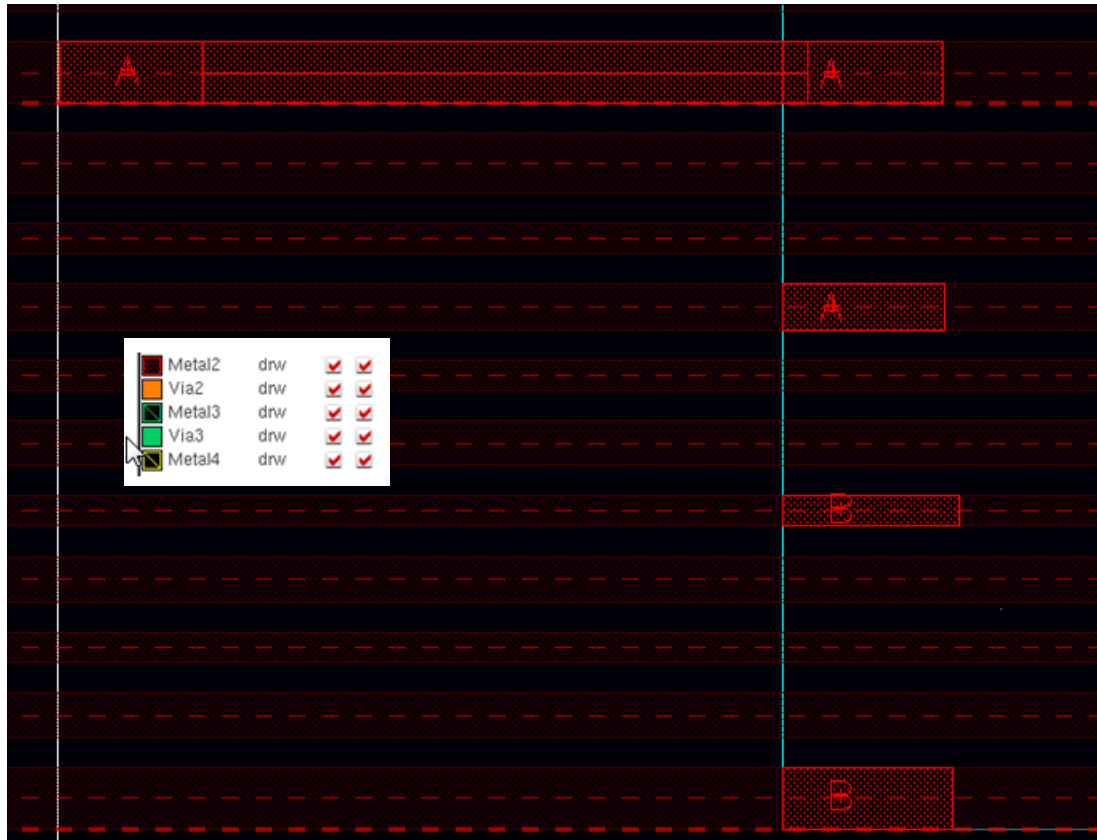
The following are some examples of connections from pins using their own patterns.

In this example, you have a `Metal2` top-level pin and two instances of the same master side by side. The following figure displays the connection that exists between the instance pin and the top-level pin.



Supporting Width Spacing Patterns

If the top-level pin has the same width spacing pattern as that of the instance pin, you can make a direct connection between the instance pin and the top-level pin without changing the metal layer, as shown in the following figure.



Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

You can also make a pin-to-pin connection between the pin on one instance to the pin on the other instance using the pin pattern.

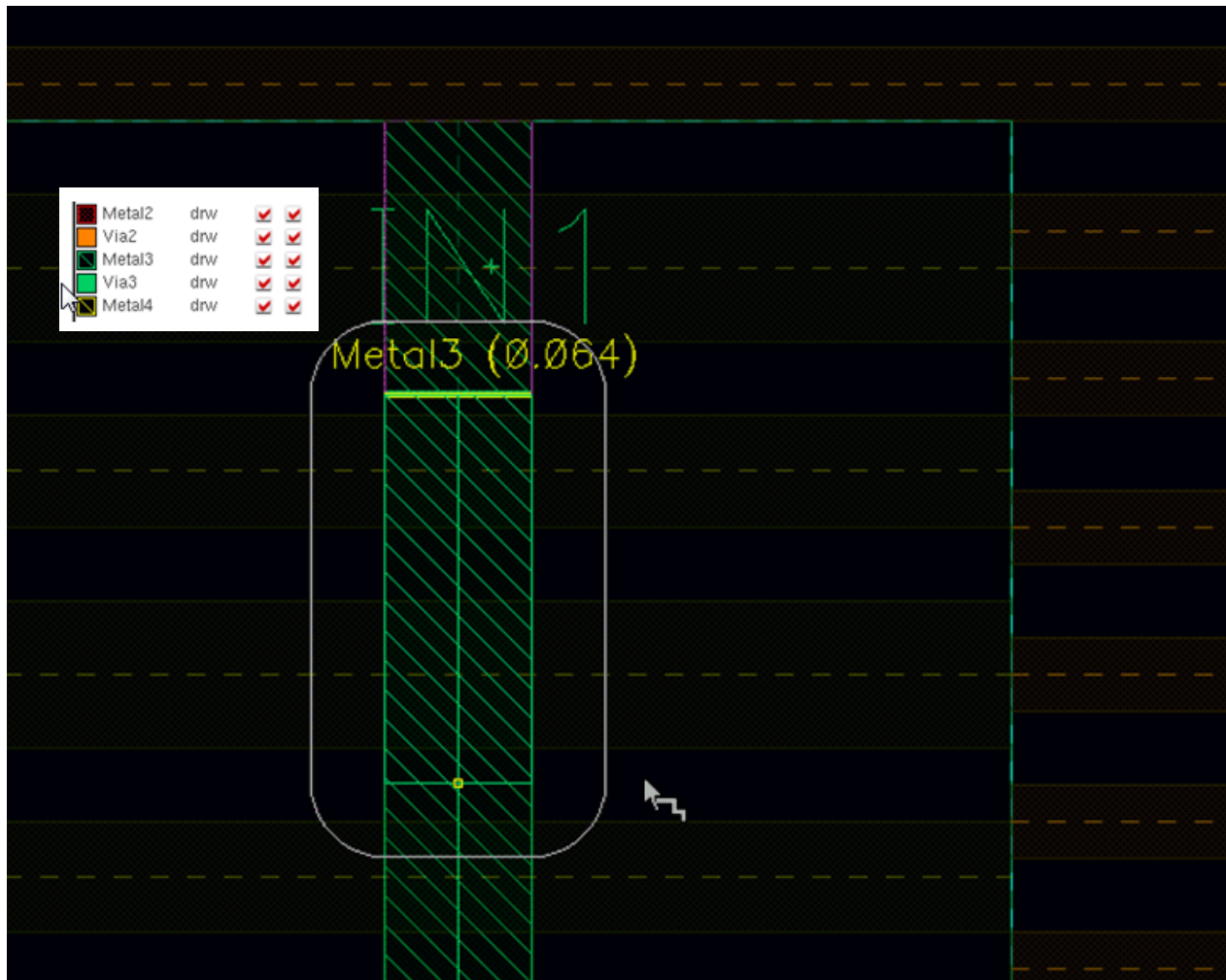


Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

The following are some examples of an instance with pins on several layers.

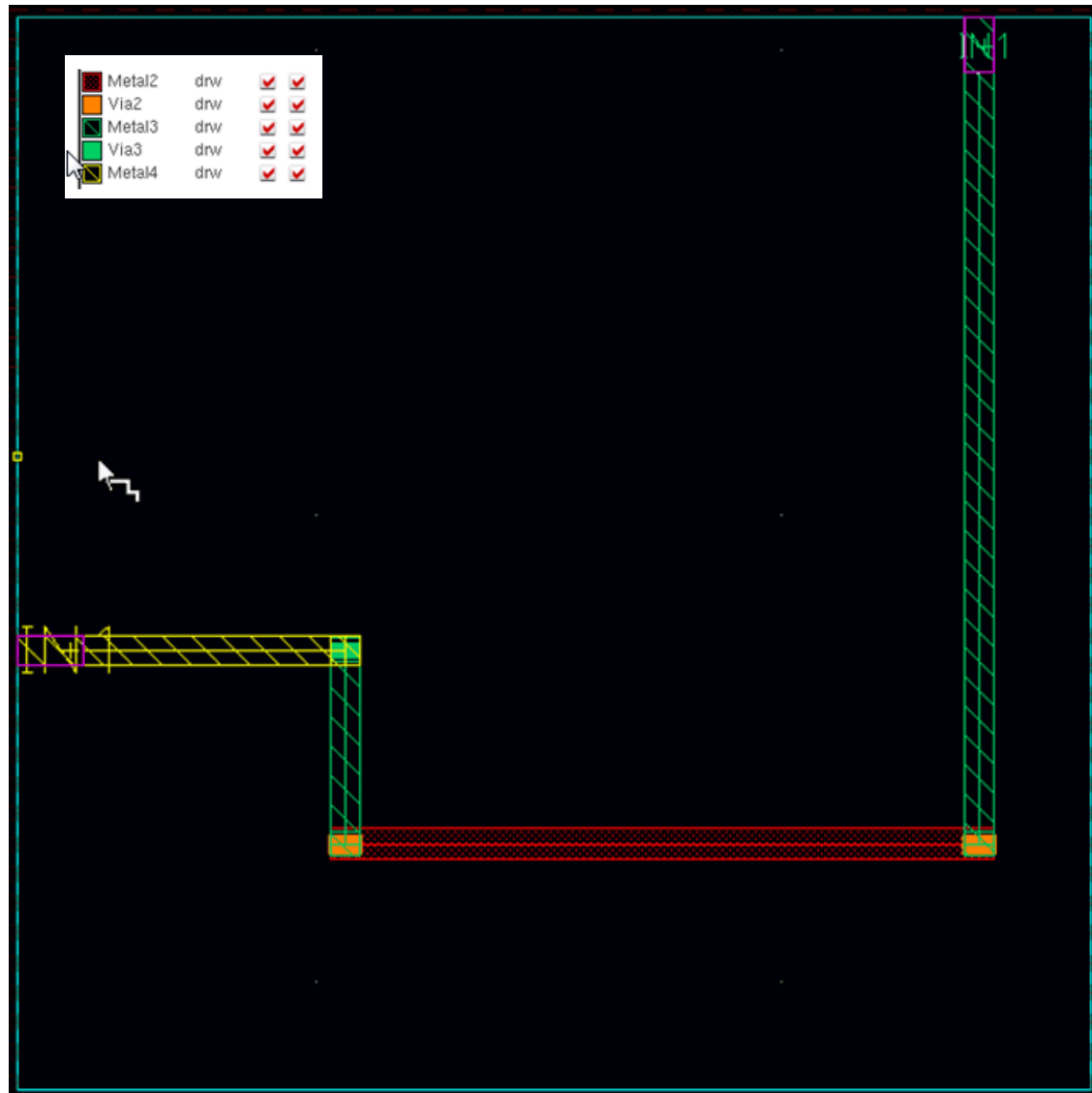
In this example, there is a pin on Metal3 and Metal4 layer. Therefore, for Metal3 and Metal4, the pin instance width spacing pattern can be used. The following figure shows the width spacing pattern of the instance pin on Metal3 layer.



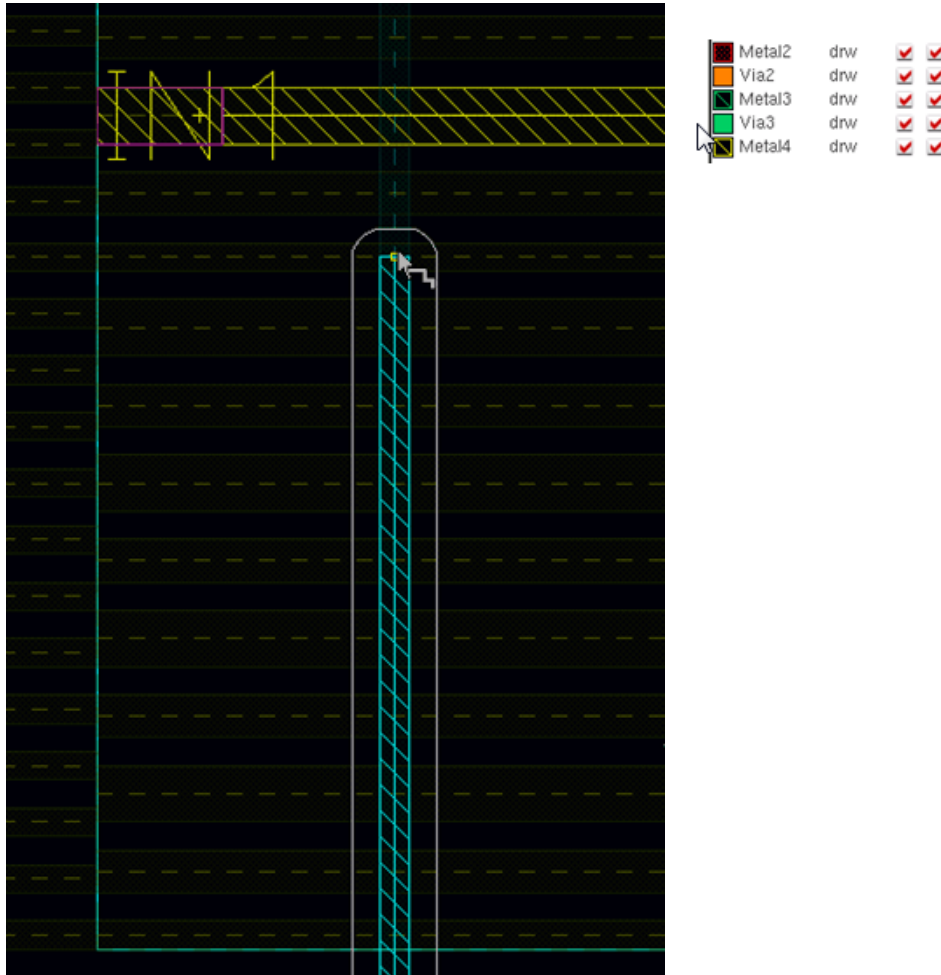
Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

Because there is no pin available on `Metal2` layer and `Metal2` is lower than the highest, or below the highest metal layer of the instance for which width spacing patterns are available, no width spacing pattern is available inside the instance.



Again, there is no pin available on the `Metal15` layer and `Metal15` is higher than the highest, or above the highest metal layer above the instance. Therefore, the global pattern is available inside the instance on `Metal15`, as shown in the following figure.



Pattern Flipping Support

Pattern flipping enables you to flip an existing WSP pattern in alternate periods in a region. WSP pattern flipping can be specified at two levels:

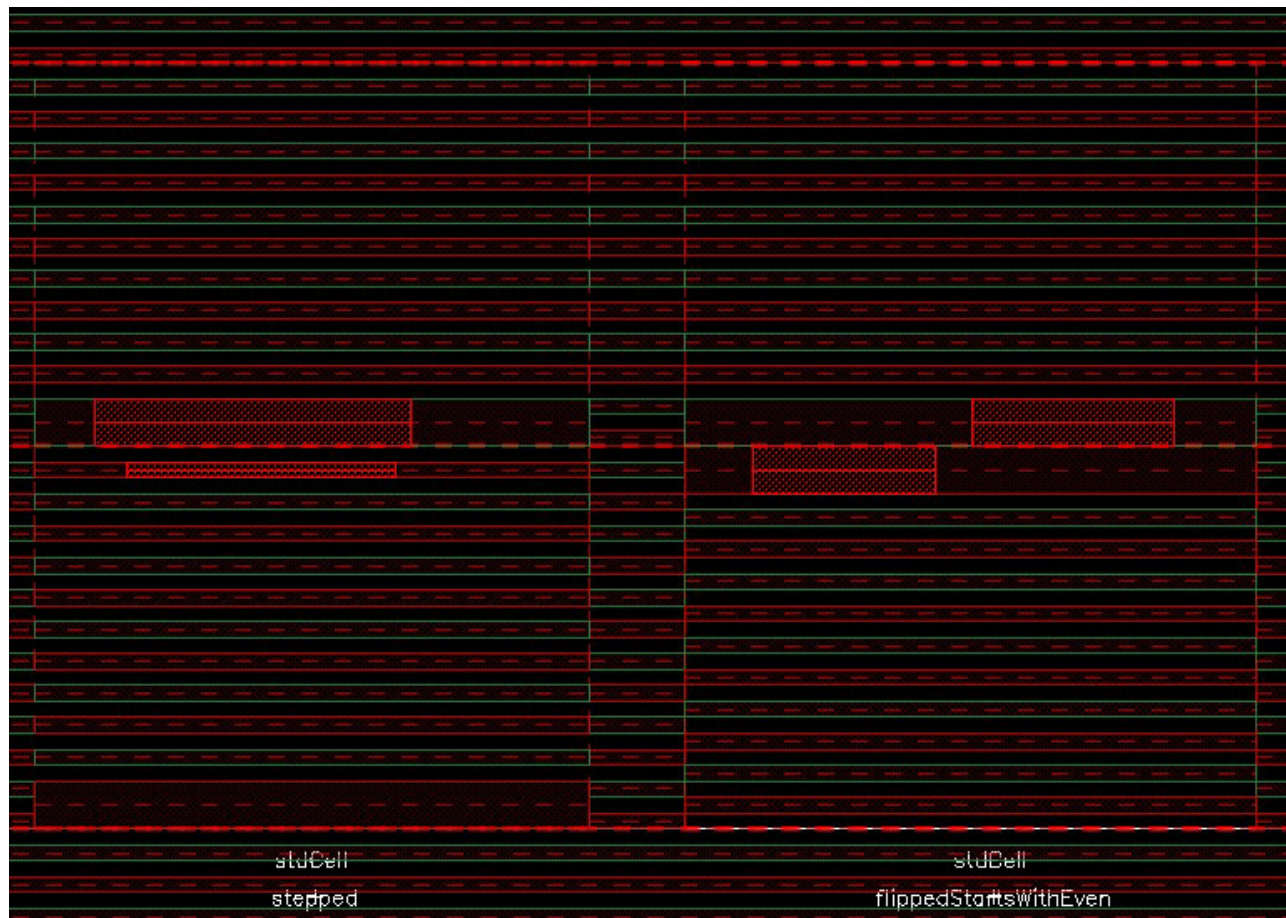
- On a WSP region
- On a WSP global grid

For more information, see [Pattern Flips](#).

Virtuoso Interactive and Assisted Routing User Guide

Supporting Width Spacing Patterns

When a pattern in a region is flipped, a wire created on a track in that region automatically adjusts to the flipped pattern, as shown in the following figure.



Stepped Pattern in the overlapping region

Create Wire supports pattern flipping

If a WSP pattern has a track with offset zero but no track at the top period line, then the track is generated on every period line. In addition, if a WSP pattern has a track with offset zero as well as a track at the top period line, then the pattern should only be used with a flipped repeat mode. In this way only compatible tracks will overlap.

If a WSP with offset zero has tracks with different widths at the top and bottom of the period, the pattern can only be flipped. This ensures that the overlapping tracks are compatible and can be merged.

Supporting Virtuoso RF

Note: The license required for Virtuoso RF is 95560 Virtuoso_RF_Option.

This chapter contains the following sections.

- [Supporting Curved Paths](#)
- [Creating a Packaging Wire](#)
 - [Inserting Via and Changing Layer](#)
- [Stretching a Curved Path](#)
 - [Automatic Snapping Using Stretch](#)
- [Curved Path Snapping Behavior in Packaging Wire](#)
- [Tapping of Curved Paths](#)
- [Creating a Packaging Bus](#)
- [Snapping Behavior of Curved Paths Using Create Packaging Bus](#)
- [Supporting Package Constraints](#)
- [Area-based Rules in Create Package Wire](#)
- [Area-based Rules in Create Packaging Bus](#)
- [Push and Shove in Interactive Routing](#)

Supporting Curved Paths

In Virtuoso RF packages, a shape is called a curved path. The curved path has round edges and lets you route on any angle similar to a path. In addition, it lets you use the context-sensitive menu to change the layers of wires.

Interactive routing has been enhanced to support curved paths that are available in cellviews along with package constraints. The bindkeys that are used for the *Create Wire* and *Create*

Bus commands are defined to create a curved path when you have a design with package. The VRF package definitions are stored either in the technology database or the design database.

However, there is a difference from the usual behavior of the *Create Wire* and *Create Bus* commands.

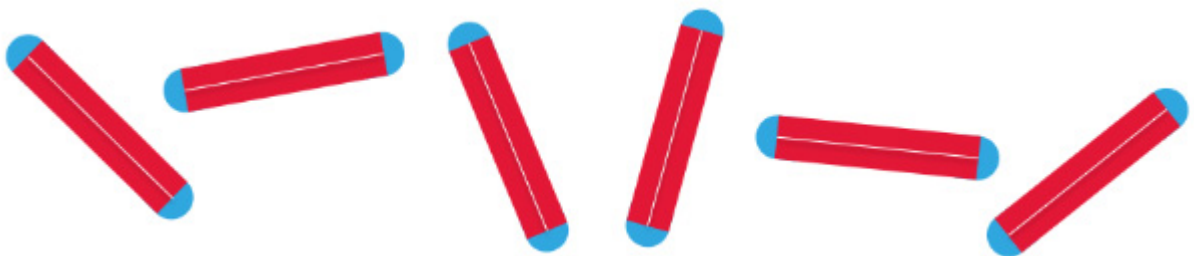
- The bindkeys and the menu options starts the *Create Packaging Wire* and the *Create Packaging Bus* commands instead of the *Create Wire* and *Create Bus* commands.

The *Create Packaging* wire form starts when the fabric type is specified as `board`, `module`, or `package`, in the technology file.

- The *Create Packaging Wire* and *Create Packaging Bus* commands automatically creates curved paths with round edges.



- The curved path supports any angle.



The *Create Packaging* wire form starts the wire option (bindkey `p`) for fabric type as `board/module/package`. The *Create Packaging Bus* form starts the bus option (bindkey `Ctrl+Shift+X`).

While interactive routing, you can create a curved path by using the *Create Packaging Wire* and *Create Packaging Bus* commands. You can start creating a curved path from:

- an empty space

- circular padstack (pin)
- an end of an existing curved path
- rounded vias
- instance term
- TLIP pcell
- bond finger

Creating a Packaging Wire

To create a packaging wire:

1. Choose *Create – Wiring – Packaging Wire* and press F3. Alternatively, use the bindkey P to start the *Create Packaging Wire* command and then press F3. You can also click the *Create Wire* icon  on the *Create* toolbar.

The Create Packaging Wire form opens. You can set the parameters in this form to create a curved path.

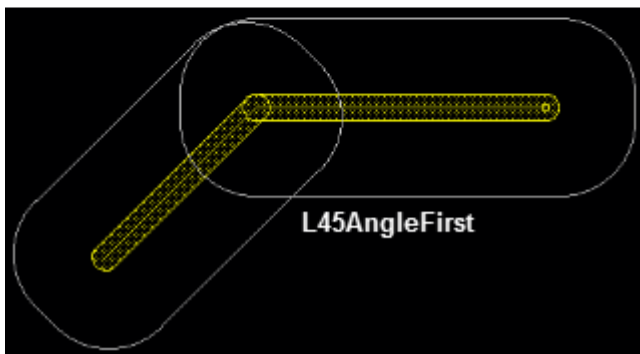
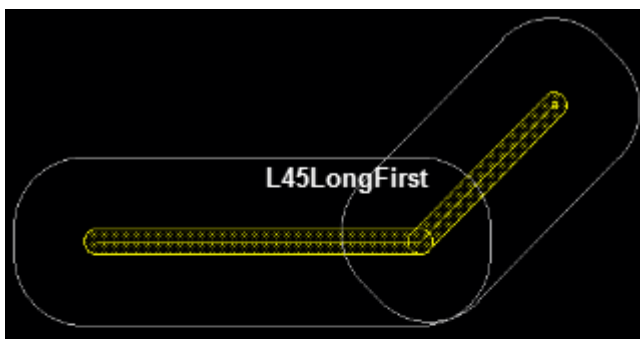


The image shows the 'Create Packaging Wire' dialog box. It has a title bar with the text 'Create Packaging Wire'. Inside, there is a section titled 'Connectivity' in red. Below this, there is a 'Net Name' text field and a checkbox labeled 'Probe Nets'. Below that is a 'Width' text field with the value '25'. Then there is a 'Snap Mode' dropdown menu currently set to 'anyAngle'. Below the dropdown are four checkboxes: 'Auto Terminate' (checked), 'Snap to Pin Center' (checked), 'Snap to Via Center' (checked), and 'Cover Pin' (unchecked). At the bottom of the dialog are four buttons: 'Hide' (in a red box), 'Cancel', 'Defaults', and 'Help'.

2. In the *Net Name* field, specify a net name for the curved path.

The curved path being created is assigned to the specified net. If the specified net does not exist in the design, the net is created.

3. To change the width of the curved path, specify the desired width in the *Width* field. The default width displayed in the *Width* field is determined from the package technology file.
4. Select *Justification* to control the direction along which to offset the curved path from the digitized points of the path.
5. Choose a value from the *Snap Mode* cyclic field. Two additional snap modes are available when editing a cellview with a packaging fabric: *L45LongFirst* and *L45AngleFirst*. If set to *L45LongFirst* or *L45AngleFirst*, two paths are created with a single click, but with a 45 degrees angle between them.



6. Specify if you want the curved path to snap to the center of a pin. This option is selected by default. In case, the *Snap to Pin Center* option is deselected, then the curved path can end-up with an invalid connection.
7. Select *Auto Terminate* while creating a curved path for terminating the path with a single click. This option is selected by default.
8. Specify if you want the curved path to snap to the center of a via. This option is selected by default.
9. Select *Cover Pin* to automatically cover the source and target pins and adjust the curved path.
10. To finish creating the curved path, double-click or press `Enter`.

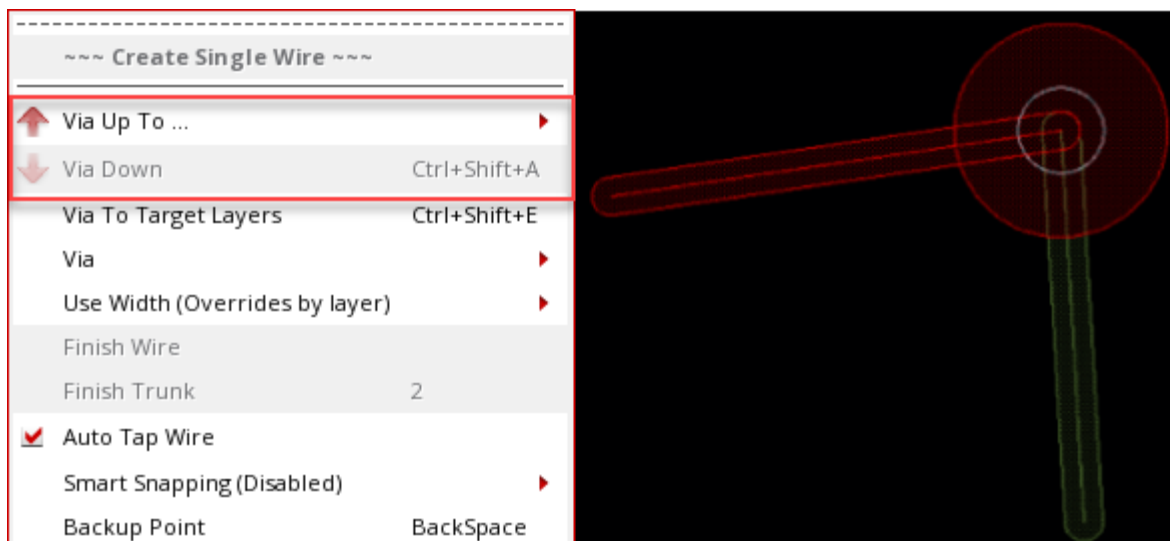
Inserting Via and Changing Layer

While creating a curved path, you can change the layer by dropping a via or a padstack. To change the layer, perform the following steps:

I) Use the *Via Down to <layer name>* and *Via Up to <layer name>* commands on the *Create Single Wire* context-sensitive menu when the *Create Packaging Wire* command is active:

1. Start the *Create Wire* command.
2. Start creating the curved path by digitizing the points.
3. Right-click to select *Via Up to <layer name>* or *Via Down to <layer name>* to change the wire layer to a higher or lower layer, respectively.
4. Continue digitizing points on the new layer.
5. Complete the command by pressing `Esc` or `Enter`.

The following figure shows a via inserted between L01 and L02.



II) By using the *Select Via* command on the *Create Single Wire* context-sensitive menu

The *Select Via* command lets you add padstacks. Padstacks have more than two accessible layers. The padstack support lets you leave a padstack from an intermediate layer.

1. Start the *Create Wire* command.
2. Start creating the curved path by digitizing the points.

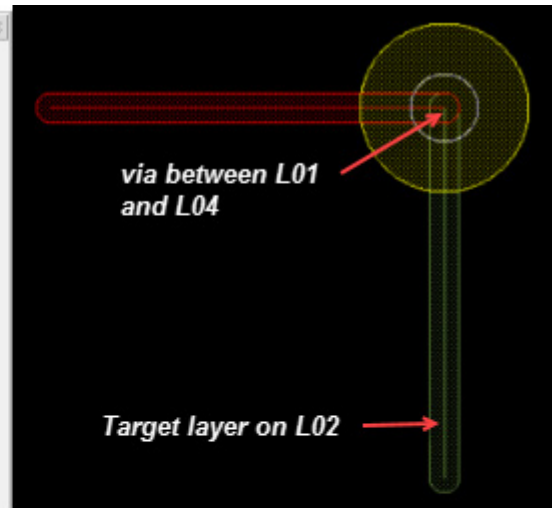
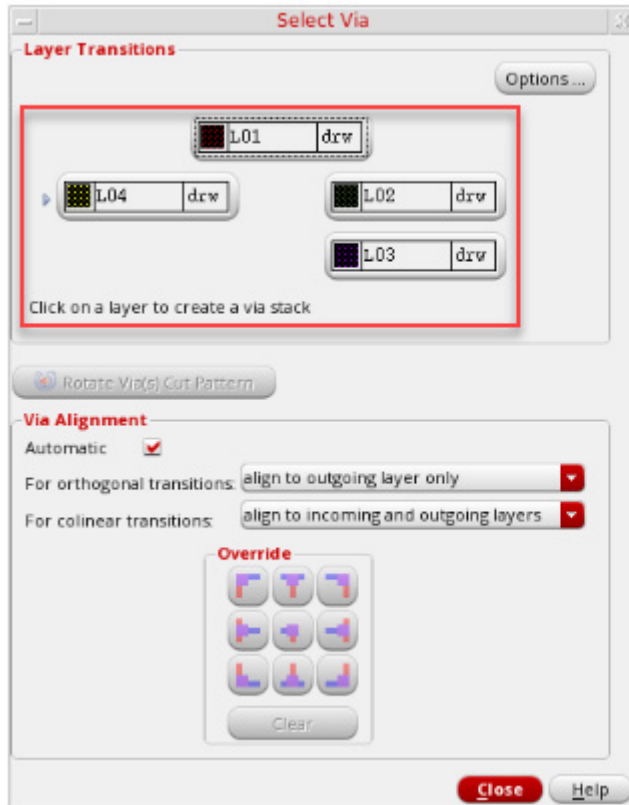
Virtuoso Interactive and Assisted Routing User Guide

Supporting Virtuoso RF

3. Right-click to select *Select Via* or press the Spacebar key.

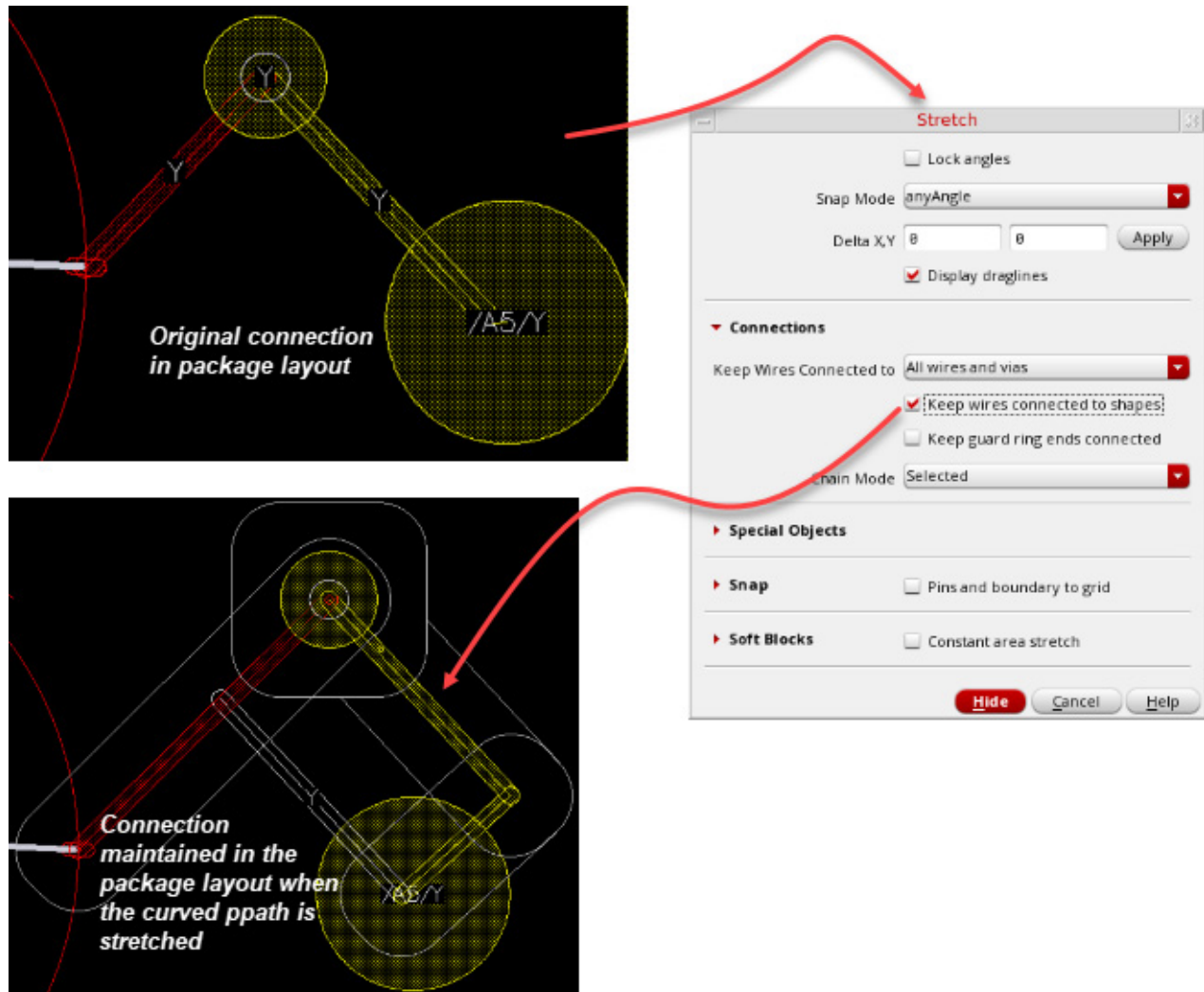
The Select Via form appears.

4. Select a layer to which you want to create a padstack. The following figure displays a padstack between L01 to L04.



Stretching a Curved Path

When stretching a curved path, the connection of the curved path is maintained when the, *Keep wires connected to shapes*, option is selected in the Stretch form.



Automatic Snapping Using Stretch

While stretching a packaging wire, you can automatically snap a wire to the center of the destination pin, via, instance pin, or bondfinger. In addition, you can automatically snap a wire to the centerline or the start and end of the path. The automatic snapping to the center of object or the path helps create a DRC connection correctly without any invalid connections.

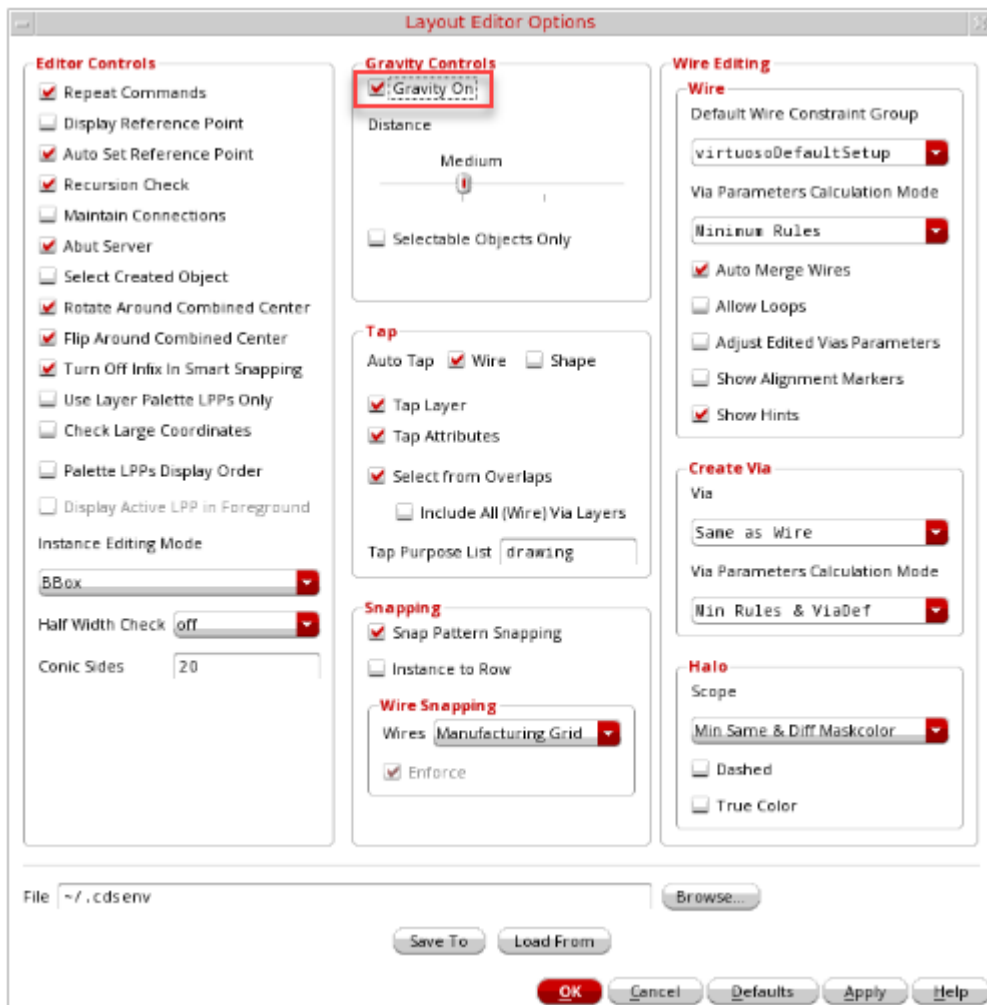
Virtuoso Interactive and Assisted Routing User Guide

Supporting Virtuoso RF

Automatic snapping ignores the connectivity as well as the metal layer of the destination object, except in case of conflict. In case of conflict, it uses the information to resolve a conflict, if any.

To automatically snap a packaging wire:

1. Choose *Options – Editor*. The Layout Editor Options form appears.
2. Select the *Gravity On* option to ensure that snapping to objects is enabled.



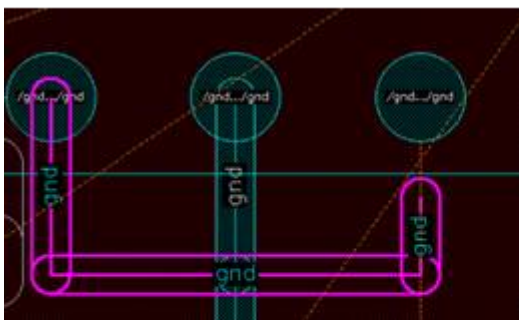
3. Choose *Edit – Stretch* and press F3. The Stretch form displays.
4. (Optional) Click *Lock Angle* to select or deselect it. The use of lock angle allows the stretch to keep the bends as they are. When the *Lock Angle* option is selected, snapping slides the bend to allow the wire to snap.

Virtuoso Interactive and Assisted Routing User Guide

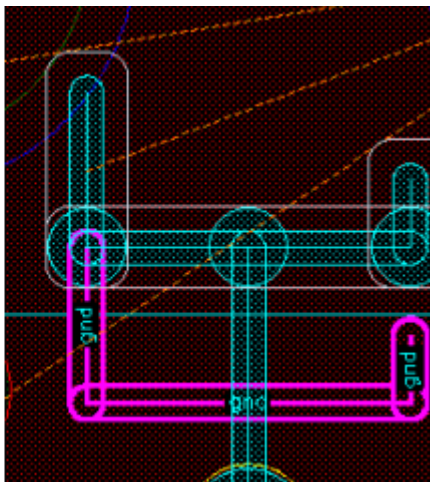
Supporting Virtuoso RF

5. Choose a value from the *Snap Mode* field. The supported snap modes are *anyAngle*, *diagonal*, and *orthogonal*.
6. Select the packaging wire that you want to stretch. You can either partially or fully select the wire.

When you fully select the wire, the *Stretch* command works similar to the *Move* command and the whole path is moved, as shown in the following figure.

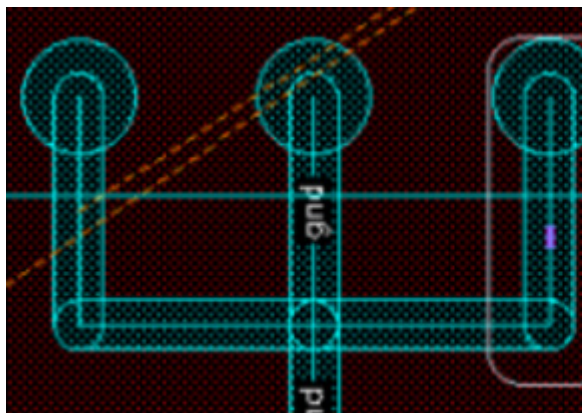


Selection of path

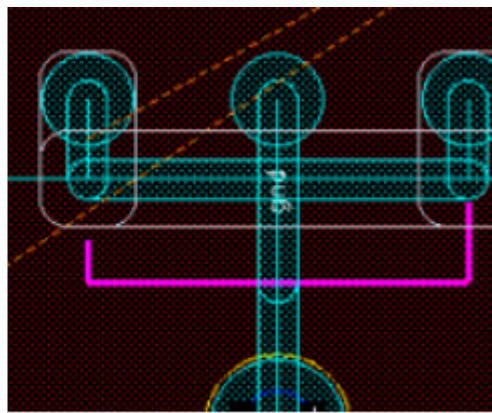


Movement of the path

When the wire is partially selected, the snapping does not add paths to the stretched wire. The path adjusts the bend location to preserve the direction and angles to avoid DRC violations.

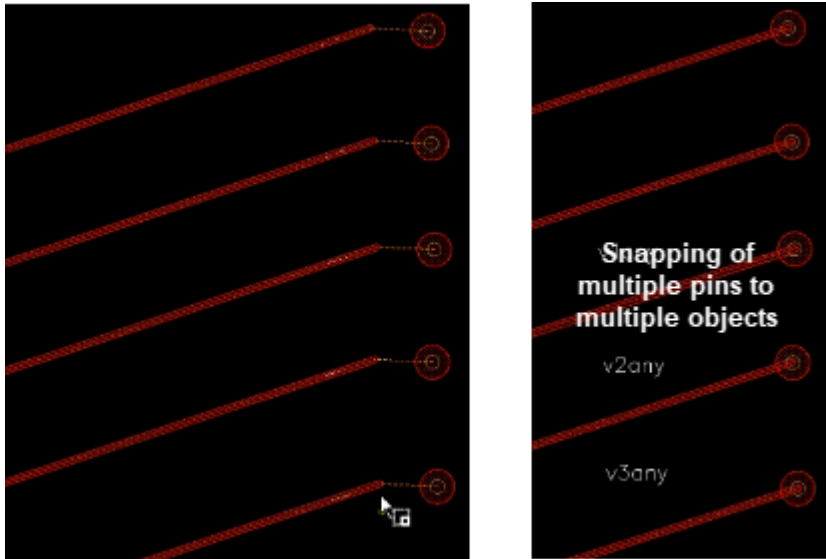


End of path



Multiple parts of the path

You can also snap multiple wires to destination objects at the same time by selecting multiple wires and overlapping them over the destination objects, as shown in the following figure.



Curved Path Snapping Behavior in Packaging Wire

When you start creating a curved path from an existing object, such as a ball grid array pin, TLIP, via, and bondfinger, the curved path snaps to the center of the object. The following examples illustrate how a curved path snaps to the center of an object.

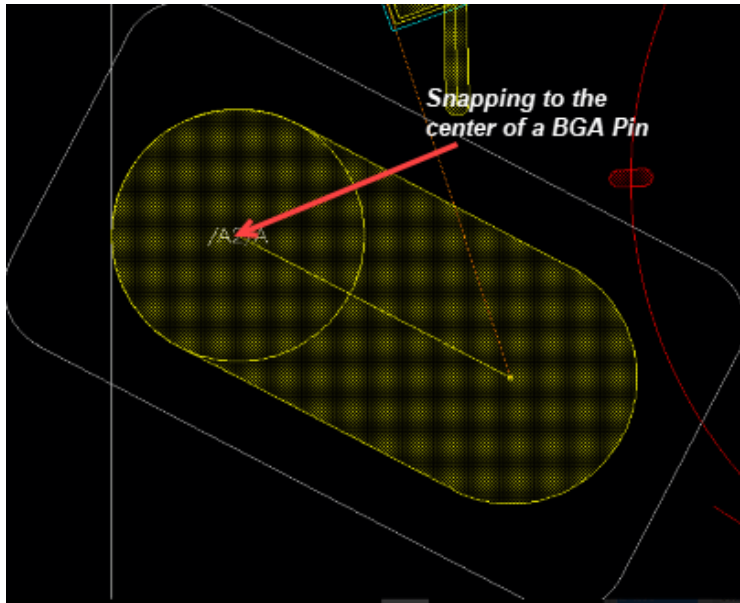
Note: The *Snap to Pin Center* and *Snap to Via Center* options in the Create Packaging Wire form are enabled, by default.

■ Snapping to a Ball Grid Array Pin

Virtuoso Interactive and Assisted Routing User Guide

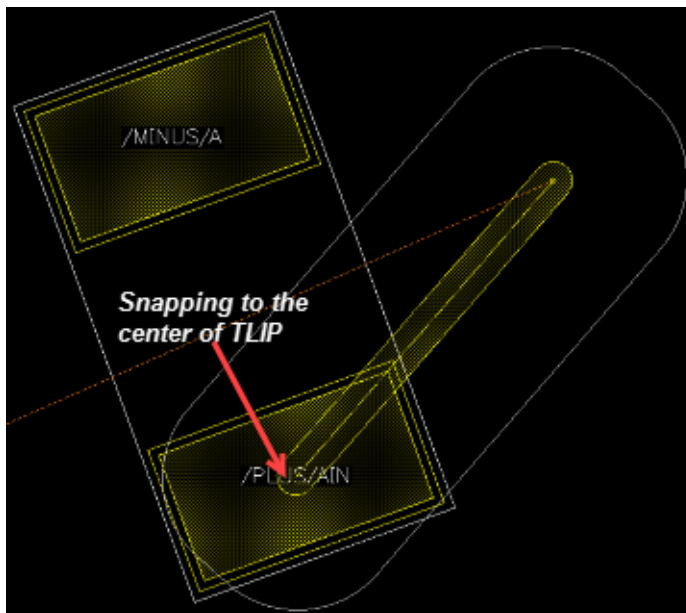
Supporting Virtuoso RF

When the *Snap to Pin Center* is selected, then curved path snaps to the center of source and destination pin. However, if the option is deselected, then the curved path starts from the first point of click on pin.



■ Snapping to Pad

Support of instances used as pads has been added in interactive routing, notably in smart snapping and in the flightlines engine.

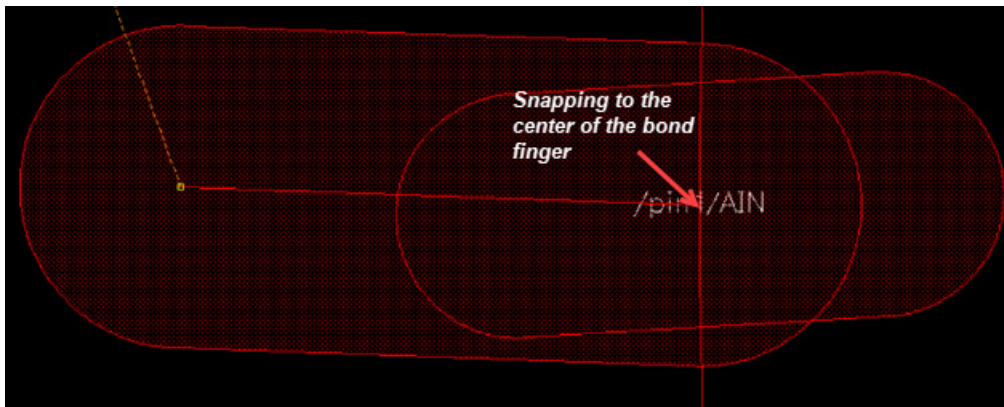


Virtuoso Interactive and Assisted Routing User Guide

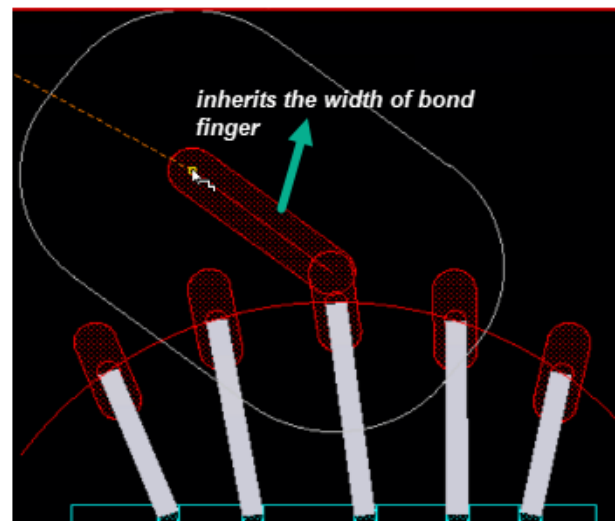
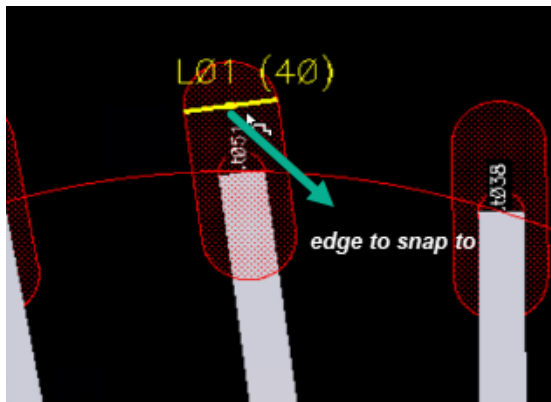
Supporting Virtuoso RF

■ Snapping to a Bond Finger

When *Snap to Pin Center* is selected, the curved path snaps to the center of the bond finger. However, if the option is deselected, then the curved path starts from the first point of click on the bond finger.



The bond finger also supports smart snapping for the curved path. In this case, the curved path is snapped to the edge of the bond finger. In addition, it also inherits the width of the bond finger.

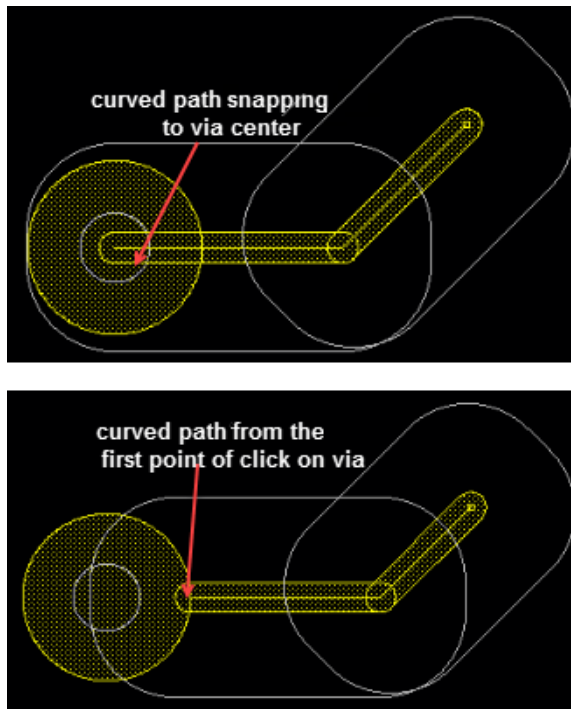


Important

Smart snapping for the bond finger works only when the *Snap to Pin Center* option in the Create Packaging Wire form is deselected.

■ Snapping from a Padstack Via

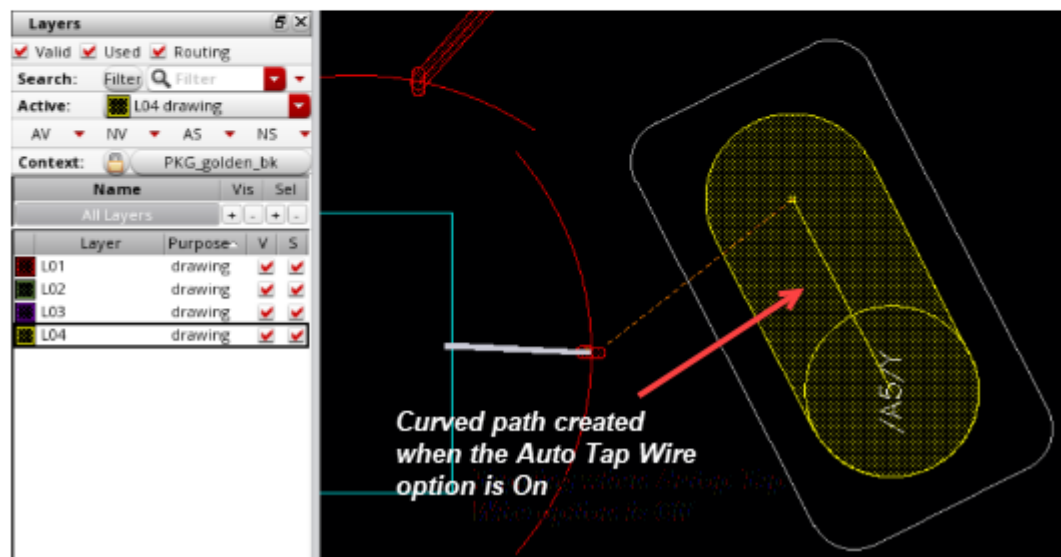
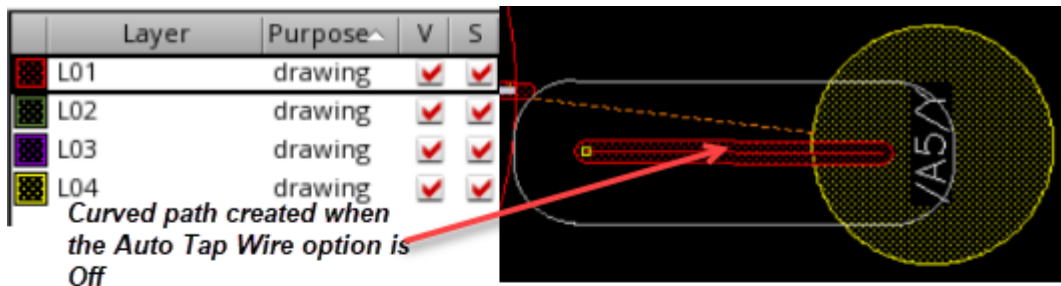
When the *Snap to Via Center* option is selected, the curved path snaps to the center of a via. However, if the option is deselected, then the curved path starts from the first point of click on the via.



Tapping of Curved Paths

The tapping behavior of a curved path is dependent on the *Auto Tap Wire* option in the *Create Single Wire* context-sensitive menu. If the *Auto Tap Wire* option is selected, the curved path automatically taps to the metal layer of the object on which the pointer is placed. Also, it inherits the width of the starting object.

The following figure shows the tapping behavior of a curved path.



Creating a Packaging Bus

To create a packaging bus:

1. Use one of the following methods to open the Create Package Bus form:
 - ☐ Choose *Create – Wiring – Packaging Bus* and press F3.
 - ☐ Use the bindkey **Ctrl+Shift+X** to start the *Create Packaging Bus* command and then press F3.
 - ☐ Click the *Create Bus* button  on the *Create* toolbar.

Virtuoso Interactive and Assisted Routing User Guide

Supporting Virtuoso RF

The Create Packaging Bus form opens. You can set the parameters in this form to create a curved path.

Create Packaging Bus

Connectivity

Net Name

☐ Probe Nets

All Nets Width

All Nets Bit Spacing ☐ Fan Out to Vias

☐ Create Bus Constraint

Number of Bits

Snap Mode

☒ Snap to Pin Center

☒ Snap to Via Center

☒ Auto Terminate

Hide **Cancel** **Defaults** **Help**

2. From the Layers assistant, select a layer on which you want to create the packaging bus.
3. To start creating a packaging bus in an empty space in the design display area:
 - a. Click at a point in the design display area to digitize the bus.
 - b. Continue to digitize the bus and create bends.
 - c. Press `Enter` to finish creating the bus.
4. To create a packaging bus that has connectivity in the empty space in the design display area:
 - a. In the *Create Packaging Bus* form, specify the list of net names separated by commas or spaces in the *Net Name* field. The curved paths that are created are assigned to the specified net. If the specified nets do not exist in the design, they are created.
 - b. Click at a point in the design display area to digitize the bus.

- c. Continue to digitize the bus and create bends.
 - d. Press `Enter` to finish creating the packaging bus.
5. To start creating a packaging bus from existing pins, use one of the following methods to select multiple pins:

☐ **Selecting Multiple Pins**

When you area-select multiple pins that are on the same layer but different nets, the pin nets are used for creating the packaging bus wires; the pin nets are populated in the *Net Name* field in the *Create Packaging Bus* form.

☐ **Adding Pins to the Selection**

Select a single pin and `Shift` + click to add additional pins to the selection.

In both the above methods, you can use the `Ctrl` and `Shift` keys to remove objects from the selection. The pin is removed only from the picked set; the originally selected set remains intact. You can use the `Ctrl` and `Shift` keys only before the first digitized point. To modify the selected set after the first point is digitized, press `Esc`.

☐ **Adding Instance Terminals**

When you select an instance terminal, the wire editor tries to find if there are qualify terminals from other instances that can be used for the bus. The create packaging bus selects all the qualifying terminals from either the horizontal or vertical instances depending on which has more qualified terminals. Following are the qualify terminals:

- The instance master of the qualifying terminal must be the same as the instance master of the picking terminal.
- The qualifying terminal must be the same terminal on the qualifying instance as the picked terminals on the instance.
- The qualifying terminal must intersect with either horizontal or vertical extent regions of the picked terminal.

The packaging bus wires start at pin centers.

☐ **Adding vias at the start of a bus**

When you select a pin to start a packaging bus on a layer that is other than the pin layer, a via is automatically dropped.

- Select a layer in the Palette. For example, `M1`.

Virtuoso Interactive and Assisted Routing User Guide

Supporting Virtuoso RF

- Choose *Create – Wiring – Packaging Bus*.
- Holding the `Ctrl` key, area select the pins that are on a layer other than the layer selected in Layers palette. Vias are dropped at the center of each pin. You can also drop a via at the start of a bus by selecting a single pin using `Ctrl` + click.

To remove a pin from the selection, hold the `Ctrl` key and area select the pin that you want to remove from the selection. If you want to add the pin to the selection, hold the `Shift` key and area select the pin that you want to add.

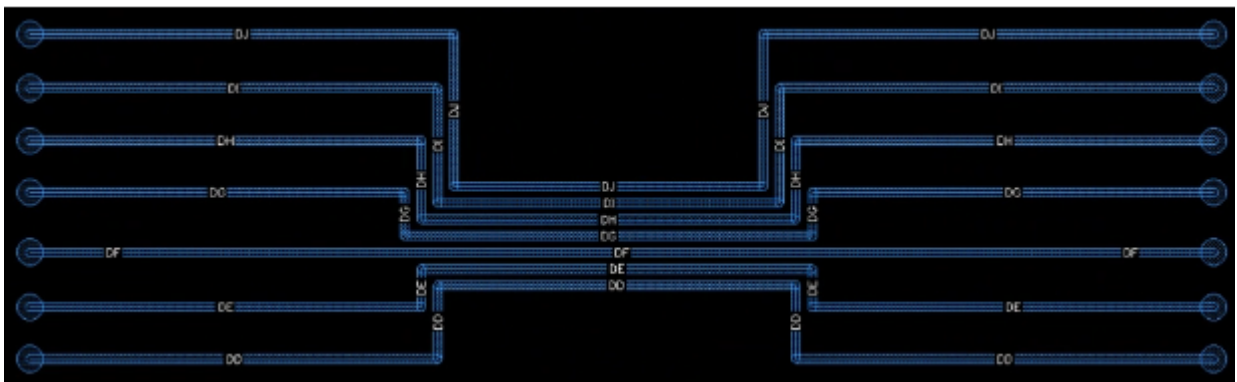
6. Right-click to display the Edit Bus Form.

If required, choose *Cycle Control Wire* from the context-sensitive menu to switch the control wire function to one of the extreme wires or the center wire of the packaging bus. For information about control wires, see Using the Control Wire.

7. You can change width and spacing overrides at any time while creating or reshaping a bus. If you select the *Default Constraints* option from the *Use Spacing* menu, then the spacing after the bend is the `minSpacing`. The default width is based on the `minWidth` and the default spacing value is based on the `minSpacing` constraint for each layer that can be used for routing.

8. Select *Gather/Spread* from the *Use Spacing* menu of the Edit Bus Form to gather or spread packaging bus wires before the first bend of wires.

If you select *Gather/Spread Bus*, the packaging bus wires are surrounded by a dotted box called the bus cursor. Individual wires are gathered or spread inside this box, which can be moved to any desired location. Jogs are added and the wires are gathered to the minimum spacing rule or specified spacing value. The command on the context-sensitive menu toggles to *Cancel Gather/Spread*.



If you select *Gather/Spread Bus* repeatedly and one of the times no gather action is performed because the *Bit Spacing* field in the form is not modified, then the *Create Packaging Bus* command restores the wire spacing that was used before the previous

gather operation. For example, if you start creating a packaging bus from pins that are non-uniformly spaced and *Bit Spacing* is 0, and select *Gather/Spread Bus* from the *Create Bus* context-sensitive menu, the wires are gathered to `minSpacing`. Click and drag to pass the obstruction. If you select *Gather/Spread Bus* again without changing the *Bit Spacing* value, the *Create Packaging Bus* command restores the spacing that was used before the previous spacing, that is, the pin spacing will be restored.

When routing a bus, each time you move the pointer, the locations of the last pathSeg leading up the pointer are computed. If you select *Cancel Gather/Spread Wires* and the wires are not fanned out to a via pattern, only one segment leads to the pointer for each wire. If you select *Gather/Spread Bus*, jogs are added to the pathSegs to bring them closer together if they are spaced farther than the minimum spacing.

Pressing `Backspace` removes the jogs and the wires are not gathered or spread.

9. To change the spacing between multiple wires, specify the new spacing value in the *All Nets Bit Spacing* field.

The spacing is updated from the last digitized point.

10. If required, change the bus wire width in the *All Nets Width* field in the *Create Packaging Bus* form or select the appropriate option from the *Use Width* menu on the Edit Bus Form.

The width is updated from the last digitized point.

11. To remove the last segment of a wire, use `Backspace`.

Bus editing supports *Undo*, *Redo*, and `Backspace`.

12. To change layers and add vias or to fan vias out to connect to the vias, see Changing Layers and Selecting Vias.
13. You can finish creating the packaging bus at any point by double clicking or by pressing `Enter`. The packaging bus ends at the pointer location.

You can check the connectivity of the packaging bus by using the *Edit Curved Path Properties* form or the *Property Editor* assistant.

Related Topics

Area-based Rules in Create Packaging Bus

Snapping Behavior of Curved Paths Using Create Packaging Bus

Edit Bus Form

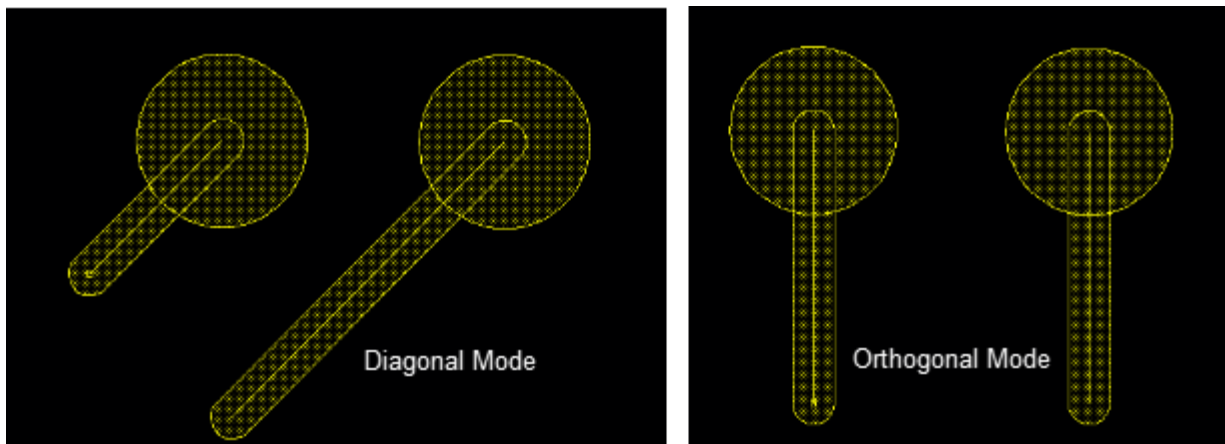
Snapping Behavior of Curved Paths Using Create Packaging Bus

When you start creating a curved path from an existing object, such as a ball-grid array pin, TILP, via, or bond finger, the curved path snaps to the center of the object. The *Snap to Pin Center* and *Snap to Via Center* options in the Create Packaging Bus form are enabled, by default.

The following examples illustrate how a curved path snaps to the center of an object using Create Packaging Bus.

■ Snapping to a Ball Grid Array Pin

When the *Snap to Pin Center* option is selected, the curved path snaps to the center of source and destination pin. However, if the option is deselected, then the curved path starts from the first point of click on pin.

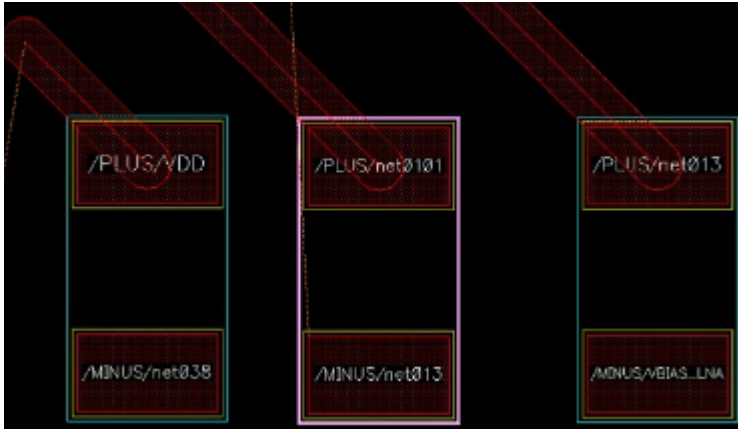


■ Snapping to a Pad

Virtuoso Interactive and Assisted Routing User Guide

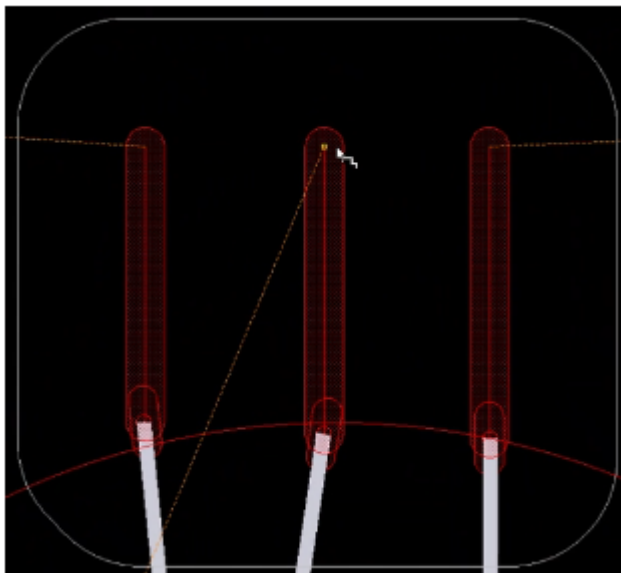
Supporting Virtuoso RF

Support of instances used as pads has been added for the *Create Packaging Bus* command, notably in smart snapping and in the flightlines engine.



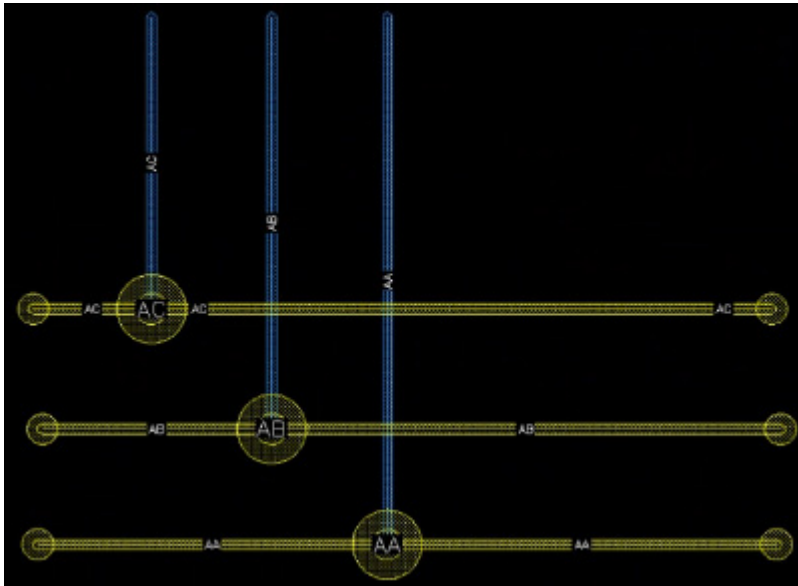
■ Snapping to a Bond Finger

When *Snap to Pin Center* is selected, the curved path snaps to the center of the bond finger. However, if the option is deselected, then the curved path starts from the first point of click on the bond finger.



■ Snapping from a Padstack Via

When the *Snap to Via Center* option is selected, the curved path snaps to the center of a via. However, if the option is deselected, then the curved path starts from the first point of click on the via.



Related Topics

[Creating a Packaging Bus](#)

[Edit Bus Form](#)

[Area-based Rules in Create Packaging Bus](#)

Supporting Package Constraints

When the associated technology is a packaging technology, interactive routing supports the following three package constraints instead of the usual spacing and width constraints.

- [pkgMaxLineWidth](#)
Specifies the maximum line width for traces in a package design.
- [pkgMinLineWidth](#)
Specifies the minimum line width for traces in a package design.
- [pkgMinSpacing](#)

Specifies the minimum spacing between shapes of the specified object types on the specified layer for advanced package designs.

Area-based Rules in Create Package Wire

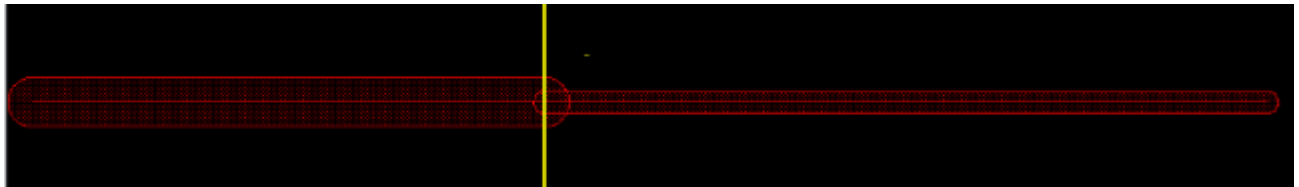
For Virtuoso RF, interactive routing supports the following area-based rules:

- minWidth
- maxWidth
- minSpacing

For more information, see [Support for Area-based Rules](#).

minWidth

The following figure shows an example of a geometric wire in two regions. It shows that when the geometric wire spans across regions, it adjusts the width of the geometric wire based on the `minWidth` value of the spanned regions.

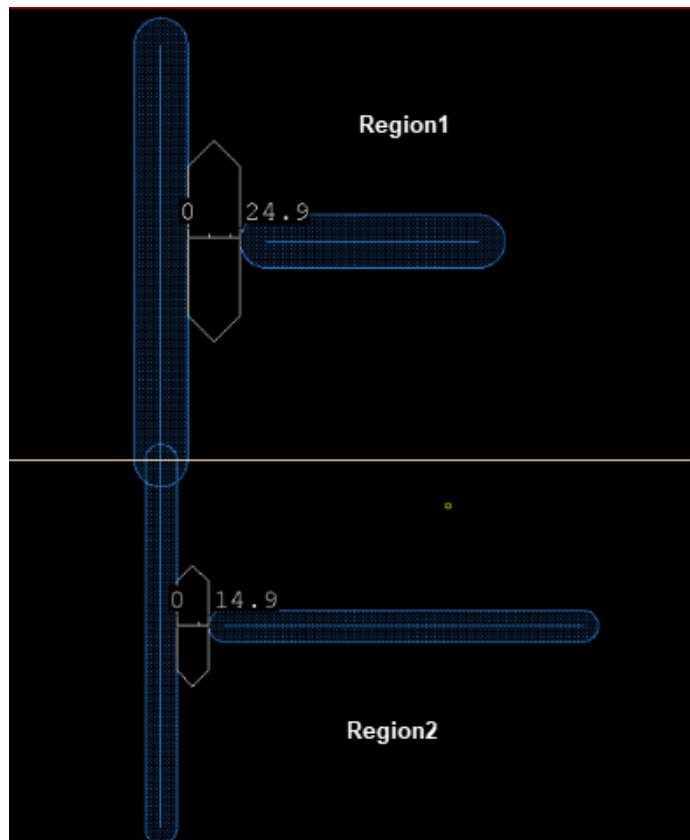


maxWidth

In case of `maxWidth`, when the geometric wire spans across regions, it adjusts the width of the geometric wire based on the `maxWidth` value defined for a region.

minSpacing

The following figure shows how different `minSpacing` values are defined for two regions.

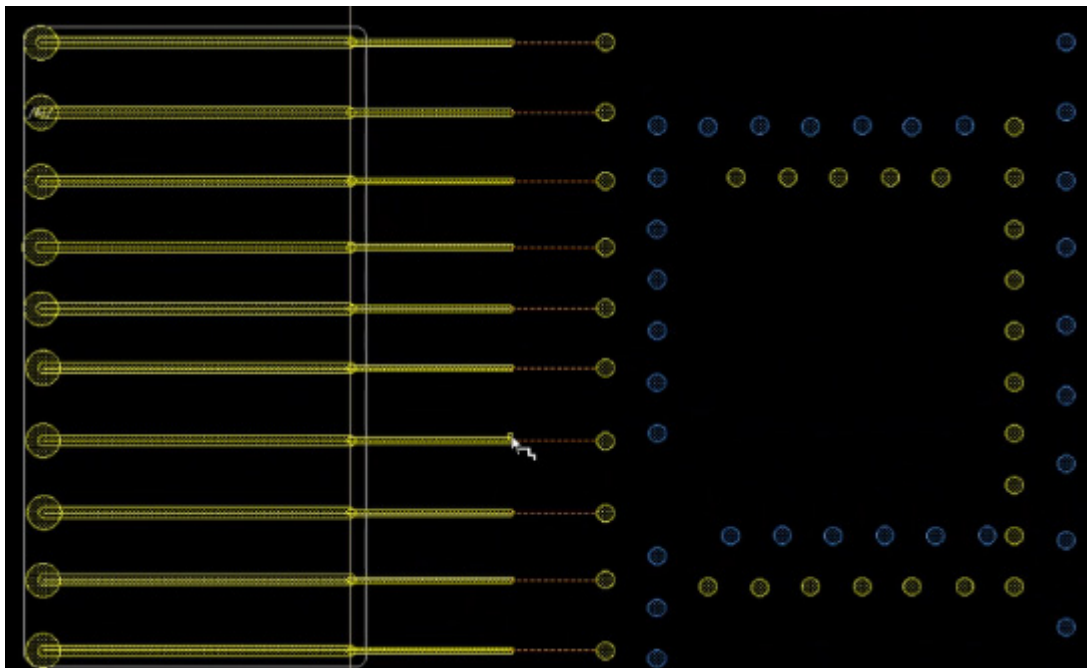


Area-based Rules in Create Packaging Bus

The Create Packaging Bus command supports the `minWidth` and `minSpacing` area-based rules.

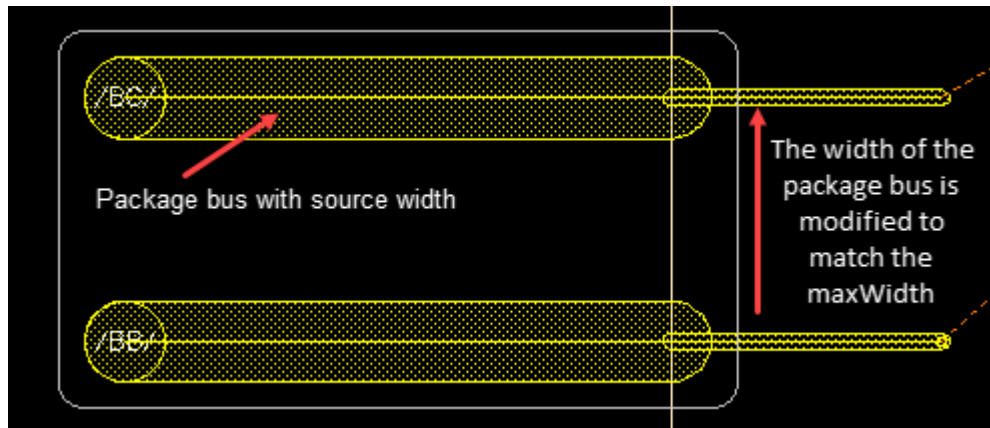
`minWidth`

In the following figure, you see how the bus spans across regions adjusting the width of the bus based on the `minWidth` value of the spanned regions.



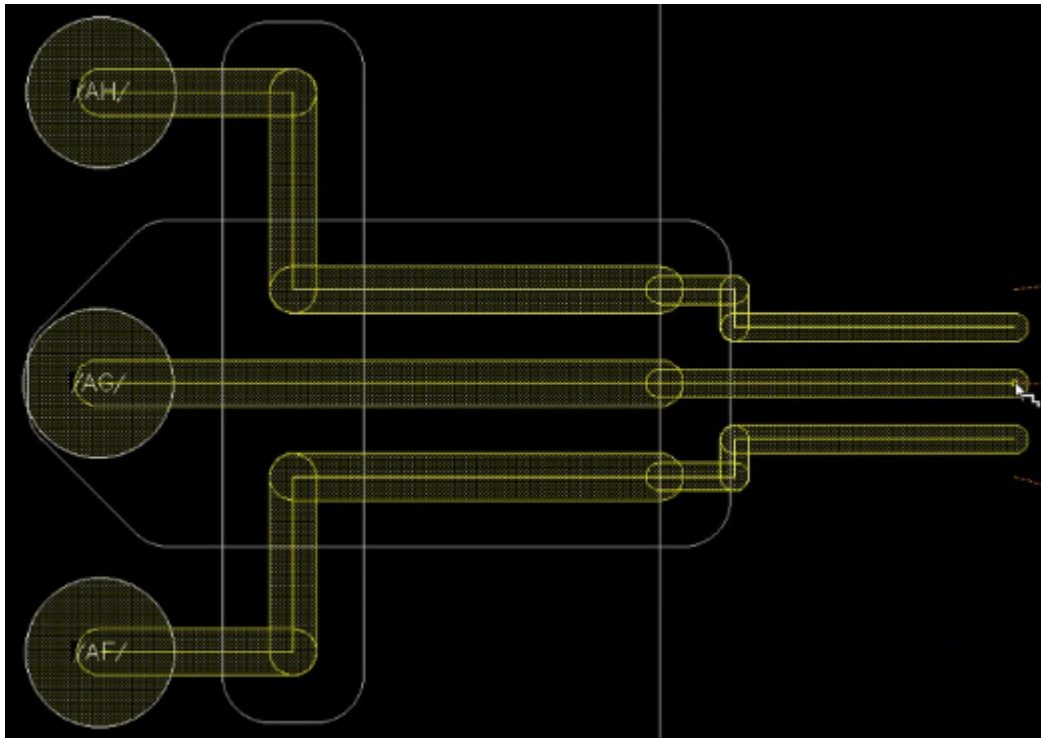
maxWidth

In case of `maxWidth`, when the packaging bus spans across regions, it adjusts the width of the bus based on the `maxWidth` value of the spanned regions.



minSpacing

The following figure shows how different `minSpacing` values are defined for two regions for the *Create Packaging Bus* command.



Related Topics

[Creating a Packaging Bus](#)

[Edit Bus Form](#)

[Snapping Behavior of Curved Paths Using Create Packaging Bus](#)

Push and Shove in Interactive Routing

The push-and-shove feature applies to routing objects, such as wires and vias. Static objects, such as pins, bumps, and bond fingers, are not considered routing objects. This capability lets you push wires out of the way by either automatically adjusting the new wire or by pushing an existing wire.

Enabling this feature enhances correct-by-construction design and makes the design DRC- and shorts-aware.

In interactive routing, the push-and-shove feature is available for the *Create Wire* and the *Stretch* commands. The push-and-shove feature also supports area-based rules and shielding constraints.

- [Using Push and Shove while Creating a Wire](#)
- [Using Push and Shove while Stretching a Wire](#)
- [Support for Area-based Rules in the Push-and-Shove Feature](#)
- [Support for Shielding Constraints in the Push-and-Shove Feature](#)

Video

You can view a video demonstration of this feature at [Routing a Package Design](#). Also read a blog related to this at [How to Route Package in Virtuoso?](#)

Using Push and Shove while Creating a Wire

While creating a wire, you can use the push-and-shove options to reroute a wire to be DRC correct.

To use the push-and-shove options while creating a wire:

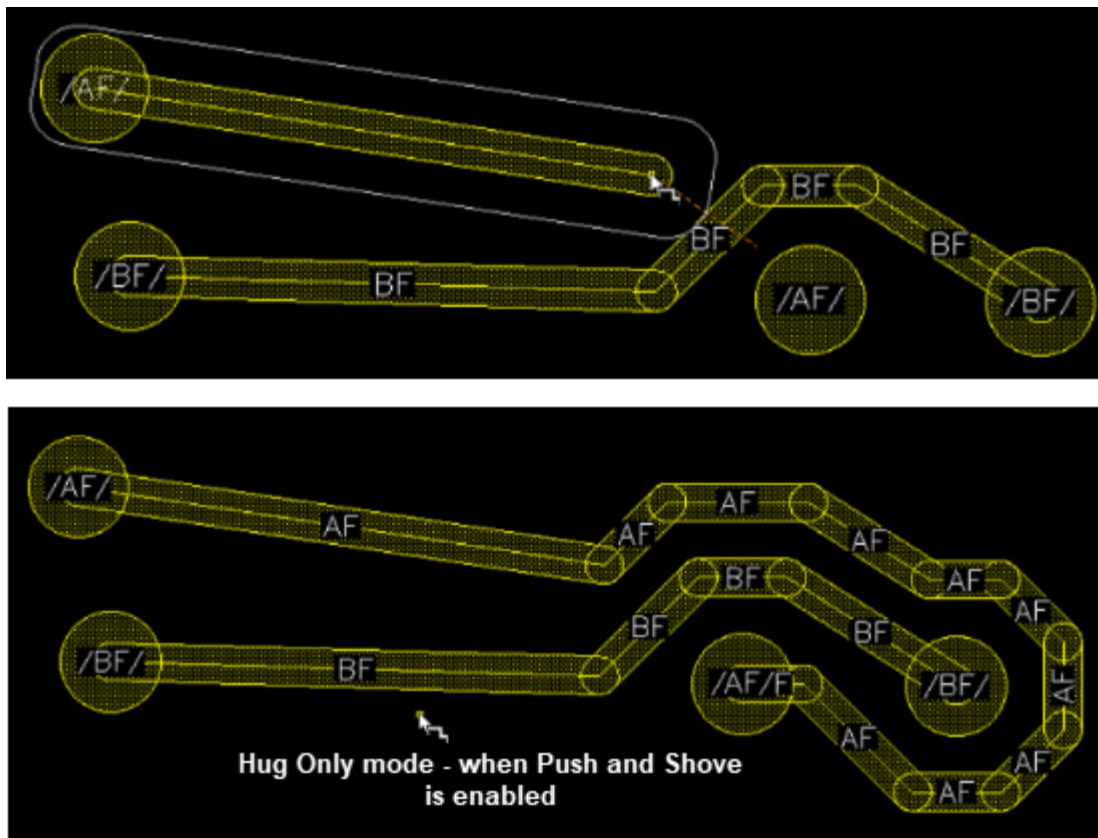
1. Choose *Create – Wiring – Wire*.
2. Press **F3** to open the Create Wire form.

Virtuoso Interactive and Assisted Routing User Guide

Supporting Virtuoso RF

The Create Packaging Wire form appears.

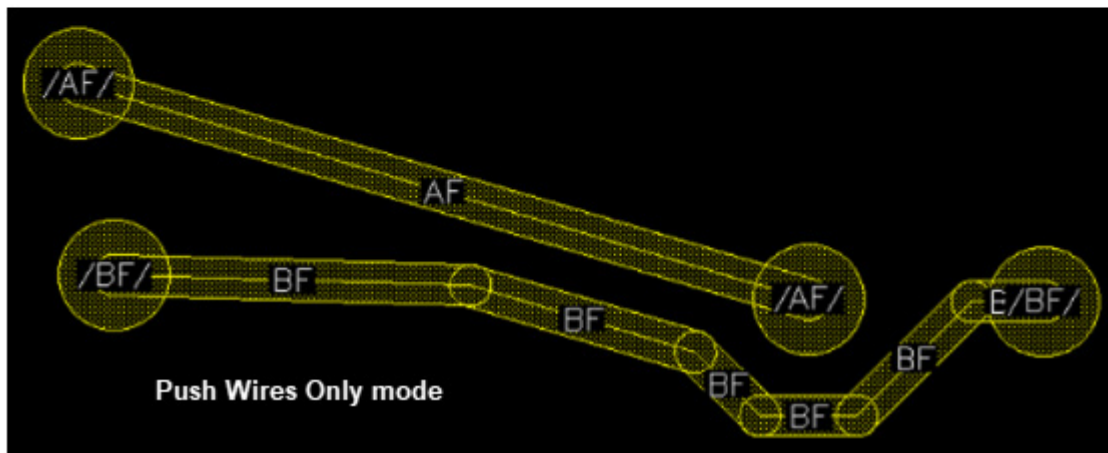
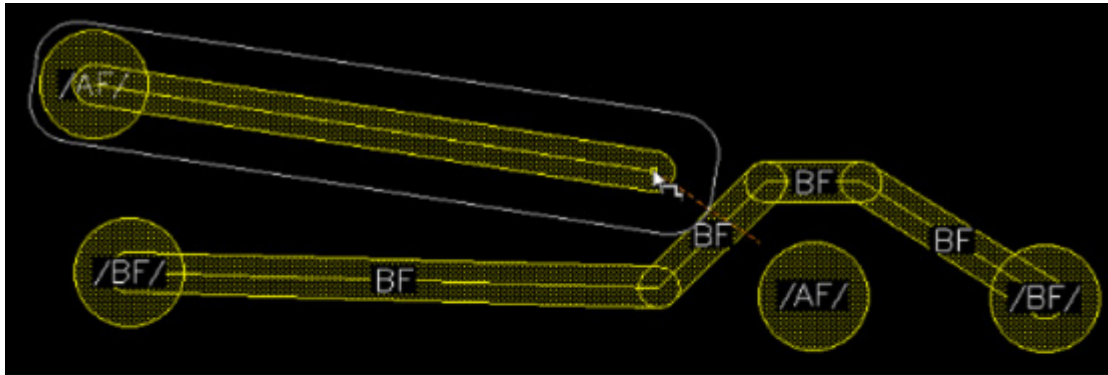
3. In the *Net Name* field, specify a net name to assign connectivity to the wire. The push-and-shove feature does not work on wires that do not have connectivity.
4. Select a value from the *Snap Mode* cyclic field. If set to *orthogonal* or *diagonal*, you can select to create orthogonal or diagonal wires. The other supported push-and-shove snap modes are *anyAngle*, *L45LongFirst*, and *L45AngleFirst*.
5. Digitize points where you want to create the wire.
6. Right-click the wire that is being created and point to the *Push and Shove* option in the *Create Single Wire* context-sensitive menu.
7. Select one of the two push-and-shove modes to enable the feature. By default, the push-and-shove mode is *Off*.
 - ☐ *Hug Only*: The wire that is being created is rerouted around the existing shapes and considers the `minSpacing` values set between itself and the existing shapes. This mode avoids other shapes.



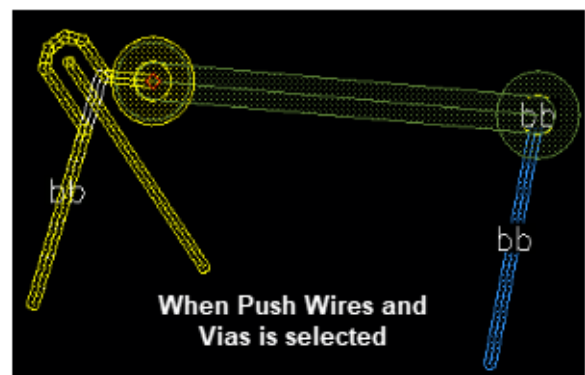
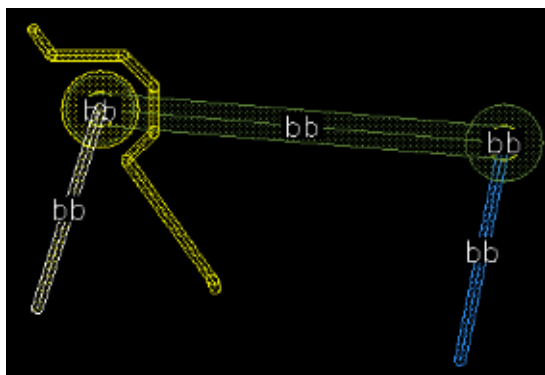
Virtuoso Interactive and Assisted Routing User Guide

Supporting Virtuoso RF

- ❑ **Push Wires Only:** The wire that is being created pushes the existing non-static shapes, such as wires, or is rerouted around the static existing shapes, such as pins. The edited wire considers the `minSpacing` values set between itself and the other existing shapes.



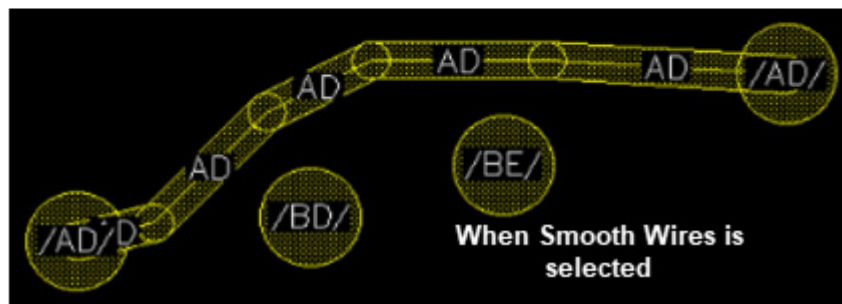
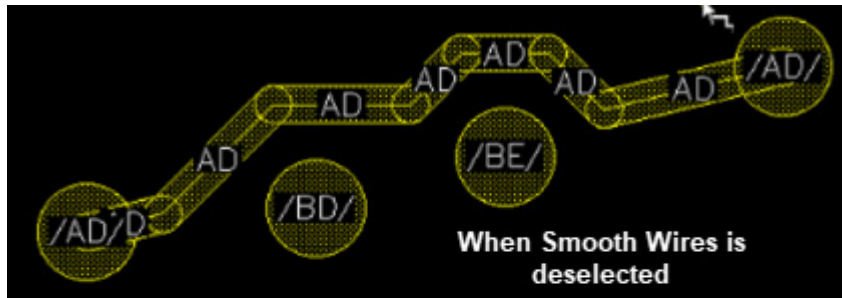
- ❑ **Push Wires and Vias:** existing wires and vias are pushed to avoid violations with the edited wire.



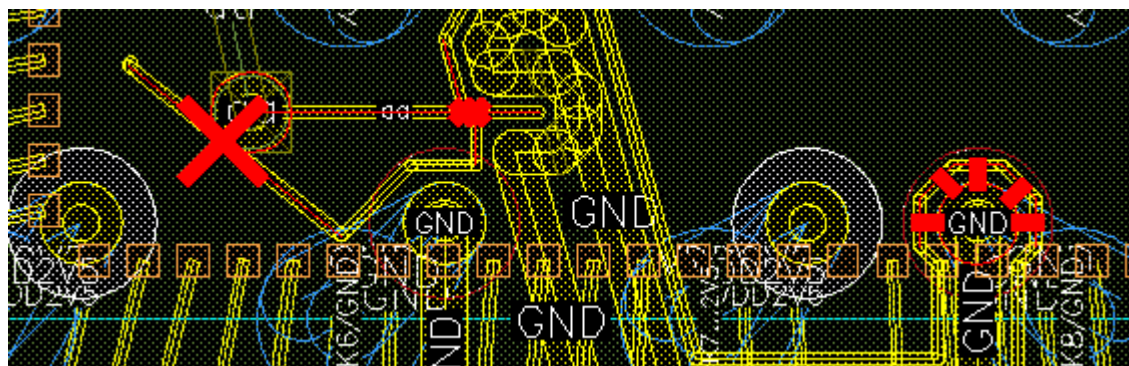
Virtuoso Interactive and Assisted Routing User Guide

Supporting Virtuoso RF

- Click *Smooth Wires* to remove unnecessary jogs and loops that might be created while pushing the wire.



- Choose *Show Remaining Violations* to view the violations that might be introduced while rerouting a wire. This option is selected by default.



Avoid congested areas and ensure that there is enough space to push wires.

Using Push and Shove while Stretching a Wire

When stretching a wire, you can use the push-and-shove options to reroute a wire and fix the violations.

To use the push-and-shove options while stretching a wire:

1. Choose *Edit – Stretch*.
2. Press **F3** to open the *Stretch* form.

The Stretch form is displayed.

3. Select the desired *Snap Mode*.

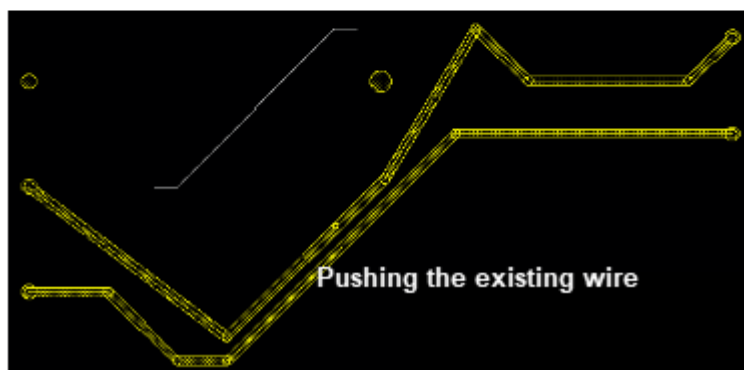
The *Snap Mode* enforces restrictions on the movement of the pointer. For example, if one end of a diagonal segment in a wire is stretched when the snap mode is horizontal, it moves only horizontally.

4. (Optional) Select *Keep Wires Connected to shapes* to maintain the wire-to-wire and wire-to-via connections while stretching.
5. Click or area select the wire or portion of wire that you want to stretch.
6. Right-click the wire that is being stretched and point to the *Push and Shove* option in the *Stretch* context-sensitive menu. There is only a single push-and-shove mode

Virtuoso Interactive and Assisted Routing User Guide

Supporting Virtuoso RF

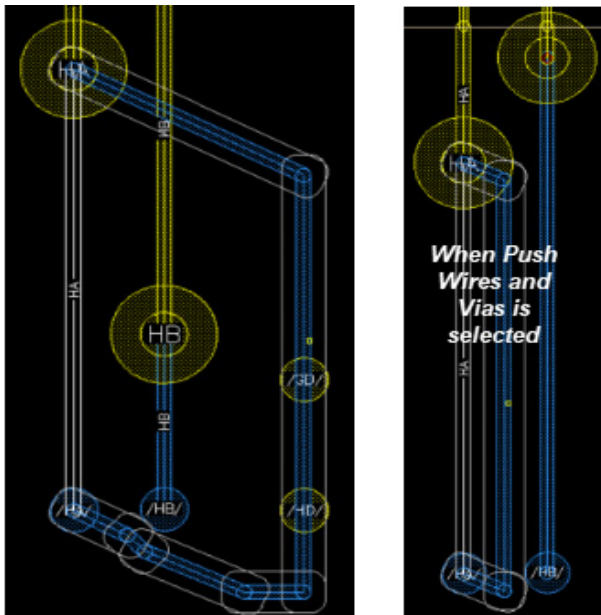
available for the *Stretch* command, which is enabled by default. The existing wires are pushed to create space for the edited wire.



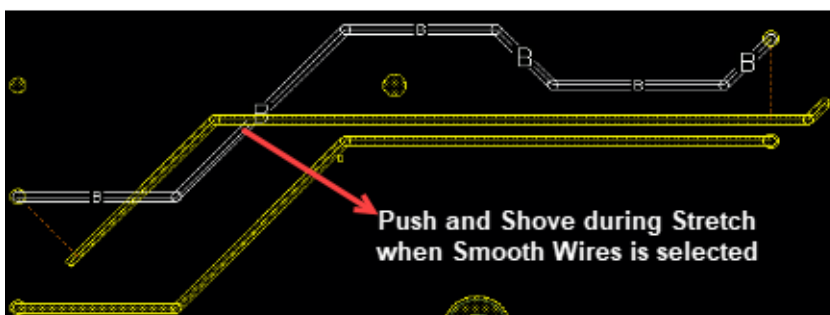
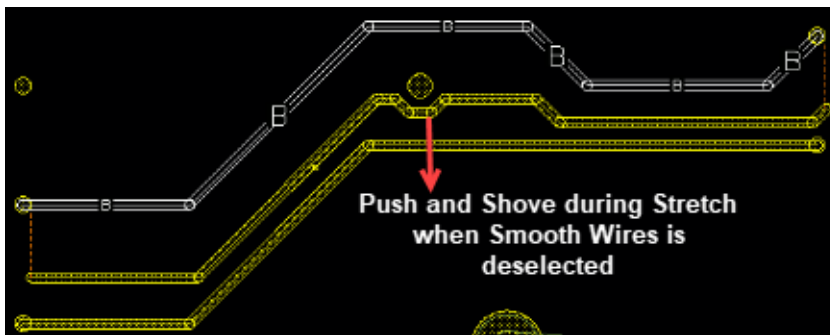
Virtuoso Interactive and Assisted Routing User Guide

Supporting Virtuoso RF

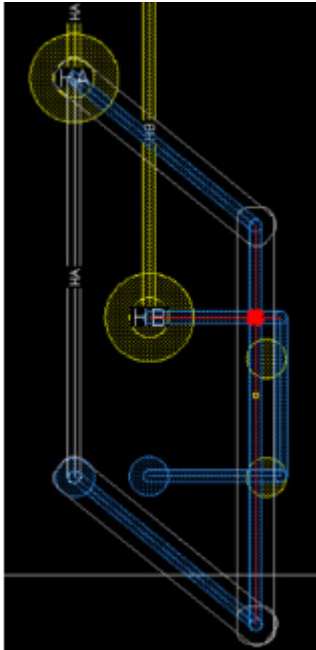
The *Push Wires and Vias* option pushes existing wires and vias to avoid violations with the edited wire.



7. Click *Smooth Wires* to remove unnecessary jogs and loops that might be created while pushing the wire.



8. Choose *Show Remaining Violations* to view the violations that might be introduced while rerouting a wire. This option is selected by default.



Support for Area-based Rules in the Push-and-Shove Feature

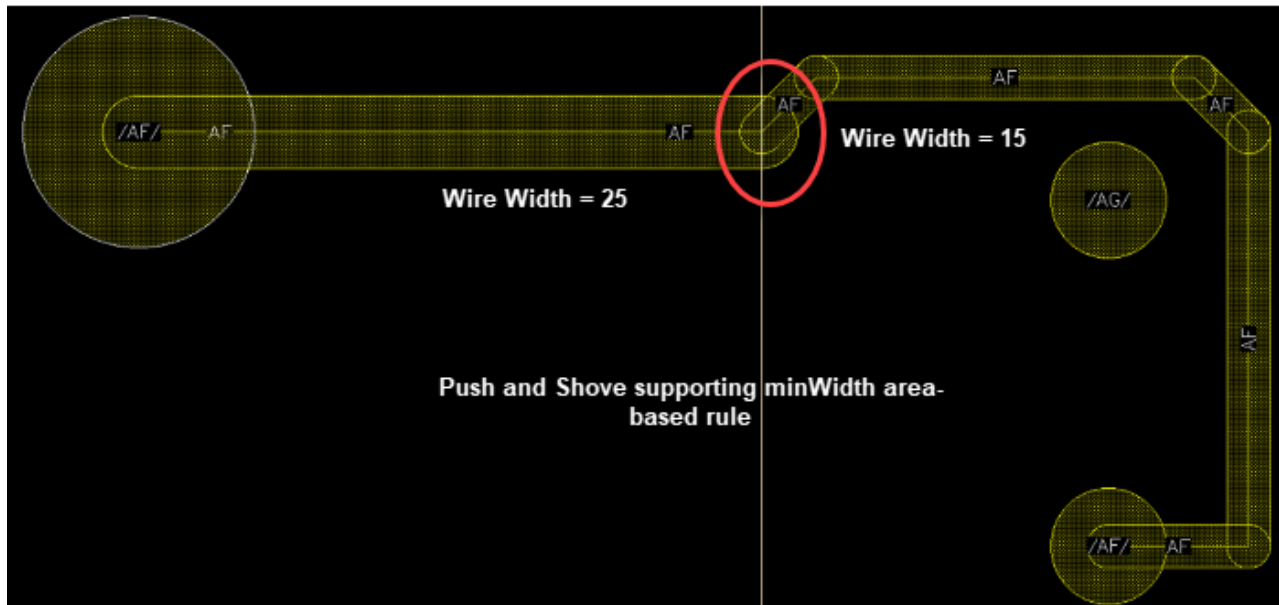
The wires that are created and edited using the push-and-shove feature supports the area-based rules and are sized and spaced accordingly. The push-and-shove feature in interactive routing supports the `minWidth`, `maxWidth`, and `minSpacing` area-based rules. For more information, see [Support for Area-based Rules](#).

The following figure shows how a geometric wire starts from a die pin and crosses different regions. In this case, when the wire spans across regions, the wire is split and pushing and

Virtuoso Interactive and Assisted Routing User Guide

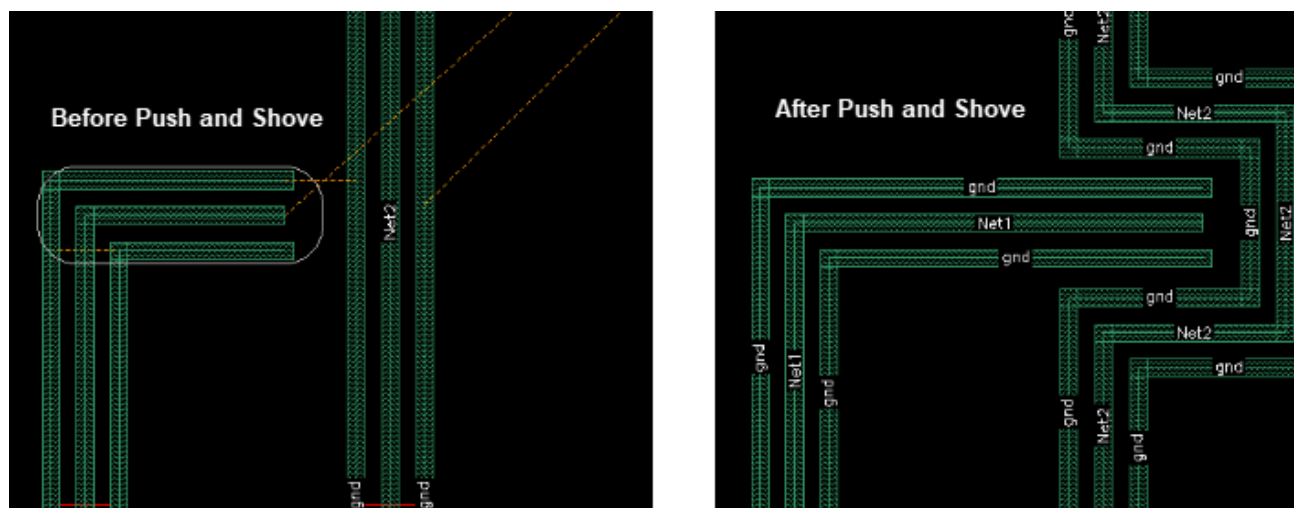
Supporting Virtuoso RF

shoving adjusts the wire width by avoiding the nearby objects and respects the `minWidth`, `maxWidth`, and `minSpacing` constraint values defined for each region.



Support for Shielding Constraints in the Push-and-Shove Feature

The push-and-shove feature supports shielding constraints that are defined in the technology file. It supports the shield spacing and width constraints and update the shielding of modified wires and vias.



Related Topics

[Introduction to the Push-and-Shove Feature](#)

[Push And Shove](#)

Virtuoso Interactive and Assisted Routing User Guide

Supporting Virtuoso RF

Interactive and Assisted Routing Forms

The following table lists the basic editing forms available for interactive and assisted routing. These forms are available in layout XL and higher tiers.

<u>Copy Route Form</u>	<u>Create Bus Form</u>	<u>Create Packaging Bus Form</u>
<u>Create Stranded Wire Form</u>	<u>Create Wire Form</u>	<u>Edit Bus Form</u>
<u>Metal Density Options Form</u>	<u>Point to Point Form</u>	<u>Select Via Form</u>

Related Topics

[Routing Assistant - Interactive Mode](#)

[Interactive and Assisted Routing Shortcut Menu Commands](#)

Create Bus Form

The following table describes the fields available on the Create Bus form.

Field	Description
Connectivity	This section lets you specify the nets for which you want to create a bus and optionally lets you create a probe for the specified net.
<i>Net Name</i>	Specifies the net names for creating a bus. You can specify the net names separated by commas or spaces or specify them in bus notation, such as net<0:3>. New nets are created if the ones specified do not already exist. If you start creating a bus from existing pins, the net names are automatically picked from the net of the pins. Environment variable: <u>netName</u>
<i>Probe Nets</i>	Creates probes for the autopicked, selected, or specified nets.
	This section lets you specify the width, direction, and style for the wire that you want to create.
<i>All Nets Width</i>	Specifies the width of each wire in the bus. You can increase or decrease the wire width by pressing <code>Ctrl + Shift</code> keys and scrolling the mouse wheel up or down, respectively. Environment variable: <u>wireWidth</u>
<i>All Nets Bit Spacing</i>	Specifies the distance between the centerlines of the bus wires. If no value is specified in this field, the <code>minSpacing</code> value is considered.
<i>Fan Out to Vias</i>	Spreads bus wires for via placement. When wires are at the minimum wire spacing but the via spacings are larger, the wires can either be fanned out to connect to the vias in the pattern or can be back tracked to intersect the previous segment in the wire. Environment variable: <u>fanOutToVias</u>
<i>Create Bus Constraint</i>	Creates a bus constraint automatically, if one does not already exist, for the nets you specify in the <i>Net Name</i> field or for the ones you pick by area-selecting multiple pins or instPins. You can view the created bus constraints in the Constraint Manager in Layout XL and higher tiers. Environment variable: <u>weCreateBusConstraint</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Number of Bits</i>	Lets you specify the number of wires in an unassigned bus. The default value is 2. ■ In the <i>Net Name</i> field, if you specify net names as a bundle name or by using the bus notation, the value in the <i>Number of Bits</i> cyclic field automatically updates to match the number of nets specified. If net names are specified in the <i>Net Name</i> field and you manually update the value in the <i>Number of Bits</i> cyclic field, the net names specified in the <i>Net Name</i> field are removed. ■ In the <i>Net Name</i> field, if you specify the net names by using the bus base name, you can set up the cardinality of the bus by using the <i>Number of Bits</i> cyclic field. The bus base name expands to display individual net names in the <i>Net Name</i> field only after you start creating the bus. ■ If you update the <i>Number of Bits</i> cyclic field after starting bus creation, the net names are removed from the <i>Net Name</i> field. ■ If you change the number of bits from the default value, which is 2, to any other value, and then specify the net name, the number of bits does not get reset to default value 2. The last changed value is retained in the <i>Number of Bits</i> cyclic field. Environment variable: <u>weBusNumBits</u>
<i>Justification</i>	Controls the direction in which to offset the pathSeg from the digitized points. The supported routing directions include <i>center</i>, <i>left</i>, and <i>right</i>. The default is <i>center</i>. Environment variable: <u>pathJustify</u>
<i>Offset</i>	Creates wire at a specified distance from the digitized points with respect to the centerline of the pathSeg, in the direction specified by <i>Justification</i>. The default is 0. Environment variable: <u>wireOffset</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Begin Style</i>	Controls how the beginning wire segment ends are created. ■ <i>auto</i>: Automatically sets the begin style of the wire. □ When the wire is Manhattan in shape, its <i>Begin Style</i> is set to <i>truncate</i>. □ Two orthogonal Manhattan wires will have their <i>Begin Style</i> and <i>End Style</i> set to <i>extend</i> to avoid notches. □ When the wire is diagonal in shape, its <i>Begin Style</i> is set to <i>custom</i>. □ Two diagonal orthogonal wires will have their <i>Begin Style</i> and <i>End Style</i> set to <i>custom</i> to avoid notches. ■ <i>truncate</i>: Creates a wire with no begin extension. ■ <i>extend</i>: Creates a wire with the default begin extension value. ■ <i>variable</i>: Lets you override the value specified in the <i>Begin</i> field for the begin extension of a wire. ■ <i>custom</i>: Constraints the vertices of the created wire to the current grid. The default value is <i>auto</i>. Environment variable: <u>pathSegBeginStyle</u>
<i>Begin</i>	Specifies the value for the begin extension of the wire. This field is enabled when <i>Begin Style</i> is set to <i>variable</i>. Environment variable: <u>pathSegBeginExt</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

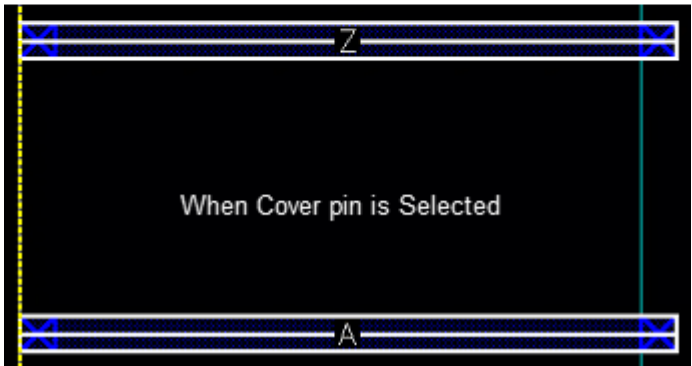
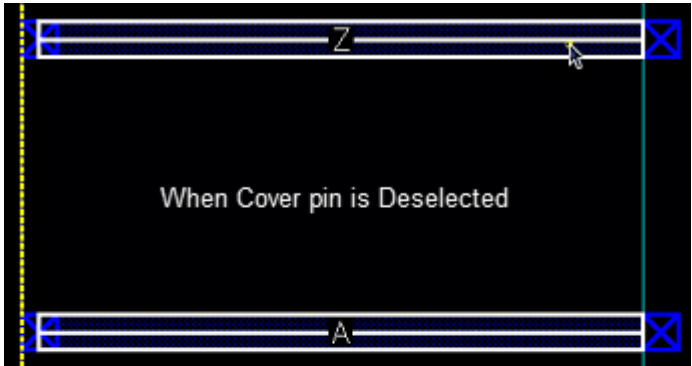
Field	Description
<i>End Style</i>	Controls how the beginning wire segment ends are created. ■ <i>auto</i>: Automatically sets the end style of the wire. □ When the wire is Manhattan in shape, its <i>Begin Style</i> is set to <i>truncate</i>. □ Two orthogonal Manhattan wires will have their <i>Begin Style</i> and <i>End Style</i> set to <i>extend</i> to avoid notches. □ When the wire is diagonal in shape, its <i>End Style</i> is set to <i>custom</i>. □ Two diagonal orthogonal wires will have their <i>Begin Style</i> and <i>End Style</i> set to <i>custom</i> to avoid notches. ■ <i>truncate</i>: Creates a wire with no begin extension. ■ <i>extend</i>: Creates a wire with the default begin extension value. ■ <i>variable</i>: Lets you override the value specified in the <i>Begin</i> field for the begin extension of a wire. ■ <i>custom</i>: Constraints the vertices of the created wire to the current grid. The default value is <i>auto</i>. Environment variable: <u>pathSegEndStyle</u>
<i>End</i>	Specifies the value for the end extension of the wire. This field is enabled when <i>End Style</i> is set to <i>variable</i>. Environment variable: <u>pathSegEndExt</u>
The following options let you specify the snapping behavior for the bus that you want to create.	

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Snap Mode</i>	Controls how the cursor snaps when you create a bus. ■ <i>diagonal</i>: Creates wires parallel to the X or Y axis or at a 45-degree angle to the axes. ■ <i>orthogonal</i>: Creates wires parallel to the X or Y axis. ■ <i>L90XFirst</i>: Creates orthogonal two-segment wires and creates the first line in the X direction. ■ <i>L90YFirst</i>: Creates orthogonal two-segment wires and creates the first line in the Y direction. The default value is <i>orthogonal</i>. Environment variable: <u>snapMode</u>
<i>Snap to Pin Center</i>	Snaps the bus wires to the center of rectangular pins or of pins that are created as polygons and have rectangular shapes. When this check box is selected, bus wires snap to the starting and ending pins centers. By default, the <i>Snap to Pin Center</i> check box is not selected. Environment variable: <u>snapToPin</u>
<i>Auto Terminate</i>	Finishes the <i>Create Bus</i> command as soon as a point is digitized on a flightline target object (pin, existing wire, or via). All wires in the bus snap to the target pin. This option is selected by default.
<i>Snap to Via Center</i>	Snaps wires to the center of vias. If this check box is selected and you click a via, the starting wire starts from the clicked point and two small segments are added to connect to the via origin. With this check box selected, tapping a via and a wire gives preference to the via. This check box is off by default. Environment variable: <u>snapToViaCenter</u>

Field	Description
<i>Cover Pin</i>	Adjusts the wire segment to automatically cover the source and target pins. Otherwise, the wire segment is not adjusted and the wire terminates at the point you chose. By default, this option is deselected.



Environment variable: [weCoverPin](#)

<i>Custom Layer Pattern</i>	Enables or disables the feature to specify a multi-layer pattern. The <i>Custom Layer Pattern</i> combo box lets you specify a multi-layer pattern or select an existing multi-layer pattern. For more information, see Creating a Multi-Layer Bus from an Empty Space .

Related Topics

[Creating a Bus](#)

[Edit Bus Form](#)

Edit Bus Form

Use the Edit Bus command to interactively change the width and spacing gap for the selected bus bits.

Field	Description
Select	
<i>Net Name</i>	Displays the net names of the selected bus bits. The net name corresponding to each selected bus bit appears in the bus bit table.
Bus Bit Table	
<i>Net</i>	Name of the net corresponding to the bus bit. The name of bus bit can be a single net name or the bit-range names.
<i>Width</i>	Minimum width of the bus bit. The default value of the bus bit width is the constraint look-up minimum width. You can change the width of a bus bit.
<i>Gap</i>	Minimum spacing between two adjacent bus bits. The default value is the constraint look-up minimum spacing. You can also modify the spacing between two bus bits.
<i>All Nets</i>	Lets you specify a value to override the width and spacing values for all the bus bits in the form. A warning message is displayed in the CIW if the value specified is less than the default look-up value.
Via	
<i>Adjust Edited Vias Parameters</i>	Calculates via parameters (such as alignment or enclosures) if you stretch the wire connected to the via or while editing width and layer properties of a pathSeg. By default the option is deselected.
<i>minNumCuts</i>	Specifies the via cuts as minimum number of cuts for creating and editing wires. If the override value is less than either the technology rule or the current constraint group on the net or on the design, then the value specified in the Edit Bus form is reset to the minimum value.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Row/Col</i>	Specifies a via cuts as the number of rows or number of columns. The increase and decrease in the number of via rows and columns is dependent on the minimum number of via cuts. number of via rows * number of via columns >= minimum number of via cuts.

Related Topics

[Editing a Bus](#)

[Create Bus Form](#)

Create Wire Form

Use the Create Wire form to create symbolic wires. Symbolic wires are created using pathSegs and vias that are contained within routes.

Field	Description
<i>Net Name</i>	Specifies the net to which the created wire is to be added. If the specified net does not exist, a new net is created.
<i>Probe Nets</i>	Creates probes for the specified net (Layout XL and higher tiers).
	This section lets you specify the width, direction, and style for the wire that you want to create.
<i>Width</i>	Specifies the width of the wire to be created. Environment variable: <u>wireWidth</u>
<i>Justification</i>	Controls the direction in which to offset the pathSeg from the digitized points of the path or pathSeg. The supported routing directions include <i>center</i> , <i>left</i> , and <i>right</i> . The default is <i>center</i> . Environment variable: <u>pathJustify</u>
<i>Offset</i>	Creates the wire at a specified distance from the digitized points with respect to the centerline of the pathSeg, in the direction specified by <i>Justification</i> . The default is 0. Environment variable: <u>wireOffset</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Begin Style</i>	Controls how the beginning wire segment ends are created. ■ <i>auto</i>: Automatically sets the begin style of the wire. □ When the wire is Manhattan in shape, its <i>Begin Style</i> is set to <i>truncate</i>. □ Two orthogonal Manhattan wires will have their <i>Begin Style</i> and <i>End Style</i> set to <i>extend</i> to avoid notches. □ When the wire is diagonal in shape, its <i>Begin Style</i> is set to <i>custom</i>. □ Two diagonal orthogonal wires will have their <i>Begin Style</i> and <i>End Style</i> set to <i>custom</i> to avoid notches. ■ <i>truncate</i>: Creates a wire with no begin extension. ■ <i>extend</i>: Creates a wire with the default begin extension value. ■ <i>variable</i>: Lets you override the value specified in the <i>Begin</i> field for the begin extension of a wire. ■ <i>custom</i>: Constraints the vertices of the created wire to the current grid. The default value is <i>auto</i>. Environment variable: <u>pathSegBeginStyle</u>
<i>Begin</i>	Specifies the value for the begin extension of the wire. This field is enabled when <i>Begin Style</i> is set to <i>variable</i>. Environment variable: <u>pathSegBeginExt</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>End Style</i>	Controls how the beginning wire segment ends are created. ■ <i>auto</i>: Automatically sets the end style of the wire. □ When the wire is Manhattan in shape, its <i>Begin Style</i> is set to <i>truncate</i>. □ Two orthogonal Manhattan wires will have their <i>Begin Style</i> and <i>End Style</i> set to <i>extend</i> to avoid notches. □ When the wire is diagonal in shape, its <i>End Style</i> is set to <i>custom</i>. □ Two diagonal orthogonal wires will have their <i>Begin Style</i> and <i>End Style</i> set to <i>custom</i> to avoid notches. ■ <i>truncate</i>: Creates a wire with no begin extension. ■ <i>extend</i>: Creates a wire with the default begin extension value. ■ <i>variable</i>: Lets you override the value specified in the <i>Begin</i> field for the begin extension of a wire. ■ <i>custom</i>: Constraints the vertices of the created wire to the current grid. The default value is <i>auto</i>. Environment variable: <u>pathSegEndStyle</u>
<i>End</i>	Specifies the value for the end extension of the wire. This field is enabled when <i>End Style</i> is set to <i>variable</i>. Environment variable: <u>pathSegEndExt</u> The following options let you specify the snapping behavior for the wire that you want to create.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Snap Mode</i>	Controls how the cursor snaps when you create the wire. ■ <i>diagonal</i>: Creates wires parallel to the X or Y axis or at a 45-degree angle to the axes. ■ <i>orthogonal</i>: Creates wires parallel to the X or Y axis. ■ <i>L90XFirst</i>: Creates orthogonal two-segment wires and creates the first line in the X direction. ■ <i>L90YFirst</i>: Creates orthogonal two-segment wires and creates the first line in the Y direction. The default value is <i>orthogonal</i>. Environment variable: <u>snapMode</u>
<i>Auto Terminate</i>	Finishes the <i>Create Wire</i> command as soon as a point is digitized on a flightline target object (pin or existing wire or via). When creating a bus and <i>Auto Terminate</i> is on, all wires in the bus snap to its target pin. This option is selected by default.
<i>Snap to Pin Center</i>	Snaps the wire to the center of rectangular pins or of pins that are created as polygons and have rectangular shapes. If this check box is selected, wires snap to the starting and ending pins. The <i>Snap to Pin Center</i> check box is deselected by default. This option is available in Layout XL and higher tiers. Environment variable: <u>snapToPin</u>
<i>Snap to Via Center</i>	Snaps a wire to the center of vias. The starting wire starts from the clicked point and two small segments are added to connect to the via origin. Tapping a via and a wire gives preference to the via. This check box is off by default. This option is available in Layout XL and higher tiers. Environment variable: <u>snapToViaCenter</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Cover Pin</i>	Adjusts the wire segment to automatically cover the source and target pins. Otherwise, the wire segment is not adjusted and the wire terminates at the point you chose. By default, this option is deselected.



When Cover Pin is Selected



When Cover Pin is Deselected

Environment variable: [weCoverPin](#)

Related topics

[Creating a Wire](#)

[Create Stranded Wire Form](#)

[Edit Bus Form](#)

[Reshape Wire Shortcut Menu Commands](#)

Create Stranded Wire Form

The *Create Stranded Wire* command is used to create a wire comprising multiple strands in a single direction on a single net.

Field	Description
<i>Net Name</i>	Specifies a net name for the current stranded wire. The stranded wire is added to the specified net. A new net is created if the one specified does not exist. Environment variable: <u>netName</u>
<i>Probe Nets</i>	Creates probes for the selected or specified net (Layout XL and higher tiers).
<i>Parameters</i>	This section lets you specify the width, direction, and style for the wire that you want to create.
<i>Strand Width</i>	Specifies the width of each strand in the stranded wire.
<i>Strand Number</i>	Specifies the number of strands in each stranded wire.
<i>Strand Spacing</i>	Specifies the spacing between the individual strands of the stranded wire.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Begin Style</i>	Controls how the stranded wire segment begins. ■ <i>auto</i>: Automatically sets the begin style of the stranded wire. □ When the stranded wire is Manhattan in shape, its <i>Begin Style</i> is set to <i>truncate</i>. □ Two orthogonal Manhattan wires will have their <i>Begin Style</i> and <i>End Style</i> set to <i>extend</i> in order to avoid notches. □ When the stranded wire is diagonal in shape, its <i>Begin Style</i> is set to <i>custom</i>. □ Two diagonal orthogonal stranded wires will have their <i>Begin Style</i> and <i>End Style</i> set to <i>custom</i> in order to avoid notches. ■ <i>truncate</i>: Creates a stranded wire with no begin extension. ■ <i>extend</i>: Creates a stranded wire with the default beginning extension value. ■ <i>variable</i>: Lets you override the value specified in the <i>Begin</i> field for the begin extension of a stranded wire. ■ <i>custom</i>: Creates a stranded wire in which the vertices are constrained to the current grid. The default value is <i>auto</i>. Environment variable: <u>pathSegBeginStyle</u>
<i>Begin</i>	Specifies the value for the begin extension of the stranded wire. This field is enabled when <i>Begin Style</i> is set to <i>variable</i>. Environment variable: <u>pathSegBeginExt</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>End Style</i>	Controls how the beginning wire segment ends are created. ■ <i>auto</i>: Automatically sets the end style of the wire. □ When the stranded wire is manhattan in shape, its <i>Begin Style</i> is set to <i>truncate</i>. □ Two orthogonal manhattan stranded wires will have their <i>Begin Style</i> and <i>End Style</i> set to <i>extend</i> in order to avoid notches. □ When the stranded wire is diagonal in shape, its <i>End Style</i> is set to <i>custom</i>. □ Two diagonal orthogonal stranded wires will have their <i>Begin Style</i> and <i>End Style</i> set to <i>custom</i> to avoid notches. ■ <i>truncate</i>: Creates a stranded wire with no begin extension. ■ <i>extend</i>: Creates a stranded wire with the default begin extension value. ■ <i>variable</i>: Lets you override the value specified in the <i>Begin</i> field for the begin extension of a wire. ■ <i>custom</i>: Constraints the vertices of the created wire to the current grid. The default value is <i>auto</i>. Environment variable: <u>pathSegEndStyle</u>
<i>End</i>	Specifies the value for the end extension of the stranded wire. This field is enabled when <i>End Style</i> is set to <i>variable</i>. Environment variable: <u>pathSegEndExt</u>
Creation Mode	The following options let you specify the snapping behavior for the stranded wire that you want to create.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Snap Mode</i>	Controls how the cursor snaps when you create the stranded wire. ■ <i>orthogonal</i>: Creates wires parallel to the X or Y axis. This is the default option. ■ <i>diagonal</i>: Generates a warning message informing the user that only orthogonal stranded wires are supported. The mode is automatically changed to <i>orthogonal</i>.



```
(LE-102008): OA exception Invalid object specified has occurred while updating wire shapes.  
(LE-102008): OA exception Invalid object specified has occurred while updating wire shapes.  
*WARNING* (WE-20029): The 'Create Wire' command supports only orthogonal stranded wires.  
Therefore, the snap mode has been changed to orthogonal.
```

Environment variable: [snapMode](#)

<i>Auto Terminate</i>	Finishes the <i>Create Stranded Wire</i> command as soon as a point is digitized on a flightline target object (pin or existing wire or via). When creating a stranded wire and <i>Auto Terminate</i> is on, all stranded wires snap to its target pin. This option is selected by default.

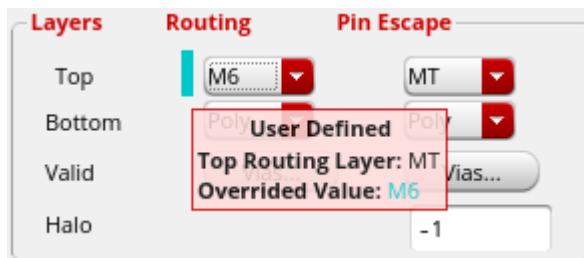
Related Topics

[Working with Stranded Wires](#)

Point to Point Form

The following table describes the fields available in the Point to Point form.

Field	Description
<i>Net Name</i>	Specifies a net name for the wire being created Environment variable: <u>netName</u>
<i>Probe Nets</i>	Creates probes for the selected or specified nets.
Wiring	This section lets you specify the width and layer for the wire that you want to create.
<i>Width</i>	Specifies the width of the wire to be created using point-to-point routing. Environment variable: <u>enableWidthOverride</u>
Layers	This section lets you specify the layer range.
<i>Routing</i>	Specifies the <i>Top</i> and <i>Bottom</i> layers to define the range of the valid routing layers. The specified layer range generally intersects the <i>Pin Escape</i> layer range. If you override the valid top or bottom layer, a color bar appears next to the layer cyclic field. Placing the cursor over the color bar displays the specified routing layer limit and the override information,



Pin Escape. Specifies the *Top* and *Bottom* layers to define the layer range. You can use this field to override the `validLayers` setting in the `virtuosoDefaultTaper` constraint group. If a `virtuosoDefaultTaper` constraint group does not already exist, it is created based on the specified *Layers*, *Vias*, and *Halo* values. If you override the valid top or bottom layers, a color bar appears next to the layer cyclic field. Placing the cursor over the color bar displays the specified pin escape layer limit and the override information.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Valid Vias</i>	Lets you select the vias to be used for creating and editing wires. Clicking the <i>Vias</i> button in the <i>Routing</i> column opens the <i>Net</i> tab in the Via Configuration form, and clicking the <i>Vias</i> button in the <i>Pin Escape</i> column opens the <i>Pin Escape</i> tab in the Vias Configuration form. The <i>Pin Escape</i> tab in the Vias Configuration form display the valid standard and custom vias from the application default constraint group.
<i>Halo</i>	Specifies the <code>taperHalo</code> value of the <code>virtuosoDefaultTaper</code> constraint group. You can override this value by specifying a positive float value.
Connection Type	
<i>Exact Location</i>	Uses the exact digitized points to create routing between two given points.
<i>Shortest Connection</i>	Uses the shortest route for connecting points and may also use pre-existing wires for creating the route. Environment variable: <code>p2pSeedStyle</code>

Related topics

[Point to Point Routing](#)

Select Via Form

The *Select Via* form consists of different layer stacks to show parallel connections. For example, layer `LiPO` can connect to `Poly`, `Metal1`, `Li2`. The displayed layer stacks and vias are defined in the active `validLayers` and `validVias` rule section in the technology file. Clicking the *Options* button in the Select Via form displays the *Select Via Options* form. The *Select Via Options* form enables the selection of a via `cutClass` and `viaDef`. The wire editor supports the use of the selected via `cutClass`. For more information about the cut class and the via definition values, see the [Create Via Options Form](#).

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

The following table describes the fields available in the Select Via form.

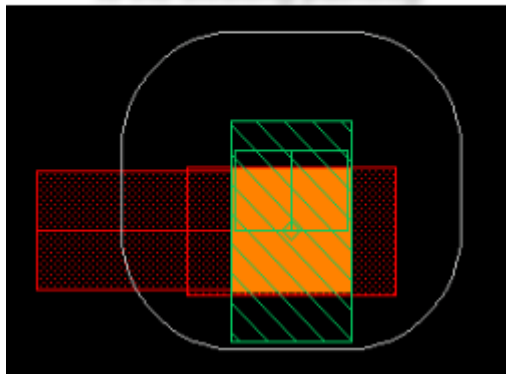
Field	Description
Layer Transitions	Lets you select a layer on which you want to create a via stack
<i>Rotate Via(s) Cut Pattern</i>	Rotates the via cuts and swaps the rows and columns, if any, while rotating the via shape by 90 degree as you place or drag the via. You can also rotate pending vias by clicking the middle mouse button. However, the top and bottom via shapes are not rotated. This option is available both in Create Stranded Wire Form and Create Bus Context-Sensitive Menu.
Via Alignment	Enables you to align vias as they are digitized during wire or bus creation. You can choose to align a via either automatically or by selecting a custom alignment option. Vias are aligned between the edges of via enclosure shapes and the edges of the connecting wires on the same layer. For example, for a Metal1 to Metal2 via, the via location is adjusted in a way that the Metal1 wire edge is aligned with the Metal1 via enclosure and the Metal2 wire edge is aligned with the Metal2 via enclosure. The alignment offsets the via origin to maintain it at the intersection of the wire end points. If you are adding and aligning a via stack, the same via alignment is applied to all the vias in the stack. You can also access the via alignment options from the Create Stranded Wire Form.

Virtuoso Interactive and Assisted Routing User Guide

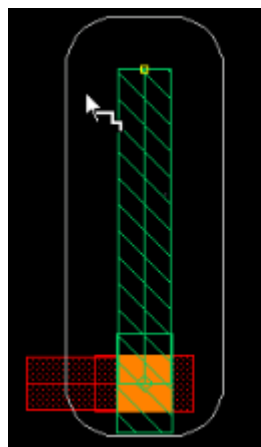
Interactive and Assisted Routing Forms

Field	Description
<i>Automatic</i>	Automatically aligns the digitized via with respect to both the existing and the new pathSeg. By default, this check box is not selected. After you select the check box, it remains selected for the remaining session while you digitize more vias. When the check box is selected, the default alignment of the next via is calculated automatically. When the check box is not selected, the default alignment of the next via is Center Center (see Override option setting). Automatic via alignment is not supported while routing a bus. The following figures illustrate the automatic via alignment in three different scenarios based on the directions in which the next pathSeg of the wire is added.

Initial via alignment with respect to the existing pathSeg



Change in via alignment with respect to the pathseg being added

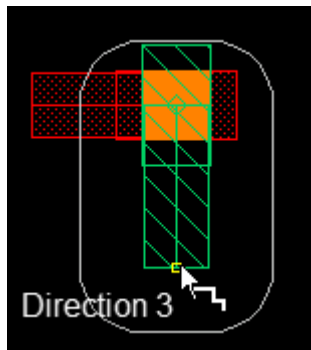
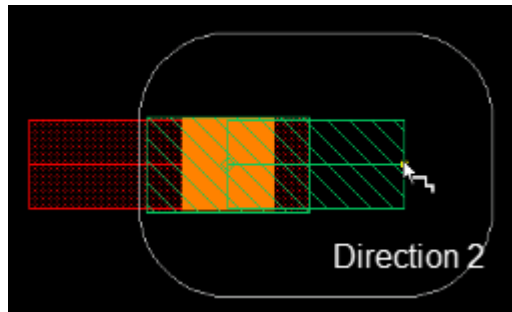


Direction 1

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description



For orthogonal transitions.

Controls alignment behavior for orthogonal transitions.

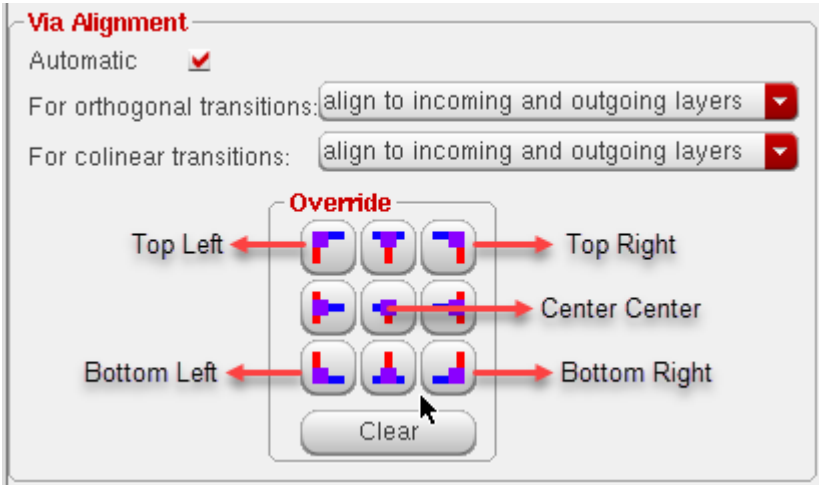
- **align to incoming and outgoing layers** aligns to the edges of both incoming and outgoing layers. This is the default option.
- **align to incoming layers** aligns to the edge of the incoming layer only.
- **align to outgoing layers** aligns to the edge of the outgoing layer only.
- **do not perform automatic alignment**, if selected no alignment is done for a via.

Environment variable: [autoViaAlignmentOrthoTransitionBehavior](#)

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
For collinear transitions.	Controls alignment behavior for colinear transitions. ■ align to incoming and outgoing layers aligns to the edges of both incoming and outgoing layers. This is the default option. ■ do not perform automatic alignment, if selected no alignment is done for a via. Environment variable: <code>autoViaAlignmentColinearTransitionBehavior</code>
Override.	Enables you to specify a custom via alignment option that overrides the automatic alignment. The <i>Override</i> buttons are available only if you click a target layer button. Each <i>Override</i> button carries the image of two pathSegs—blue and red—and a rectangular via. The two pathSeg colors indicate two layers. You move the wire you are creating from one layer to the other after digitizing a via. The image on the button represents the alignment of the via with respect to the two layers. Clicking a button displays a check mark on the button; the check mark is cleared when you digitize a via with the selected alignment setting. The Center button is available only if the <i>Automatic</i> check box is selected.



Clear.	Environment variable: <code>viaAlignment</code> Removes the check mark displayed on the selected <i>Override</i> button.

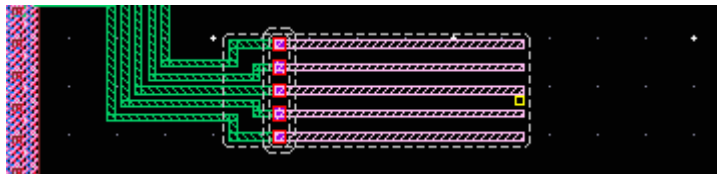
Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

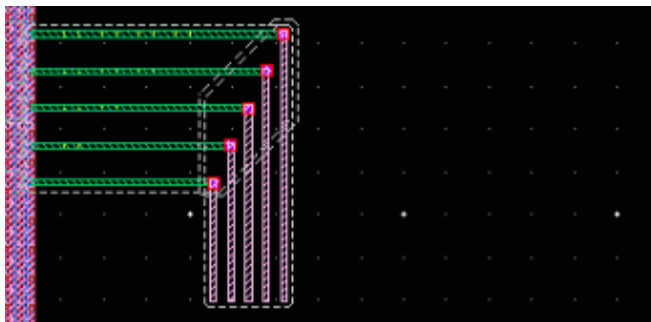
Field	Description
Via Pattern	Helps you specify a pre-defined manner in which multiple vias can be added simultaneously when routing multiple wires. Environment variable: viaPattern

Mode

- **Auto No Crossing:** Automatically places the vias in a diagonal pattern in such a way that bus wires do not cross when making orthogonal transitions. This pattern retains the bus bit order and might prevent the wire editor from creating jogs if *Via Alignment* is on.
- **Auto Crossing:** Automatically places the vias in a diagonal pattern in such a way that bus wires do cross when making orthogonal transitions. This pattern reverses the bus bit, which might be useful if the starting and targets pins are reversed.
- **Perpendicular:** Vias are created along a line perpendicular to the incoming direction of the wires.

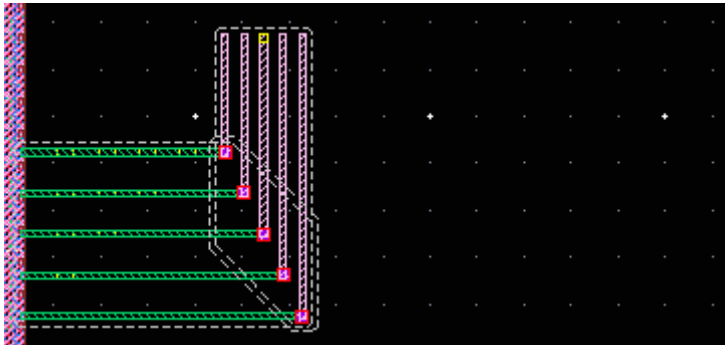
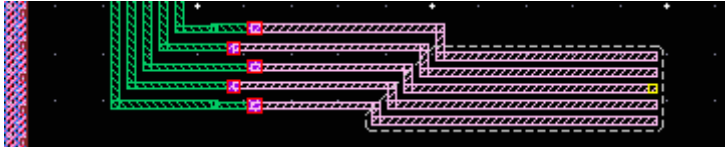
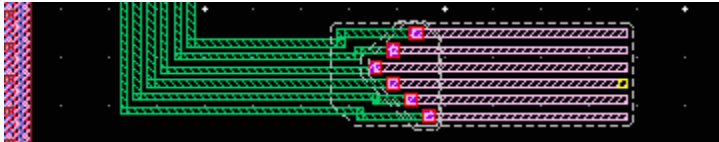
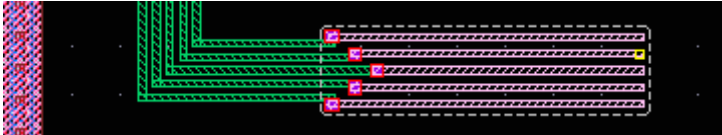


- **Diagonal Down:** The right-most or top bus is the longest segment to drop via. This pattern is most appropriate for right-hand turns.



Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
	Diagonal Up: The left-most and bottom bus is the longest segment to drop via. This pattern is most appropriate for left-hand turns. 
	Stagger 
	In Taper 
	Out Taper 

Via patterns and *Fan Out to Vias* support both orthogonal and diagonal wire directions.

Metal Density Options Form

The Metal Density Options form lets you specify the options for the metal density flow: checks, analysis, and fixing. The form contains the following tabs.

Tab	Description
<u>Layers</u>	Lets design engineers specify analysis type, aging options, start and stop time of reliability simulation, and options related to device masking, degradation ratio, and lifetime calculation.
<u>Slot Geometry</u>	Lets model developers specify the modeling type to be used during reliability simulation.
<u>Slotting Style</u>	Lets model developers specify options to print device and subcircuit degradation information into a .bt0 file.
<u>Slot Consistency</u>	Lets design engineers specify the degradation reports to be generated and methods to filter degradation results in these reports.
<u>Fillet</u>	Lets design engineers specify the degradation reports to be generated and methods to filter degradation results in these reports.

Layers

The following table describes the fields available on the *Layers* tab of the Metal Density Options form.

Field	Description
<i>Use Palette Layers</i>	Enables selection of palette layers for slotting. Environment variable: <u>usePaletteLayers</u>
<i>Material</i>	Lets you select the layer category to be processed. Once a specific material is selected, only the corresponding valid layers are displayed in the <i>Layers</i> list box. Environment variable: <u>material</u>
<i>Layers</i>	Lets you select the names of the layers that you want to process. You can select single or multiple layers from the list box by using the <i>Shift</i> or the <i>Ctrl</i> key. Environment variable: <u>metalDensityLayers</u>

Slot Geometry

The following table describes the fields available on the *Slot Geometry* tab of the Metal Density Options form.

Field	Description
<i>Length</i>	Specifies the length of the slot. Default is $-0.1\mu\text{m}$, which means that the parameter is not specified. Environment variable: <u>slotLength</u>
<i>Width</i>	Specifies the width of the slot. Default is $-0.1\mu\text{m}$, which means that the parameter is not specified. Environment variable: <u>slotWidth</u>
<i>Length Spacing</i>	Specifies the slot spacing along the length. Default is $-0.1\mu\text{m}$, which means that the parameter is not specified. Environment variable: <u>lSpacing</u>
<i>Width Spacing</i>	Specifies the slot spacing along the width. Default is $-0.1\mu\text{m}$, which means that the parameter is not specified. Environment variable: <u>wSpacing</u>
<i>Distance to Edge</i>	Specifies the minimum distance from the slot to the edge. Default is $-0.1\mu\text{m}$, which means that the parameter is not specified. Environment variable: <u>slotToEdge</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Specification Cell</i>	Specifies a cell name (specification cell). It is a cellview that contains, for each concerned layer, the representation of the directional slot shapes and the turn areas. The syntax for a specification cell is <code>libName/cellName/ViewName</code>. If specified, it overrides any entries from the other fields of the option form. All the geometric information is derived from the specification cell that is mentioned. The check box next to the <i>Specification Cell</i> field is used to enable or disable the field. In case a list of contexts is defined using the sltSetContextSpecificationCells SKILL function, instead of the string field for the <i>Specification Cell</i>, the <i>Context</i> field is available, which lists the defined contexts. For more information, see Specification Cells. Environment variables: slotSpecCell, enableSlotSpecCell, useOneSlotSpecCell, and slottingContext

Slotting Style

The following table describes the fields available on the *Slotting Style* tab of the Metal Density Options form.

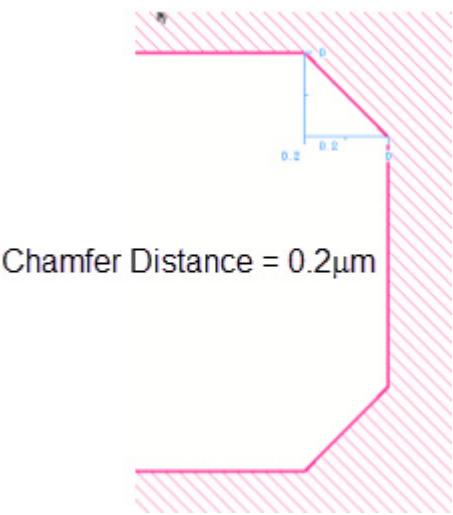
Field	Description
<i>Width Threshold</i>	Specifies the width to filter out the shapes with a width smaller than the one specified. Default is $-0.1\mu\text{m}$, which means that the parameter is not specified. Environment variable: widthThreshold
<i>Length Width Ratio</i>	Specifies a threshold ratio value that determines whether a slot must be created as a square or a rectangular shape. If the length-to-width ratio of the object is greater than the defined threshold, then a rectangular slot is created. If the length-to-width ratio of the object is less than or equal to the specified ratio, then a square slot is created. The default is $-0.1\mu\text{m}$, which means that the parameter is not specified. Environment variable: lengthWidthRatio

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Stagger</i>	Enables you to define the offset mode used to add slots. If the option is deselected, slots are in line. However, if the option is selected, slots are staggered. By default the option is deselected. Environment variable: <u>slotStaggered</u>
<i>Slot Via</i>	Enables slotting of vias. By default, this option is selected. Environment variable: <u>slotVia</u>
<i>Exclude Pin</i>	Lets you exclude pin objects from being slotted. By default the option is deselected. Environment variable: <u>excludePins</u>
<i>Density Aware</i>	Lets you only slot the shapes that violate the maximum density rules. The prerequisite is to make these rules available in the technology library file. Additionally, if any of the parameters, such as <i>Length</i>, <i>Width</i>, <i>Length Spacing</i>, <i>Width Spacing</i>, or <i>Distance to Edge</i> is specified as 0, the appropriate value to closely match the target density is automatically computed. Environment variable: <u>densityAwareMode</u>

Field	Description
<i>Directional Pattern</i>	Defines a slot pattern that is dropped in the direction of the current over the orthogonal portion of the shape. The direction is given by the from and to points of the wire. For other shapes, the direction is given by the longest dimension of the bounding box, the orthogonal portion. You can specify one of the following slot patterns: ■ Rectangle this is the default value. ■ Points when selected, the text field next to it is enabled. In this field, you can specify a list of coordinates for each corner of the required polygon. ■ Chamfer when selected, the text field next to it is enabled. In this field, you can specify the chamfer value for each corner of the rectangles (or squares) defined by <code>slotWidth</code> and <code>slotLength</code>.



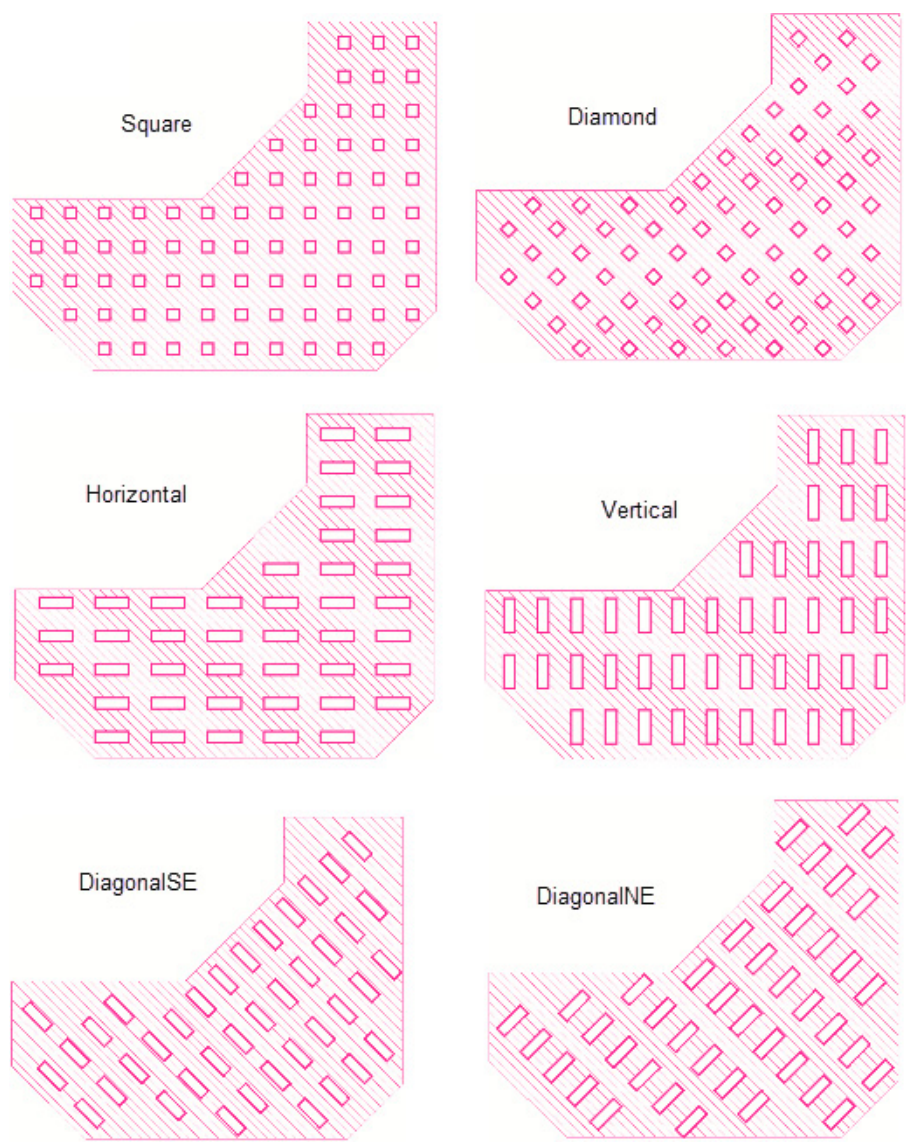
- **Radial** when selected, the text field next to it is enabled. In this field, you can specify the radius value to round each corner of the rectangles (or squares) defined by `slotWidth` and `slotLength`.

Environment variable: [directionalSlotPattern](#)

Pattern at Turn Lets you define a slot pattern at the turn of a wire by defining a list of points. You can select from one of the following slot patterns: *Square*, *Points*, *Chamfer*, and *Radial*.

Environment variable: [turnSlotPattern](#)

Field	Description
Slots Orientation	lets you define a slot orientation for the shapes to be slotted. You can select one of the following orientations for the slots: <i>Default</i> , <i>Square</i> , <i>Diamond</i> , <i>Horizontal</i> , <i>Vertical</i> , <i>diagonalNE</i> , and <i>diagonalSE</i> . If the slot orientation is other than default, then the slotting on an object respects the specified orientation and overrides the directional pattern as well as the pattern at turn. The following figure shows the various slot orientation patterns.



Environment variable: slotOrientationMode

Slot Consistency

The following table describes the fields available on the *Slot Consistency* tab of the Metal Density Options form.

Field	Description
<i>Slot Out of Group</i>	Reports the slot shapes that are not part of a slotting fig group. Slot shapes are added to slotting fig groups when they are created using the Slot Consistency form or the lxShapeSlotting function.
<i>Slot Over Blockage</i>	Reports the slot shapes that overlap on the same layer and the blockage type is <code>slot</code> .
<i>Slot Over Instance</i>	Reports the slot shapes that overlap an instance of a cell.
<i>Floating Slot</i>	Reports the slot shapes that do not overlap a metal shape and are not connected.

Fillet

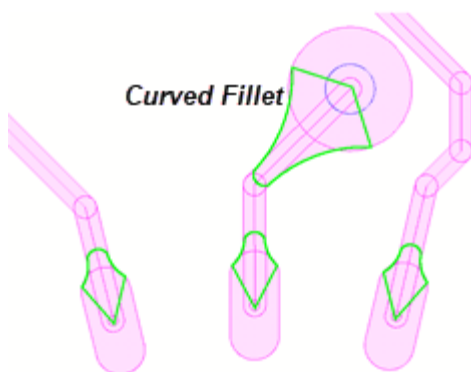
The *Fillet* tab lets you create and control where a fillet shape is inserted. You can select options to filter out shapes and objects and specify the parameters for fillet creation. This tab is only available for Virtuoso RF Solution. The license required for Virtuoso RF is 95560 Virtuoso_RF_Option.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

The following table describes the fields available on the *Fillet* tab of the Metal Density Options form.

Field	Description
<i>Curved Fillet</i>	ets you create a fillet using an arc instead of a line as part of the shape outline from the center of the shape to the intersection of the shape. By default this option is deselected.



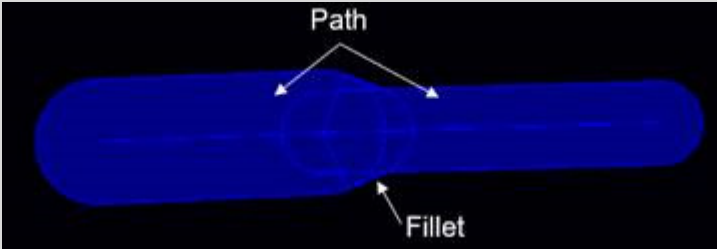
<i>Unassigned Shapes</i>	Lets you filter the shapes that do not have any net connectivity. By default this option is deselected.

Parameters

<i>Desired Angle</i>	The target angle of the fillet shape. The default is 90 degrees. In some cases, the value can be increased to maximum angle to get DRC correct results or optimal fillet shape to ensure a fully covered object edge.
<i>Max Angle</i>	Specifies the maximum limit for the fillet. The default is 90 degrees. This value must always be equal to or greater than the <i>Desired Angle</i> .
<i>Max Offset</i>	Specifies the maximum distance between the intersection of the shape edge and the connecting line forming the fillet length. The default is 150 μm . A larger <i>Desired Angle</i> and a smaller <i>Max Offset</i> creates a short fillet. A smaller <i>Desired Angle</i> and larger <i>Max Offset</i> creates a long fillet.
<i>Min Line Width</i>	Specifies the minimum width of the wire entering the shape. If the width of the wire is less than the specified value, the fillet is not created. The default is 1 μm .
<i>Max Line Width</i>	Specifies the maximum width of the wire entering the shape. If the width of the wire is greater than the specified value, the fillet is not created. The default is 635 μm .

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
Objects	Lets you select the objects for which a fillet is to be created. The valid object types are <i>Paths</i> , <i>Bond Fingers</i> , <i>Vias</i> , <i>Pins</i> , <i>Pad without drills</i> , and <i>T connections</i> . By default, <i>Paths</i> , <i>Vias</i> , <i>Pins</i> , and <i>T connections</i> are selected. The <i>Bond Fingers</i> and <i>Pad without drills</i> objects are deselected.
Shapes	Lets you choose options for the fillet shapes: <i>Circle</i> , <i>Square</i> , <i>Rectangle</i> , and <i>Oblong</i> .
Max Size	Specifies the maximum size of the fillet, which is the larger dimension of the width or length of the shape's bounding box. The default is 2540 μm . These shapes are filtered out based on the specified maximum size. If the size of the shape is greater than the maximum specified size, then it is not considered as a valid object for fillet insertion.
Tapered Traces	Tapered topology is a topology in which the wire width is not the same, as shown in the following figure. 
Tapered traces	Creates a fillet on the tapered traces. This option is selected by default.
Min Segment angle	Specifies the minimum segment angle. The default is 135 μm .
Desired angle	Is the value of the tapered angle to be created by the fillet. The default is 60 degrees.
Max offset	Is the maximum distance the fillet may extend from vertex to the point of cline width transition. The default is 635 μm .

Slotting Button

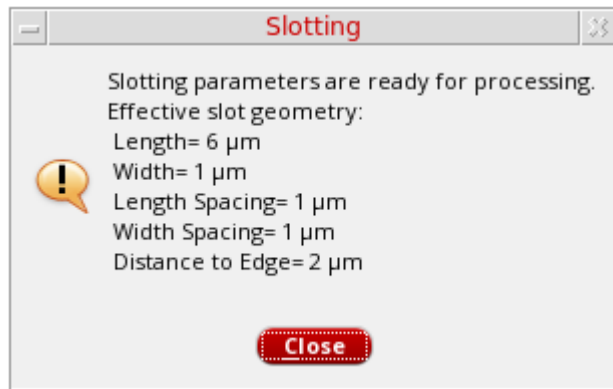
A status button in the Metal Density Options form. The *Slotting* button has three states: Green, Yellow, and Red.



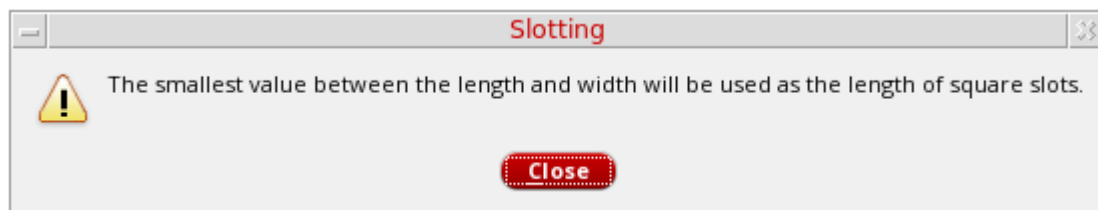
Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

The green status indicates that slotting can be done. When you click the *Slotting* button with the green status, a Slotting message box appears mentioning the effective slot geometry parameters, as shown below.



The yellow status indicates that slotting can be done, but there is an undesired parameter, or value of a parameter. When you click the *Slotting* button with the yellow status, a warning message appears mentioning the undesired parameter, or the value of the parameter, as shown below.



The red status indicates that slotting cannot be done and parameters are missing. When you click the *Slotting* button with the red status, a message box appears mentioning the rationale behind the red status, as shown below.



Related Topics

[Managing Metal Density](#)

Create Packaging Bus Form

Use the Create Packaging Bus form to set the parameters for creating a curved path. The following table describes the fields available on the Create Packaging Bus form.

Field	Description
<i>Connectivity</i>	This section lets you specify the nets for which you want to create a bus and optionally lets you create a probe for the specified net.
<i>Net Name</i>	Specifies the net names for creating a bus. You can specify the net names separated by commas or spaces or specify them in bus notation, such as net<0:3>. New nets are created if the ones specified do not already exist. If you start creating a bus from existing pins, the net names are automatically picked from the net of the pins. Environment variable: netName
<i>Probe Nets</i>	Creates probes for the autopicked, selected, or specified nets.
	This section lets you specify the width, direction, and style for the wire that you want to create.
<i>All Nets Width</i>	Specifies the width of each wire in the bus. You can increase or decrease the wire width by pressing Ctrl + Shift keys and scrolling the mouse wheel up or down, respectively. Environment variable: wireWidth
<i>All Nets Bit Spacing</i>	Specifies the distance between the centerlines of the bus wires. If no value is specified in this field, the <code>minSpacing</code> value is considered.
<i>Fan Out to Vias</i>	Spreads bus wires for via placement. When wires are at the minimum wire spacing but the via spacings are larger, the wires can either be fanned out to connect to the vias in the pattern or can be back tracked to intersect the previous segment in the wire. Environment variable: fanOutToVias

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Number of Bits</i>	Enables you to specify the number of wires in an unassigned bus. The default value is 2. Environment variable: weBusNumBits
	The following options let you specify the snapping behavior for the bus that you want to create.
<i>Snap Mode</i>	Controls how the cursor snaps when you create a bus. ■ <i>diagonal</i>: Creates wires parallel to the X or Y axis or at a 45-degree angle to the axes. ■ <i>orthogonal</i>: Creates wires parallel to the X or Y axis. Environment variable: snapMode
<i>Snap to Pin Center</i>	Snaps the bus wires to the center of rectangular pins or of pins that are created as polygons and have rectangular shapes. When this check box is selected, bus wires snap to the starting and ending pins centers. By default, the <i>Snap to Pin Center</i> check box is not selected. Environment variable: snapToPin
<i>Snap to Via Center</i>	Snaps wires to the center of vias. If this check box is selected and you click a via, the starting wire starts from the clicked point and two small segments are added to connect to the via origin. With this check box selected, tapping a via and a wire gives preference to the via. This check box is off by default. Environment variable: snapToViaCenter
<i>Auto Terminate</i>	Finishes the <i>Create Bus</i> command as soon as a point is digitized on a flightline target object (pin, existing wire, or via). All wires in the bus snap to the target pin. This option is selected by default.

Related Topics

[Creating a Packaging Bus](#)

[Area-based Rules in Create Packaging Bus](#)

[Snapping Behavior of Curved Paths Using Create Packaging Bus](#)

Copy Route Form

The Copy Route form lets you specify the options for running copy route. The form contains the following tabs.

Tab	Description
<u>General</u>	Lets you specify the general settings for copy route.
<u>Options</u>	Lets you specify less frequently used options for copy route.

General

The following table describes the fields available on the *General* tab of the Copy Route form.

Field	Description
Target	Describes the selected reference wire state. The target wires have the following modes. ■ <i>No reference wire</i>: the reference wire is not yet selected. ■ <i>Connected to Pins</i>: the reference wire is either pin-to-space or pin-to-pin. This is the default option. You can switch to <i>In Space</i> mode and the result would be generated from space-to-space. ■ <i>In Space</i>: the reference wire is space-to-space.
<i>Number of Copies</i>	Defines the number of copies when the first and last transitions of route copies are not connected to pins. Environment variable: <u>copyRouteNumCopiesInSpace</u>
Spacing	
<i>Spacing</i>	Defines the copy spacing mode for the intermediate sections. ■ <i>Default Constraint</i>: uses the <code>minSpacing</code> defined in technology file. This is the default option. ■ <i>Override All Layers</i>: uses the value specified for the <code>copyRouteSpacingOverrideAllLayers</code>. ■ <i>Keep Pin Space</i>: follows the pin spacing for copy route. Environment variable: <u>copyRouteSpacingMode</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Spacing Override Value</i>	Defines the spacing value to be used for copy route. This field is enabled only when <i>Override All Layers</i> option is selected. Environment variable: <u>copyRouteSpacingOverrideAllLayers</u>
<i>Spacing At ends</i>	Enables spacing at the first and last sections. Environment variable: <u>copyRouteEnableSpacingAtEnds</u>
<i>Mode</i>	Specifies the spacing at the first and last sections that are not connected to pins. ■ <i>Default Constraint</i>: uses the <code>minSpacing</code> defined in technology file. This is the default option. ■ <i>Override All Layers</i>: uses the value specified for the <code>copyRouteSpacingOverrideAllLayersAtEnds</code>. Environment variable: <u>copyRouteSpacingModeAtEnds</u>
<i>Spacing Override Value at Ends</i>	Lets you define spacing value to be used at the first and last sections that are not connected to pins for copy route. This field is enabled only when <i>Override All Layers</i> option is selected. Environment variable: <u>copyRouteSpacingOverrideAllLayersAtEnds</u>
Width	
<i>Width</i>	Defines the copy width mode for intermediate sections. Intermediate sections are the sections that are not connected to pins. ■ <i>Same As Reference</i>: the copy width is exactly the same as the reference width. This is the default option. ■ <i>Default Constraint</i>: uses the <code>minWidth</code> that is defined in the technology file. ■ <i>Override All Layers</i>: uses the value defined for the <code>copyRouteWidthOverrideAllLayers</code>. ■ <i>Keep Pin Width</i>: uses the pin width if the wire is starting from a pin. Environment variable: <u>copyRouteWidthMode</u>
<i>Width Override Value</i>	Defines the width to be used for copy route. This field is enabled only when <i>Override All Layers</i> option is selected. Environment variable: <u>copyRouteSpacingOverrideAllLayers</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Width At Ends</i>	Enables the width of the first and last sections of the copied wire. Environment variable: <code>copyRouteEnableWidthAtEnds</code>
<i>Mode</i>	Defines the copy width mode for first and last sections. ■ <i>Same As Reference</i>: the width of the copies is exactly the same as the reference width. This is the default option. ■ <i>Same As Reference or Pin</i>: if the reference wire matches the pin width, then copies also match that pin width. Else the reference width is used. ■ <i>From Tech Only</i>: the copies use the <code>minWidth</code> value that is defined in the technology file. ■ <i>From Tech Or Pin</i>: if the reference wire matches the pin width, then copies also match that pin width. Else, the <code>minWidth</code> defined in the technology file is used. ■ <i>From User Only</i>: the copies use the value defined for <code>copyRouteWidthOverrideAllLayersAtEnds</code>. ■ <i>From User Or Pin</i>: if the reference wire matches the pin width, then copies also match that pin width. Else, the copies use the value defined for <code>copyRouteWidthOverrideAllLayersAtEnds</code>. ■ <i>Same As Pin</i>: the copies use the same width value as that of the pin. Environment variable: <code>copyRouteWidthModeAtEnds</code>
<i>Width Override Value at Ends</i>	Lets you define the width value to be used at the first and last sections. This field is enabled only when the <i>From User Only</i> or <i>From User Or Pin</i> option from the <i>Mode</i> drop-down list is selected. Environment variable: <code>copyRouteWidthOverrideAllLayersAtEnds</code>
Vias	
<i>Preserve Cut Classes</i>	Generates copies with the same cut class as that of the reference wire. Environment variable: <code>copyRoutePreserveCutClasses</code>
<i>Options</i>	Displays the Select Via Options form. This button is enabled only when the <i>Preserve Cut Classes</i> option is deselected.
Layers	

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Field	Description
<i>Layers</i>	Specifies the layer mode for copy route. ■ <i>SameAs Reference</i>: This is a single layer mode whose layer is different from the reference pin. This is the default option. ■ <i>Are Based on Pins</i>: Similar to <i>Follow Reference Layer Pattern Except Pins</i>, but the intermediate section layer is adjusted based on the pin layer difference. ■ <i>Are Based on Pattern</i>: specifies the user-defined pattern. Environment variable: <u>copyRouteLayersMode</u>
<i>Pattern</i>	Specifies the layer pattern to be followed for copy route. The layer pattern is a list of integers. Environment variable: <u>copyRouteLayerPattern</u>
<i>Top/Bottom Layers</i>	Enables the top and bottom layers check. The layer for route copies should be in the range of specified top and bottom layer. Else, it is a violation. Environment variable: <u>copyRouteEnableValidLayer</u>
<i>Top Layer</i>	Specifies the top layer for copy route. Environment variable: <u>copyRouteTopLayer</u>
<i>Bottom Layer</i>	Specifies the bottom layer for copy route. Environment variable: <u>copyRouteBottomLayer</u>
<i>Routing Channel</i>	Enables and lets you specify the routing channel layer. Environment variable: <u>copyRouteEnableRoutingChannel</u>, <u>copyRouteRoutingChannelLPP</u>

Options

The following table describes the fields available on the *Options* tab of the Copy Route form.

Field	Description
<i>Generate Copy With Errors</i>	Selects one of the following options for the copy route results with violations: ■ <i>All</i>: generates the result no matter how many violations are present. This is the default option. ■ <i>None</i>: generates the result without any violations. The copy route is stopped as soon as there is a single violation. ■ <i>First Only</i>: generates the result with any violation. The copy route is stopped as soon as there is a violation, but the first copy route with violations is kept. Environment variable: <u>copyRouteGenerateCopyWithErrors</u>
<i>Group Copies</i>	Creates the route copies in a fig group. Environment variable: <u>copyRouteCreateFigGroup</u>
<i>Allow copies Without Matching Pins</i>	Selects and creates copies of the unpaired net pin. Environment variable: <u>copyRouteAllowCopiesWithoutMatchingPins</u>
<i>Remove Last dangling section</i>	Removes the last dangling section of copy route. This option is enabled only when <i>Allow copies Without Matching Pins</i> is selected. Environment variable: <u>copyRouteTrimmingFixer</u>

Related Topics

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Variables to Customize the Settings for Copy Route](#)

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Forms

Interactive and Assisted Routing Shortcut Menu Commands

The following table lists the basic editing shortcut menu commands that are available for interactive and assisted routing.

Create Bus Shortcut Menu Commands	Create Single Wire Shortcut Menu Commands
Create Stranded Wire Shortcut Menu Commands	Edit Bus Shortcut Menu Commands
Point to Point Shortcut Menu Commands	Reshape Wire Shortcut Menu Commands
Split Wire Shortcut Menu Commands	Stretch Shortcut Menu Commands

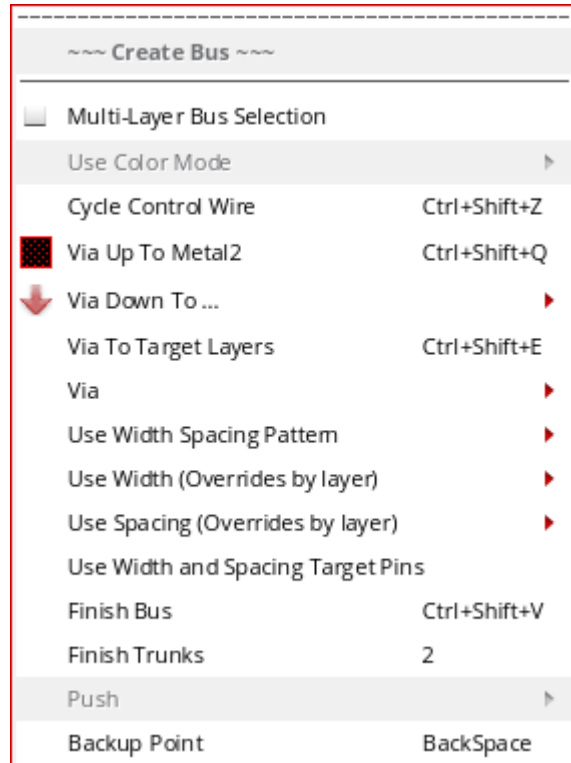
Related Topics

[Routing Assistant - Interactive Mode](#)

[Interactive and Assisted Routing Forms](#)

Create Bus Shortcut Menu Commands

The *Create Bus* shortcut menu commands appear on right-clicking in the design display area when the *Create Bus* command is running and a bus is being routed. The following commands are available for Create Bus.



Field	Description
<i>Multi-Layer Bus Selection</i>	Provides support to create and digitize the bits of a bus on different metal layers. For more information, see Supporting Multi-Layer Bus .
<i>Use Color Mode</i>	Controls the color mode of the bus. For more information, see Specifying the Bus Color Mode
<i>Cycle Control Wire</i>	Cycles the control wire function among the extreme and middle wires of a bus. For more information, see Using the Control Wire .

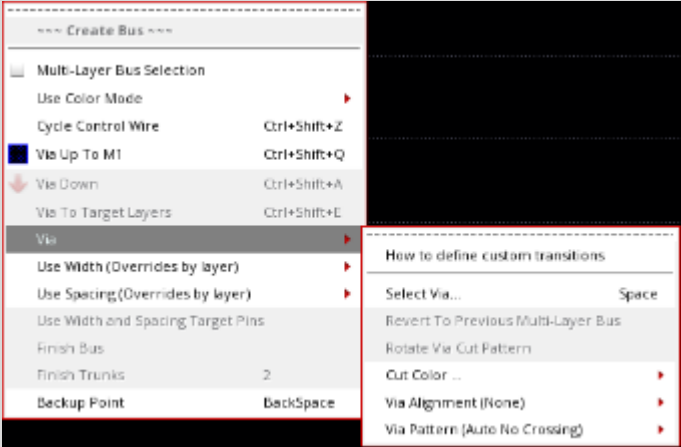
Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Via Up To</i> <layer_name>	Changes the layer to the next layer up and places the corresponding default via. A layer can now connect to several layers in the layers stack: ■ When current layer connects to two or more upper layers in the layer stack, a slider is created with the connected layers. ■ When current layer connects to a single upper layer in the layers stack, the layer name appears in the <i>Create Bus</i> Context Sensitive menu. ■ When current layer is unable to connect to any upper layer in the layers stack, the <i>Via Up</i> field appears disabled in the Create Bus Context Sensitive menu.
<i>Via Down To</i> <layer_name>	Changes the layer to the next layer down and places the corresponding default via. A layer can now connect to several layers in the layers stack: ■ When current layer connects to two or more lower layers in the layer stack, a slider is created with the connected layers. ■ When current layer connects to a single lower layer in the layers stack, the layer name appears in the <i>Create Bus</i> Context Sensitive menu. ■ When current layer is unable to connect to any lower layer in the layers stack, the <i>Via Up</i> field appears disabled in the Create Bus Context Sensitive menu.
<i>Via To Target Layers</i>	Adds via stacks to the layer of the target objects. The target objects are all the objects that are pointed to by the flightlines when the <i>Create Bus</i> command is run. However, if the flightline option in the Wire Assistant is off, then vias are not added. If a bus bit is already on the same layer as its respective target object, then a via is not added and a message is displayed in CIW as follows: "WARNING* (LE-105035): Already on specified layer."

Virtuoso Interactive and Assisted Routing User Guide

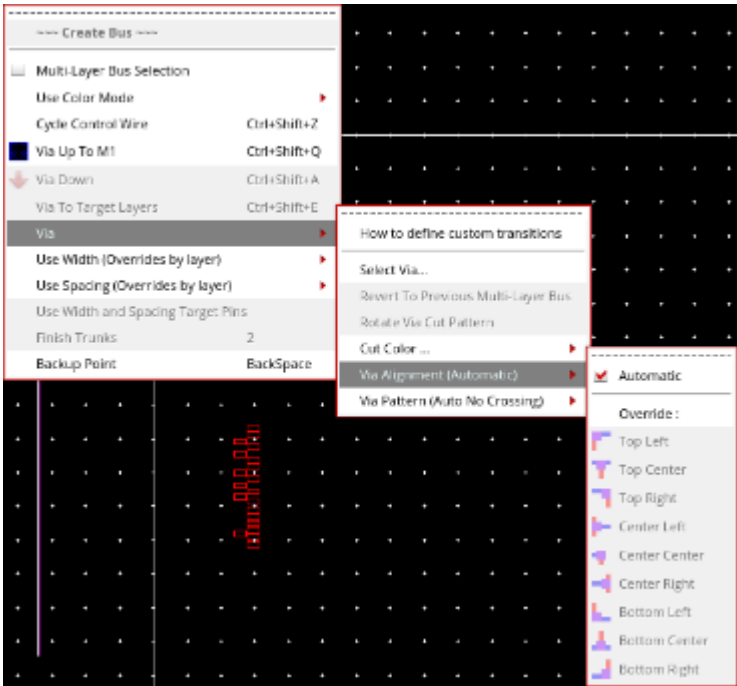
Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Via</i>	Let's you select the via options.  The screenshot shows the 'Create Bus' menu with the 'Via' option selected. The 'Via' submenu is open, displaying options such as 'Use Width (Overrides by layer)', 'Use Spacing (Overrides by layer)', 'Use Width and Spacing Target Pins', 'Finish Bus', 'Finish Trunks', and 'Backup Point'. The 'Via' option is highlighted with a red box.
<i>How to define custom transitions</i>	Lets you define customized SKILL procedures to create complex multi-layer bus transition patterns. For more information, see Customizing Multi-Layer Bus Transition.
<i>Select Via</i>	Opens the Select Via form. Use the Select Via form to change the bus layer. You can rotate the via by using the <i>Select Via</i> form or by using the <i>Rotate Via Cut Pattern</i> command on this shortcut menu.
<i>Revert To Multi-Layer Bus</i>	Enables you to switch to a multi-layer bus configuration from the bus bits on the same metal layer. For more information, see Switching Bus Bits to the Same Layer.
<i>Rotate Via Cut Pattern</i>	Rotates the via cuts and swaps the rows and columns, if any, while rotating the via shape by 90 degrees as you place or drag the via. You can also rotate pending vias by clicking the middle mouse button. This command is available only while you are adding a via. You can also rotate a via by using the <i>Rotate Via(s) Cut Pattern</i> button on the Select Via Form.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

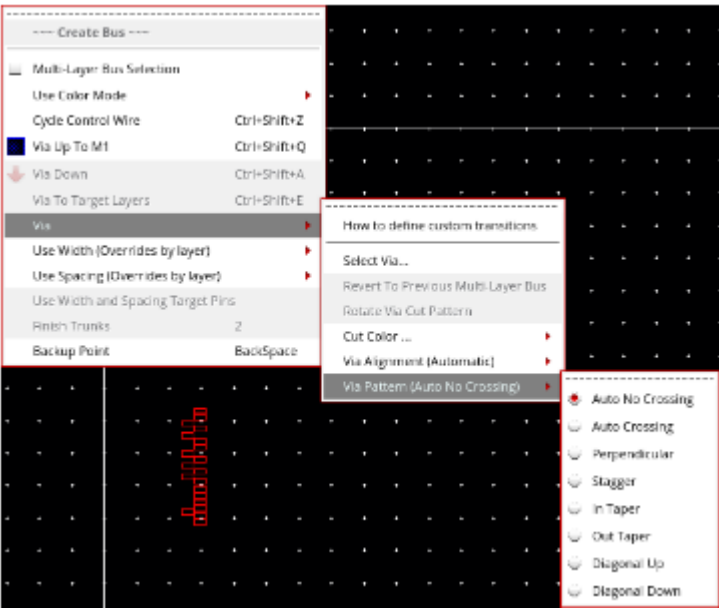
Field	Description
<i>Cut Color</i>	Enables cut coloring and cycles the color of each cut when a via is inserted. The color of the cut is based on information set by the <code>mptSetLayerDefaultColor</code> SKILL API. The <i>Cycle</i> option in the submenu is enabled when a via is being dragged. To use cut coloring patterns applicable for the <i>Create Bus</i> command, see the <code>weWACmdCalculatedColor</code> environment variable.
<i>Via Alignment</i>	Provides options to align vias while adding them to the layout. You can choose to align a via automatically or by selecting a custom alignment option. You can also align a via by using the <i>Via Alignment</i> options on the Select Via form. The Override options on the Via Alignment shortcut command are available while you are adding a via.



Virtuoso Interactive and Assisted Routing User Guide

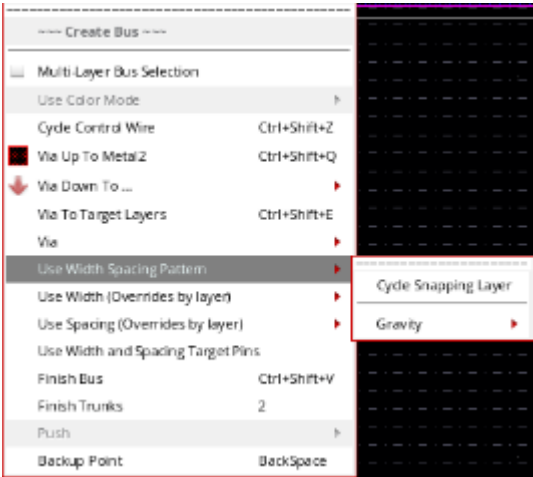
Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
Via Pattern	Provides predefined via patterns to choose from to add multiple vias simultaneously while you are routing multiple wires. For more information about the available patterns, see the Select Via Form .



Use Width Spacing Pattern	Use Width Spacing Pattern has been introduced in the <i>Create Bus</i> context-sensitive menu to work with various width spacing patterns.

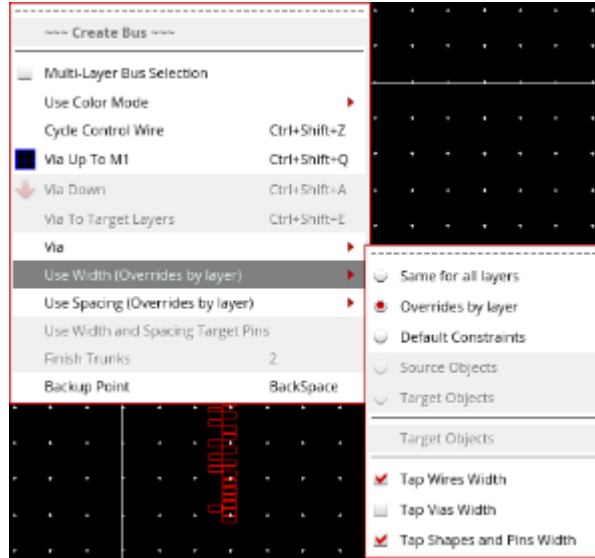
For more information on this, refer to [Working with Width Spacing Patterns During Wire Editing](#).



Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Use Width (Overrides by Layer)</i>	The following options are available to specify the bus width.



- **Same for all layers:** Specifies to use the same bus wire width for all layers. The width value is determined by the value specified in the *Width* field of the Create Wire form.
- **Overrides by layer:** Sets the bus wire width the same as the result of the constraint lookup for `minWidth` in the technology file when the layer is not used. Else, the width is determined by the last width used for the layer. The width value from the Wire Assistant table can also be used for the bus wires on each layer. You can also use the value specified in the *Width* field for the bus wires on each layer.
- **Default Constraints:** Sets the bus wire width same as the result of the constraint lookup for `minWidth` value in the technology file or in any applicable constraint group defined at the design or object level.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
	■ <i>Source Objects</i>: Specified to use the width determined by the width of the tapped object. This is the default option. The source object can be pins or wires. Selecting the <i>Source Objects</i> option for the first segment of a bus wire ensures that <i>Tap Wires Width</i> or <i>Tap Pins Width</i> is selected, depending on the source object. You can also select <i>Source Objects</i> even if the first segments of the wires does not have the source object width. In such a case, the <i>Tap Wires Width</i> and <i>Tap Pins Width</i> option does not change. ■ <i>Target Objects</i>: Sets each bus wire to match the width of the same edge of each respective target pin. If the pins are vertically aligned to bus wires, the bus wire widths match the width of the vertical edges of the respective target pins. Similarly, if the pins are horizontally aligned to bus wires, the bus wire widths match the width of the horizontal edges of the respective target pins. The horizontal and vertical edges might correspond to narrow edges for some target pins and wide edges for other target pins. This option cannot be selected and you need to use the <i>Target Pin Narrow Edge</i> or <i>Target Pin Wide Edge</i> options depending on the edge whose width you want. The narrow and wide edges for target pins is only applicable in case of single wire creation. ■ <i>Tap Wires Width</i>: When selected, enables tapping the width on the starting object for new wire when the starting object is a wire. The <i>Tap Wires Width</i> option is disabled when the <i>Auto Tap Wire</i> option is deselected in the Layout Editor Options form. To enable the <i>Tap Wires Width</i> option in the <i>Use Width</i> menu, select the <i>Auto Tap Wire</i> option in the Layout Editor Options form. ■ <i>Tap Vias Width</i>: When selected, enables tapping the width on the starting object for new wire when the starting object is a via. The edge of the via determines the width of the started wire.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description

- *Tap Shapes and Pins Width*: When selected, automatically taps a top-level shape, a top-level pin, or a level-1 instance layer and width attributes. The objects type is either a rectangle, a polygon, or a path. All valid routing layers can be tapped. The *Tap Shapes and Pins Width* option is disabled when the *Auto Tap Wire* option is deselected in the Layout Editor Options form. To enable the *Tap Shapes and Pins Width* option in the *Use Width* menu, select the *Auto Tap Wire* option in the Layout Editor Options form.

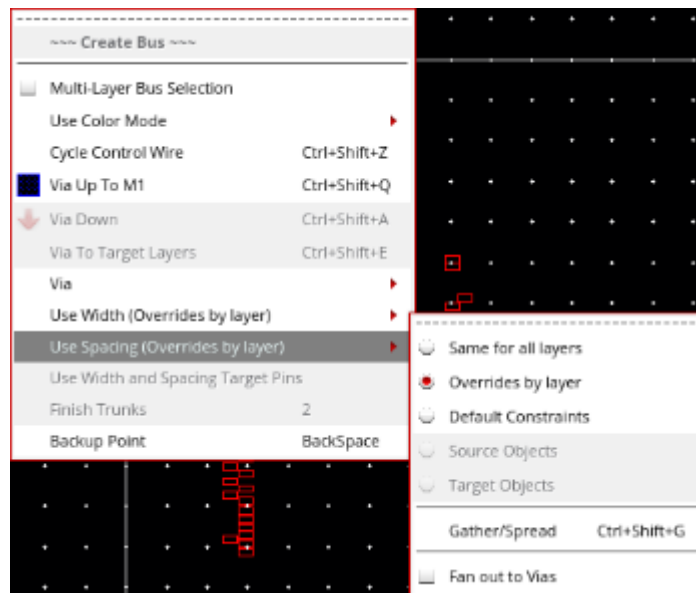
Environment variables: weTapWidthOnPins, weTapWidthOnWires

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description

<i>Use Spacing (Overrides by Layer)</i>	The following options are available to specify the spacing between the bus bits.



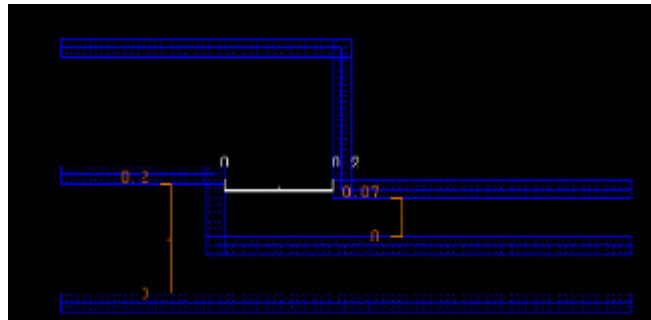
- **Same for all layers:** Specifies to use the same bus wire spacing on all the layers. The value specified in the *Bit Spacing* field is used for the bus wires on all layers. The constraint overrides for all layers are available in the Wire Assistant table.
- **Overrides by layer:** Sets the bus wire spacing same as the spacing value from the Wire Assistant table on each layer. By default, the result of the constraint lookup for `minSpacing` is used. You can also use the value specified in the *Bit Spacing* field for the bus wires on each layer.
- **Default Constraints:** Sets the bus wire spacing the same as the result of the constraint lookup for `minSpacing` in the technology file or in any applicable constraint group defined at the design or object level.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
	■ <i>Source Objects</i>: This is the default option when starting from pins or wires and not when starting from space. The spacing of the wires is adjusted to keep the centerlines of the wires aligned with the pins center. ■ <i>Target Objects</i>: Sets the same spacing between the centerlines of bus wires as the spacing between the centers of the target pins. You can also set the spacing between the pins which belong to the same instance or different instances. ■ <i>Gather/Spread</i>: Gathers or spreads bus wires before the next digitized point. It toggles to the <i>Cancel Gather/Spread</i> command. When spreading or gathering a bus, you can control the spacing of a bus.

The following figure shows the spacing in a bus while gathering or spreading a bus.

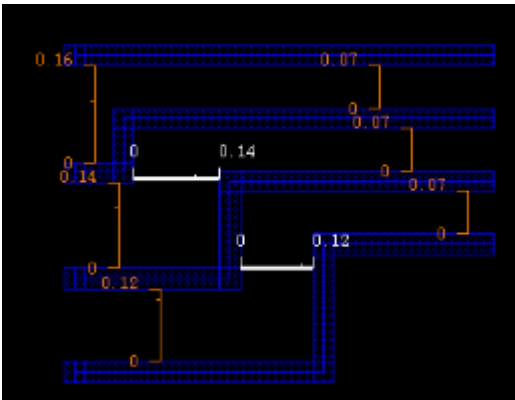


The following figure displays the spacing on the middle segment when the `gatherAndSpreadSpacing` environment variable is set to `largest`.

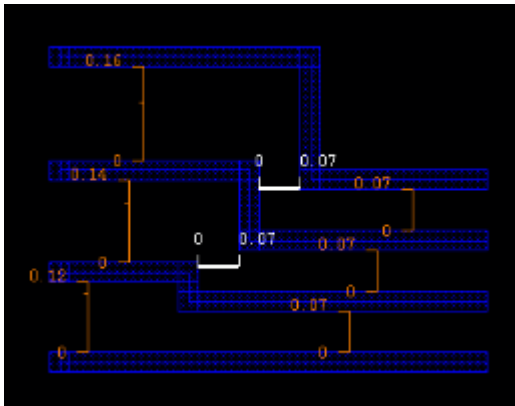
Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description



The following figure displays the spacing on the middle segment when the `gatherAndSpreadSpacing` environment variable is set to `smallest`.

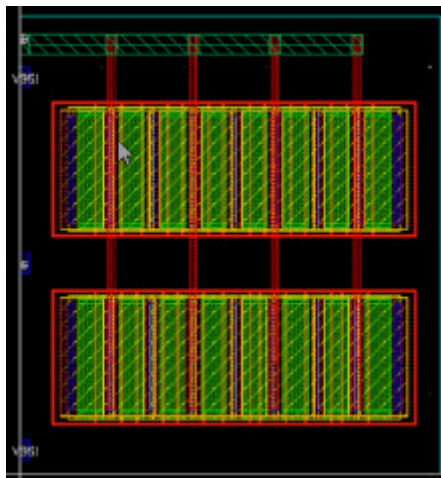


Environment variable: `gatherAndSpreadSpacing`

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

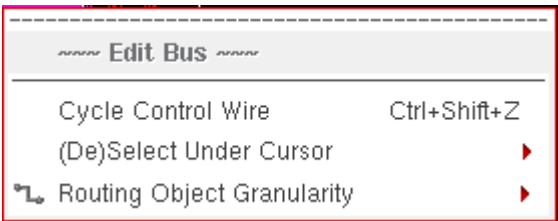
Field	Description
<i>Fan Out To Vias</i>	Spreads bus wires for via placement. When wires are at the minimum wire spacing but the via spacings are larger, the wires can either be fanned out to connect to the vias in the pattern or can be back tracked to intersect the previous segment in the wire. When the Fan Out to Vias check box is selected, jogs are created near the vias to respect the minimum spacing between the vias and wires. Else, the via spacing that respect the minimum spacing is propagated until the previous bend or vias. The fanout checkbox has an effect only if the spacing that would be used to place the vias in the previous segment alignment is smaller than <code>minSpacing</code>. Environment variable: <code>fanOutToVias</code>
<i>Use Width and Spacing Target Pins</i>	Sets both <i>Use Width: Target Pins</i> and <i>Use Spacing: Target Pins</i> .
<i>Finish Trunk</i>	The Finish Trunks option automatically converts the pathSegs that are being created using the Create Wire or Create Bus command into trunks. Also, it then runs Pin To Trunk routing in the same step. The default bindkey is 2. The following figure shows a layout design on which the Finish Trunk command has successfully completed.



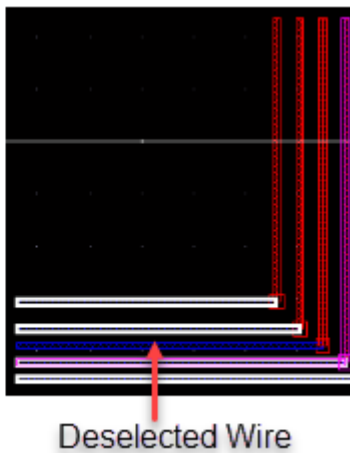
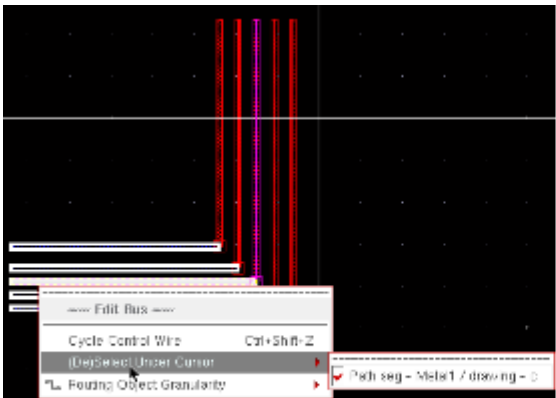
<i>Backup Point</i>	Removes the last digitized point. If a via was added to the last digitized point, that via is also removed.

Edit Bus Shortcut Menu Commands

Selecting and right-clicking the bus in the design canvas when the *Edit – Wiring – Bus* command is active displays Edit Bus shortcut menu commands.



Field	Description
<i>Cycle Control Wire</i>	Changes the control wire of the selected part of a bus. For example, if there are 5 wires selected in a bus, the control wire moves from top to bottom and then to centre. This means using the Cycle Control Wire option in the Edit Bus context-sensitive menu, you can select only the top, bottom, and the centre wire as the control wire. The second and fourth wire will never be selected as the control wire. To set the second or fourth wire as a control wire, click the corresponding net name in the Wire Assistant Bus Bit table. For more information, refer to Using the Control Wire .
<i>(De)Select Under Cursor</i>	Selects or deselects the wire(s) on which the mouse cursor exists as shown in the following figure.



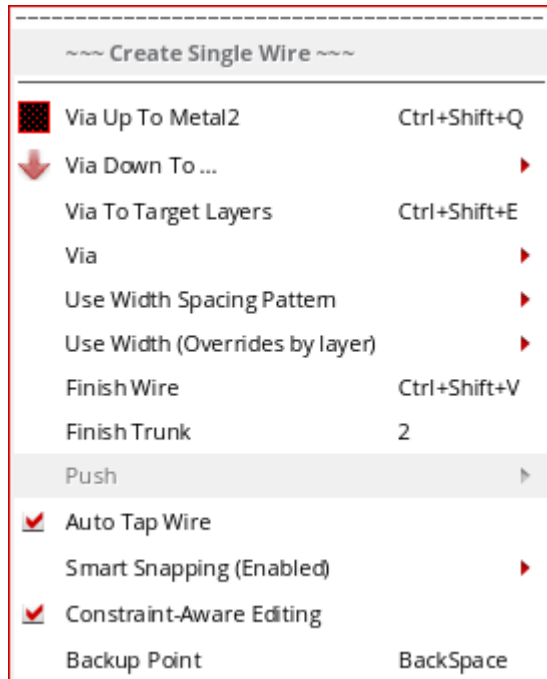
Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Routing Object Granularity</i>	Helps select routing objects by automatically expanding the selection based on the modes. For more information on modes, refer to Selection Options form.

Create Single Wire Shortcut Menu Commands

Right-clicking in the design canvas when the *Create Wire* command is active displays the *Create Single Wire* shortcut menu commands. The following commands are available for Create Wire.



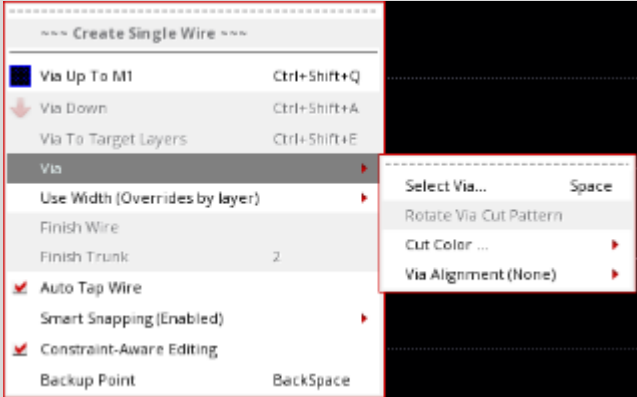
Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Via Up To</i> <layer_name>	Changes the layer to the next layer up and places the corresponding default via. A layer can now connect to several layers in the layers stack: ■ When current layer connects to two or more upper layers in the layer stack, a slider is created with the connected layers. ■ When current layer connects to a single upper layer in the layers stack, the layer name appears in the <i>Create Wire</i> shortcut command. ■ When current layer is unable to connect to any upper layer in the layers stack, the <i>Via Up</i> field appears disabled in the <i>Create Wire</i> shortcut command.
<i>Via Down To</i> <layer_name>	Changes the layer to the next layer down and places the corresponding default via. A layer can now connect to several layers in the layers stack: ■ When current layer connects to two or more lower layers in the layer stack, a slider is created with the connected layers. ■ When current layer connects to a single lower layer in the layers stack, the layer name appears in the <i>Create Wire</i> shortcut command. ■ When current layer is unable to connect to any lower layer in the layers stack, the <i>Via Up</i> field appears disabled in the <i>Create Wire</i> shortcut command.
<i>Via To Target Layers</i>	Adds via stack to the layer of the target object. The target object is the object that is pointed to by the flightline when the <i>Create Wire</i> command is run. However, if the flightline option in the Wire Assistant is off, then vias are not added. If a wire is already on the same layer as its respective target object, then a via is not added and a message is displayed in CIW as follows: "WARNING* (LE-105035): Already on specified layer."

Virtuoso Interactive and Assisted Routing User Guide

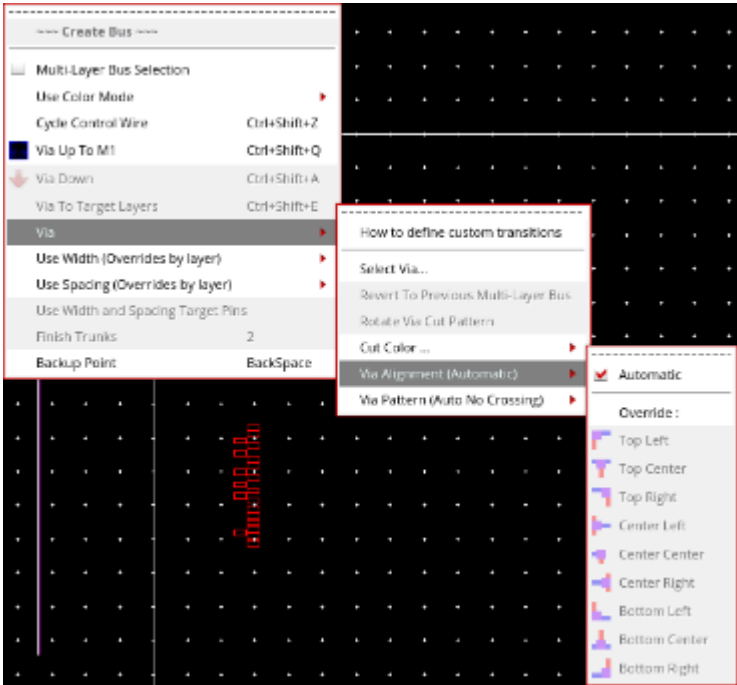
Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
Via	Let's you select the via options. 
<i>Select Via</i>	Opens the <u>Select Via Form</u> to change to other layers. Using this form, you can also rotate the via and also control via alignment.
<i>Rotate Via Cut Pattern</i>	Rotates the via cuts and swaps the rows and columns, if any, while rotating the via shape by 90 degrees as you place or drag the via. You can also rotate pending vias by clicking the middle mouse button. This menu command is available while you are adding a via. You can also rotate a via by using the <i>Rotate Via(s) Cut Pattern</i> button on the <u>Select Via Form</u> .
<i>Cut Color</i>	Enables cut coloring and cycles the color of each cut when a via is inserted. The color of the cut is based on information set by the <u>mptSetLayerDefaultColor</u> SKILL API. The <i>Cycle</i> option in the shortcut command is enabled when a via is being dragged.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Via Alignment</i>	Provides options to align vias while adding them to the layout. You can choose to align a via either automatically or by selecting a custom alignment option. You can also align a via by using the <i>Via Alignment</i> options on the <i>Select Via Form</i> . For more information about the <i>Automatic</i> and <i>Override</i> alignment options, see the <i>Select Via Form</i> . The <i>Override</i> options on the <i>Via Alignment</i> sub-menu are available while you are adding a via.

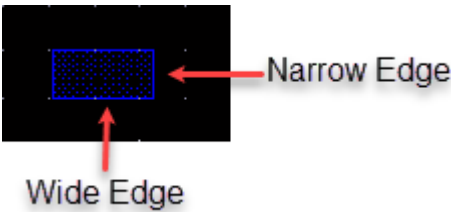


<i>Use Width Spacing Pattern</i>	Lets you work with various width spacing patterns. ■ <i>Cycle Snapping Layer</i> ■ <i>Dynamic Pattern Switching</i> ■ <i>Allow Crossed-regions Off Track Wires</i> ■ <i>Gravity</i> For more information on these options, see Working with Width Spacing Patterns During Wire Editing.

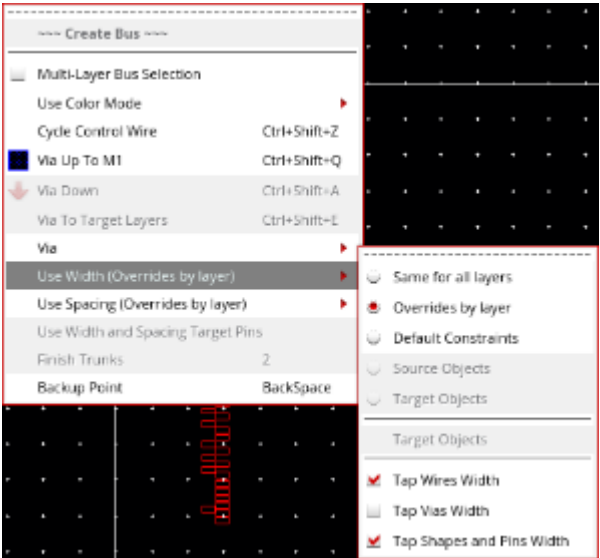
Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Use Width (Overrides by Layer)</i>	For all the <i>Use Width</i> options, except for <i>Target Pin Narrow Edge</i> and <i>Target Pin Wide Edge</i>, a pin edge highlights when it is within a specific distance from the pointer. If you click, the wire snaps to the highlighted pin edge. This is the smart snapping feature of wire. You can configure the snap distance (<i>Aperture</i>) in the <i>Layout Editor Options</i> form. The highlight represents the starting point of wire creation and the length of the highlight determines wire width. The center of the highlight is represented by a yellow box. You can cycle the highlight through the available edges within the <i>Aperture</i> distance by using the <i>Cycle Edge</i> command on the <i>Smart Snapping</i> sub-menu. Only the <i>Aperture</i> setting from the <i>Gravity Controls</i> section in the <i>Layout Editor Options</i> form is used during wire creation. Other settings in the <i>Gravity Controls</i> section are not considered.



The following *Use Width* commands are available for Create Wire.



Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
	■ <i>Same for all layers</i>: Specifies to use the same wire width for all layers. The width value is determined by the value specified in the <i>Width</i> field of the Create Wire form. The constraint overrides for all layers are available in the Wire Assistant table. ■ <i>Overrides by layer</i>: Sets the wire width the same as the result of the constraint lookup for <code>minWidth</code> in the technology file when the layer is not used. Else, the width is determined by the last width used for the layer. The width value from the Wire Assistant table can also be used for the wires on each layer. You can also use the value specified in the <i>Width</i> field for the wires on each layer. ■ <i>Default Constraints</i>: Sets the wire width same as the result of the constraint lookup for <code>minWidth</code> in the technology file or in any applicable constraint group defined at the design or object level. ■ <i>Source Objects</i>: This is the default option. The source object can be pins or wires. It is specified to use the maximum width determined by the width of the tapped object. Selecting the <i>Source Objects</i> option for the first segment of a wire ensures that <i>Tap Wires Width</i> or <i>Tap Pins Width</i> is selected, depending on the source object. You can also select <i>Source Objects</i> even if the first segments of the wires does not have the source object width. In such a case, the <i>Tap Wires Width</i> and <i>Tap Pins Width</i> option does not change. ■ <i>Target Objects</i>: Sets each bus wire to match the width of the same edge of each respective target pin. If the pins are vertically aligned to bus wires, the bus wire widths match the width of the vertical edges of the respective target pins. Similarly, if the pins are horizontally aligned to bus wires, the bus wire widths match the width of the horizontal edges of the respective target pins. The horizontal and vertical edges might correspond to narrow edges for some target pins and wide edges for other target pins. ■ <i>Target Object Narrow</i>: This option matches the last segment wire width to the length of the shortest edge of the target pin. The width of the remaining wire is unaffected.

Virtuoso Interactive and Assisted Routing User Guide

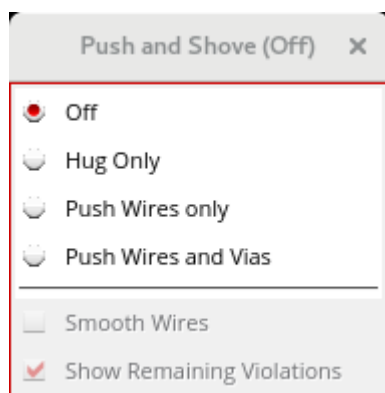
Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
	■ <i>Tap Wires Width</i>: Automatically tap an object's layers and attributes when digitizing the first coordinate of a wire. Layers that can be tapped are determined from the technology file. The <i>Tap Wires Width</i> option is disabled when the <i>Auto Tap Wire</i> option is deselected in the Layout Editor Options form. To enable the <i>Tap Wires Width</i> option in the <i>Use Width</i> menu, select the <i>Auto Tap Wire</i> option in the Layout Editor Options form. ■ <i>Tap Vias Width</i>: When selected, enables tapping the width on the starting object for new wire when the starting object is a via. The edge of the via determines the width of the started wire. ■ <i>Tap Shapes and Pins Width</i>: When selected, automatically taps a top-level shape, a top-level pin, or a level-1 instance layer and width attributes. The objects type is either a rectangle, a polygon, or a path. All valid routing layers can be tapped. The <i>Tap Shapes and Pins Width</i> option is disabled when the <i>Auto Tap Wire</i> option is deselected in the Layout Editor Options form. To enable the <i>Tap Shapes and Pins Width</i> option in the <i>Use Width</i> menu, select the <i>Auto Tap Wire</i> option in the Layout Editor Options form.
	Environment variables: weTapWidthOnPins, weTapWidthOnWires
<i>Finish Trunk</i>	Converts the pathSeg that is being created using the <i>Create Wire</i> or <i>Create Bus</i> command into a trunk. The default bindkey is 2. For more information, see Finishing a Trunk. The <i>Finish Trunk</i> command is disabled when the wire segment created using the <i>Create Wire</i> command has no nets associated with it or when there are no more unconnected pins on the net for which a wire segment is created.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Push and Shove</i>	This command is available only in the Virtuoso RF flow. The license required for Virtuoso RF is <code>Virtuoso_RF_Option</code> (95560). The following figure shows the <i>Push and Shove</i> options for the <i>Create Wire</i> command.



- **Off:** The push-and-shove feature is disabled. This is the default mode.
- **Hug Only:** The wire that is edited moves and avoids the other existing wires. The existing shapes are considered as static and are not pushed.
- **Push Wires only:** The existing wires are pushed to create space for the edited wire. The wire that is edited moves and avoids the shapes that cannot be pushed, such as pins, bumps, and bond fingers.
- **Push Wires and Vias:** Lets you push existing vias on a wire. The *Push Vias* option is enabled only when the mode is *Prefer Shove*.
- **Smooth Wires:** The extra curves and bends of the wire edited by push-and-shove are smoothened. This option is available in case of Virtuoso RF designs.
- **Show Remaining Violations:** The violations that exist on a wire as a result of push and shove are shown.

Environment variables: `pushAndShoveMode`,
`pushAndShoveSmooth`, `pushAndShoveCheckerStopLevel`

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Auto Tap Wire</i>	This option is on by default. See the <i>Auto Tap</i> option in the Layout Editor Options form.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description

<i>Smart Snapping</i>	Lets you toggle between <i>Cycle Edge</i> and <i>Use Current Position</i> modes by using the corresponding bindkeys, Ctrl + Spacebar and Ctrl + Shift + Spacebar, respectively. Create Single Wire <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px 5px;">Via Up To M1</td> <td style="text-align: right; padding: 2px 5px;">Ctrl+Shift+Q</td> </tr> <tr> <td style="padding: 2px 5px;">Via Down</td> <td style="text-align: right; padding: 2px 5px;">Ctrl+Shift+A</td> </tr> <tr> <td style="padding: 2px 5px;">Via To Target Layers</td> <td style="text-align: right; padding: 2px 5px;">Ctrl+Shift+E</td> </tr> <tr> <td style="padding: 2px 5px;">Via</td> <td style="text-align: right; padding: 2px 5px;">▶</td> </tr> <tr> <td style="padding: 2px 5px;">Use Width (Overrides by layer)</td> <td style="text-align: right; padding: 2px 5px;">▶</td> </tr> <tr> <td style="padding: 2px 5px;">Finish Trunk</td> <td style="text-align: right; padding: 2px 5px;">2</td> </tr> <tr> <td style="padding: 2px 5px;">Auto Tap Wire</td> <td style="text-align: right; padding: 2px 5px;">▶</td> </tr> <tr style="background-color: #f0f0f0;"> <td style="padding: 2px 5px;">Smart Snapping (Enabled)</td> <td style="text-align: right; padding: 2px 5px;">▶</td> </tr> <tr> <td style="padding: 2px 5px;">Backup Point</td> <td style="text-align: right; padding: 2px 5px;">BackSpace</td> </tr> </table> Smart Snapping Enabled Cycle Edge Use Current Position	Via Up To M1	Ctrl+Shift+Q	Via Down	Ctrl+Shift+A	Via To Target Layers	Ctrl+Shift+E	Via	▶	Use Width (Overrides by layer)	▶	Finish Trunk	2	Auto Tap Wire	▶	Smart Snapping (Enabled)	▶	Backup Point	BackSpace
Via Up To M1	Ctrl+Shift+Q																		
Via Down	Ctrl+Shift+A																		
Via To Target Layers	Ctrl+Shift+E																		
Via	▶																		
Use Width (Overrides by layer)	▶																		
Finish Trunk	2																		
Auto Tap Wire	▶																		
Smart Snapping (Enabled)	▶																		
Backup Point	BackSpace																		

- **Enabled:** Enables the smart snapping mode. If off, the *Cycle Edge* and *Use Current Position* options are not available. If smart snapping is off, you can start creating the wire by clicking anywhere on a pin; the wire starts from the clicked position.
- **Cycle Edge:** Enables you to highlight and cycle through the edges available within the *Aperture* distance from the cursor. You can, therefore, quickly and precisely choose the appropriate edge of a level 1 or top-level pin both while starting wire creation and while completing the wire creation. This command also helps to select the edge of the required pin from a group of overlapping pins. You can configure *Aperture* in the Layout Editor Options form.

Only the *Aperture* setting from the *Gravity Controls* section in the *Layout Editor Options* form is used during wire creation. Other settings in the *Gravity Controls* section are not considered.

- ❑ Wire creation begins from the edge that is highlighted at the time of click.
- ❑ Wire creation completes on the edge to which the wire is snapped at the time of click.

Field	Description
	The length of the highlight on the starting pin edge depends on the <i>Use Width</i> option selection, which also determines the wire width. You can cycle the highlight through the prospective starting and ending pin edges in the <i>Cycle Edge</i> mode by pressing <code>Ctrl + Spacebar</code> keys. You can switch to <i>Use Current Position</i> by using the Create Single Wire shortcut command or by pressing the <code>Ctrl + Shift + Spacebar</code> bindkeys. ■ <i>Use Current Position</i>: Resets the point of starting wire creation and completing wire creation to the cursor's current location. In this mode, you can start or end a wire in empty space as well as from or at a pin. Therefore, select this command when you do not want to start or end a wire by snapping to a pin edge. This is the default smart snapping option when <i>Smart Snapping</i> is <i>Enabled</i>.
<i>Backup Pont</i>	Removes the last digitized point. If a via had been added to the last digitized point, the via is also removed.

Reshape Wire Shortcut Menu Commands

Selecting and right-clicking the wire in the design canvas when the *Reshape* command is active displays the *Reshape Wire* shortcut menu commands.



The following *Reshape Wire* shortcut menu commands are available.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Via Up To</i> <layer_name>	Changes the layer to the next layer up and places the corresponding default via.
<i>Via Down To</i> <layer_name>	Changes the layer to the next layer down and places the corresponding default via.
<i>Select Via</i>	Opens the <u>Select Via Form</u> to change to other layers. Using the Select Via form, you can rotate the via and also control via alignment. You can also rotate the via by using the <i>Via</i> command on this context-sensitive menu.
<i>Rotate Via Cut Pattern</i>	Rotates the via cuts and swaps the rows and columns, if any, while rotating the via shape by 90 degree as you place or drag the via. You can also rotate pending vias by clicking the middle mouse button. This menu command is available while you are adding a via. You can also rotate a via by using the <i>Rotate Via(s) Cut Pattern</i> button on the <u>Select Via Form</u> .
<i>Via Alignment</i>	Provides options to align vias while adding them to the layout. You can choose to align a via either automatically or by selecting a custom alignment option. You can also align a via by using the <i>Via Alignment</i> options on the <u>Select Via Form</u> . For more information about the <i>Automatic</i> and <i>Override</i> alignment options, see the <u>Select Via Form</u> . The <i>Override</i> options on the <i>Via Alignment</i> sub-menu are available while you are adding a via.
<i>Backup Point</i>	Removes the last digitized point. If a via had been added to the last digitized point, the via is also removed.
<i>(De)Select Under Cursor</i>	Selects or deselects the wire(s) on which the mouse cursor exists.
<i>Routing Object Granularity</i>	Helps select routing objects by automatically expanding the selection based on the modes. For more information on modes, refer to <u>Selection Options</u> form.

Split Wire Shortcut Menu Commands

Selecting and right-clicking the wire in the design canvas when the *Split* command is active displays the *Split* shortcut menu commands.



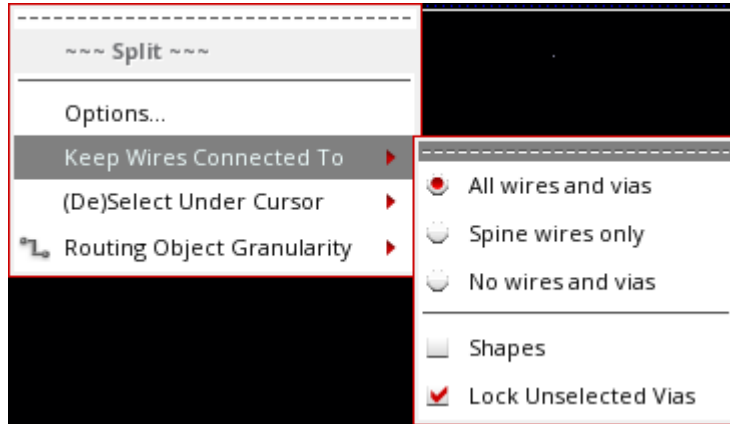
The following *Split Wire* shortcut menu commands are available.

Field	Description
<i>Options</i>	Opens the Split form. You can also press F3 to open the Split form.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

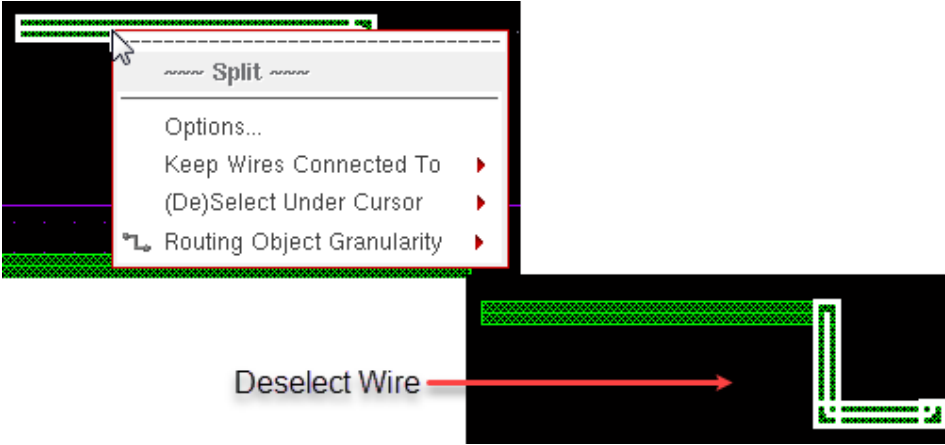
Field	Description
<i>Keep Wires Connected To</i>	Maintains the connectivity even when wires and vias are split. The options available to maintain connectivity are shown in the following figure.



- *All wires and vias*: maintains pathSeg-to-pathSeg and pathSeg-to-via connections during a split. This option is selected by default.
- *Spine wires only*: maintains connections to spine pathSegs but not to connecting vias when splitting a spine. For information about selecting a spine, see [Selecting a Path Vertex](#).
- *No wires and vias*: does not maintain pathSeg-to-pathSeg and pathSeg-to-via connections during split. This option is selected by default.
- *Shapes*: maintains the shape-to-via connection during the stretch.
- *Lock Unselected Vias*: locks the vias that are not selected while stretching a wire. Since the via is locked, the wire is modified to keep the shape connected. In case, the wire could not be modified to maintain the connection, the via is anyway moved to maintain the connection. By default, this option is disabled. It is enabled only when the *All Wires and Vias* option is selected from the *Keep Wires Connected To* field.

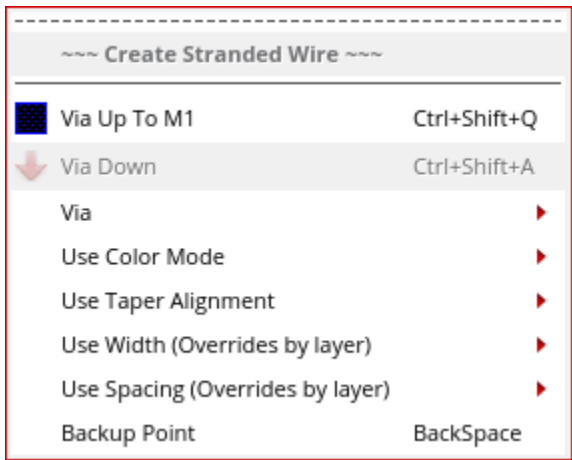
Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
(De)Select Under Cursor	This option helps to select or deselect the wire(s) on which the mouse cursor exists as shown in the following figure.
	
Routing Object Granularity	Helps select routing objects by automatically expanding the selection based on the modes. For more information on modes, refer to Selection Options form.

Create Stranded Wire Shortcut Menu Commands

The *Create Stranded Wire* context-sensitive menu appears on right-clicking in the design display area when the *Create Stranded Wire* command is running and multiple wires are being routed.



The following *Reshape Wire* shortcut menu commands are available.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Via Up To</i> <layer_name>	Changes the layer to the next layer up and automatically places the corresponding default via. A layer can now connect to several layers in the layers stack: ■ When the current layer connects to two or more upper layers in the layer stack, a slider is created with the connected layers. ■ When the current layer connects to a single upper layer in the layers stack, the layer name appears in the <i>Create Stranded Wire</i> menu. ■ When the current layer is unable to connect to any upper layer in the layers stack, the <i>Via Up</i> field appears disabled in the <i>Create Stranded Wire</i> menu.
<i>Via Down To</i> <layer_name>	Changes the layer to the next layer down and automatically places the corresponding default via. A layer can now connect to several layers in the layers stack: ■ When the current layer connects to two or more lower layers in the layer stack, a slider is created with the connected layers. ■ When the current layer connects to a single lower layer in the layers stack, the layer name appears in the <i>Create Stranded Wire</i> menu. ■ When the current layer is unable to connect to any lower layer in the layers stack, the <i>Via Up</i> field appears disabled in the <i>Create Stranded Wire</i> menu.

Virtuoso Interactive and Assisted Routing User Guide

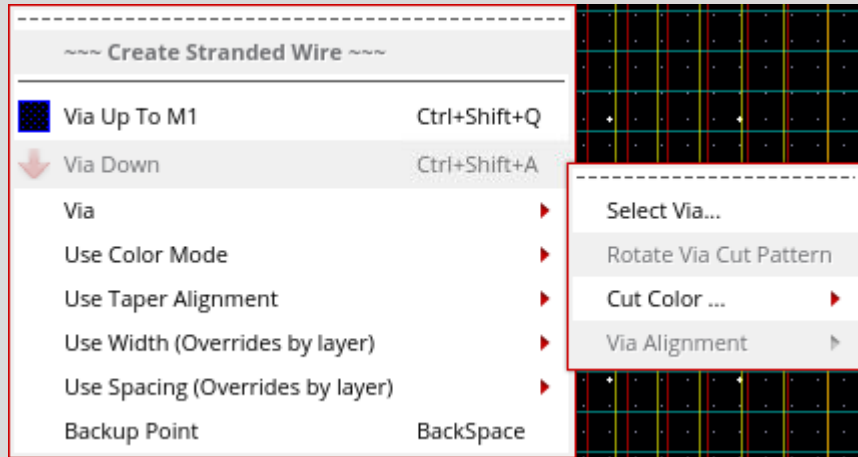
Interactive and Assisted Routing Shortcut Menu Commands

Field

Description

Via

Let's you select the via options.



- *Select Via*: opens the Select Via form. Use the Select Via form to change the layer of the stranded wire. You can rotate the via by using the Select Via form or by using the *Rotate Via Cut Pattern* command.
- *Rotate Via Cut Pattern*: rotates the via cuts and swaps the rows and columns, if any, while rotating the via shape by 90 degrees as you place or drag the via. You can also rotate the existing vias by clicking the middle mouse button. This command is available only while you are adding a via. You can also rotate a via by using the *Rotate Via(s) Cut Pattern* button on the Select Via form.
- *Cut Color*: enables cut coloring and cycles the color of each cut when a via is inserted. The color of the cut is based on information set by the [mptSetLayerDefaultColor](#) SKILL API. The *Cycle* option in the submenu is enabled when a via is being dragged.

To use cut coloring patterns applicable for the Create Stranded Wire command, see the [weWACmdCalculatedColor](#) environment variable.
- *Via Alignment*: this option is disabled as via alignment is not supported for stranded wire.

Use Color Mode

Controls the color mode of the bus. For more information, see [Specifying the Bus Color Mode](#)

Virtuoso Interactive and Assisted Routing User Guide

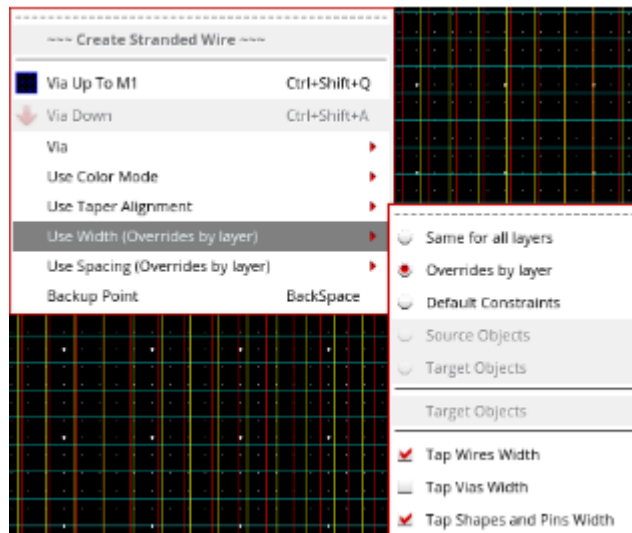
Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Use Taper Alignment</i>	Lets you align the strands in a stranded wire when the number of strands are changed on each section of a wire in the same direction on the same layer or different metal layers. You can change the taper alignment mode to top/right, center, bottom/left. For more information on how to taper the alignment of the stranded wire, see <u>Tapering in Stranded Wire</u> .

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Use Width (Overrides by Layer)</i>	The following <i>Use Width</i> commands are available for Create Stranded Wire.



- **Same for all layers:** Specifies that the same stranded wire width is to be used for all layers. The width value is determined by the value specified in the *Strand Width* field in the Create Stranded Wire form.
- **Overrides by layer:** Sets the stranded wire width to be the same as the result of the constraint lookup for `minWidth` in the technology file when the layer is not used. Otherwise, the width is determined by the last width used for the layer. The width value from the Wire Assistant table can also be used for the stranded wires on each layer. You can also use the value specified in the *Strand Width* field for the stranded wires on each layer.
- **Default Constraints:** Sets the stranded wire width to be the same as the result of the constraint lookup for `minWidth` in the technology file or in any applicable constraint group defined at the design or object level.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
	■ <i>Source Objects</i>: Sets to use the width that is determined by the width of the tapped source object. This is the default option. The source object can be a pin or another wire. Selecting the <i>Source Objects</i> option for the first segment of a stranded wire ensures that <i>Tap Wires Width</i> or <i>Tap Pins Width</i> is selected, depending on the source object. You can also select <i>Source Objects</i> even if the first segments of the stranded wires does not have the source object width. In such a case, the <i>Tap Wires Width</i> and <i>Tap Pins Width</i> option does not change. ■ <i>Target Objects</i>: Sets each stranded wire to match the width of the same edge of each respective target pin. If the pins are vertically aligned to stranded wires, the stranded wire widths match the width of the vertical edges of the respective target pins. Similarly, if the pins are horizontally aligned to stranded wires, the stranded wire widths match the width of the horizontal edges of the respective target pins. The horizontal and vertical edges might correspond to narrow edges for some target pins and wide edges for other target pins. This option cannot be selected and you need to use the <i>Target Pin Narrow Edge</i> or <i>Target Pin Wide Edge</i> options depending on the edge whose width you want. For more information, see Inserting Vias Automatically. The narrow and wide edges for target pins are only applicable in case of single wire creation. ■ <i>Tap Wires Width</i>: Taps the width on the starting object for new stranded wire when the starting object is a wire. The <i>Tap Wires Width</i> option is disabled when the <i>Auto Tap Wire</i> option is deselected in the Layout Editor Options form. To enable the <i>Tap Wires Width</i> option in the <i>Use Width</i> menu, select the <i>Auto Tap Wire</i> option in the Layout Editor Options form. For more information, see Creating a Stranded Wire from an Existing Wire. ■ <i>Tap Vias Width</i>: Taps the width on the starting object for new stranded wire when the starting object is a via. The edge of the via determines the width of the started wire. For more information, see Creating a Stranded Wire from an Existing Wire.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description

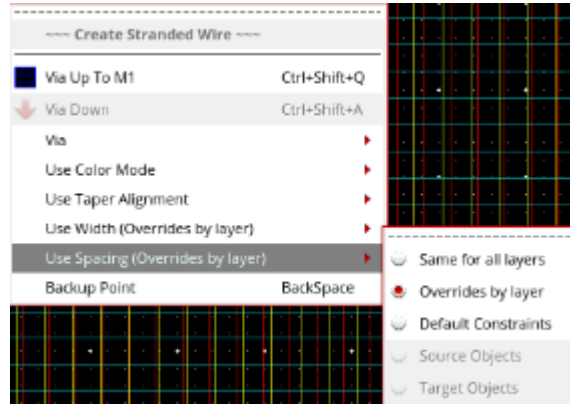
	■ <i>Tap Shapes and Pins Width</i>: When selected, the stranded wires are spread evenly to cover the pin. However, when the <i>Tap Shapes and Pins Width</i> option is deselected, the stranded wires are created from the click point. For more information, see Creating a Stranded Wire from an Existing Pin.

Environment variables: [weTapWidthOnWires](#), [weTapWidthOnPins](#)

Use Spacing

(Overrides by Layer)

The following options are available to specify the spacing between the bus bits.



- *Same for all layers*: Specifies that the same stranded wire spacing is to be used on all the layers. The value specified in the *Stranded Spacing* field is used for the stranded wires on all layers. The constraint overrides for all layers are available in the Wire Assistant table.
- *Overrides by layer*: Sets the stranded wire spacing to be same as the spacing value from the Wire Assistant table on each layer. By default, the result of the constraint lookup for `minSpacing` is used. You can also use the value specified in the *Strand Spacing* field for the stranded wires on each layer.
- *Default Constraints*: Sets the stranded wire spacing to be the same as the result of the constraint lookup for `minSpacing` in the technology file or in any applicable constraint group defined at the design or object level.

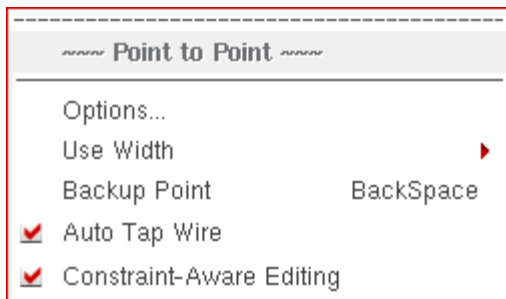
Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
	■ <i>Source Objects</i>: This is the default option when starting from pins or wires and not when starting from an empty space. The spacing of the stranded wires is adjusted to keep the centerlines of the stranded wires aligned with the center lines of the pins. ■ <i>Target Objects</i>: Sets the same spacing between the centerlines of stranded wires as the spacing between the centerlines of the target pins. You can also set the spacing between the pins which belong to the same instance or different instances.
<i>Backup Point</i>	Removes the last digitized point. If a via was added to the last digitized point, that via is also removed.

Point to Point Shortcut Menu Commands

Right-clicking in the design canvas when the *Point to Point* command is active displays the *Point to Point* context-sensitive menu. The following *Point to Point* shortcut menu commands are available.

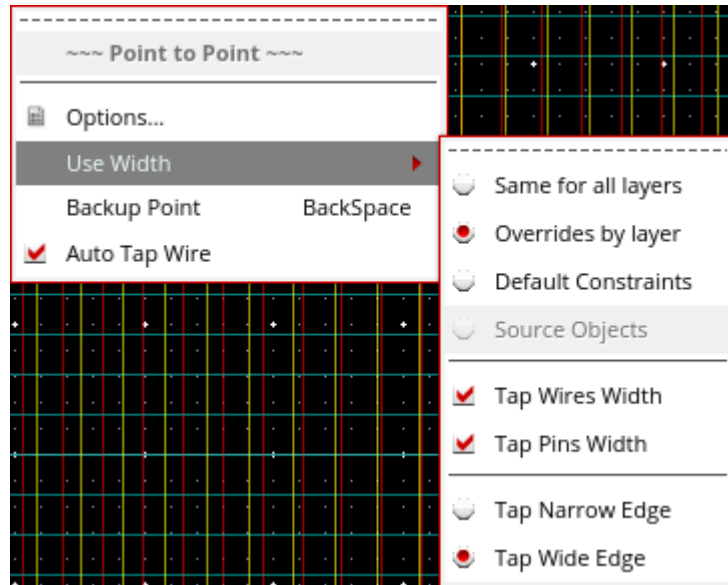


Field	Description
<i>Options</i>	Opens the <u>Point to Point Form</u> . You can also use the F3 bind key to open the options form.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Use Width</i> (<i>Overrides by Layer</i>)	The following <i>Use Width</i> commands are available for Point to Point.



- **Same for all layers:** Specifies to use the same wire width for all layers. The width value is determined by the value specified in the *Width* field of the Point to Point form. The constraint overrides for all layers are available in the Wire Assistant table.
- **Overrides by layer:** Sets the wire width the same as the result of the constraint lookup for `minWidth` in the technology file when the layer is not used. Else, the width is determined by the last width used for the layer. The width value from the Wire Assistant table can also be used for the wires on each layer. You can also use the value specified in the *Width* field for the wires on each layer.
- **Default Constraints:** Sets the stranded wire width to be the same as the result of the constraint lookup for `minWidth` in the technology file or in any applicable constraint group defined at the design or object level.

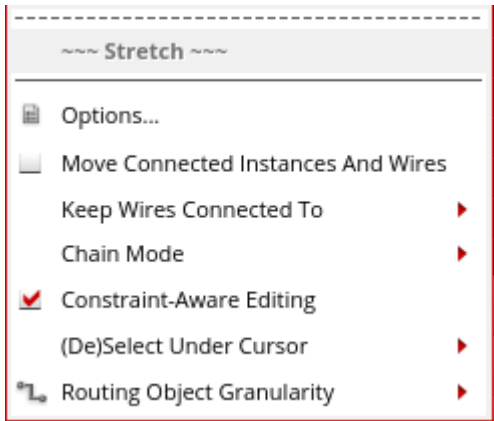
Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
	■ <i>Source Objects</i>: Sets to use the width that is determined by the width of the tapped source object. This is the default option. The source object can be a pin or another wire. Selecting the <i>Source Objects</i> option for the first segment of a stranded wire ensures that <i>Tap Wires Width</i> or <i>Tap Pins Width</i> is selected, depending on the source object. You can also select <i>Source Objects</i> even if the first segments of the stranded wires does not have the source object width. In such a case, the <i>Tap Wires Width</i> and <i>Tap Pins Width</i> option does not change. ■ <i>Tap Wires Width</i>: Enables tapping the wire width when digitizing the first coordinate of a point-to-point command. The <i>Tap Wires Width</i> option is disabled when the <i>Auto Tap Wire</i> option is deselected in the Layout Editor Options form. To enable the <i>Tap Wires Width</i> option in the <i>Use Width</i> menu, select the <i>Auto Tap Wire</i> option in the Layout Editor Options form. ■ <i>Tap Pins Width</i>: Enables tapping the pin width when digitizing the first coordinate of the point-to-point command. The <i>Tap Pins Width</i> option is disabled when the <i>Auto Tap Shape</i> option is deselected in the Layout Editor Options form. To enable the <i>Tap Pins Width</i> option in the <i>Use Width</i> menu, select the <i>Auto Tap Shape</i> option in the Layout Editor Options form. ■ <i>Tap Narrow Edge</i>: Matches the last segment wire width to the length of the shortest edge of the target pin. The width of the remaining wire is unaffected. ■ <i>Tap Wide Edge</i>: Matches the last segment wire width to the length of the longest edge of the target pin. The width of the remaining wire is unaffected.
<i>Backup Point</i>	Removes the last digitized point. If a via had been added to the last digitized point, the via is also removed.
<i>Auto Tap Wire</i>	This option is on by default. See the <i>Auto Tap</i> option in the Layout Editor Options form.

Stretch Shortcut Menu Commands

Right-clicking in the design canvas when the *Stretch* command is active displays the *Stretch* shortcut menu commands.



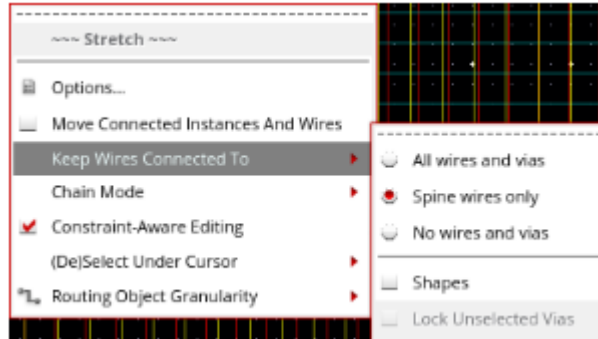
The following *Split Wire* shortcut menu commands are available.

Field	Description
<i>Options</i>	Opens the <u>Stretch Form</u> . You can also press F3 to open the options form.
<i>Move Connected Instances and Wires</i>	Maintains the connectivity between instances and pathSegs and the selected object while moving or stretching the instances or pathSegs. When this option is selected, the <i>Keep Wires Connected to</i> and <i>Lock Unselected Vias</i> options are disabled. To maintain the connection from selected objects: ■ The selected objects move and stretch in both x and y directions. ■ The connected objects stretch or move to keep them connected to the selected objects. Environment variable: <u>moveConnectedInstancesAndWires</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Keep Wires Connected To</i>	Maintains the connectivity when the wires and vias are stretched.

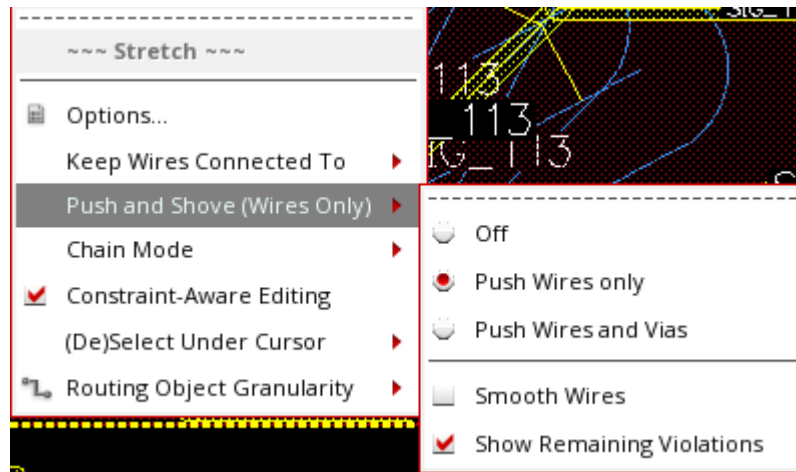


- *All wires and vias*: maintains pathSeg-to-pathSeg and pathSeg-to-via connections during stretch.
- *Spine wires only*: maintains connections to spine pathSegs but not to connecting vias when stretching a spine. This option is selected by default. For information about selecting a spine, see [Selecting a Path Vertex](#).
- *No wires and vias*: does not maintain pathSeg-to-pathSeg and pathSeg-to-via connections during split. This option is selected by default.
- *Shapes*: maintains the shape-to-via connection during the stretch.
- *Lock Unselected Vias*: locks the vias that are not selected while stretching a wire. Since the via is locked, the wire is modified to keep the shape connected. In case, the wire could not be modified to maintain the connection, the via is anyway moved to maintain the connection. By default, this option is disabled. It is enabled only when the *All Wires and Vias* option is selected from the *Keep Wires Connected To* field.

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Push and Shove</i>	This command is available only in the Virtuoso RF flow. The license required for Virtuoso RF is <code>Virtuoso_RF_Option</code> (95560). In the <i>Stretch</i> command you can enable or disable push-and-shove. The following figure shows the <i>Push and Shove</i> options that are available for the Stretch command.



- **Off:** The push-and-shove feature is disabled. This is the default mode.
- **Push Wires only:** The existing wires are pushed to create space for the edited wire. The wire that is edited moves and avoids the shapes that cannot be pushed, such as pins, bumps, and bond fingers.
- **Push Wires and Vias:** Lets you push existing vias on a wire. The *Push Vias* option is enabled only when the mode is *Prefer Shove*.
- **Smooth Wires:** The extra curves and bends of the wire edited by push-and-shove are smoothened. This option is available in case of Virtuoso RF designs.
- **Show Remaining Violations:** The violations that exist on a wire as a result of push and shove are shown.

Environment variables: [pushAndShoveMode](#),
[pushAndShoveSmooth](#), [pushAndShoveCheckerStopLevel](#)

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Shortcut Menu Commands

Field	Description
<i>Chain Mode</i>	Specifies how to stretch chained devices. For more information refer to, Stretch Form .
<i>Constraint Aware Editing</i>	Enforces constraints for nets grouped in a bus constraint and for shielding wires. This option is on by default and is available in Layout XL and higher tiers.
<i>(De)Select Under Cursor</i>	Selects or deselects the wire(s) on which the mouse cursor exists.
<i>Routing Object Granularity</i>	Helps select routing objects by automatically expanding the selection based on the modes. For more information on modes, refer to Selection Options form.

Interactive and Assisted Routing Environment Variables

The system looks for the system-defined default settings in the following file:

```
install_dir/etc/tools/we/.cdsenv  
install_dir/etc/tools/layout/.cdsenv
```

This topic contains the Virtuoso interactive and assisted routing environment variable names, descriptions, types, and values. The environment variables are categorized as follows:

- [Variables to Customize the Interactive and Assisted Routing Settings](#)
- [Variables to Customize the Settings for Slotting](#)
- [Variables to Customize the Settings for Copy Route](#)

Variables to Customize the Interactive and Assisted Routing Settings

Use the following variables to configure the interactive and assisted routing settings.

<u>adjustEditedViasParams</u>	<u>adjustEditedViasParamsScope</u>
<u>allLayersWrongWay</u>	<u>allLayersWrongWayCustomValue</u>
<u>allowLoops</u>	<u>allowOffTrackForCrossRegion</u>
<u>autoDisableProbeDuringIARouting</u>	<u>autoLengthControl</u>
<u>autoTap</u>	<u>autoTapActiveLayer</u>
<u>autoTerminate</u>	<u>autoUpdateFlightlines</u>
<u>autoViaAlignment</u>	<u>autoViaAlignmentColinearTransitionBehavior</u>
<u>autoViaAlignmentOrthoTransitionBehavior</u>	<u>blockageAvoidanceSearchThreshold</u>
<u>blockageUseMinWidth</u>	<u>checkOffGridEditedWires</u>
<u>constraintGroupCustomFilterMode</u>	<u>copyRoutingAllowCrossings</u>
<u>copyRoutingPreserveSpacing</u>	<u>createBusAllowsMultiLayerSelection</u>
<u>createWireMode</u>	<u>createWireStitchAware</u>
<u>displayMinAreaMarker</u>	<u>displayOrthogonalGridEndPoints</u>
<u>dynamicDisplayEnabled</u>	<u>enableDynamicPatternSwitching</u>
<u>enableFinishRoutingCmds</u>	<u>enableWidthOverride</u>
<u>enforceStretchWidthConstraints</u>	<u>fanOutToVias</u>
<u>flightLinesLimit</u>	<u>flightLinesPolygonPointsLimit</u>
<u>flightLinesShapesLimit</u>	<u>gatherAndSpreadSpacing</u>
<u>ignoreSameMaskColor</u>	<u>keepWiresConnected</u>
<u>layerTapPickWireVia</u>	<u>lockUnselectedVias</u>
<u>maxNumConnWireElements</u>	<u>mixedSignalWireEditingEnvironment</u>
<u>moveConnectedInstancesAndWires</u>	<u>moveConnectedObjectType</u>
<u>p2pSeedStyle</u>	<u>pathSegBeginExt</u>
<u>pathSegBeginStyle</u>	<u>pathSegEndExt</u>
<u>pathSegEndStyle</u>	<u>patternFillAlpha</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

<u>patternGravity</u>	<u>patternGravityLayerTransitionType</u>
<u>patternOutlineAlpha</u>	<u>patternRenderingAperture</u>
<u>pushAndShoveMode</u>	<u>pushAndShoveSmooth</u>
<u>pushAndShoveCheckerStopLevel</u>	<u>setTaperMode</u>
<u>showAlignmentMarkers</u>	<u>snapResizeWireLabels</u>
<u>snapToPin</u>	<u>snapToViaCenter</u>
<u>snapWireGrid</u>	<u>spaceFuncName</u>
<u>taperToPinWidth</u>	<u>taperToPinWidthStyle</u>
<u>tapPreference</u>	<u>useMergeBusAndWire</u>
<u>useOverlapRegionForVia</u>	<u>viaAlignment</u>
<u>viaCheckDRC</u>	<u>viaCutColorMode</u>
<u>viaCutColorPatterns</u>	<u>viaExtendEnclosureBeyondOverlap</u>
<u>viaFillEnclosureOverlap</u>	<u>viaMaxNumCuts</u>
<u>viaPattern</u>	<u>weAllowOddWidth</u>
<u>weAreaBasedRulesEnabled</u>	<u>weAreaBasedRulesHierLevel</u>
<u>weAreaBasedRulesWidthMode</u>	<u>weARWidthMode</u>
<u>weARTapWideEdge</u>	<u>weARTapWidthOnPins</u>
<u>weARTapWidthOnWires</u>	<u>weAutoDropVia</u>
<u>weAutoDropViaForOverlaps</u>	<u>weAutoMergeWires</u>
<u>weAutoWidthModeAfterVias</u>	<u>weBusColorMode</u>
<u>weBusNumBits</u>	<u>weChangeWireDirectionAfterVia</u>
<u>weCoverPin</u>	<u>weCreateBusConstraint</u>
<u>weCustomLayerPattern</u>	<u>weCWStretchLastPointModifier</u>
<u>weCWWidthMode</u>	<u>weEnableDanglingTargetsFL</u>
<u>weEnableXLCheckoutInL</u>	<u>weEnforceDirFromRoutingPattern</u>
<u>weFlightLinesHierLevel</u>	<u>weGuidedRouteGatherAtBend</u>
<u>weHalo</u>	<u>weHaloDashStyle</u>
<u>weHaloEnabled</u>	<u>weHaloEOLSpacing</u>
<u>weHaloTrueColor</u>	<u>weInstancePinPattern</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

<u>weInteractiveShielding</u>	<u>weKeepSelectViaOrder</u>
<u>weMemoryOptimization</u>	<u>weMfgGridWidthIncrement</u>
<u>weNoSmartSnap</u>	<u>wePostCriticWires</u>
<u>weShowCtrlWireAlignMarkerOnly</u>	<u>weShowHints</u>
<u>weSmartSnappingDepth</u>	<u>weStrandedAlignCollinearMode</u>
<u>weStrandedColorMode</u>	<u>weStrandedLadderAtTurn</u>
<u>weStrandNum</u>	<u>weStrandSpacing</u>
<u>weStrandWidth</u>	<u>weTapWidthOnPins</u>
<u>weTapWidthOnVias</u>	<u>weTapWidthOnWires</u>
<u>weTrimBridgePurpose</u>	<u>weTrimInsertion</u>
<u>weTrimInsertionOnPRBoundary</u>	<u>weTrimPatchExtension</u>
<u>weTrimPatchPurpose</u>	<u>weTrimTrimPurpose</u>
<u>weWACmdCalculatedColor</u>	<u>weWAConflictColor</u>
<u>weWADefaultSeedCGName</u>	<u>weWASeedCGColor</u>
<u>weWAUserDefColor</u>	<u>wireConstraintGroup</u>
<u>wireOffset</u>	<u>wireTaperConstraintGroup</u>
<u>wireViaStackMode</u>	<u>wireWidth</u>

Related Topics

[Interactive and Assisted Routing Environment Variables](#)

adjustEditedViasParams

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

Description

Re-calculates via parameters (such as alignment or enclosures) if you stretch the wire connected to the via or while editing width and layer properties of a pathSeg. The default is `nil`.

GUI Equivalent

Command	<i>Options – Editor</i> <i>Window – Assistant – Wire Assistant – Net – Via Configuration.</i>
Form Field	<i>Adjust Edited Vias Params</i>

Examples

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

Related Topics

[Via Configuration Form](#)

[Point to Point Form](#)

[Variables to Customize the Interactive and Assisted Routing Settings](#)

adjustEditedViasParamsScope

```
layout adjustEditedViasParamsScope string "functionname"
```

Description

Defines the interactive routing command that is allowed to adjust the via parameters. The valid commands are: stretch, reshape, quickAlign, split, editProperties, editBus, pickedVia, and iRoute.

GUI Equivalent

None

Examples

```
envGetVal("layout" "adjustEditedViasParamsScope")  
envSetVal("layout" "adjustEditedViasParamsScope" 'string "stretch")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

allLayersWrongWay

```
layout allLayersWrongWay cyclic { "unknown" | "As Is" | "Allow" | "None" | "Custom"
}
```

Description

Controls wrong way routing.

- `As Is` allows specific wrong way routing, if required.
- `Allow` permits wrong way routing freely.
- `None` does not allow wrong way routing.
- `Custom` lets you control the extent of wrong way routing you want to permit. When set to `Custom`, use the [`allLayersWrongWayCustomValue`](#) environment variable to specify a value.

Default is `As Is`.

GUI Equivalent

Command	<i>Window – Assistant – Wire Assistant – Interactive – Finish</i>
Field	<i>Wrong Way Tax</i>

Examples

```
envGetVal("layout" "allLayersWrongWay")
envSetVal("layout" "allLayersWrongWay" 'cyclic "Allow")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Interactive Section](#)

[Finish](#)

allLayersWrongWayCustomValue

```
layout allLayersWrongWayCustomValue int wrongWayCustomValue
```

Description

Specifies the extent of wrong way routing you want to permit. You can specify a value between 0 to 100, 0 being the level of maximum freedom for creating wrong way routing and 100 being the level of maximum restriction for creating wrong way routing. The default value is 1.

GUI Equivalent

Command	<i>Window – Assistant – Wire Assistant – Interactive – Finish</i>
Field	<i>Wrong Way Tax Value</i>

Examples

```
envGetVal("layout" "allLayersWrongWayCustomValue")  
envSetVal("layout" "allLayersWrongWayCustomValue" 'int 50)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Interactive Section](#)

[Finish](#)

allowLoops

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

Description

Allows the creation of loops when creating or editing wires. The default is `nil`.

GUI Equivalent

Command	<i>Options – Editor</i> <i>Windows – Assistant – Wire Assistant – Interactive – Create Wire</i>
Form Field	<i>Allow Loops</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

[Interactive Section](#)

allowOffTrackForCrossRegion

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

Description

Lets a wire cross regions regardless of it being on track or off track. When `nil` and the WSP tracks are not identical, the `pathSeg` stops at local region boundaries. When `t`, the end point of the `pathSeg` can be across a region boundary, such that it is off track or on track with different color and width. The default is `nil`.

GUI Equivalent

Command	<i>Create – Wiring – Wire – Create Single Wire context-sensitive menu – Use Width Spacing Pattern</i>
Form Field	<i>Allow Crossed-region Off Track Wires.</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Supporting pathSegs with Different WSPs across Regions](#)

autoDisableProbeDuringIARouting

```
layoutXL autoDisableProbeDuringIARouting boolean { t | nil }
```

Description

Disables the probe nets feature in interactive and assisted routing commands, such as Create Wire, Create Bus, Guided Routing, and Point to Point, automatically. The default is `t` and the Probe Nets check box is deselected.

If the *Probe Nets* check box is explicitly selected or deselected in the displayed form `t` when one of the interactive or assisted routing command is run, automatic disabling of probe nets is no longer valid. This means that the value specified for the *Probe Nets* check box is used.

GUI Equivalent

Command	<i>Create – Wiring – Wire/Bus/Stranded Wire/Point to Point Routing</i>
Form Field	<i>Probe Nets</i>

Examples

```
envGetVal("layoutXL" "autoDisableProbeDuringIARouting")  
envSetVal("layoutXL" "autoDisableProbeDuringIARouting" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Create Stranded Wire Form](#)

[Edit Bus Form](#)

[Create Stranded Wire Form](#)

[Point to Point Form](#)

autoLengthControl

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

Description

Controls whether or not the wire editor follows `allowedLengthRange` constraint. The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "autoLengthControl")  
envSetVal("layout" "autoLengthControl" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

autoTap

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

Description

Controls whether a design-tap occurs on the first entry point (`t`) or not (`nil`), when you use the *Create – Wire* command. When auto tap is on, the current entry layer will seed the form and update attributes of the wire you create based on the first point you enter. The tap command does not seed the Net Name field of the Create Wire form the layout editor. The *Net Name* field is seeded in Virtuoso XL. The default is `t`.

GUI Equivalent

Command	<i>Options – Layout Editor</i>
Form Field	<i>Auto Tap Wire</i>

Examples

```
envGetVal("layout" "autoTap")  
envSetVal("layout" "autoTap" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

autoTapActiveLayer

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

Description

When set to `nil`, the Choose Object to Tap form is displayed on clicking the overlapping area. The form is displayed irrespective of the layer selected in the *Layers* assistant in the Palette Assistant. The default is `t`.

GUI Equivalent

Command	<i>Options – Layout Editor</i>
Form Field	<i>Tap Layer</i>

Examples

```
envGetVal("layout" "autoTapActiveLayer")  
envSetVal("layout" "autoTapActiveLayer" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

autoTerminate

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

Description

When `t`, and creating a wire, the wire will automatically terminate when you create a vertex on an existing object with the same net name. The default is `nil`.

GUI Equivalent

Command	<i>Create – Wiring – Wire/Bus/Stranded Wire</i> <i>Window – Assistant – Wire Assistant – Interactive – Create Wire</i>
Field	<i>Auto Terminate</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Create Stranded Wire Form](#)

[Edit Bus Form](#)

[Create Stranded Wire Form](#)

[Interactive Section](#)

autoUpdateFlightlines

```
layout autoUpdateFlightlines cyclic { "on" | "off" | "minimum" }
```

Description

Controls the display of flightlines during wire creation. You can hide the display of flightlines by setting it to `off` or display only three flightlines for each bus by setting it to `minimum`. The default is `on`.

GUI Equivalent

Command	<i>Window – Assistant – Wire Assistant – Interactive – Create Wire</i>
Field	<i>Show Flightline</i>

Examples

```
envGetVal("layout" "autoUpdateFlightlines")  
envSetVal("layout" "autoUpdateFlightlines" 'cyclic "off")  
envSetVal("layout" "autoUpdateFlightlines" 'cyclic "minimum")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Interactive Section](#)

autoViaAlignment

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

Description

Specifies a via to be automatically aligned with respect to both the existing and the new pathSeg. The default is `nil`.

GUI Equivalent

Command	<i>Create Wire context-sensitive menu – Via – Select Via</i>
Field	<i>Via Alignment - Automatic</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Metal Density Options Form](#)

[Create Stranded Wire Form](#)

autoViaAlignmentColinearTransitionBehavior

```
layout autoViaAlignmentColinearTransitionBehavior cyclic { "none" | "incoming" |  
    "outgoing" | "both" }
```

Description

Controls the alignment behavior for collinear transitions. The default is `both`.

GUI Equivalent

Command	<i>Create Wire context-sensitive menu – Via – Select Via</i>
Field	<i>For collinear transitions</i>

Examples

```
envGetVal("layout" "autoViaAlignmentColinearTransitionBehavior")  
envSetVal("layout" "autoViaAlignmentColinearTransitionBehavior" 'cyclic "none")  
envSetVal("layout" "autoViaAlignmentColinearTransitionBehavior" 'cyclic  
"incoming")  
envSetVal("layout" "autoViaAlignmentColinearTransitionBehavior" 'cyclic  
"outgoing")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Metal Density Options Form](#)

[Create Stranded Wire Form](#)

autoViaAlignmentOrthoTransitionBehavior

```
layout autoViaAlignmentOrthoTransitionBehavior cyclic { "none" | "incoming" |  
    "outgoing" | "both" }
```

Description

Controls the alignment behavior for orthogonal transitions. The default is both.

GUI Equivalent

Command	<i>Create Wire context-sensitive menu – Via – Select Via</i>
Field	<i>For orthogonal transitions</i>

Examples

```
envGetVal("layout" "autoViaAlignmentOrthoTransitionBehavior")  
envSetVal("layout" "autoViaAlignmentOrthoTransitionBehavior" 'cyclic "none")  
envSetVal("layout" "autoViaAlignmentOrthoTransitionBehavior" 'cyclic "incoming")  
envSetVal("layout" "autoViaAlignmentOrthoTransitionBehavior" 'cyclic "outgoing")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Metal Density Options Form](#)

blockageAvoidanceSearchThreshold

`we blockageAvoidanceSearchThreshold int BlockageAvoidanceThresholdValue`

Description

Controls the threshold value for the search limit for blockage avoidance.

When the value is greater than 0, the routing path is searched until the threshold value is reached. When the value is less than or equal to 0, the search is disabled. The default value is 100.

GUI Equivalent

None

Examples

```
envGetVal("we" "blockageAvoidanceSearchThreshold")  
envSetVal("we" "blockageAvoidanceSearchThreshold" 'int 5)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

blockageUseMinWidth

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

Description

Considers unassigned blockages as metal shapes and use their real width to calculate spacing requirements. When set to `t`, blockages are treated as metal shapes with minimum width. The default is `nil`.

GUI Equivalent

Command	<i>Options – Editor</i>
Form Field	<i>Blockage: Use Minimum Width</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

checkOffGridEditedWires

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

Description

Verifies whether the wire being edited using the *Stretch* command ends on an appropriately colored track. As a result, the *Stretch* command might prevent wires to be edited to avoid color shorts. The default is `nil`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

constraintGroupCustomFilterMode

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

Description

Ignores the custom API and all the constraint groups that are listed. When set to `t`, the custom API `constraintGroupCustomFilter` is called, if it has been defined, to display the list of constraint groups in the *Default Wire Constraint Group* cyclic field. The default is `nil`.

You must define the signature API `t/nil`

```
constraintGroupCustomFilter(editedCellViewId constraintGroupId).
```

GUI Equivalent

Command	<i>Wire Assistant Toolbar icon.</i>
Form Field	<i>Seed Attributes From a Constraint Group.</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

copyRoutingAllowCrossings

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

Description

Controls the placement of the unlocked pathSegs that are not connected to pins. The default is `nil`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

copyRoutingPreserveSpacing

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

Description

Preserves the spacing of the unlocked pathSegs that are not connected to pins. The default is t.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

createBusAllowsMultiLayerSelection

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

Description

When set to `t`, lets you select pathsegs, pins, and instance pins on different metal layers, thereby enabling multi-layer bus feature. The default is `nil`.

You can also enable the multi-layer bus feature by selecting *Multi-Layer Bus Selection* option from the *Create Bus* context-sensitive menu.

GUI Equivalent

Command	<i>Create – Wiring – Bus – Create Bus</i> context-sensitive menu
Form Field	<i>Multi-Layer Bus Selection</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Supporting Multi-Layer Bus](#)

createWireMode

```
layout createWireMode cyclic { "unknown" | "paths" | "segments" | "Paths" |  
    "Segments" }
```

Description

You can create wires as paths or pathSegs (Segments). Paths are not supported by Innovus as part of a route element. When saving data to Innovus you must use pathSegs rather than paths. Using paths for non-orthogonal routing, you can create off-grid vertices, depending on the width and angle being used. You can create on grid 45 degree wires with pathSegs. PathSegs cannot be used for routing to any angle and are limited to orthogonal and diagonal routing. The default is `segments`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "createWireMode")  
envSetVal("layout" "createWireMode" 'cyclic "unknown")  
envSetVal("layout" "createWireMode" 'cyclic "paths")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Create Wire Form](#)

[leHiCreateGeometricWire](#)

createWireStitchAware

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

Description

Controls whether the *Create Wire* command is cognizant of the existing stitches formed by collinear wires of different mask colors. If so, the command maintains the stitches. The default is `nil`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

displayMinAreaMarker

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

Description

Displays markers for minimum area violations while creating wires. The default is `t`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

displayOrthogonalGridEndPoints

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

Description

Enables the display of grid end points to which the pathSeg is extended to while creating a wire, bus, and stretch. The default is `t`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

dynamicDisplayEnabled

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

Description

Enables the dynamic display of tracks, snapPatterns, and width spacing patterns. The default is `t`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Dynamic Display](#)

enableDynamicPatternSwitching

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

Description

Specifies the default state of dynamic pattern switching. When dynamic pattern switching is enabled, it lets you cross region-to-global, global-to-region, and region-to-region boundary without manually setting active pattern. The default is t.

GUI Equivalent

Command	<i>Create – Wiring – Wire – Create Single Wire context-sensitive menu – Use Width Spacing Pattern.</i>
Form Field	<i>Dynamic Pattern Switching.</i>

Examples

```
envGetVal("we" "enableDynamicPatternSwitching")
envSetVal("we" "enableDynamicPatternSwitching" 'boolean t)
envSetVal("we" "enableDynamicPatternSwitching" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Supporting Width Spacing Patterns](#)

[Create Stranded Wire Form](#)

[Switching Patterns Dynamically](#)

enableFinishRoutingCmds

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

Description

Enables or disables the *Finish Wire* and *Finish Bus* commands in the *Create Wire* and *Create Bus* context-sensitive menus respectively. The default is `t`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Create Stranded Wire Form](#)

enableWidthOverride

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

Description

Allows the wire width specified in the *Width* field to be used by the point to point router when generating wire segments. The default is `nil`.

GUI Equivalent

Command	<i>Create – Wiring – Point to Point</i>
Field	<i>Width</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Routing](#)

[Point to Point Form](#)

enforceStretchWidthConstraints

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

Description

Controls width enforcement to fit width constraints, such as `minWidth`, `maxWidth`, and `allowedWidthRanges`, when a `pathSeg` is stretched. When set to `nil`, the width of the `pathSeg` is not updated even if it is not valid. The default is `t`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

fanOutToVias

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

Description

Spreads out the bus wires for via placement. The default is `nil`.

GUI Equivalent

Command	<i>Create – Wiring – Bus</i>
Field	<i>Fan Out to Vias</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Fanning Out Wires to Vias](#)

[Create Bus Form](#)

flightLinesLimit

```
we flightLinesLimit int flightLines
```

Description

Determines the threshold value above which the display mode `ON` (for all flightlines) automatically switches to the display mode `minimum` (for bus extremities and control wires flightlines). The default is 64.

GUI Equivalent

None

Examples

```
envGetVal("we" "flightLinesLimit")  
envSetVal("we" "flightLinesLimit" 'int 100)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

flightLinesPolygonPointsLimit

```
we flightLinesPolygonPointsLimit int flightLinesPointsLimit
```

Description

Determines the maximum numbers of points for polygons being considered while computing dynamic flightlines. The default is 64.

GUI Equivalent

None

Examples

```
envGetVal("we" "flightLinesPolygonPointsLimit")  
envSetVal("we" "flightLinesPolygonPointsLimit" 'int 100)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

flightLinesShapesLimit

`we flightLinesShapesLimit int flightLineShapeNo`

Description

Determines the maximum number of shapes that can be traversed while computing flightlines when using shape chasing. The default is 1000.

GUI Equivalent

None

Examples

```
envGetVal("we" "flightLinesShapesLimit")  
envSetVal("we" "flightLinesShapesLimit" 'int 100)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

gatherAndSpreadSpacing

```
layout gatherAndSpreadSpacing cyclic { "largest" | "smallest" }
```

Description

Controls the spacing used when spreading or gathering a bus. The default is `smallest`.

You can set this environment variable as follows:

```
leSetEnv("gatherAndSpreadSpacing" "largest")
```

GUI Equivalent

Command	<i>Create – Wiring – Bus – Create Bus context-sensitive menu – Use Spacing</i>
Field	<i>Gather/Spread</i>

Examples

```
envGetVal("layout" "gatherAndSpreadSpacing")  
envSetVal("layout" "gatherAndSpreadSpacing" 'cyclic "largest")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Edit Bus Form](#)

ignoreSameMaskColor

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

Description

Used by the Create Bus command to ignore `minCutClassSpacing` constraints with the same mask color parameter. This environment variable should be used in synchronization with the environment variable `ignore_same_mask_spacing`, that is, both environment variables must be set to either `t` or `nil` to obtain the same behavior between DRD and create bus command. The default is `nil`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Interactive Coloring for Create Bus](#)

keepWiresConnected

```
layout keepWiresConnected cyclic { "all" | "spine" | "none" }
```

Description

Specifies whether to keep pathSegs and vias connected when using the *Stretch* command. In the layout editor, wires remain connected if the end points meet correctly. In the XL layout editor, wires remain connected even if end points do not meet completely.

- `all` maintains pathSeg-to-pathSeg and pathSeg-to-via connections during stretch.
- `spine` maintains only spine pathSeg connections during stretch.
- `none` the connections are not maintained.

The default is `all`.

GUI Equivalent

Command	<i>Edit – Stretch</i>
Field	<i>Keep Wires Connected to</i>

Examples

```
envGetVal("layout" "keepWiresConnected")  
envSetVal("layout" "keepWiresConnected" 'cyclic "spine")  
envSetVal("layout" "keepWiresConnected" 'cyclic "none")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Using Keep Wires Connected to Split Command](#)

layerTapPickWireVia

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

Description

The default behavior when tapping on a via is, if there is an existing segment connected to one layer of the via (or stack of vias) the intend wire is tap from the unoccupied layer of the via. This option can be used to disable the default behavior causing the *Choose Object to Tap* form to open. The default is `nil`.

GUI Equivalent

Command	<i>Options – Layout Editor</i>
Form Field	<i>Include All (Wire) Via Layers</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

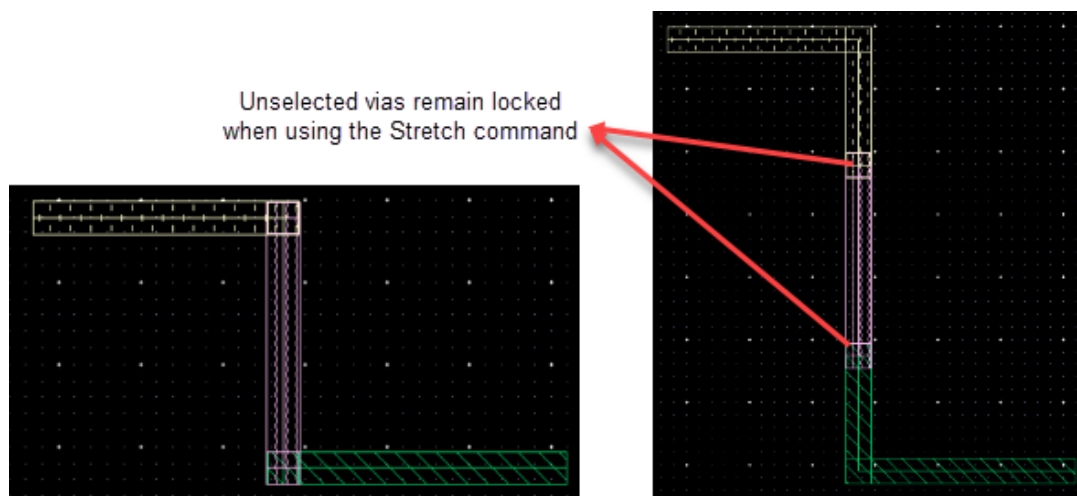
lockUnselectedVias

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

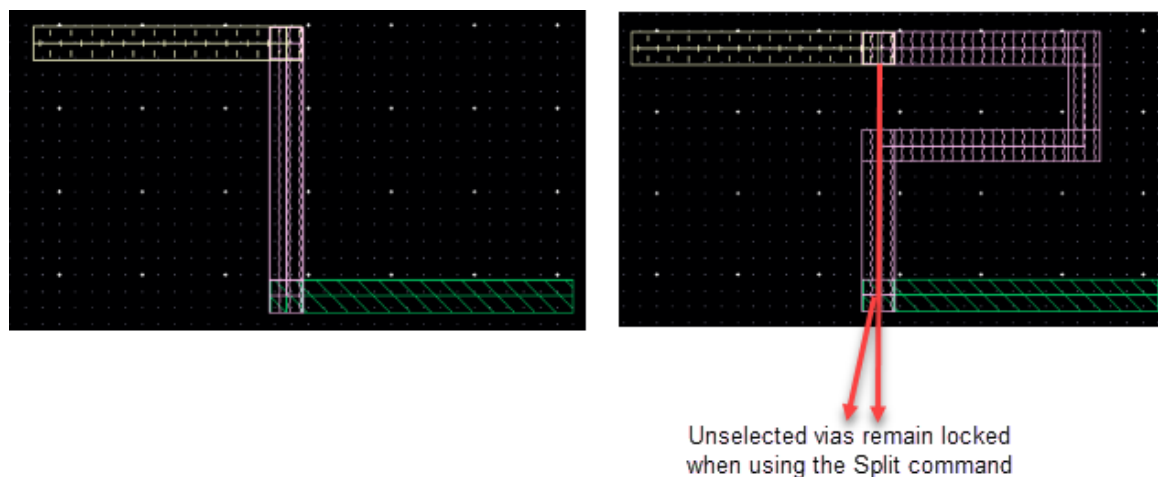
Description

Locks the vias that are not selected while splitting or stretching a wire. The default is `nil`.

The following figure shows the behavior of how the wire is stretched when the unselected vias are locked.



Similarly, the following figure shows the behavior of how the wire is split when the unselected vias are locked.



GUI Equivalent

Command *Edit – Stretch*

Edit – Split

Form Field *Lock Unselected Vias*

Note: The *Lock Unselected Vias* option is enabled only when the *All Wires and Vias* option is selected from the *Keep Wires Connected To* field.

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Metal Density Options Form](#)

[Stretch Shortcut Menu Commands](#)

[Split Wire Shortcut Menu Commands](#)

maxNumConnWireElements

`layout maxNumConnWireElements int wireElements`

Description

Controls the maximum number of connected elements when stretching wires. Value should be a non-negative integer. The default is 5000.

GUI Equivalent

None

Examples

```
envGetVal("layout" "maxNumConnWireElements")  
envSetVal("layout" "maxNumConnWireElements" 'int 1000)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

mixedSignalWireEditingEnvironment

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

Description

Adds the `Geometric Wire` command to the *Create – Shape* menu. The default is `nil`, which means that the `Geometric Wire` command is hidden from the *Create – Shape* menu. You can add the `Geometric Wire` command to the menu by setting this environment variable.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

moveConnectedInstancesAndWires

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

Description

Maintains the connectivity between instances and pathSegs and the selected object while moving or stretching the instances or pathSegs. When enabled, the *Keep Wires Connected to* and *Lock Unselected Vias* options are disabled. The default is `t`.

GUI Equivalent

Command	<i>Edit – Stretch</i>
Form Field	<i>Move Connected Instances and Wires</i>

Examples

```
envGetVal("layout" "moveConnectedInstancesAndWires")  
envSetVal("layout" "moveConnectedInstancesAndWires" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Metal Density Options Form](#)

[Stretch Shortcut Menu Commands](#)

moveConnectedObjectType

```
we moveConnectedObjectType cyclic { "instances" | "pins" | "all" }
```

Description

Maintains connectivity between instances and wires and the selected object type while moving or stretching the instances or pathSegs. This environment variable is used by the *Stretch* command when the *Move Connected Instances and Wires* option is selected.

- `instances`: instances connected by wires are moved together while stretching. T
- `pins`: top-level pins connected by wires are moved together while stretching. Other object types are not moved.
- `all`: instances and pins connected by wires are moved while stretching.

The default is `instances`.

GUI Equivalent

None

Examples

```
envGetVal("we" "moveConnectedObjectType")  
envSetVal("we" "moveConnectedObjectType" 'cyclic "pins")  
envSetVal("we" "moveConnectedObjectType" 'cyclic "all")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Metal Density Options Form](#)

p2pSeedStyle

```
layout p2pSeedStyle cyclic { "Exact Location" | "Shortest Connection" }
```

Description

Controls the type of connection to be followed when creating a point to point route.

- `Exact Location` uses the exact digitized points to create routing between two given points.
- `Shortest Connection` uses the shortest route for connecting points and may also use pre-existing wires for creating the route.

The default is `Shortest Connection`.

GUI Equivalent

Command	<i>Create – Wiring – Point to Point</i>
Field	<i>Connection Type</i>

Examples

```
envGetVal("layout" "p2pSeedStyle")  
envSetVal("layout" "p2pSeedStyle" 'cyclic "Exact Location")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Routing](#)

[Point to Point Form](#)

pathSegBeginExt

layout pathSegBeginExt float *pathSegBeginExtValue*

Description

Sets the value for the beginning extension when you create a path segment with a variable extension. The default is the constraint group value for the path width for the active layer. The value should be a positive integer.

GUI Equivalent

Command	<i>Create – Wiring – Wire/Bus/Stranded Wire</i>
Field	<i>Begin</i>

Examples

```
envGetVal("layout" "pathSegBeginExt")  
envSetVal("layout" "pathSegBeginExt" 'float 1.0)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Edit Bus Form](#)

[Create Stranded Wire Form](#)

[Create Wire Form](#)

[Create Bus Form](#)

pathSegBeginStyle

```
layout pathSegBeginStyle cyclic { "truncate" | "extend" | "variable" | "diagonal" }
```

Description

Sets the beginning extension style that must be used when you create a path segment. The default is `truncate`.

GUI Equivalent

Command	<i>Create – Wiring – Wire/Bus/Stranded Wire</i>
Field	<i>Begin Style</i>

Examples

```
envGetVal("layout" "pathSegBeginStyle")  
envSetVal("layout" "pathSegBeginStyle" 'cyclic "extend")  
envSetVal("layout" "pathSegBeginStyle" 'cyclic "variable")  
envSetVal("layout" "pathSegBeginStyle" 'cyclic "diagonal")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Edit Bus Form](#)

[Create Stranded Wire Form](#)

[Create Bus Form](#)

[Create Wire Form](#)

pathSegEndExt

layout pathSegEndExt float *pathSegEndExtValue*

Description

Sets the value for the ending extension that will be used when you create a path segment with a variable extension. The default is the constraint group value for the path width for the active layer. The value should be a positive integer.

GUI Equivalent

Command	<i>Create – Wiring – Wire/Bus/Stranded Wire</i>
Field	<i>End</i>

Examples

```
envGetVal("layout" "pathSegEndExt")  
envSetVal("layout" "pathSegEndExt" 'float 1.0)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Edit Bus Form](#)

[Create Stranded Wire Form](#)

[Create Wire Form](#)

[Create Bus Form](#)

pathSegEndStyle

```
layout pathSegEndStyle cyclic { "truncate" | "extend" | "variable" | "diagonal" }
```

Description

Sets which ending extension style that must be used when you create a path segment. The default is `truncate`.

GUI Equivalent

Command	<i>Create – Wiring – Wire/Bus/Stranded Wire</i>
Field	<i>End Style</i>

Examples

```
envGetVal("layout" "pathSegEndStyle")
envSetVal("layout" "pathSegEndStyle" 'cyclic "extend"')
envSetVal("layout" "pathSegEndStyle" 'cyclic "variable"')
envSetVal("layout" "pathSegEndStyle" 'cyclic "diagonal"')
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Edit Bus Form](#)

[Create Stranded Wire Form](#)

[Create Wire Form](#)

[Create Bus Form](#)

patternFillAlpha

`we patternFillAlpha int patternFillValue`

Description

Determines the `fillAlpha` parameter given to the `hiliteSet` used for WSP dynamic display (see [geSetHiliteSetAlphaBlendingParameters](#)). The default is 500.

GUI Equivalent

None

Examples

```
envGetVal("we" "patternFillAlpha")
envSetVal("we" "patternFillAlpha" 'int 50)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[geSetHiliteSetAlphaBlendingParameters](#)

patternGravity

```
layout patternGravity cyclic { "Min" | "Standard" | "Max" }
```

Description

Controls the magnet on `snapPattern` and `trackPattern`. It can take three values.

- **Min:** No snapping on track and snap pattern intersections.
- **Standard:** Snaps on the track and snap pattern intersections if the mouse pointer projection on the track axis is within 15 pixels around the intersection.
- **Max:** Always snaps on the track and snap pattern intersections.

The default is `Standard`.

GUI Equivalent

Command	<i>Create Wire/Create Bus Context Sensitive – Use Width Spacing Pattern</i>
Field	<i>Gravity</i>

Examples

```
envGetVal("layout" "patternGravity")  
envSetVal("layout" "patternGravity" 'cyclic "min")  
envSetVal("layout" "patternGravity" 'cyclic "max")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Wire Editor Gravity](#)

patternGravityLayerTransitionType

```
we patternGravityLayerTransitionType cyclic { "patternGravityLayerUp" |  
      "patternGravityLayerDown" | "patternGravityLayerAny" }
```

Description

Controls the dynamic display of the preferred layers on WSP tracks.

- `patternGravityLayerUp` enables the dynamic display of the current layer and the layers above the current layer on WSP tracks. For example, if the current layer is `metal2`, the `metal1` tracks are not displayed. The WSP tracks for layers greater than or equal to `metal2` are displayed.
- `patternGravityLayerDown` enables the dynamic display of the current layer and the layers below the current layer on WSP tracks. For example, if the current layer is `metal2`, the `metal3` tracks are not displayed. The WSP tracks for layers less than or equal to `metal2` are displayed.
- `patternGravityLayerAny` enables the dynamic display of the current layer and the layers above and below the current layer on WSP tracks. For example, if the current layer is `metal2`, the `metal1` and `metal3` layer tracks are displayed.

The default is `patternGravityLayerAny`.

GUI Equivalent

None

Examples

```
envGetVal("we" "patternGravityLayerTransitionType")  
envSetVal("we" "patternGravityLayerTransitionType" 'cyclic  
"patternGravityLayerUp")  
envSetVal("we" "patternGravityLayerTransitionType" 'cyclic  
"patternGravityLayerDown")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[weCyclePatternGravityLayerTransitionType](#)

patternOutlineAlpha

`we patternOutlineAlpha int patternOutlineValue`

Description

Determines the `outlineAlpha` parameter given to the `hiliteSet` used for WSP dynamic display (see [geSetHiliteSetAlphaBlendingParameters](#)). The default is 600.

GUI Equivalent

None

Examples

```
envGetVal("we" "patternOutlineAlpha")
envSetVal("we" "patternOutlineAlpha" 'int 50)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

patternRenderingAperture

`layout patternRenderingAperture int patternRenderingApertureValue`

Description

Controls the display of patterns in the design. If aperture is equal to 0, no adjacent patterns are displayed. However, if a value is specified for aperture, some patterns around the aperture value centered by the snap cursor are displayed. To display all pattern use 10000 or more. The default is 250.

GUI Equivalent

None

Examples

```
envGetVal("layout" "patternRenderingAperture")
envSetVal("layout" "patternRenderingAperture" 'int 500)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Supporting Width Spacing Patterns](#)

pushAndShoveMode

```
we pushAndShoveMode cyclic { "none" | "hugOnly" | "pushWires" | "pushWiresAndVias" }
```

Description

(Layout EXL and higher tiers) Lets you toggle between the push-and-shove modes for wires.

- `none`: indicates that push-and-shove is off.
- `hugOnly`: indicates that the edited wire moves and avoids the other existing shapes. This means that the existing shapes are considered as static and are not pushed.

Note: This mode is not valid for the *Stretch* command. If set, it works similar to `pushWires`.
- `pushWires`: indicates that the existing wires are pushed to create empty space for the edited wire. The wire that is edited moves and avoids the shapes that cannot be pushed, such as pins, bumps, and bond fingers.
- `pushWiresAndVias`: indicates that the existing wires and vias are pushed to avoid violations with the edited wire.

Default is `none`.

GUI Equivalent

Command	<i>Create – Wiring – Wire</i> <i>Edit – Stretch</i>
Form Field	<i>Push and Shove</i>

Examples

```
envGetVal("we" "pushAndShoveMode")  
envSetVal("we" "pushAndShoveMode" 'cyclic "hugOnly")  
envSetVal("we" "pushAndShoveMode" 'cyclic "pushWires")  
envSetVal("we" "pushAndShoveMode" 'cyclic "pushWiresAndVias")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

Push and Shove in Interactive Routing

pushAndShoveSmooth

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

Description

(Layout EXL Only) Controls the *Smooth Wires* setting for the wires modified by push and shove. This applies only to Virtuoso RF designs. The default is `nil`.

GUI Equivalent

Command	<i>Create – Wiring – Wire – Push and Shove</i> context-sensitive menu
	<i>Edit – Stretch – Push and Shove</i> context-sensitive menu
Form Field	<i>Smooth Wires</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Push and Shove in Interactive Routing](#)

pushAndShoveCheckerStopLevel

`we pushAndShoveCheckerStopLevel int pushAndShoveCheckerStopLevelValue`

Description

(Layout EXL Only) Controls the hierarchical levels of the DRC violations that are identified and fixed by push-and-shove:

- 0: avoids violations between edited wires and shapes at the top level and instance pins.
- N: avoids violations between edited wires and shapes from top level to level N.
- -1: uses the top level to control the display of hierarchical levels in DRC checks.

The default is 0.

GUI Equivalent

Command	<i>Create – Wiring – Wire – Push and Shove</i> context-sensitive menu
	<i>Edit – Stretch – Push and Shove</i> context-sensitive menu
Field	<i>Show Remaining Violations</i>

Examples

```
envGetVal("we" "pushAndShoveCheckerStopLevel")  
envSetVal("we" "pushAndShoveCheckerStopLevel" 'int 1)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Push and Shove in Interactive Routing](#)

setTaperMode

```
rte setTaperMode boolean { t | nil }
```

Description

Controls the global taper routing mode. The setting determines whether or not the `virtuosoDefaultTaper` constraint group is used during the *Point to Point* command. The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("rte" "setTaperMode")  
envSetVal("rte" "setTaperMode" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

showAlignmentMarkers

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

Description

Allows display of alignment marks when the pointer aligns with a nearby wire, pin, or via that is on the net you are routing. The default is `nil`.

GUI Equivalent

Command	<i>Options - Editor</i> <i>Wire Assistant – Interactive – Create Wire</i>
Form Field	<i>Show Alignment Markers</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Interactive Section](#)

snapResizeWireLabels

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

Description

Snaps the child labels of pins and pathSegs to the center or centerline and resizes their height to match the width of the pins and pathSegs. The default is `nil`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

snapToPin

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

Description

Snaps the starting point of wires to the origin of pins. The default is `nil`.

GUI Equivalent

Command	<i>Create – Wiring – Wire/Bus</i>
Field	<i>Snap to Pin Center</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Edit Bus Form](#)

[Create Stranded Wire Form](#)

snapToViaCenter

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

Description

Snaps the wire to the center of via. The default is `nil`.

GUI Equivalent

Command	<i>Create – Wiring – Wire/Bus</i>
Field	<i>Snap to Via Center</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Edit Bus Form](#)

[Create Stranded Wire Form](#)

snapWireGrid

```
layout snapWireGrid cyclic { "Manufacturing Grid" | "Routing Grid" | "Off grid" |  
    "Track Pattern" | "Snap Pattern" }
```

Description

Controls the snapping of wires to a grid while using the interactive wire editing and assisted routing commands. The default is `Manufacturing Grid`.

GUI Equivalent

Command	<i>Options – Layout Editor</i>
Field	<i>Wires</i>

Examples

```
envGetVal("layout" "snapWireGrid")  
envSetVal("layout" "snapWireGrid" 'cyclic "Routing Grid")  
envSetVal("layout" "snapWireGrid" 'cyclic "Off grid")  
envSetVal("layout" "snapWireGrid" 'cyclic "Track Pattern")  
envSetVal("layout" "snapWireGrid" 'cyclic "Snap Pattern")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

spaceFuncName

```
layout spaceFuncName string "functionname"
```

Description

Binds the name of the specified function to the space bar. The `leArrowFunc` function executes the function named by this environment variable, and the `leHiCreateWire` function automatically sets `spaceFuncName` to `leHiAddWireVia`, which accepts `Up` or `Down` to select the default via leading to the next layer above or below the current layer. The default value is an empty string `""`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "spaceFuncName")  
envSetVal("layout" "spaceFuncName" 'string "myfunction")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

taperToPinWidth

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

Description

When set to `t`, the *Taper Pin Width* check box is selected by default. The default is `nil`.

GUI Equivalent

Command	<i>NA</i>
Field	<i>Taper Pin Width</i> check box

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

taperToPinWidthStyle

```
layout taperToPinWidthStyle cyclic { "to_first_via" | "min_taper_halo" |  
    "within_pin_halo" }
```

Description

Enables the router to create pathSegs connecting to top-level pins or instance pins using a width that matches the target pin. The default is `within_pin_halo`.

GUI Equivalent

Command	<i>Window – Assistant – Wire Assistant – Override Constraints – Pin Escape tab</i>
Field	<i>Mode</i>

Examples

```
envGetVal("layout" "taperToPinWidthStyle")  
envSetVal("layout" "taperToPinWidthStyle" 'cyclic "to_first_via")  
envSetVal("layout" "taperToPinWidthStyle" 'cyclic "min_taper_halo")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

tapPreference

```
layout tapPreference cyclic { "noPrefer" | "wire" | "pin" }
```

Description

Controls the tapped objects that are filtered out if there is more than one object in the tapped list that contains both preferred and non-preferred objects.

- If set to `noPrefer`, no filtering is done.
- If set to `wire`, filters out any non-wire tapped objects.
- If set to `pin`, filters out any non-pin tapped objects.

If the tapped list contains valid route objects but none of them is a preferred object, then the interactive routing ignores the `tapPreference` setting.

The default is `noPrefer`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "tapPreference")  
envSetVal("layout" "autoTapActiveLayer" 'cyclic "wire")  
envSetVal("layout" "autoTapActiveLayer" 'cyclic "pin")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

useMergeBusAndWire

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

Description

When set to `t`, lets you select multiple pins and when you start the create wire command it switches to create bus automatically. However, when this variable is set to `nil` and you start the create wire command, only a single wire is created. The default is `nil`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

useOverlapRegionForVia

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

Description

Creates square vias to connect collinear wires. The default is `t`.

GUI Equivalent

Command	<i>Route – Design Setup – Routability Check</i> <i>Window – Assistant – Wire Assistant – Automatic – Optimization</i> <i>Window – Assistant – Wire Assistant – Net – Via Configuration</i>
Field	<i>Maximize Cuts</i>

Examples

```
envGetVal("layout" "useOverlapRegionForVia")  
envSetVal("layout" "useOverlapRegionForVia" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

viaAlignment

```
layout viaAlignment cyclic { "none" | "top-left" | "top-center" | "top-right" |  
    "center-left" | "center-center" | "center-right" | "bottom-left" | "bottom-  
    center" | "bottom-right" }
```

Description

Specifies a custom via alignment option. The default is `auto`.

GUI Equivalent

Command	<i>Create Wire Context Sensitive menu – Via</i> <i>Create Bus Context Sensitive menu – Via</i> <i>Select Via form</i>
Field	<i>Via Alignment - Override</i>

Examples

```
envGetVal("layout" "viaAlignment")  
envSetVal("layout" "viaAlignment" 'cyclic "none")  
envSetVal("layout" "viaAlignment" 'cyclic "top-left")  
envSetVal("layout" "viaAlignment" 'cyclic "center-right")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Create Stranded Wire Form](#)

[Edit Bus Form](#)

[Metal Density Options Form](#)

Virtuoso Interactive and Assisted Routing User Guide

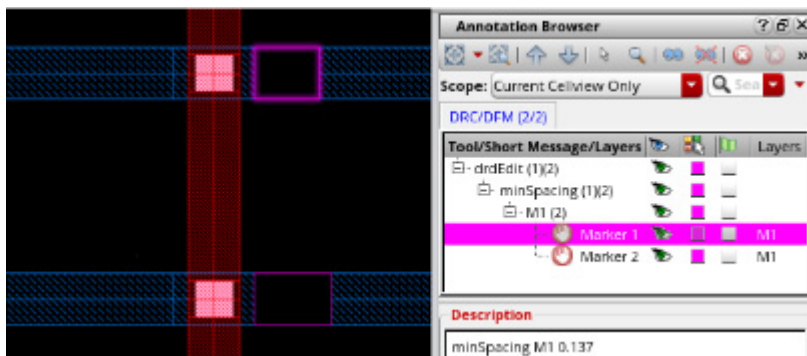
Interactive and Assisted Routing Environment Variables

viaCheckDRC

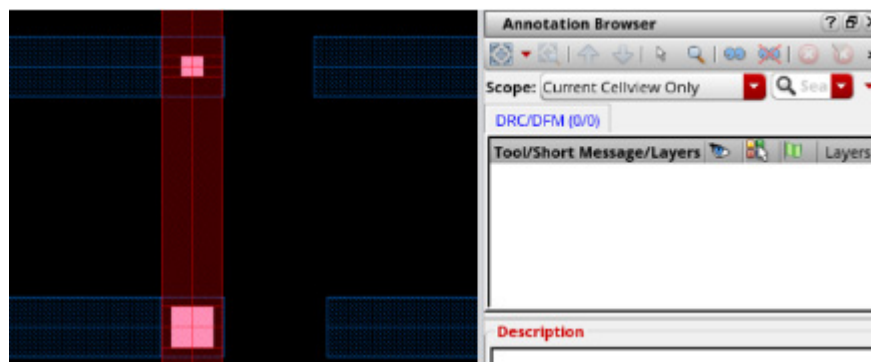
```
layout viaCheckDRC boolean { t | nil }
```

Description

Creates DRC clean vias for routed wires with neighboring shapes. The default is `nil`. In this case, the different solutions to create vias are not considered and results in DRC violations.



When the variable is set to `t`, the different solutions to create vias are considered and prevent DRC violations.



GUI Equivalent

Command	Routing assistant – <i>Interactive</i> – <i>Wire</i> tab
Field	<i>DRC Check</i>

Examples

```
envGetVal("layout" "viaCheckDRC")
```

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

```
envSetVal("layout" "viaCheckDRC" 'boolean t)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

viaCutColorMode

```
we viaCutColorMode cyclic { "off" | "on" | "pseudo" }
```

Description

Lets you cycle the color of each cut when a via is inserted during the *Create Wire*, *Create Bus*, and *Create Stranded Wire* commands.

- `off`: disables cut coloring, which means that the same color is used for each cut when a via is inserted.
- `on`: enables cut coloring and cycles the color of each cut when a via is inserted.
- `pseudo`: considers cut coloring in internal interactive routing data structures while applying routing constraints but does not write out cut coloring in interactive routing commands.

The default is `off`.

GUI Equivalent

Command	<i>Create – Wiring – Wire/Bus/Stranded Wire</i>
Field	<i>Via – Cut Color</i>

Examples

```
envGetVal("we" "viaCutColorMode")
envSetVal("we" "viaCutColorMode" 'cyclic "on")
envSetVal("we" "viaCutColorMode" 'cyclic "psuedo")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

viaCutColorPatterns

```
we viaCutColorPatterns string "patternName"
```

Description

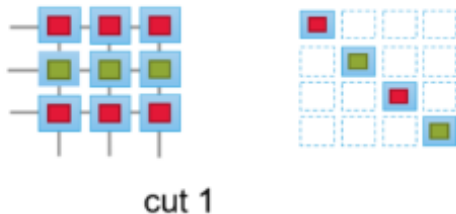
Defines a cut coloring pattern to be used for layer transition. Using this environment variable, you can specify the list of cut layer names. The cut layer uses a basic color pattern: alternate row, column, chessboard, and uniform. If no coloring pattern is defined for a cut layer, by default the uniform color pattern is considered.

For example, to define the coloring pattern for a cut layer you could use the following:

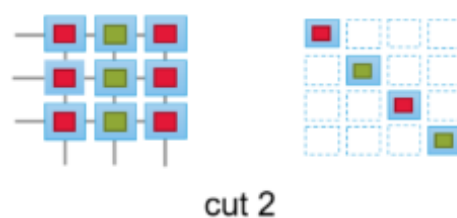
```
envSetVal ("we" "weViaCutColorPatterns" 'string "(cut1 row) (cut2 column)")
```

The pattern defined in the above example specifies that for `cut1` layer apply alternate row and for `cut2` layer apply alternate column.

row: alternate row (start to mask1)



column: alternate column (start to mask1)



GUI Equivalent

None

Examples

```
envGetVal ("we" "viaCutColorPatterns")  
envSetVal ("we" "viaCutColorPatterns" 'string "(cut1LayerName col) (cut2LayerName  
row) ")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

viaExtendEnclosureBeyondOverlap

```
we viaExtendEnclosureBeyondOverlap cyclic { "no" | "yes" | "sameAsCreateVia" }
```

Description

Lets you extend the via enclosures beyond the overlap region between two intersecting shapes. The shape layer that exists beyond the overlap region helps to accommodate the extended enclosures.

- `no` does not extend the via enclosure to cover the complete overlap area between the two wires.
- `yes` extends the via enclosure.
- `sameAsCreateVia` extends or does not extend the via enclosure depending on the value of the `viaFillEnclosureOverlap` environment variable.

The default is `sameAsCreateVia`.

GUI Equivalent

None

Examples

```
envGetVal("we" "viaExtendEnclosureBeyondOverlap")  
envSetVal("we" "viaExtendEnclosureBeyondOverlap" 'cyclic "no")  
envSetVal("we" "viaExtendEnclosureBeyondOverlap" 'cyclic "yes")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

viaFillEnclosureOverlap

```
we viaFillEnclosureOverlap cyclic { "no" | "yes" | "sameAsCreateVia" }
```

Description

Controls how vias are generated in the interactive routing commands, such as *Create Wire*, *Stretch*, *Create Bus*, and *Create Stranded Wire*.

- `no` does not extend the via enclosure to cover the complete overlap area between the two wires.
- `yes` extends the via enclosure.
- `sameAsCreateVia` extends or does not extend the via enclosure depending on the value of the `viaFillEnclosureOverlap` environment variable.

The default is `no`.

GUI Equivalent

None

Examples

```
envGetVal("we" "viaFillEnclosureOverlap")  
envSetVal("we" "viaFillEnclosureOverlap" 'cyclic "sameAsCreateVia")  
envSetVal("we" "viaFillEnclosureOverlap" 'cyclic "yes")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

viaMaxNumCuts

```
layout viaMaxNumCuts int viaMaxNumCutsValue
```

Description

Specifies the maximum number of cuts that can be added to a via.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

viaPattern

```
layout viaPattern cyclic { "unknown" | "perp" | "diag_up" | "diag_down" | "stagger"  
    | "out_taper" | "in_taper" | "auto_no_crossing" | "auto_crossing" }
```

Description

Specifies a pre-defined manner to add multiple vias simultaneously when routing multiple wires. The default is `auto_no_crossing`.

GUI Equivalent

Command	<i>Create Bus Context Sensitive menu – Via</i> <i>Create Bus Context Sensitive menu – Via</i> <i>Select Via form</i>
Field	<i>Via Pattern</i>

Examples

```
envGetVal("layout" "viaPattern")  
envSetVal("layout" "viaPattern" 'cyclic "perp")  
envSetVal("layout" "viaPattern" 'cyclic "in_taper")  
envSetVal("layout" "viaPattern" 'cyclic "stagger")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Create Stranded Wire Form](#)

[Edit Bus Form](#)

[Metal Density Options Form](#)

weAllowOddWidth

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

Description

Enables the *Create Wire* command to create wires with odd width, such as 0.27. When set to *nil*, a wire with an even width is created. The default is *t*.

GUI Equivalent

None

Examples

```
envGetVal("layout" "weAllowOddWidth")  
envSetVal("layout" "weAllowOddWidth" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weAreaBasedRulesEnabled

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

Description

Enables the area-based rules support during interactive routing. The default is `t`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Support for Area-based Rules](#)

weAreaBasedRulesHierLevel

```
we weAreaBasedRulesHierLevel int areaBasedRulesHierLevel
```

Description

Specifies the traverse level to detect the ABR layer region and to establish how deep inside the hierarchy wire editor should check for area-based rules. The default is 1.

GUI Equivalent

None

Examples

```
envGetVal("we" "weAreaBasedRulesHierLevel")  
envSetVal("we" "weAreaBasedRulesHierLevel" 'int 2)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Support for Area-based Rules](#)

weAreaBasedRulesWidthMode

```
we weAreaBasedRulesWidthMode cyclic { "commonWidth" | "nonCommonWidth" }
```

Description

Controls the width of a pathSeg when the pathSeg spans across different regions.

- `commonWidth` uses the same common width when a pathSeg spans across regions.
- `nonCommonWidth` splits the wire across regions. Each part of the pathSeg can have different widths in different regions.

The default is `commonWidth`.

GUI Equivalent

None

Examples

```
envGetVal("we" "weAreaBasedRulesWidthMode")  
envSetVal("we" "weAreaBasedRulesWidthMode" 'cyclic "nonCommonWidth")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Support for Area-based Rules](#)

weARWidthMode

```
layout weARWidthMode cyclic { "constraints" | "overrideAllLayers" |  
    "overrideByLayer" | "sources" }
```

Description

Sets a value for the width of the current wire depending on the width mode specified when the *Point to Point* or *Guided Routing* command is selected.

- `constraint` sets the wire width same as the result of the constraint lookup for `minWidth` value in the technology file or in any applicable constraint group defined at the design or object level.
- `overrideAllLayers` uses the same wire width for all layers.
- `overrideByLayer` sets the wire width the same as the result of the constraint lookup for `minWidth` in the technology file when the layer is not used.
- `sources` uses the width determined by the width of the tapped object.

The default is `overrideByLayer`.

GUI Equivalent

Command	<i>Create – Wiring – Point to Point</i>
Field	<i>Context Sensitive menu – Use Width</i>

Examples

```
envGetVal("layout" "weARWidthMode")  
envSetVal("layout" "weARWidthMode" 'cyclic "constraints")  
envSetVal("layout" "weARWidthMode" 'cyclic "overrideAllLayers")  
envSetVal("layout" "weARWidthMode" 'cyclic "sources")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Shortcut Menu Commands](#)

weARTapWideEdge

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

Description

Matches the last segment wire width to the length of the longest edge of the target pin. The width of the remaining wire is unaffected. The default is `t`.

GUI Equivalent

Command	<i>Point to Point Context Sensitive menu – Use Width</i>
Field	<i>Tap Wide Edge</i>

Examples

```
envGetVal("layout" "weARTapWideEdge")  
envSetVal("layout" "weARTapWideEdge" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Shortcut Menu Commands](#)

weARTapWidthOnPins

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

Description

Allows tapping the width on the starting object for a new route when the starting object is a pin. This environment variable can be set for width tapping when *Point to Point* command is selected. The default is *t*.

GUI Equivalent

Command	<i>Point to Point Context Sensitive menu – Use Width</i>
Field	<i>Tap Pins Width</i>

Examples

```
envGetVal("layout" "weARTapWidthOnPins")  
envSetVal("layout" "weARTapWidthOnPins" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Shortcut Menu Commands](#)

weARTapWidthOnWires

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

Description

Allows tapping the width on the starting object for a new route when the starting object is a wire. This environment variable can be set for width tapping when *Point to Point* command is selected. The default is `t`.

GUI Equivalent

Command	<i>Point to Point Context Sensitive menu – Use Width</i>
Field	<i>Tap Wires Width</i>

Examples

```
envGetVal("layout" "weARTapWidthOnWires")  
envSetVal("layout" "weARTapWidthOnWires" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Shortcut Menu Commands](#)

weAutoDropVia

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

Description

Enables automatic insertion of a via while crossing a pathseg on a layer above or below the same net. The default is `nil`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weAutoDropViaForOverlaps

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

Description

Enables automatic insertion of a via while crossing one pathseg over the other pathSeg on a different layer and between pathSeg and a top-level pin or instance pin. The default is `nil`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weAutoMergeWires

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

Description

Merges the connected geometric paths automatically. Also, allows you to choose whether or not to merge the collinear geometric pathsegs while creating or stretching a geometric wire. The default is `t`.

GUI Equivalent

Command	<i>Options – Editor</i>
Form Field	<i>Auto Merge Wires</i>

Examples

```
envGetVal("layout" "weAutoMergeWires")  
envSetVal("layout" "weAutoMergeWires" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

weAutoWidthModeAfterVias

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

Description

Controls if the width mode should automatically change after dropping a via or if it should preserve the width mode set before dropping the via. The default is `nil`, which means that the width mode before the via drop is preserved.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weBusColorMode

```
layout weBusColorMode cyclic { "optimized" | "uniform" }
```

Description

Controls the color mode for the *Create Bus* command.

- **optimized** the control wire acquires the same mask color as that specified for the active layer in the Palette. The other bus bits acquire the following coloring pattern:
 - The bus bits above the control wire acquire the mask color as `maskColor(controlWire)+1`, `maskColor(controlWire)+2`, and so on. This pattern continues until the highest mask color available for the selected layer-purpose pair is assigned to a bus bit. The pattern is then repeated starting at the mask color of control wire.
 - The bus bits below the control wire acquire the mask color as `maskColor(controlWire)-1`, `maskColor(controlWire)-2`, and so on. If the control wire has the lowest available mask color assigned to it, then the bus bits below the control wire are colored starting with the highest mask color available for the layer on which the bus is being placed. The pattern is then repeated for the remaining bus bits.
- **uniform** assigns the same mask color as that of the control wire to all bus bits. The mask color assigned to the control wire and other bus bits is the same as the active layer in the Palette.

The default is `optimized`.

GUI Equivalent

Command	<i>Create – Wiring – Bus – Create Bus</i> context-sensitive menu
Form Field	<i>Use Color Mode</i>

Examples

```
envGetVal("we" "weBusColorMode")  
envSetVal("we" "weBusColorMode" 'cyclic "uniform")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

Interactive Coloring for Create Bus

weBusNumBits

```
layout weBusNumBits int busNumBits
```

Description

Specifies the number of wires in the bus. The default is 2.

GUI Equivalent

Command	<i>Create – Wiring – Bus</i>
Field	<i>Number of Bits</i>

Examples

```
envGetVal("layout" "weBusNumBits")  
envSetVal("layout" "weBusNumBits" 'int 100)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Edit Bus Form](#)

weChangeWireDirectionAfterVia

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

Description

Determines whether the direction of the outgoing wires changes after vias are inserted. When vias are being dropped, hold down the `Ctrl` key and press the middle mouse button to toggle the value of the environment variable between `t` and `nil`.

When set to `t`, the default via pattern, during via up and via down, is diagonal for *Auto No Crossing*. Also, the direction of the outgoing wires is orthogonal to the incoming wires.

When set to `nil`, the via pattern, during via up and via down, is staggered for *Auto No Crossing*. Also, the direction of the outgoing wires is the same as that of the incoming wires.

The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "weChangeWireDirectionAfterVia")  
envSetVal("layout" "weChangeWireDirectionAfterVia" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weCoverPin

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

Description

When set to `t`, adjusts the wire segment to automatically cover the source and target pins. Else, the wire segment is not adjusted and the wire terminates at a point chosen by you. The default is `nil`.

GUI Equivalent

Command	<i>Create – Wiring – Bus</i>
Form Field	<i>Cover Pin</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Edit Bus Form](#)

weCreateBusConstraint

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

Description

Creates a bus constraint automatically for the nets you specify by using the `netName` environment variables or for the ones you pick by selecting multiple pins. The default is `nil`.

GUI Equivalent

Command	<i>Create – Wiring – Bus</i>
Field	<i>Create Bus Constraint</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Edit Bus Form](#)

weCustomLayerPattern

```
layout weCustomLayerPattern string "patternName"
```

Description

Specifies the pattern to be used for creating a multi-layer bus starting from an empty space. In case, no pattern is specified for a multi-layer bus creation, then the custom layer pattern is disabled and a bus is created on a single layer.

Setting the variable, updates the corresponding option in the Create Bus form. Similarly, specifying a pattern in the Create Bus form also updates the variable setting.

GUI Equivalent

Command	<i>Create – Wiring – Bus</i>
Field	<i>Custom Layer Pattern</i>

Examples

```
envGetVal("layout" "weCustomLayerPattern")  
envSetVal("layout" "weCustomLayerPattern" 'string "(M1 M2) (M2 M3)")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Edit Bus Form](#)

[Creating a Multi-Layer Bus from an Empty Space](#)

[Defining a Custom Layer Pattern](#)

weCWStretchLastPointModifier

```
layout weCWStretchLastPointModifier cyclic { "disabled" | "shift" | "ctrl" |  
    "ctrlShift" }
```

Description

Controls how to enable the stretch from the last point on the wire. For example:

```
envSetVal("layout" "weCWStretchLastPointModifier" 'cyclic "shift" )
```

In this case, you can only stretch a wire while keeping the `Shift` key pressed. Therefore, when you press the `Ctrl+Shift` key and scroll the mouse wheel up or down, the stretch is not enabled.

The default is `ctrlShift`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "weCWStretchLastPointModifier")  
envSetVal("layout" "weCWStretchLastPointModifier" 'cyclic "disabled")  
envSetVal("layout" "weCWStretchLastPointModifier" 'cyclic "shift")  
envSetVal("layout" "weCWStretchLastPointModifier" 'cyclic "ctrl")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weCWWidthMode

```
layout weCWWidthMode cyclic { "constraints" | "overrideAllLayers" |  
    "overrideByLayer" | "sources" | "targets" }
```

Description

Sets a value for the width of the current wire depending on the width mode specified.

- **constraint:** Sets the wire width same as the result of the constraint lookup for `minWidth` value in the technology file or in any applicable constraint group defined at the design or object level.
- **overrideAllLayers:** Uses the same wire width for all layers.
- **overrideByLayer:** Sets the wire width the same as the result of the constraint lookup for `minWidth` in the technology file when the layer is not used.
- **sources:** Uses the width determined by the width of the tapped object.
- **targets:** Sets the wire to match the width of the same edge of each respective target pin.

The default is `overrideByLayer`.

GUI Equivalent

Command	<i>Create – Wiring – Wire/Bus/Stranded Wire</i>
Field	<i>Context Sensitive menu – Use Width</i>

Examples

```
envGetVal("layout" "weCWWidthMode")  
envSetVal("layout" "weCWWidthMode" 'cyclic "overrideAllLayers")  
envSetVal("layout" "weCWWidthMode" 'cyclic "constraints")  
envSetVal("layout" "weCWWidthMode" 'cyclic "sources")  
envSetVal("layout" "weCWWidthMode" 'cyclic "targets")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weEnableDanglingTargetsFL

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

Description

Enables the flightline engine to connect to the end of the island instead of connecting to the middle of the island. The default is `nil`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weEnableXLCheckoutInL

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

Description

Checks out an XL license for some interactive routing features, such as the flightlines. These features are also available in Layout L. The default is `nil`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weEnforceDirFromRoutingPattern

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

Description

Controls the width and the direction of the active WSP pattern. The default is `t`.

GUI Equivalent

Command	<i>Options – Layout Editor</i>
Form Field	<i>Enforce</i>

Examples

```
envGetVal("layout" "weEnforceDirFromRoutingPattern")  
envSetVal("layout" "weEnforceDirFromRoutingPattern" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

[Snapping Behavior in WSP During Wire Editing](#)

weFlightLinesHierLevel

layout weFlightLinesHierLevel int *flightLinesHierLevel*

Description

Determines the threshold value above which the display mode `ON` (for all flightlines) automatically switches to the display mode `minimum` (for bus extremities and control wires flightlines). The default is 0.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weGuidedRouteGatherAtBend

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

Description

Matches the width of the first and last segments of the routing envelope to the width of the pin envelopes they connect to. The default is `t`.

GUI Equivalent

Command	<i>Create – Wiring – Guided Routing</i>
Field	<i>Gather At Bend</i>

Examples

```
envGetVal("layout" "weGuidedRouteGatherAtBend")  
envSetVal("layout" "weGuidedRouteGatherAtBend" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

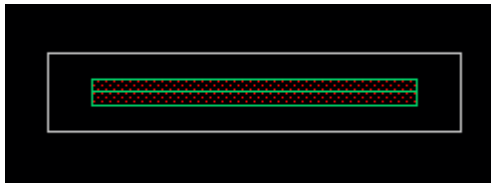
weHalo

```
we weHalo cyclic { "minEOL" | "maxEOL" | "bothEOL" }
```

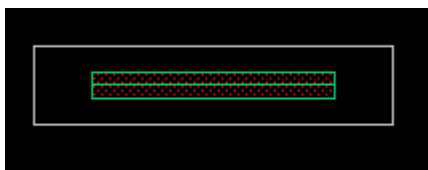
Description

Includes the `minEndOfLineSpacing` constraint value to the halo. When `minEndOfLineSpacing` constraint is available for a via or a path segment, the halo becomes a polygon and is not a rounded halo.

- `minEOL` the halo shows the `diffMask` or the `sameMask` spacing for the `minEndOfLineSpacing` constraint.
If `diffMask` spacing is less than the `sameMask` spacing, `minEOL` shows the `diffMask` spacing.
If `sameMask` spacing is less than the `diffMask` spacing, `minEOL` shows the `sameMask` spacing.



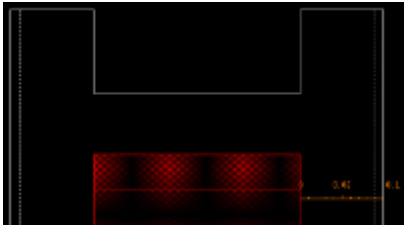
- `maxEOL` the halo shows the `diffMask` or the `sameMask` spacing for the `minEndOfLineSpacing` constraint.
If `diffMask` spacing is greater than the `sameMask` spacing, `maxEOL` shows the `diffMask` spacing.
If `sameMask` spacing is greater than the `diffMask` spacing, `maxEOL` shows the `sameMask` spacing.



Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

- `bothEOL` the halo shows the minimum (`diffMask`, `sameMask`) spacing and the maximum (`diffMask`, `sameMask`) spacing of the available `minEndOfLineSpacing` constraint.



The Default is `minEOL`.

GUI Equivalent

Command	<i>Options – Editor</i>
Form Field	<i>Halo</i>

Examples

```
envGetVal("we" "weHalo")
envSetVal("we" "weHalo" 'cyclic "maxEOL")
envSetVal("we" "weHalo" 'cyclic "bothEOL")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

weHaloDashStyle

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

Description

Creates a dashed halo around a wire, bus, or stranded wire while creating or editing an object that violates a spacing constraint. The default is `nil`.

GUI Equivalent

Command	<i>Options – Editor</i>
Form Field	<i>Dashed</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

weHaloEnabled

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

Description

Lets you enable or disable a halo display in the layout canvas. When the option is unselected, the halo is not shown when the wire is created. The default is `t`.

GUI Equivalent

Command	<i>Options – Editor</i>
Form Field	<i>Halo – Enable</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

weHaloEOLSpacing

```
we weHaloEOLSpacing cyclic { "endToSide" | "endToEnd" | "both" }
```

Description

Determines which halo to show for `minEndOfLineSpacing` constraint.

- `endToSide` shows the end-to-side halo spacing for the `minEndOfLineSpacing` constraint.
- `endToEnd` shows the end-to-end halo spacing for the `minEndOfLineSpacing` constraint.
- `both` shows both end-to-side and end-to-end halo spacing for the `minEndOfLineSpacing` constraint.

The default is `endToSide`.

GUI Equivalent

Command	<i>Options – Editor</i>
Form Field	<i>Halo</i>

Examples

```
envGetVal("we" "weHaloEOLSpacing")  
envSetVal("we" "weHaloEOLSpacing" 'cyclic "endToEnd")  
envSetVal("we" "weHaloEOLSpacing" 'cyclic "both")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

weHaloTrueColor

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

Description

Creates a halo in the color of that layer for each layer. This makes it easier to differentiate between halos on different layers. The default is `nil`.

GUI Equivalent

Command	<i>Options – Editor</i>
Form Field	<i>True Color</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

weInstancePinPattern

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

Description

Controls the usage of the patterns in the hierarchy. The default is `nil`.

GUI Equivalent

Command	<i>Options – Layout Editor</i>
Form Field	<i>Instance Pins Patterns</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Wire Editor Hierarchy Support](#)

[Snapping Behavior in WSP During Wire Editing](#)

weInteractiveShielding

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

Description

Enables interactive shielding in case the shielding constraint has been defined in the constraint manager for a net. The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("layout" " weInteractiveShielding ")  
envSetVal("layout" " weInteractiveShielding " 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weKeepSelectViaOrder

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

Description

Displays vias in the *Select Via* form in the same order as they are defined in the `validVias` section of the technology file. When set to `t`, the order is not maintained and the more frequently selected vias are displayed at the top of the list in the *Select Via* form. The default is `nil`.

GUI Equivalent

None

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weMemoryOptimization

```
layout weMemoryOptimization cyclic { "none" | "low" | "medium" | "high" }
```

Description

Enables performance optimization during Wire Editing in Layout XL and higher tiers. The following Wire editing commands are impacted: *Create Wire*, *Create Bus*, *Stretch*, and *Reshape*. A dialog box suggesting you to set `weMemoryOptimization` to `high` is displayed once per session when using a wire editing command and `weMemoryOptimization` is set to `low` or `medium`.

- `none`: When set to `none`, the advanced routing data-structures are built for the entire design with all hierarchical levels when opening a cellview in Layout XL and higher tiers. This improves the results of assisted routing commands but increases memory as well.
- `low`: when set to `low`, the interactive Wire Editing commands as well as the stretch command on wires build internal routing data-structures for the entire design. This requires additional time and the memory increases the first time the command is invoked. However, the next editions are faster.

The memory is populated with all the routing data only once in the same session when in Layout XL and higher tiers. Discarding the edits cleans up the routing data from the memory.

- `medium`: when set to `medium`, the interactive Wire Editing commands as well as the stretch command on wires build internal routing data-structures only in the edited area needed for the current edition. Therefore, there should be no performance difference between the first and other editing commands in the design. The first edition is then faster as compared to `low` mode. However, the other editions are a bit slower when compared to `low`, as the memory is populated incrementally based on the new editing contexts.
- `high`: when `weMemoryOptimization` is set to `high`, the advanced routing data-structures are loaded only when using the advanced routing functionalities, such as speciality routing, or DRD. This improves the performance and memory when the interactive wire editing commands, such as *Create Wire*, *Create Bus*, *Stretch*, *Reshape*, are used.

The default is `high`.

GUI Equivalent

None

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

Examples

```
envGetVal("layout" "weMemoryOptimization")
envSetVal("layout" "weMemoryOptimization" 'cyclic "none")
envSetVal("layout" "weMemoryOptimization" 'cyclic "low")
envSetVal("layout" "weMemoryOptimization" 'cyclic "medium")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weMfgGridWidthIncrement

layout weMfgGridWidthIncrement int *MfgGridWidthValue*

Description

Specifies the value to increase or decrease the width of the wires being created by the *Create Wire*, *Create Bus*, or *Create Stranded Wire* commands. To apply the increment or decrement value, use the middle mouse button. The default value is 2 dbu.

GUI Equivalent

None

Examples

```
envGetVal("layout" "weMfgGridWidthIncrement")  
envSetVal("layout" "weMfgGridWidthIncrement" 'int 5)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weNoSmartSnap

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

Description

Disables the smart snapping mode while creating a wire. The default is `nil`.

GUI Equivalent

Command	<i>Create Single Wire Context Sensitive menu</i>
Field	<i>Smart Snapping – Enabled</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Create Stranded Wire Form](#)

wePostCriticWires

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

Description

Lets you straighten wires where possible after routing. The default is `nil`.

GUI Equivalent

Command	<i>Window – Assistant – Wire Assistant – Interactive – Finish</i>
Field	<i>Straighten Wires</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Interactive Section](#)

weShowCtrlWireAlignMarkerOnly

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

Description

Displays the alignment marker only if the control wire is aligned with its target pin. The default is t.

GUI Equivalent

Command	<i>Window – Assistant – Wire Assistant – Interactive – Create Bus</i>
Field	<i>Controlled Wire Only</i>

Examples

```
envGetVal("layout" "weShowCtrlWireAlignMarkerOnly")  
envSetVal("layout" "weShowCtrlWireAlignMarkerOnly" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weShowHints

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

Description

Controls the display of the *Interactive Routing Bindkey* hint box in Layout EXL when you start the *Create Wire*, *Create Bus*, or *Create Stranded Wire* command. In Layout XL, the hint box is displayed only for the *Create Stranded Wire* command. The default is `t`.

GUI Equivalent

Command	<i>Options – Editor</i>
Form Field	<i>Show Hints</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weSmartSnappingDepth

```
layout weSmartSnappingDepth int smatSnappingDepth
```

Description

Specifies the depth for smart snapping of wires. It is available in Layout XL and higher tiers. The default is 1.

GUI Equivalent

None

Examples

```
envGetVal("layout" " weSmartSnappingDepth ")  
envSetVal("layout" " weSmartSnappingDepth " 'int 5)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weStrandedAlignCollinearMode

```
layout weStrandedAlignCollinearMode cyclic { "topRight" | "center" | "bottomLeft" }
```

Description

Controls the tapering alignment mode of the stranded wire.

- `topRight` aligns the stranded wire to the top when the wire is dragged in the horizontal direction and to the right when wire is dragged in the vertical direction.
- `center` aligns the stranded wire to the center when the number of strands are reduced or increased.
- `bottomLeft` aligns the stranded wire to the bottom when the wire is dragged in the horizontal direction and to the left when wire is dragged in the vertical direction.

The default is `center`.

GUI Equivalent

Command	<i>Create – Wiring – Bus – Create Bus</i> context-sensitive menu
Form Field	<i>Use Taper Alignment</i>

Examples

```
envGetVal("we" "weStrandedAlignCollinearMode")  
envSetVal("we" "weStrandedAlignCollinearMode" 'cyclic "topCenter")  
envSetVal("we" "weStrandedAlignCollinearMode" 'cyclic "bottomLeft")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Creating a Stranded Wire](#)

[Tapering in Stranded Wire](#)

weStrandedColorMode

```
layout weStrandedColorMode cyclic { "optimized" | "uniform" }
```

Description

Controls the color mode for the *Create Stranded Wire* command.

- **optimized** the control wire acquires the same mask color as that specified for the active layer in the Palette. The other stranded wires acquire the following coloring pattern:
 - The stranded wires above the control wire acquire the mask color as `maskColor(controlWire)+1, maskColor(controlWire)+2`, and so on. This pattern continues until the highest mask color available for the selected layer-purpose pair is assigned to the stranded wire. The pattern is then repeated starting at the mask color of control wire.
 - The stranded wires below the control wire acquire the mask color as `maskColor(controlWire)-1, maskColor(controlWire)-2`, and so on. If the control wire has the lowest available mask color assigned to it, then the stranded wires below the control wire are colored starting with the highest mask color available for the layer on which the stranded wire is being placed. The pattern is then repeated for the remaining stranded wires.
- **uniform** assigns the same mask color as that of the control wire to all stranded wires. The mask color assigned to the control wire and other stranded wires is the same as the active layer in the Palette.

The default is `optimized`.

GUI Equivalent

Command	<i>Create – Wiring – Stranded Wire– Create Stranded Wire</i> context-sensitive menu
Form Field	<i>Use Color Mode</i>

Examples

```
envGetVal("we" "weStrandedColorMode")  
envSetVal("we" "weStrandedColorMode" 'cyclic "uniform")
```

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Interactive Coloring for Create Stranded Wire](#)

weStrandedLadderAtTurn

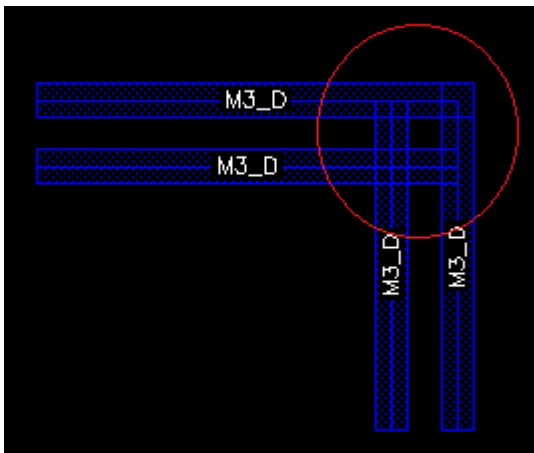
```
layout weStrandedLadderAtTurn cyclic { "always" | "none" | "switchLayer" }
```

Description

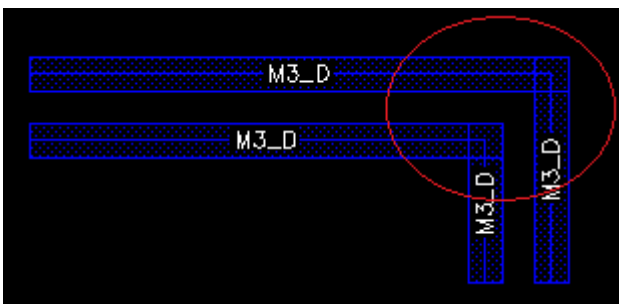
Controls the creation of ladder at the bend when creating a stranded wire. The default is `always`.

This environment variable can take three values:

- `always` creates a ladder at the turn, as shown in the following figure.



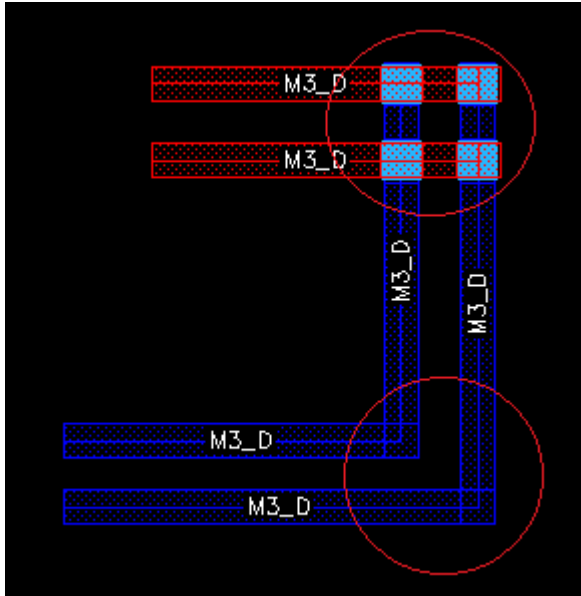
- `none` does not create a ladder at the turn.



Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

- `switchLayer` creates a ladder only when switching between layers.



GUI Equivalent

None

Examples

```
envGetVal("we" "weStrandedLadderAtTurn")
envSetVal("we" "weStrandedLadderAtTurn" 'cyclic "none")
envSetVal("we" "weStrandedLadderAtTurn" 'cyclic "switchLayer")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Creating a Stranded Wire](#)

weStrandNum

`layout weStrandNum int stradNumValue`

Description

Defines the number of stranded wires in interactive stranded routing. The default is 2.

GUI Equivalent

Command	<i>Create – Wiring – Stranded Wire - Create Stranded Wire form.</i>
Field	<i>Strand Number.</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Creating a Stranded Wire](#)

weStrandSpacing

layout weStrandSpacing float *strandSpacingValue*

Description

Defines the spacing between the two stranded wires. The default is 0.0, which is the value in the technology file).

GUI Equivalent

Command	<i>Create – Wiring – Stranded Wire - Create Stranded Wire</i> form.
Field	<i>Strand Spacing.</i>

Examples

```
envGetVal("layout" "weStrandSpacing")  
envSetVal("layout" "weStrandSpacing" 'float 1.0)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Creating a Stranded Wire](#)

weStrandWidth

layout weStrandWidth float *strandWidthValue*

Description

Defines the width of the stranded wire. The default value is 0, which means `maxWidth` is considered. If `maxWidth` is not found, `minWidth` is considered.

GUI Equivalent

Command	<i>Create – Wiring – Stranded Wire - Create Stranded Wire</i> form.
Field	<i>Strand Width</i> .

Examples

```
envGetVal("layout" " weStrandWidth ")  
envSetVal("layout" " weStrandWidth " 'float 1.0)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Creating a Stranded Wire](#)

weTapWidthOnPins

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

Description

Allows tapping the width on the starting object for new wire when the starting object is a pin. The default is `t`.

GUI Equivalent

Command	<i>Create Single Wire Context Sensitive menu – Use Width</i>
Field	<i>Tap Shapes and Pins Width</i>

Examples

```
envGetVal("layout" "weTapWidthOnPins")  
envSetVal("layout" "weTapWidthOnPins" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Create Stranded Wire Form](#)

weTapWidthOnVias

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

Description

Allows tapping the width on the starting object for new wire when the starting object is a via. The default is `nil`.

GUI Equivalent

Command	<i>Create Single Wire Context Sensitive menu – Use Width</i>
Field	<i>Tap Vias Width</i>

Examples

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

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Create Stranded Wire Form](#)

weTapWidthOnWires

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

Description

Allows tapping the width on the starting object for new wire when the starting object is a wire. The default is `t`.

GUI Equivalent

Command	<i>Create Single Wire Context Sensitive menu – Use Width</i>
Field	<i>Tap Wires Width</i>

Examples

```
envGetVal("layout" "weTapWidthOnWires")  
envSetVal("layout" "weTapWidthOnWires" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Create Stranded Wire Form](#)

weTrimBridgePurpose

```
we weTrimBridgePurpose string "purpose_name"
```

Description

Specifies the purpose on which the bridge should be inserted. If no value is specified for the purpose, the bridge is not inserted. However, if the specified purpose is invalid, the bridge is inserted on the default purpose. The default is `drawing`.

GUI Equivalent

None

Examples

```
envGetVal("we" "weTrimBridgePurpose")  
envSetVal("we" "weTrimBridgePurpose" 'string "type2")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Inserting TrimMetal Layers in Interactive Colored Routing](#)

weTrimInsertion

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

Description

Controls the interactive and post-edit insertion of trim shapes in interactive routing. The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("we" "weTrimInsertion")
```

```
envSetVal("we" "weTrimInsertion" 'boolean nil)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Inserting TrimMetal Layers in Interactive Colored Routing](#)

weTrimInsertionOnPRBoundary

```
we weTrimInsertionOnPRBoundary cyclic { "none" | "inside" | "outside" | "straddled"
}
```

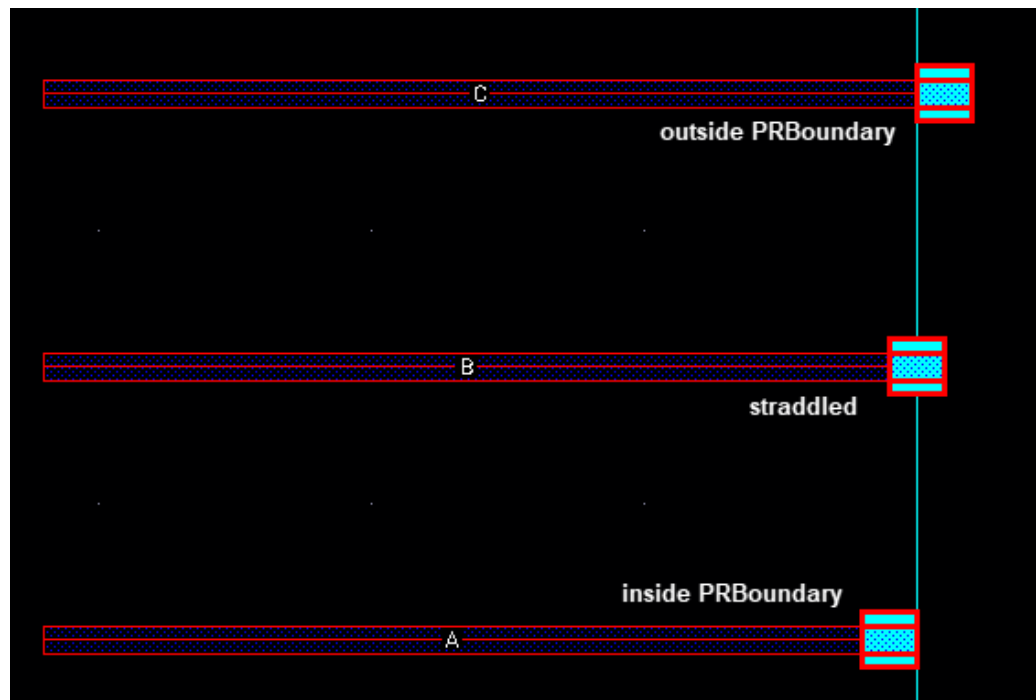
Description

Insert trims at the PR Boundary even when there are no segments outside.

This environment variable can take four values:

- `none` no trims are inserted. This is the default.
- `inside` inserts the trims inside the PR Boundary.
- `straddled` inserts the trims straddled on the PR Boundary.
- `outside` inserts the trims outside the PR Boundary.

The following figure shows the trim insertions for various values of the environment variable.



GUI Equivalent

None

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

Examples

```
envGetVal("we" "weTrimInsertionOnPRBoundary")
envSetVal("we" "weTrimInsertionOnPRBoundary" 'cyclic "inside")
envSetVal("we" "weTrimInsertionOnPRBoundary" 'cyclic "straddled")
envSetVal("we" "weTrimInsertionOnPRBoundary" 'cyclic "outside")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weTrimPatchExtension

```
we weTrimPatchExtension cyclic { "variable" | "truncate" }
```

Description

Controls how the patch is implemented when inserting trim interactively.

- `variable`: The patch shape is a variable extension of the digitized wire.
- `truncate`: The patch is merged with the digitized wire to form a single wire with a truncated extension.

The default is `variable`.

GUI Equivalent

None

Examples

```
envGetVal("we" "weTrimPatchExtension")  
envSetVal("we" "weTrimPatchExtension" 'cyclic "truncate")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Inserting TrimMetal Layers in Interactive Colored Routing](#)

weTrimPatchPurpose

```
we weTrimPatchPurpose string "purpose_name"
```

Description

Specifies the purpose on which the patch segment should be inserted. If no value is specified for the purpose or if the specified purpose is invalid, the patch segment is inserted on the default purpose. The default purpose is inherited from the wire or via purpose that extends up to the trim shape.

GUI Equivalent

None

Examples

```
envGetVal("we" "weTrimPatchPurpose")  
envSetVal("we" "weTrimPatchPurpose" 'string "type3")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Inserting TrimMetal Layers in Interactive Colored Routing](#)

weTrimTrimPurpose

```
we weTrimTrimPurpose string "purpose_name"
```

Description

Specifies the purpose on which the trim shape should be inserted. If no value is specified for the purpose or if the purpose is invalid, the trim shape is inserted on the default purpose. The default purpose is `drawing`.

GUI Equivalent

None

Examples

```
envGetVal("we" "weTrimTrimPurpose")  
envSetVal("we" "weTrimTrimPurpose" 'string "type2")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Inserting TrimMetal Layers in Interactive Colored Routing](#)

weWACmdCalculatedColor

```
layout weWACmdCalculatedColor string "color"
```

Description

The `weWACmdCalculatedColor` environment variable is used on the bus bit table of the Wire Assistant when the *Create Wire* or *Create Bus* command is enabled and sends the notifications. Consider an example, where the constraint lookup has 1.0 as the width of one particular bit, but the tapped pin has 1.5 as the width. In this case, the bus bit displays the `weWACmdCalculatedColor` on that particular bus bit's width cell.

GUI Equivalent

None

Examples

```
envGetVal("layout" "weWACmdCalculatedColor")  
envSetVal("layout" "weWACmdCalculatedColor" 'string "NavyBlue")  
envSetVal("layout" "weWACmdCalculatedColor" 'string "255 245 10")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weWAConflictColor

```
layout weWAConflictColor string "color"
```

Description

Enables you to set up the color of the color bar when there are any constraint overrides in the *Wire Assistant*.

You can specify the color as an RGB value or as the color name. The RGB value must be between 0 and 255. The color name should be specified as one word. The default is an empty string "".

GUI Equivalent

None

Examples

```
envGetVal("layout" "weWAConflictColor")  
envSetVal("layout" "weWAConflictColor" 'string "NavyBlue")  
envSetVal("layout" "weWAConflictColor" 'string "255 245 10")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weWADefaultSeedCGName

```
layout weWADefaultSeedCGName string "constraintGroup_names"
```

Description

Enables you to set a user-defined default constraint group for the *Wire Assistant*. The default is an empty string "".

GUI Equivalent

Command	<i>Window – Assistant – Wire Assistant</i>
Field	<i>Seed Attributes From a Constraint group</i>

Examples

```
envGetVal("layout" "weWADefaultSeedCGName")  
envSetVal("layout" "weWADefaultSeedCGName" 'string "wireConstraintgroup")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weWASeedCGColor

```
layout weWASeedCGColor string "color"
```

Description

Enables you to set up the color of the color bar when a constraint group from *Seed Attributes From a Constraint Group* icon on the *Wire Assistant* toolbar is loaded.

You can specify the color as an RGB value or as the color name. The RGB value must be between 0 and 255. The color name should be specified as one word. The default is an empty string "".

GUI Equivalent

None

Examples

```
envGetVal("layout" "weWASeedCGColor")  
envSetVal("layout" "weWASeedCGColor" 'string "NavyBlue")  
envSetVal("layout" "weWASeedCGColor" 'string "255 245 10")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

weWAUserDefColor

```
layout weWAUserDefColor string "color"
```

Description

Enables you to set up the color of the color bar when a user-defined constraint group is loaded in the *Wire Assistant*.

You can specify the color as an RGB value or as the color name. The RGB value must be between 0 and 255. The color name should be specified as one word. The default is an empty string "".

GUI Equivalent

None

Examples

```
envGetVal("layout" "weWAUserDefColor")  
envSetVal("layout" "weWAUserDefColor" 'string "NavyBlue")  
envSetVal("layout" "weWAUserDefColor" 'string "255 245 10")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

wireConstraintGroup

```
layout wireConstraintGroup string "constraintGroup_name"
```

Description

Specifies the constraint group used to supply a default set of constraints for both interactive and automatic routing. The default constraint group is `virtuosoDefaultSetup`. The `virtuosoDefaultSetup` constraint group should contain `validVias` and `validLayers` for routing and possibly other process rules that would override the foundry rules.

GUI Equivalent

Command	<i>Options – Editor</i>
Form Field	<i>Default Wire Constraint Group</i>

Examples

```
envGetVal("layout" "wireConstraintGroup")  
envSetVal("layout" "wireConstraintGroup" 'string "myConstraint")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Point to Point Form](#)

wireOffset

```
layout wireOffset float wireOffsetValue
```

Description

Allows you to create a wire a specified distance away from the cursor. The default is 0.0 user units. Value should be a positive float number.

GUI Equivalent

Command	<i>Create – Wiring – Wire</i>
Field	<i>Offset</i>

Examples

```
envGetVal("layout" "wireOffset")  
envSetVal("layout" "wireOffset" 'float 1.0)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

[Create Stranded Wire Form](#)

wireTaperConstraintGroup

```
layout wireTaperConstraintGroup string "constraintGroup_name"
```

Description

The taper constraint group can be used for several purposes. The primary purpose is to taper the width of a wire to match the width of the pin to which it connects. You can also adjust the widths of taper routing. Many different constraints can be included in the taper constraint group.

Another purpose is to restrict routing on the poly layer to the taper and set a taperHalo to control the length of a wire.

The default value for `wireTaperConstraintGroup` is `virtuosoDefaultTaper`. When the desired routing requires tapering, you must define the `virtuosoDefaultTaper` constraint group. Doing so will ensure that the correct layers and widths are being used for the tapered wires.

GUI Equivalent

None

Examples

```
envGetVal("layout" "wireTaperConstraintGroup")  
envSetVal("layout" "wireTaperConstraintGroup" 'string "wireEditorDefaultSetup")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

wireViaStackMode

```
layout wireViaStackMode cyclic { sameFootPrint | sameRowCol | manual }
```

Description

Enables standard via and cdsVia enclosures to align with wire edges. The default value is `manual`.

- `sameFootPrint` generates all vias in the stack with the same dimensions as the top via. This setting does not always ensure that all vias in the stack are of the same size. For example, some DRC rules might cause lower vias to be bigger than the top via.
- `sameRowCol` generates all vias in the stack with the same number of rows and columns.
- `manual` sets the row and column numbers only on the top via. All the vias below stay at their defaults (minimum rules).

The default is `manual`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "wireViaStackMode")  
envSetVal("layout" "wireViaStackMode" 'cyclic "sameRowCol")  
envSetVal("layout" "wireViaStackMode" 'cyclic "sameFootPrint")
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

wireWidth

layout wireWidth float wireWidthValue

Description

Sets a value for the width of the current wire. The default is 0.6 user units. Value should be a positive float number.

GUI Equivalent

Command	<i>Create – Wiring – Wire/Point to Point/Guided Routing</i>
Field	<i>Width</i>

Examples

```
envGetVal("layout" "wireWidth")  
envSetVal("layout" "wireWidth" 'float 1.0)
```

Related Topics

[Variables to Customize the Interactive and Assisted Routing Settings](#)

Variables to Customize the Settings for Slotting

The environment variables listed below are `slt` environment variables that provide a way to predefine some of the parameters prior to running the slotting SKILL command.

- If an environment variable has an invalid value (negative or out of range), the default behavior applies.
- The value specified in the environment variable can be overridden if the associated SKILL argument is specified in the command line.

Use the following variables to configure the slotting settings..

<u>addFilletOnBondFingers</u>	<u>addFilletOnCircle</u>	<u>addFilletOnComplexShapes</u>
<u>addFilletOnOblong</u>	<u>addFilletOnOctagon</u>	<u>addFilletOnPaths</u>
<u>addFilletOnPadWithoutDrills</u>	<u>addFilletOnPins</u>	<u>addFilletOnRectangle</u>
<u>addFilletOnSquare</u>	<u>addFilletOnTconnections</u>	<u>addFilletOnVias</u>
<u>areaDelimitedMode</u>	<u>curvedFillet</u>	<u>desiredAngle</u>
<u>densityAwareMode</u>	<u>directionalSlotPattern</u>	<u>enableSlotSpecCell</u>
<u>excludePins</u>	<u>floatingSlot</u>	<u>hideConvertToPolygon</u>
<u>hideReconstruct</u>	<u>lengthWidthRatio</u>	<u>lSpacing</u>
<u>mainSlotFigGroupName</u>	<u>material</u>	<u>maxAngle</u>
<u>maxLineWidth</u>	<u>maxOffset</u>	<u>maxSizeOnCircle</u>
<u>maxSizeOnComplexShapes</u>	<u>maxSizeOnOblong</u>	<u>maxSizeOnOctagon</u>
<u>maxSizeOnRectangle</u>	<u>maxSizeOnSquare</u>	<u>metalDensityLayers</u>
<u>minLineWidth</u>	<u>minPolygonEdgeLength</u>	<u>slottingContext</u>
<u>slotLength</u>	<u>slotOverBlockage</u>	<u>slotOverInstance</u>
<u>slotOrientationMode</u>	<u>slotOutOfGroup</u>	<u>slotSpecCell</u>
<u>slotStaggered</u>	<u>slotToEdge</u>	<u>slotVia</u>
<u>slotWidth</u>	<u>taperedTraces</u>	<u>tapDesiredAngle</u>
<u>tapMaxOffset</u>	<u>tapMinSegAngle</u>	<u>touchBlockageMode</u>
<u>turnSlotPattern</u>	<u>unassignedShapes</u>	<u>useOneSlotSpecCell</u>
<u>usePaletteLayers</u>	<u>wSpacing</u>	<u>widthThreshold</u>

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

Related Topics

[Interactive and Assisted Routing Environment Variables](#)

addFilletOnBondFingers

```
slt addFilletOnBondFingers boolean { t | nil }
```

Description

Creates fillets on bond finger objects. The default is `nil`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Objects – Bond Fingers</i>

Examples

```
envGetVal("slt" "addFilletOnBondFingers")  
envSetVal("slt" "addFilletOnBondFingers" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

addFilletOnCircle

```
slt addFilletOnCircle boolean { t | nil }
```

Description

Creates fillets on circle shapes. The default is `t`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Shapes – Circle</i>

Examples

```
envGetVal("slt" "addFilletOnCircle")  
envSetVal("slt" "addFilletOnCircle" 'boolean nil)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

addFilletOnComplexShapes

```
slt addFilletOnComplexShapes boolean { t | nil }
```

Description

Creates fillets on complex shapes. The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("slt" "addFilletOnComplexShapes")  
envSetVal("slt" "addFilletOnComplexShapes" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

addFilletOnOblong

```
slt addFilletOnOblong boolean { t | nil }
```

Description

Creates fillets on oblong shapes. The default is `t`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Shapes – Oblong</i>

Examples

```
envGetVal("slt" "addFilletOnOblong")  
envSetVal("slt" "addFilletOnOblong" 'boolean nil)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

addFilletOnOctagon

```
slt addFilletOnOctagon boolean { t | nil }
```

Description

Creates fillets on octagon shapes. The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("slt" "addFilletOnOctagon")  
envSetVal("slt" "addFilletOnOctagon" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

addFilletOnPaths

```
slt addFilletOnPaths boolean { t | nil }
```

Description

Creates fillets on path objects. The default is `t`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Objects – Paths</i>

Examples

```
envGetVal("slt" "addFilletOnPaths")  
envSetVal("slt" "addFilletOnPaths" 'boolean nil)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

addFilletOnPadWithoutDrills

```
slt addFilletOnPadWithoutDrills boolean { t | nil }
```

Description

Creates fillets on pads without drills. The default is `nil`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Objects – Pad without drills</i>

Examples

```
envGetVal("slt" "addFilletOnPadWithoutDrills")  
envSetVal("slt" "addFilletOnPadWithoutDrills" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

addFilletOnPins

```
slt addFilletOnPins boolean { t | nil }
```

Description

Creates fillets on pin objects. The default is `t`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Objects – Pins</i>

Examples

```
envGetVal("slt" "addFilletOnPins")  
envSetVal("slt" "addFilletOnPins" 'boolean nil)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

addFilletOnRectangle

```
slt addFilletOnRectangle boolean { t | nil }
```

Description

Creates fillets on rectangles. The default is `t`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Shapes – Rectangle</i>

Examples

```
envGetVal("slt" "addFilletOnRectangle")  
envSetVal("slt" "addFilletOnRectangle" 'boolean nil)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

addFilletOnSquare

```
slt addFilletOnSquare boolean { t | nil }
```

Description

Creates fillets on squares. The default is `t`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Shapes – Square</i>

Examples

```
envGetVal("slt" "addFilletOnSquare")  
envSetVal("slt" "addFilletOnSquare" 'boolean nil)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

addFilletOnTconnections

```
slt addFilletOnTconnections boolean { t | nil }
```

Description

Creates fillets on T connections. The default is `t`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Objects – T connections</i>

Examples

```
envGetVal("slt" "addFilletOnTconnections")  
envSetVal("slt" "addFilletOnTconnections" 'boolean nil)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

addFilletOnVias

```
slt addFilletOnVias boolean { t | nil }
```

Description

Creates fillets on vias. The default is `t`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Objects – Vias</i>

Examples

```
envGetVal("slt" "addFilletOnVias")  
envSetVal("slt" "addFilletOnVias" 'boolean nil)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

areaDelimitedMode

```
slt areaDelimitedMode cyclic { "adjacent" | "enclosedOnly" }
```

Description

Defines different ways of finding shapes. It considers the shapes that are partially enclosed within the area and lets you create, delete, and check slotting shapes based on the area.

- Create slot - the target is shape, therefore, only the shapes within the specified area or window are slotted.
- Delete Slot - the target is slot, therefore only the slot shapes within the specified area or window are deleted.
- Checking - only the objects within the specified area or window are checked.

Note: The area or window specified, defines the region in which the command is performed. However, the type of object on which it applies is determined by the command selected.

The two modes that can be specified for the variable are:

- `adjacent`, in this mode, the regions of shapes that are in contact with the area or window are also considered and slotted.
- `enclosedOnly`, in this mode, only the portion of shapes that are within the area or window are slotted.

The default is `adjacent`.

GUI Equivalent

None

Examples

```
envGetVal("slt" "areaDelimitedMode")  
envSetVal("slt" "areaDelimitedMode" 'cyclic "enclosedOnly")
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

Create Packaging Bus Form

lxShapeSlotting

lxConvertSlotToPolygon

curvedFillet

```
slt curvedFillet boolean { t | nil }
```

Description

Creates a fillet using an arc instead of a line as part of the shape outline from the center of the shape to the intersection of the shape. The default is `nil`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Curved Fillet</i>

Examples

```
envGetVal("slt" "curvedFillet")  
envSetVal("slt" "curvedFillet" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

desiredAngle

slt desiredAngle float *float_number*

Description

Specifies the target angle of the fillet shape. In some cases, the value can be increased to the maximum angle to achieve DRC correct results or optimal fillet shapes to ensure a fully covered object edge. The default is 90.0 degrees.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Desired Angle</i>

Examples

```
envGetVal("slt" "desiredAngle")  
envSetVal("slt" "desiredAngle" 'float 3.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

densityAwareMode

```
slt densityAwareMode boolean { t | nil }
```

Description

Enables slotting only those shapes that violate the maximum density rules. These rules should be available in the technology library file. The default is `nil`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slotting Style</i>
Field	<i>Density Aware</i>

Examples

```
envGetVal("slt" "densityAwareMode")  
envSetVal("slt" "densityAwareMode" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[1xShapeSlotting](#)

[1xConvertSlotToPolygon](#)

directionalSlotPattern

```
slt directionalSlotPattern string "list_of_points"
```

Description

Defines a slot pattern (for example, complex chamfered rectangles) by defining a list of points. When this environment variable is set, the pattern slots are inserted in the direction of the current. The string is a list of points, such as X1:Y1 X2:Y2 X3:Y3. The default value is an empty string "".

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slotting Style</i>
Field	<i>Directional Pattern</i>

Examples

```
envGetVal("slt" "directionalSlotPattern")  
envSetVal("slt" "directionalSlotPattern" 'string "X1:Y1 X2:Y2 X3:Y3 X4:Y4 X5:Y5  
X6:Y6 X7:Y7 X8:Y8")
```

The above example defines the list of points for the following shape pattern.



Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

[IxShapeSlotting](#)

[IxConvertSlotToPolygon](#)

enableSlotSpecCell

```
slt enableSlotSpecCell boolean { t | nil }
```

Description

Uses the `slotSpecCell` environment variable to override the default slotting parameters. The default is `t`. When set to `t`, it enables the *Specification Cell* field. Otherwise, the *Specification Cell* field is disabled.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slot Geometry</i>
Field	Check box next to the <i>Specification Cell</i> field

Examples

```
envGetVal("slt" "enableSlotSpecCell")  
envSetVal("slt" "enableSlotSpecCell" 'boolean nil)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

excludePins

```
slt excludePins boolean { t | nil }
```

Description

Enables excluding pin objects from being slotted. The default is `nil`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i>
	<i>Options – Slotting Style</i>
Field	<i>Exclude Pin.</i>

Examples

```
envGetVal("slt" "excludePins")  
envSetVal("slt" "excludePins" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[IxShapeSlotting](#)

[IxConvertSlotToPolygon](#)

floatingSlot

```
slt floatingSlot boolean { t | nil }
```

Description

Reports the slot shapes, with `slot` purpose, that do not overlap a metal shape and are not connected. The default is `nil`, which means that the slotting command never creates slot shapes on the space (floating shape).

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slot Consistency</i>
Field	<i>Floating Slot</i>

Examples

```
envGetVal("slt" "floatingSlot")  
envSetVal("slt" "floatingSlot" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

hideConvertToPolygon

```
slt hideConvertToPolygon boolean { t | nil }
```

Description

Hides the options to convert the original shapes covered by the slot shapes into slotted polygons. The default is `nil`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i>
	<i>Tools – Convert Slots</i>

Examples

```
envGetVal("slt" "hideConvertToPolygon")  
envSetVal("slt" "hideConvertToPolygon" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Metal Density Toolbar](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

hideReconstruct

```
slt hideReconstruct boolean { t | nil }
```

Description

Hides the options to reconstruct a slotted polygon to a regular polygon. The default is `nil`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i>
	<i>Tools – Reconstruct Slots</i>

Examples

```
envGetVal("slt" "hideReconstruct")  
envSetVal("slt" "hideReconstruct" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Metal Density Toolbar](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

lengthWidthRatio

slt lengthWidthRatio float *lengthWidthRatioValue*

Description

Specifies a threshold ratio value that determines whether a slot must be created as a square or a rectangular shape. If the length-to-width ratio of the object is greater than the defined threshold, then a rectangular slot is created. If the length-to-width ratio of the object is less than or equal to the specified ratio, then a square slot is created. The default is $-0.1\mu\text{m}$.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slotting Style</i>
Field	<i>Length Width Ratio</i>

Examples

```
envGetVal("slt" "lengthWidthRatio")  
envSetVal("slt" "lengthWidthRatio" 'float 3.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

ISpacing

`slt lSpacing float lSpacingValue`

Description

Defines the slot spacing along the length. The default is $-0.1\mu\text{m}$.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slot Geometry</i>
Field	<i>Length Spacing</i>

Examples

```
envGetVal("slt" "lSpacing")  
envSetVal("slt" "lSpacing" 'float 1.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

mainSlotFigGroupName

```
slt mainSlotFigGroupName string "figgroupname"
```

Description

Creates the main fig group and all the fig groups of slots are considered as the members of this main fig group. The name of this main fig group is defined by this variable. The default value is an empty string " ".

GUI Equivalent

None

Examples

```
envGetVal("slt" "mainSlotFigGroupName")  
envSetVal("slt" "mainSlotFigGroupName" 'string "figGroup1")
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

material

```
slt material string "layercategoryname"
```

Description

Specifies the layer category to be processed. The default is `metal`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Layers</i>
Field	<i>Material</i>

Examples

```
envGetVal("slt" "material")  
envSetVal("slt" "material" 'string "nwell")
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[IxShapeSlotting](#)

[IxConvertSlotToPolygon](#)

maxAngle

```
slt maxAngle float maxAngleValue
```

Description

Specifies the maximum angle for the fillet. The default value is 90.0 degrees.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Max Angle</i>

Examples

```
envGetVal("slt" "maxAngle")  
envSetVal("slt" "maxAngle" 'float 45.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[IxShapeSlotting](#)

[IxConvertSlotToPolygon](#)

maxLineWidth

```
slt maxLineWidth float maxLineWidthValue
```

Description

Specifies the maximum width of the wire entering the shape. If the width of the wire is greater than the specified value, the fillet is not created. The default value is 635.0 μm .

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Max Line Width</i>

Examples

```
envGetVal("slt" "maxLineWidth")  
envSetVal("slt" "maxLineWidth" 'float 45.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

maxOffset

`slt maxOffset float maxOffsetValue`

Description

Specifies the maximum distance between the intersection of the shape edge and the connecting line forming the fillet length. A larger `desiredAngle` and a smaller `maxOffset` creates a short fillet. A smaller `desiredAngle` and larger `maxOffset` creates a long fillet. The default value is 150.0 μm .

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Max Offset</i>

Examples

```
envGetVal("slt" "maxOffset")  
envSetVal("slt" "maxOffset" 'float 120.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

maxSizeOnCircle

```
slt maxSizeOnCircle float maxSizeOnCircleValue
```

Description

Specifies the maximum size of a fillet for a circle shape. The default value is 2540.0 μm .

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Shapes – Circle – Max Size</i>

Examples

```
envGetVal("slt" "maxSizeOnCircle")  
envSetVal("slt" "maxSizeOnCircle" 'float 1000.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

maxSizeOnComplexShapes

```
slt maxSizeOnComplexShapes float maxSizeOnComplexShapes
```

Description

Specifies the maximum size of a fillet for a complex shape. The default value is 2540.0 μm .

GUI Equivalent

None

Examples

```
envGetVal("slt" "maxSizeOnComplexShapes")  
envSetVal("slt" "maxSizeOnComplexShapes" 'float 1200.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

maxSizeOnOblong

slt maxSizeOnOblong float *maxSizeOnOblong*

Description

Specifies the maximum size of a fillet for an oblong shape. The default value is 2540.0 μm .

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Shapes – Oblong – Max Size</i>

Examples

```
envGetVal("slt" "maxSizeOnOblong")  
envSetVal("slt" "maxSizeOnOblong" 'float 1200.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

maxSizeOnOctagon

```
slt maxSizeOnOctagon float maxSizeOnOctagon
```

Description

Specifies the maximum size of a fillet for an octagon shape. The default value is 2540.0 μm .

GUI Equivalent

None

Examples

```
envGetVal("slt" "maxSizeOnOctagon")  
envSetVal("slt" "maxSizeOnOctagon" 'float 1200.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

maxSizeOnRectangle

slt maxSizeOnRectangle float *maxSizeOnRectangle*

Description

Specifies the maximum size of a fillet for a rectangle shape. The default value is 2540.0 μm .

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Shapes – Rectangle – Max Size</i>

Examples

```
envGetVal("slt" "maxSizeOnRectangle")  
envSetVal("slt" "maxSizeOnRectangle" 'float 1100.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

maxSizeOnSquare

```
slt maxSizeOnSquare float maxSizeOnSquare
```

Description

Specifies the maximum size of a fillet for a square shape. The default value is 2540.0 μm .

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Shapes – Square – Max Size</i>

Examples

```
envGetVal("slt" "maxSizeOnSquare")  
envSetVal("slt" "maxSizeOnSquare" 'float 1300.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

metalDensityLayers

```
slt metalDensityLayers string "list_of_layernames"
```

Description

Specifies the names of the layers that you want to process. The default value is an empty string " ".

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Layers</i>
Field	<i>Layers.</i>

Examples

```
envGetVal("slt" "metalDensityLayers")  
envSetVal("slt" "metalDensityLayers" 'string "Metal1 Metal2")
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[IxShapeSlotting](#)

[IxConvertSlotToPolygon](#)

minLineWidth

`slt minLineWidth float minLineWidth`

Description

Specifies the minimum width of the wire entering the shape. If the width of the wire is less than the specified value, the fillet is not created. The default value is 1 μm .

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Min Line Width</i>

Examples

```
envGetVal("slt" "minLineWidth")  
envSetVal("slt" "minLineWidth" 'float 2.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

minPolygonEdgeLength

`slt minPolygonEdgeLength float minPolygonEdgeLength`

Description

Truncates the slot with edges inferior to the value passed into the environment variable. The default is 0.0 μ m.

GUI Equivalent

None

Examples

```
envGetVal("slt" "minPolygonEdgeLength")  
envSetVal("slt" "minPolygonEdgeLength" 'float 1.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[IxShapeSlotting](#)

[IxConvertSlotToPolygon](#)

slottingContext

```
slt slottingContext string "slottingcontextname"
```

Description

Lets you define the context for slotting only if [useOneSlotSpecCell](#) is set to `nil`. It represents the value of the *Context* field in the Metal Density Options form. The default value is "default".

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density Options – Slot Geometry</i>
Field	<i>Context</i>

Examples

```
envGetVal("slt" "slottingContext")  
envSetVal("slt" "slottingContext" 'string "context2")
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[IxShapeSlotting](#)

[IxConvertSlotToPolygon](#)

[sltSetContextSpecificationCells](#)

slotLength

```
slt slotLength float slotLength
```

Description

Defines the length of the slot. The default is $-0.1\mu\text{m}$.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slot Geometry</i>
Field	<i>Length</i>

Examples

```
envGetVal("slt" "slotLength")  
envSetVal("slt" "slotLength" 'float 1.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[IxShapeSlotting](#)

[IxConvertSlotToPolygon](#)

slotOverBlockage

```
slt slotOverBlockage boolean { t | nil }
```

Description

Reports slot shapes that overlap on the same layer and the blockage type is `slot`. The default is `nil`, which means that the slotting command never creates slot shapes over blockage.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slot Consistency</i>
Field	<i>Slot Over Blockage</i>

Examples

```
envGetVal("slt" "slotOverBlockage")  
envSetVal("slt" "slotOverBlockage" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

slotOverInstance

```
slt slotOverInstance boolean { t | nil }
```

Description

Reports slot shapes, with `slot` purpose, that overlap an instance of a cell. The default is `nil`, which means that the slotting command never creates slot shapes over an instance.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slot Consistency</i>
Field	<i>Slot Over Instance</i>

Examples

```
envGetVal("slt" "slotOverInstance")  
envSetVal("slt" "slotOverInstance" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

slotOrientationMode

```
slt slotOrientationMode cyclic { "default" | "square" | "diamond" | "horizontal" |  
    "vertical" | "diagonalNE" | "diagonalSE" }
```

Description

Defines slot orientation for the shapes to be slotted. You can specify one of the following orientations for the slots: default, square, diamond, horizontal, vertical, diagonalNE, and diagonalSE. The default is "default".

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slotting Style</i>
Field	<i>Slots Orientation</i>

Examples

```
envGetVal("slt" "slotOrientationMode")  
envSetVal("slt" "slotOrientationMode" 'cyclic "diamond")  
envSetVal("slt" "slotOrientationMode" 'cyclic "vertical")
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[IxShapeSlotting](#)

[IxConvertSlotToPolygon](#)

slotOutOfGroup

```
slt slotOutOfGroup boolean { t | nil }
```

Description

Reports the slot shapes with `slot` purpose, which are not part of a slotting `figGroup`. The slotting commands always creates slot shapes in a `figGroup`. The default is `nil`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slot Consistency</i>
Field	<i>Slot Out Of Group</i>

Examples

```
envGetVal("slt" "slotOutOfGroup")  
envSetVal("slt" "slotOutOfGroup" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

slotSpecCell

```
slt slotSpecCell string "speccellname"
```

Description

Specifies a cell name (specification cell). A specification cellview contains representation of the directional slot shapes and the turn areas for each concerned layer. The syntax for a specification cell is `libName/cellName/ViewName`. For example, `lib1/cell1/view1`. The default value is an empty string `" "`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slot Geometry</i>
Field	<i>Specification Cell</i>

Examples

```
envGetVal("slt" "slotSpecCell")  
envSetVal("slt" "slotSpecCell" 'string "lib1/cell1/view2")
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

slotStaggered

```
slt slotStaggered boolean { t | nil }
```

Description

Defines the offset mode used to add slots. If the offset mode is set to `nil`, slots are inline. However, if the offset mode is set to `t`, slots are staggered. The default is `t`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slotting Style</i>
Field	<i>Stagger</i>

Examples

```
envGetVal("slt" "slotStaggered")  
envSetVal("slt" "slotStaggered" 'boolean nil)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

slotToEdge

slt slotToEdge float *slotToEdgeValue*

Description

Defines the minimum distance from the slot to the edge. The default is $-0.1\mu\text{m}$.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slot Geometry</i>
Field	<i>Distance to Edge</i>

Examples

```
envGetVal("slt" "slotToEdge")  
envSetVal("slt" "slotToEdge" 'float 1.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

slotVia

```
slt slotVia boolean { t | nil }
```

Description

Enables slotting of vias. The default is `t`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slotting Style</i>
Field	<i>Slot Via</i>

Examples

```
envGetVal("slt" "slotVia")  
envSetVal("slt" "slotVia" 'boolean nil)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[IxShapeSlotting](#)

[IxConvertSlotToPolygon](#)

slotWidth

slt slotWidth float *slotWidthValue*

Description

Defines the width of the slot. The default is -0.1 μm .

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slot Geometry</i>
Field	<i>Width</i>

Examples

```
envGetVal("slt" "slotWidth")  
envSetVal("slt" "slotWidth" 'float 1.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[IxShapeSlotting](#)

[IxConvertSlotToPolygon](#)

taperedTraces

```
slt taperedTraces boolean { t | nil }
```

Description

Creates fillets on tapered traces. The default is `t`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density Options – Fillet</i>
Field	<i>Tapered Traces – Tapered traces</i>

Examples

```
envGetVal("slt" "taperedTraces")  
envSetVal("slt" "taperedTraces" 'boolean nil)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

tapDesiredAngle

slt tapDesiredAngle float tapDesiredAngleValue

Description

Specifies the value of the tapered angle to be created by the fillet. The default is 60 μm .

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Tapered Traces – Desired angle</i>

Examples

```
envGetVal("slt" "tapDesiredAngle")  
envSetVal("slt" "tapDesiredAngle" 'float 20)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

tapMaxOffset

slt tapMaxOffset float tapMaxOffsetValue

Description

Specifies the maximum distance the fillet may extend from the vertex to the point of cline width transition. The default is 635 μm .

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Tapered Traces – Max offset</i>

Examples

```
envGetVal("slt" "tapMaxOffset")  
envSetVal("slt" "tapMaxOffset" 'float 500)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

tapMinSegAngle

```
slt tapMinSegAngle float tapMinSegAngleValue
```

Description

Specifies the minimum segment angle. The default is 135 μ m.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Tapered Traces – Min segment angle</i>

Examples

```
envGetVal("slt" "tapMinSegAngle")  
envSetVal("slt" "tapMinSegAngle" 'float 100)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

touchBlockageMode

```
slt touchBlockageMode boolean { t | nil }
```

Description

Places slots within the regions that are partially covered by the blockage. The regions that are fully covered by the blockage are not slotted. When the value is `t`, no slot is placed within the region covered by the blockage, fully or partially. The default is `nil`.

The behavior of slotting using the `lxShapeSlotting` SKILL API depends on the value of the `touchBlockageMode` environment variable.

GUI Equivalent

None

Examples

```
envGetVal("slt" "touchBlockageMode")  
envSetVal("slt" "touchBlockageMode" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

turnSlotPattern

```
slt turnSlotPattern string "list_of_points"
```

Description

Defines a slot pattern (for example, complex chamfered squares) by defining a list of points. When this environment variable is set, the pattern slots are inserted at the turn. The string is a list of points, such as X1:Y1 X2:Y2 X3:Y3. The default value is an empty string "".

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slotting Style</i>
Field	<i>Pattern at Turn</i>

Examples

```
envGetVal("slt" "turnSlotPattern")  
envSetVal("slt" "turnSlotPattern" 'string "X1:Y1 X2:Y2 X3:Y3 X4:Y4")
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

unassignedShapes

```
slt unassignedShapes boolean { t | nil }
```

Description

Creates fillets on objects without connectivity. The default is `nil`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Fillet</i>
Field	<i>Unassigned Shapes</i>

Examples

```
envGetVal("slt" "unassignedShapes")  
envSetVal("slt" "unassignedShapes" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

useOneSlotSpecCell

```
slt useOneSlotSpecCell boolean { t | nil }
```

Description

When set to `nil`, a list of defined contexts is visible in the *Context* field of the *Slot Geometry* tab of the Metal Density Options form. Instead of the string field for the *Specification Cell*, a drop-down list is available, which lists the contexts that have been defined using the `sltSetContextSpecificationCells` SKILL function. The default is `t`.

If set to `t`, you can specify the slotting parameters only for a single specification cell. In the Metal Density Options form, a string field is available to specify the specification cell name in the `library/cell/layout` format.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slot Geometry</i>
Field	<i>Context/Specification Cell</i>

Examples

```
envGetVal("slt" "useOneSlotSpecCell")  
envSetVal("slt" "useOneSlotSpecCell" 'boolean nil)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

[sltSetContextSpecificationCells](#)

usePaletteLayers

```
slt usePaletteLayers boolean { t | nil }
```

Description

Enables using only the visible palette layers for slotting. The default is `nil`.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i>
	<i>Options – Layers</i>
Field	<i>Use Palette Layers</i>

Examples

```
envGetVal("slt" "usePaletteLayers")  
envSetVal("slt" "usePaletteLayers" 'boolean t)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

wSpacing

```
slt wSpacing float wSpacingValue
```

Description

Defines the slot spacing along the width. The default is $-0.1\mu\text{m}$.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slot Geometry</i>
Field	<i>Width Spacing</i>

Examples

```
envGetVal("slt" "wSpacing")  
envSetVal("slt" "wSpacing" 'float 1.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

widthThreshold

`slt widthThreshold float widthThresholdValue`

Description

Defines the width to filter out the shapes with a width smaller than the one specified. The default is $-0.1\mu\text{m}$.

GUI Equivalent

Command	<i>Window – Toolbar – Metal Density</i> <i>Options – Slotting Style</i>
Field	<i>Width Threshold</i>

Examples

```
envGetVal("slt" "widthThreshold")  
envSetVal("slt" "widthThreshold" 'float 1.0)
```

Related Topics

[Variables to Customize the Settings for Slotting](#)

[Managing Metal Density](#)

[Create Packaging Bus Form](#)

[lxShapeSlotting](#)

[lxConvertSlotToPolygon](#)

Variables to Customize the Settings for Copy Route

Use the following variables to configure the copy route options in the *CopyRoute* tab of the Routing assistant and the Copy Route form..

<u>copyRouteAllowCopiesWithoutMatchingPins</u>	<u>copyRouteBottomLayer</u>
<u>copyRouteClearPreviewWhenEsc</u>	<u>copyRouteCreateFigGroup</u>
<u>copyRouteEnableRoutingChannel</u>	<u>copyRouteEnableSpacingAtEnds</u>
<u>copyRouteEnableValidLayer</u>	<u>copyRouteEnableWidthAtEnds</u>
<u>copyRouteGenerateCopyWithErrors</u>	<u>copyRouteLayerPattern</u>
<u>copyRouteLayersMode</u>	<u>copyRouteNumCopiesInSpace</u>
<u>copyRoutePreserveCutClasses</u>	<u>copyRoutePreviewMode</u>
<u>copyRouteRoutingChannelLPP</u>	<u>copyRouteSpacingMode</u>
<u>copyRouteSpacingModeAtEnds</u>	<u>copyRouteSpacingOverrideAllLayers</u>
<u>copyRouteSpacingOverrideAllLayersAtEnds</u>	<u>copyRouteTopLayer</u>
<u>copyRouteTrimmingFixer</u>	<u>copyRouteWidthMode</u>
<u>copyRouteWidthModeAtEnds</u>	<u>copyRouteWidthOverrideAllLayers</u>
<u>copyRouteWidthOverrideAllLayersAtEnds</u>	

Related Topics

[Interactive and Assisted Routing Environment Variables](#)

copyRouteAllowCopiesWithoutMatchingPins

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

Description

Lets you select and create copies of unpaired net pins.

The default is `nil`.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Allow without matching pins</i> <i>Options – Allow Copies Without Matching Pins</i>

Example

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

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteTrimmingFixer](#)

copyRouteBottomLayer

```
we copyRouteBottomLayer string "bottomLayer"
```

Description

Specifies the bottom layer for copy route. This variable is effective only if the [copyRouteEnableValidLayer](#) variable is set to `t`.

The default is empty string "".

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Annotate Copies – Bottom</i> <i>General – Top/Bottom Layers</i>

Examples

```
envGetVal("we" "copyRouteBottomLayer")  
envSetVal("we" "copyRouteBottomLayer" 'string "Metall")
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Form](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[copyRouteTopLayer](#)

[copyRouteEnableValidLayer](#)

copyRouteClearPreviewWhenEsc

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

Description

Clears the route preview when the copy route command is canceled.

The default is `t`.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i>
Field	<i>Preview – Clear preview when cmd cancelled</i>

Example

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

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRoutePreviewMode](#)

copyRouteCreateFigGroup

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

Description

Creates the route copies in a fig group.

The default is `nil`.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Copies – In group</i> <i>Options – Group Copies</i>

Example

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

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

copyRouteEnableRoutingChannel

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

Description

Enables the routing channel check. The routing channel is an LPP defined using the `copyRouteRoutingChannelLPP` environment variable.

The route copies must be created inside the routing channel. If any copies are created outside the defined channel, it is considered as a violation and annotations or markers are created.

The default is `nil`.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Outside routing area</i> <i>General – Routing Channel</i>

Example

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

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteRoutingChannelLPP](#)

copyRouteEnableSpacingAtEnds

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

Description

Enables spacing at the first and last sections.

The default is `nil`.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Spacing – First/last Sections – Enable</i> <i>General – Spacing At Ends</i>

Example

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

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteSpacingModeAtEnds](#)

[copyRouteSpacingOverrideAllLayers](#)

copyRouteEnableValidLayer

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

Description

Enables the top and bottom layers check. The layer for route copies must be in the range of specified top and bottom layer, otherwise a violation is created.

The default is `nil`.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Out of layer range</i> <i>General – Top/Bottom Layers</i>

Example

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

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteBottomLayer](#)

[copyRouteTopLayer](#)

copyRouteEnableWidthAtEnds

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

Description

Enables to define the width of the first and last sections of the copied wire.

The default is `nil`.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Width – First/last Sections – Enable</i> <i>General – Width At Ends</i>

Examples

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

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteWidthModeAtEnds](#)

[copyRouteWidthOverrideAllLayersAtEnds](#)

copyRouteGenerateCopyWithErrors

```
we copyRouteGenerateCopyWithErrors cyclic { "all" | "none" | "firstOnly" }
```

Description

Generates the copy route results with violations.

- **all**: generates the result no matter how many violations are present. This is the default option.
- **none**: generates the result without any violations. The copy route is stopped as soon as there is a single violation.
- **firstOnly**: generates the result with any violation. The copy route is stopped as soon as there is a violation, but the first copy route with violations is kept.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Copies – With Errors</i> <i>Options – Generate Copy With Errors</i>

Examples

```
envGetVal("we" "copyRouteGenerateCopyWithErrors")  
envSetVal("we" "copyRouteGenerateCopyWithErrors" 'cyclic "none")  
envSetVal("we" "copyRouteGenerateCopyWithErrors" 'cyclic "firstOnly")
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

Copy Route Form

copyRouteLayerPattern

```
we copyRouteLayerPattern string "layerPattern"
```

Description

Specifies the layer pattern that should be followed for copy route. The layer pattern is a list of integers.

The default is an empty string.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Layers – Pattern</i> <i>General – Pattern</i>

Examples

```
envGetVal("we" "copyRouteLayerPattern")  
envSetVal("we" "copyRouteLayerPattern" 'string "1 0 -1")
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteLayersMode](#)

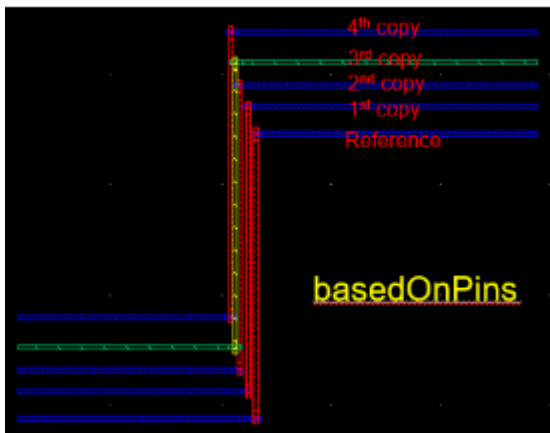
copyRouteLayersMode

```
we copyRouteMultilayerMode cyclic { "sameAsReference" | "basedOnPins" |  
    "basedOnPattern" }
```

Description

Specifies the layer mode for copy route.

- **sameAsReference:** Lets you select different layer pins, where different layers means different from the reference pin layer. The first and last pathSeg layer is the connected pin layer. The intermediate section layer is the same as the reference pin.
- **basedOnPins:** It is similar to `forPinsOnly`, but the intermediate section layer is adjusted based on the pin layer difference. For example, in the following figure, the reference wire is M1 - M2 - M1. The copy from M3 pin would be M3 - M4 - M3. All the layers are two layers ahead because the pin layer difference is $M3 - M1 = 2$.

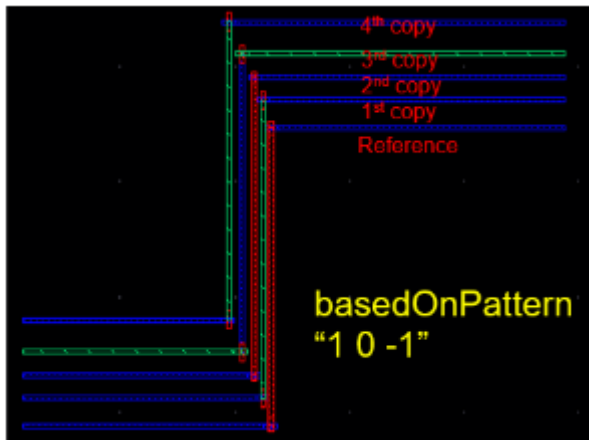


- **basedOnPattern:** specifies the user-defined pattern. This is dependent on the pattern specified in the `copyRouteLayerPattern` environment variable. For example, in the following figure, the middle layer of reference wire is M2 and the pattern is "1 0 -1". The first middle layer copy is M3 (+1 layer). The second middle layer copy is M2 (+0 layer).

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

The third middle layer copy is M1 (-1 layer). Then we repeat the pattern, so the 4th middle layer copy is M3 (+1 layer).



GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Layers – Intermediate Sections</i> <i>General – Target Layers</i>

Examples

```
envGetVal("we" "copyRouteMultilayerMode")
envSetVal("we" "copyRouteMultilayerMode" 'cyclic "forPinsOnly")
envSetVal("we" "copyRouteMultilayerMode" 'cyclic "basedOnPins")
envSetVal("we" "copyRouteMultilayerMode" 'cyclic "basedOnPattern")
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

Copy Route Form

copyRouteLayerPattern

copyRouteNumCopiesInSpace

`we copyRouteNumCopiesInSpace int numberOfCopies`

Description

Defines the number of copies when the first and last transitions of route copies are not connected to pins.

The default value is 1.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Number of Copies</i> <i>General – Number of Copies</i>

Examples

```
envGetVal("we" "copyRouteNumCopiesInSpace")  
envSetVal("we" "copyRouteNumCopiesInSpace" 'int 2)
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

copyRoutePreserveCutClasses

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

Description

Generates copies with the same cutClass as that of the reference wire.

The default is `t`.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Vias – Preserve cut classes</i> <i>General – Preserve Cut Classes</i>

Example

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

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

copyRoutePreviewMode

```
we copyRoutePreviewMode cyclic { "off" | "low" | "high" }
```

Description

Displays the preview of the copy route result.

- **off**: Displays only the flightlines for the copy route result. This is the default.
- **low**: Displays a preview. The preview computation is faster than **high** but less accurate.
- **high**: Displays a preview. The preview computation is more accurate than **low** but slower.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i>
Field	<i>Preview – Show Preview, Effort</i>

Examples

```
envGetVal("we" "copyRoutePreviewMode")  
envSetVal("we" "copyRoutePreviewMode" 'cyclic "low")  
envSetVal("we" "copyRoutePreviewMode" 'cyclic "high")
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteClearPreviewWhenEsc](#)

copyRouteRoutingChannelLPP

```
we copyRouteRoutingChannelLPP string "LPP"
```

Description

Defines the routing channel layer.

The default is y0 drawing.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Annotate Copies – Routing layer</i> <i>General – Routing Channel</i>

Examples

```
envGetVal("we" "copyRouteRoutingChannelLPP")  
envSetVal("we" "copyRouteRoutingChannelLPP" 'string "Nwell drawing")
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteEnableRoutingChannel](#)

copyRouteSpacingMode

```
we copyRouteSpacingMode cyclic { "defaultConstraint" | "overrideAllLayers" |  
    "keepPinSpace" }
```

Description

Defines the copy spacing mode for the intermediate sections.

- **defaultConstraint:** uses the minSpacing defined in technology file. This is the default option.
- **overrideAllLayers:** uses the value specified for the copyRouteSpacingOverrideAllLayers.
- **keepPinSpace:** follows the pin spacing for copy route.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Spacing – Intermediate Sections</i> <i>General – Spacing</i>

Examples

```
envGetVal("we" "copyRouteSpacingMode")  
envSetVal("we" "copyRouteSpacingMode" 'cyclic "overrideAllLayers")  
envSetVal("we" "copyRouteSpacingMode" 'cyclic "keepPinSpace")
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

Copy Route Form

copyRouteSpacingOverrideAllLayers

copyRouteSpacingModeAtEnds

```
we copyRouteSpacingModeAtEnds cyclic { "defaultConstraint" | "overrideAllLayers" }
```

Description

Specifies the spacing at the first and last sections that are not connected to pins.

- **defaultConstraint:** uses the `minSpacing` defined in technology file. This is the default option.
- **overrideAllLayers:** uses the value specified for the `copyRouteSpacingOverrideAllLayersAtEnds`.

GUI Equivalent.

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Spacing – First/last Sections</i> <i>General – Spacing</i>

Examples

```
envGetVal("we" "copyRouteSpacingModeAtEnds")  
envSetVal("we" "copyRouteSpacingModeAtEnds" 'cyclic "overrideAllLayers")
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteEnableSpacingAtEnds](#)

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

copyRouteSpacingOverrideAllLayersAtEnds

copyRouteSpacingOverrideAllLayers

`we copyRouteSpacingOverrideAllLayers float overrideValue`

Description

Defines the spacing value to be used for copy route.

The default is 0.0.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i>
Field	<i>Spacing – Value</i> (intermediate sections)

Examples

```
envGetVal("we" "copyRouteSpacingOverrideAllLayers")  
envSetVal("we" "copyRouteSpacingOverrideAllLayers" 'float 0.05)
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteSpacingMode](#)

copyRouteSpacingOverrideAllLayersAtEnds

`we copyRouteSpacingOverrideAllLayersAtEnds float overrideValue`

Description

Defines the spacing value to be used at the first and last section that are not connected to pins for copy route.

The default is 0.0.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i>
Field	<i>Spacing – Value (first/last sections)</i>

Examples

```
envGetVal("we" "copyRouteSpacingOverrideAllLayersAtEnds")  
envSetVal("we" "copyRouteSpacingOverrideAllLayersAtEnds" 'float 0.05)
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteEnableSpacingAtEnds](#)

[copyRouteSpacingModeAtEnds](#)

copyRouteTopLayer

```
we copyRouteTopLayer string "topLayer"
```

Description

Specifies the top layer for copy route. This variable is effective only if the [copyRouteEnableValidLayer](#) variable is set to `t`.

The default is empty string "".

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Annotate Copies – Top</i> <i>General – Top/Bottom Layers</i>

Examples

```
envGetVal("we" "copyRouteBottomLayer")  
envSetVal("we" "copyRouteBottomLayer" 'string "Metall")
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteEnableValidLayer](#)

[copyRouteBottomLayer](#)

copyRouteTrimmingFixer

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

Description

Removes the last dangling section of copy route. This variable is effective only if the [copyRouteAllowCopiesWithoutMatchingPins](#) variable is set to `t`.

The default is `t`.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Remove last transition</i> <i>Options – Removes last dangling section</i>

Example

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

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteAllowCopiesWithoutMatchingPins](#)

copyRouteWidthMode

```
we copyRouteWidthMode cyclic { "sameAsReference" | "defaultConstraint" |  
    "overrideAllLayers" | "keepPinWidth" }
```

Description

Defines the copy width mode for the intermediate sections. Intermediate sections means the sections which are not connected to pins.

- **sameAsReference**: the copy width is exactly the same as the reference width. This is the default option.
- **defaultConstraint**: uses the minWidth that is defined in the technology file.
- **overrideAllLayers**: uses the value defined for the copyRouteWidthOverrideAllLayers.
- **keepPinWidth**: uses the pin width if the wire is coming from a pin.

GUI Equivalent.

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Width – Intermediate Sections</i> <i>General – Width</i>

Examples

```
envGetVal("we" "copyRouteWidthMode")  
envSetVal("we" "copyRouteWidthMode" 'cyclic "defaultConstraint")  
envSetVal("we" "copyRouteWidthMode" 'cyclic "overrideAllLayers")  
envSetVal("we" "copyRouteWidthMode" 'cyclic "keepPinWidth")
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

Virtuoso Interactive and Assisted Routing User Guide

Interactive and Assisted Routing Environment Variables

Setting Copy Route Options

Copying Pin-to-Pin Routes

Copy Route Form

copyRouteWidthOverrideAllLayers

copyRouteWidthModeAtEnds

```
we copyRouteWidthModeAtEnds cyclic { "sameAsReference" | "sameAsReferenceOrPin" |  
    "fromTechOnly" | "fromTechOrPin" | "fromUserOnly" | "fromUserOrPin" |  
    "sameAsPin" }
```

Description

Defines the width for connected pin sections.

- **sameAsReference**: sets the width of the copies to exactly the same as that of the reference width. This is the default option.
- **sameAsReferenceOrPin**: if the reference wire matches the pin width, then copies also match that pin width. Else, the reference width is considered.
- **fromTechOnly**: Sets the width of the copies to the `minWidth` value that is defined in the technology file.
- **fromTechOrPin**: if the reference wire matches the pin width, then copies also match that pin width. Else the `minWidth` defined in the technology file is considered.
- **fromUserOnly**: the copies use the value defined for `copyRouteWidthOverrideAllLayersAtEnds`.
- **fromUserOrPin**: if the reference wire matches the pin width, then copies also match that pin width. Else, the copies use the value defined for `copyRouteWidthOverrideAllLayersAtEnds`.
- **sameAsPin**: sets the width of the copies to use the same width value as that of the pin.

GUI Equivalent:

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Width – First/last Sections</i> <i>General – Width At Ends</i>

Examples

```
envGetVal("we" "copyRouteWidthModeAtEnds")  
envSetVal("we" "copyRouteWidthModeAtEnds" 'cyclic "sameAsPin")
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteEnableWidthAtEnds](#)

[copyRouteWidthOverrideAllLayersAtEnds](#)

copyRouteWidthOverrideAllLayers

`we copyRouteWidthOverrideAllLayers float overrideValue`

Description

Defines the width to be used for copy route for the intermediate sections.

The default is 0.0.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Width – Value</i> (intermediate sections) <i>General – Width</i>

Examples

```
envGetVal("we" "copyRouteWidthOverrideAllLayers")  
envSetVal("we" "copyRouteWidthOverrideAllLayers" 'float 0.05)
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteWidthMode](#)

copyRouteWidthOverrideAllLayersAtEnds

`we copyRouteWidthOverrideAllLayersAtEnds float overrideValue`

Description

Defines the width of copy routes for connected pin sections.

The default is 0.0.

GUI Equivalent

Command	<i>Window – Assistant – Routing – CopyRoute</i> <i>Edit – Wiring – Copy Route</i>
Field	<i>Width – Value (First/last sections)</i> <i>General – Width At Ends</i>

Examples

```
envGetVal("we" "copyRouteWidthOverrideAllLayersAtEnds")  
envSetVal("we" "copyRouteWidthOverrideAllLayersAtEnds" 'float 0.5)
```

Related Topics

[Variables to Customize the Settings for Copy Route](#)

[CopyRoute](#)

[Copy Route Command](#)

[Setting Copy Route Options](#)

[Copying Pin-to-Pin Routes](#)

[Copy Route Form](#)

[copyRouteEnableWidthAtEnds](#)

[copyRouteWidthModeAtEnds](#)

Bindkey Keyboard Map

To view the bindkeys commands, see Virtuoso Layout Suite [Bindkeys](#). For information about bindkeys, see [Bindkeys and Access Keys](#).

Virtuoso Interactive and Assisted Routing User Guide

Bindkey Keyboard Map

Routing Assistant - Interactive Mode

The Routing assistant is a simplified, consistent, flow-based assistant for all design types. It has the same look and feel as that of the automatic mode. The Routing assistant is a dockable assistant pane that provides various options to let you perform tasks related to interactive routing. The Routing Assistant for interactive mode is available in Layout EXL and higher tiers.

The Routing Assistant has the following components:

<u>Routing Assistant Tabs</u>	Lets you specify the options for running interactive routing commands.
<u>Routing Assistant Command Buttons</u>	Lets you access the buttons on each tab to complete an interactive routing task.

Routing Assistant Tabs

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

Tab	Description
<u>Setup</u>	Lets you specify the settings for routing in interactive mode.
<u>Wire</u>	Lets you specify the options for creating a wire.
<u>Bus</u>	Lets you specify the options for creating a bus.
<u>CopyRoute</u>	Lets you specify the options for copying routes.

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Setup

The following table describes the fields available on the *Setup* tab of the Routing assistant for routing in interactive mode.

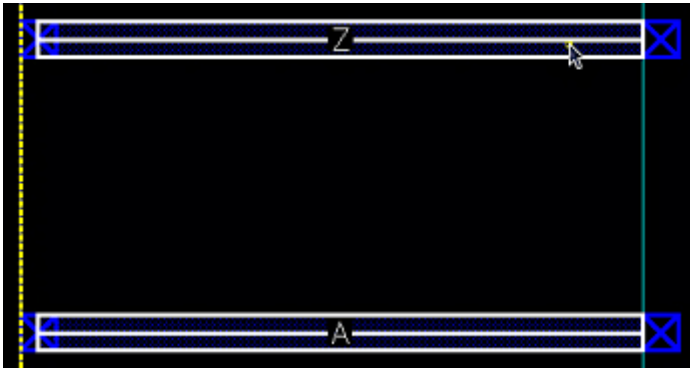
Field	Description
General	
<i>Snap Mode</i>	Controls how the cursor snaps to the grid when you create a bus. ■ <i>anyangle</i>: Creates wires at any angle. Can only be specified when using paths to create wires. ■ <i>diagonal</i>: Creates wires parallel to the X or Y axis or at a 45-degree angle to the axes. ■ <i>orthogonal</i>: Creates wires parallel to the X or Y axis. ■ <i>L90XFirst</i>: Creates orthogonal two-segment wires and creates the first line in the X direction. ■ <i>L90YFirst</i>: Creates orthogonal two-segment wires and creates the first line in the Y direction. Environment variable: <u>snapMode</u>

Virtuoso Interactive and Assisted Routing User Guide

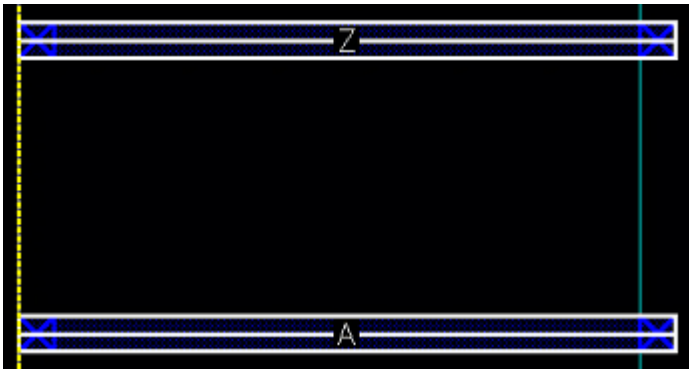
Routing Assistant - Interactive Mode

Field	Description
<i>Cover pin</i>	Adjusts the wire segment to automatically cover the source and target pins. Otherwise, the wire segment is not adjusted and the wire terminates at a point you had chosen. By default, this option is deselected.

Environment variable: weCoverPin



When Cover Pin is Deselected



When Cover Pin is Selected

<i>Auto terminate</i>	Finishes the interactive routing command as soon as a point is digitized on a flightline target object (pin, existing wire, or via). All wires snap to the target pin.
<i>Allow loops</i>	Permits creation of loops when creating or editing wires. The option is off by default. When off, the Create Wire command does not allow the creation of loops on the same layer. When the layer is changed while wire creation, loop checking is reset and applies only to the new layer.

Environment variable: allowLoops

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Auto merge wires</i>	Automatically merges the connected paths and collinear pathsegs while creating or stretching a wire. The check box is selected by default. To prevent automatic merging of collinear (symbolic or geometric) wires, disable the <i>Auto Merge Wires</i> option. Environment variable: <u>weAutoMergeWires</u>
Via	
<i>Adjust edited via params</i>	Enables the re-calculation of via alignment of the vias if you stretch the wires connected to them. This check box is not selected by default. Environment variable: <u>adjustEditedViasParams</u>
<i>Override variants</i>	Lets you specify a custom via alignment option that overrides the automatic alignment.
<i>Cut Enclosure Minimum</i>	Specifies the via cuts as minimum number of cuts for creating and editing wires. If the override value is less than either the technology rule or the current constraint group on the net or on the design, then the specified value is reset to the minimum value.
<i>Cut Spacing</i>	Lets you select the spacing for the number of cuts that can be added to a via. You can either select the minimum spacing between the via cuts or evenly distribute the via cuts.

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Calculation Mode</i>	Sets the calculation mode for via parameters to one of the following: ■ <i>Min Rules & ViaDef</i>: For all the via parameter fields, such as the number of cut rows and columns, and cut sizes, the minimum rule or the viaDef default values, whichever is greater, are used. If minimum rules are not defined, viaDef defaults are used. The spacing and enclosure values are calculated automatically from the minimum rules. The minimum rules are picked from the constraint group that you select from the Via list in the <i>Create Via</i> section. The calculated values for the spacing and enclosure fields are never less than the viaDef defaults. For example, if the minExtensionDistance rule is 0.05 and the default enclosure in viaDef is 0.1, 0.1 is used as the enclosure value in the <i>Create Via</i> form. Similarly, if the minSpacing rule is 0.2 and the default cut spacing in viaDef is 0.18, 0.2 is used as the cut spacing X and Y in the <i>Create Via</i> form. ■ <i>Minimum Rules</i>: For all the via parameter fields, only the minimum rules, such as minWidth and minSpacing, are used. For example, if the minExtensionDistance rule is 0.05 and the default enclosure in viaDef is 0.1, 0.05 is used as the enclosure value in the <i>Create Via</i> form. The viaDef default enclosure value is ignored. ■ <i>ViaDef Defaults</i>: For all the via parameter fields, viaDef defaults are used. When <i>ViaDef Defaults</i> option is selected, the via engine does not check any rules.
<i>Via Alignment</i>	Lets you align vias as they are digitized during wire or bus creation. You can choose to align a via either automatically or by selecting a custom alignment option. Vias are aligned between the edges of via enclosure shapes and the edges of the connecting wires on the same layer. For example, for a Metal1 to Metal2 via, the via location is adjusted in a way that the Metal1 wire edge is aligned with the Metal1 via enclosure and the Metal2 wire edge is aligned with the Metal2 via enclosure. The alignment offsets the via origin to maintain it at the intersection of the wire end points. If you are adding and aligning a via stack, the same via alignment is applied to all the vias in the stack. You can also access the via alignment options from the Create Stranded Wire Form. Environment variable: autoViaAlignment

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Orthogonal Transitions</i>	Controls alignment behavior for orthogonal transitions. ■ <i>In and out layers</i>: Aligns to the edges of both incoming and outgoing layers. This is the default option. ■ <i>In layer only</i>: aligns to the edge of the incoming layer only. ■ <i>Out layer only</i>: aligns to the edge of the outgoing layer only. ■ <i>None</i>: if selected no alignment is done for a via. Environment variable: autoViaAlignmentOrthoTransitionBehavior
<i>Collinear Transitions</i>	For collinear transitions controls alignment behavior for collinear transitions. ■ <i>In and out layers</i>: Aligns to the edges of both incoming and outgoing layers. This is the default option. ■ <i>None</i>: If selected no alignment is done for a via. Environment variable: autoViaAlignmentCollinearTransitionBehavior
Halo	Select a scope for creating a halo. ■ <i>End to Side EOL Spacing</i>: Shows the end-to-side halo spacing for the <code>minEndOfLineSpacing</code> constraint. ■ <i>End to End EOL Spacing</i>: Shows the end-to-end halo spacing for the <code>minEndOfLineSpacing</code> constraint. ■ <i>Both EOL Spacing</i>: Shows both end-to-side and end-to-end halo spacing for the <code>minEndOfLineSpacing</code> constraint. Environment variable: weHaloEOLSpacing ■ <i>Min Same & Diff Maskcolor</i> shows the minimum <code>diffMask</code> or the <code>sameMask</code> spacing for the <code>minEndOfLineSpacing</code> constraint. ■ <i>Max Same and Diff Maskcolor</i> shows the maximum <code>diffMask</code> or the <code>sameMask</code> spacing for the <code>minEndOfLineSpacing</code> constraint. ■ <i>Both Maskcolors</i> shows the minimum (<code>diffMask</code>, <code>sameMask</code>) spacing and the maximum (<code>diffMask</code>, <code>sameMask</code>) spacing of the available <code>minEndOfLineSpacing</code> constraint. Environment variable: weHalo

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Dashed</i>	Creates a dashed halo around a wire, bus, or stranded wire while creating or editing an object that violates a spacing constraint. By default, this option is deselected. Environment variable: weHaloDashStyle
<i>True Color</i>	Creates a halo in the color of that layer for each layer. By default, this option is deselected. Environment variable: weHaloTrueColor
Display	
<i>Orthogonal WSP grid</i>	Enables the display of grid end points to which the pathSeg is extended to while creating a wire, bus, and stretch.
<i>Dynamic WSP</i>	Enables the dynamic display of tracks, snapPatterns, and width spacing patterns.
<i>Snap Pattern</i>	Enables snap pattern snapping. Environment variable: snapPatternSnapping
<i>Path Display</i>	Determines whether you see path borders and centerlines, path borders only, or path centerlines only.
Trim Shape	
<i>Trim Purpose</i>	Specifies the purpose on which the trim shape should be inserted. If no value is specified for the purpose or if the purpose is invalid, the trim shape is inserted on the default purpose. The default purpose is <code>drawing</code> . Environment variable: weTrimTrimPurpose
<i>Bridge Purpose</i>	Specifies the purpose on which the bridge should be inserted. If no value is specified for the purpose, the bridge is not inserted. However, if the specified purpose is invalid, the bridge is inserted on the default purpose. The default purpose is <code>drawing</code> . Environment variable: weTrimBridgePurpose

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Patch Purpose</i>	Specifies the purpose on which the patch segment should be inserted. If no value is specified for the purpose or if the specified purpose is invalid, the patch segment is inserted on the default purpose. The default purpose is inherited from the wire or via purpose that extends up to the trim shape. Environment variable: <u>weTrimPatchPurpose</u>

Wire

The following table describes the fields available on the *Wire* tab of the Routing assistant for routing in interactive mode.

Field	Description
<i>Constraint Group</i>	Specifies the name of the constraint group that contains the foundry constraints.
<i>Nets</i>	The net name for the current wire.
<i>Width</i>	

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Width Mode</i>	Sets the width mode to one of the following: ■ <i>Same for all layers</i>: Specifies to use the same wire width for all layers. The width value is determined by the value specified in the <i>Width</i> field of the Create Wire form. The constraint overrides for all layers are available in the Wire Assistant table. ■ <i>Overrides by layer</i>: Sets the wire width the same as the result of the constraint lookup for <code>minWidth</code> in the technology file when the layer is not used. Else, the width is determined by the last width used for the layer. The width value from the Wire Assistant table can also be used for the wires on each layer. You can also use the value specified in the <i>Width</i> field for the wires on each layer. ■ <i>Default Constraints</i>: Sets the wire width same as the result of the constraint lookup for <code>minWidth</code> in the technology file or in any applicable constraint group defined at the design or object level. ■ <i>Source Objects</i>: This is the default option. The source object can be pins or wires. It is specified to use the maximum width determined by the width of the tapped object. Selecting the <i>Source Objects</i> option for the first segment of a wire ensures that <i>Tap Wires Width</i> or <i>Tap Pins Width</i> is selected, depending on the source object. You can also select <i>Source Objects</i> even if the first segments of the wires does not have the source object width. In such a case, the <i>Tap Wires Width</i> and <i>Tap Pins Width</i> option does not change. ■ <i>Target Object Narrow</i>: This option matches the last segment wire width to the length of the shortest edge of the target pin. The width of the remaining wire is unaffected. ■ <i>Target Object Wide</i>: This option matches the last segment wire width to the length of the longest edge of the target pin. The width of the remaining wire is unaffected.
<i>Width</i>	Displays the width of the current wire. You can override this value.
Automatic Insertion	
<i>Via on WSP Crossing</i>	Enables automatic insertion of via while crossing a WSP on a layer above or below.

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Trim</i>	Controls the interactive and post-edit insertion of trim shapes in interactive routing.
Via	
<i>DRC Fix</i>	Lets you create DRC clean vias for wires.
<i>minNumCuts</i>	Specifies the minimum number of cuts that can be added to a via. You can override the <code>minNumCut</code> rule value for creating and editing wires. If the override value is less than either the technology rule or the current constraint group on the net or on the design, then the value is reset to the minimum value.
<i>Row/col</i>	Specifies the via cuts as the number of rows or the number of columns. By doing this, you can override the <code>minNumCut</code> rule with a more explicit configuration for creating and editing wires. The increase and decrease in the number of via row and column is dependent on the minimum number of via cuts. number of via rows * number of via columns >= minimum number of via cuts.
<i>Maximize cuts</i>	Lets you get a larger cut size depending on the added metal.
Snapping & Tapping	
<i>Smart Snapping</i>	Controls how the cursor snaps when you create the wire.
<i>Tap Layer</i>	Taps the routing layer. Layers that can be tapped are determined from the technology file.
<i>Tap Width Enable</i>	Enables width tapping for objects: wire, via, and shapes and pins.
<i>Wire</i>	Taps an object's layers and attributes automatically when digitizing the first coordinate of a wire.
<i>Via</i>	Taps the width on the starting object for new wire when the starting object is a via. The edge of the via determines the width of the started wire.
<i>Shape and pin</i>	Taps a top-level shape, a top-level pin, or a level-1 instance layer and width attributes. The objects type is either a rectangle, a polygon, or a path. All valid routing layers can be tapped.

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Snap To</i>	Controls the snapping of wires to a grid. ■ <i>Off Grid</i> segments and vias can be created or moved anywhere without snapping. ■ <i>Manufacturing Grid</i>, snaps the wires to the manufacturing grid. This is the default option. ■ <i>Routing Grid</i> snaps the segments and vias other than the ones directly connecting to the off-grid pins to the routing grid. ■ <i>Track Pattern</i> snaps the segments and vias other than the ones directly connecting to the off-grid pins to track. The track patterns are saved on the design and so each design can have its own set of non-uniform track patterns. ■ <i>Snap Pattern</i> enables snapping of wires to the nearest snap pattern track.
<i>Pin center</i>	Snaps wires to the center of rectangular pins or of pins that are created as polygons and have rectangular shapes. If this check box is selected, wires snap to the starting and ending pins. The <i>Pin Center</i> check box is deselected by default.
<i>Via center</i>	Snaps wires to the center of vias. If this check box is selected and you click a via, the starting wire starts from the clicked point and two small segments are added to connect to the via origin. With this check box selected, tapping a via and a wire gives preference to the via. This check box is deselected by default.
<i>Halo</i>	Controls the display of halos in the layout canvas in order to make it easier to view them in the design.
<i>Show hints</i>	Controls the display of the <i>Interactive Routing Bindkey</i> hint box in Layout EXL when you start the <i>Create Wire</i>, <i>Create Bus</i>, or <i>Create Stranded Wire</i> commands. In Layout XL, the hint box is displayed only for the <i>Create Stranded Wire</i> command. The <i>Interactive Routing Bindkey</i> hint box describes the useful bindkeys and shortcuts for the interactive routing commands. This option is selected by default.
<i>Alignment Markers</i>	Dynamically displays an alignment arrow and an alignment marker. The arrow points to the closest object on the active flightline. The marker snaps the cursor to the target object edges and center as you move the cursor. Environment variable: <u>showAlignmentMarkers</u>

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Flightline</i>	Controls the display of flightlines during wire creation. You can hide the display of flightlines by setting it to <code>off</code> or display only three flightlines for each bus by setting it to <code>minimum</code> . The default is <code>on</code> .

Bus

The following table describes the fields available on the *Bus* tab of the Routing assistant for interactive mode routing.

Field	Description
<i>Constraint Group</i>	Specifies the name of the constraint group that contains the foundry constraints.

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Nets</i>	Lets you specify the net names for creating a bus. You can specify the net names in one of the following ways: ■ By specifying a bundle name, such as <code>netA</code>, <code>netB</code>, <code>netC<1></code>, <code>netD</code>, where the net names are separated by commas. ■ By using the bus notation, such as <code>net<0:3></code>. If you specify the net names in bus notation, the notation expands to display individual net names, such as <code>net<0></code>, <code>net<1></code>, <code>net<2></code>, <code>net<3></code>, in the <i>Net Name</i> field when you move the cursor to another field in the form. The number in the <i>Number of Bits</i> cyclic field updates based on the number of nets specified in the <i>Net Name</i> field. For example, the <i>Number of Bits</i> cyclic field updates to 4 for the bus notation <code>net<0:3></code>. ■ By specifying a bus base name, such as <code>netA</code> or <code>myBus</code>, and additionally setting up the bus cardinality in the <i>Number of Bits</i> cyclic field. The bus base name expands to display individual net names in the <i>Net Name</i> field only after you start creating the bus. For example, if you start creating a bus by specifying the bus base name as <code>netA</code> and setting the <i>Number of Bits</i> cyclic field to 10, the bus creation begins with 10 wires that are assigned to the <code>netA<0></code>, <code>netA<1></code>, <code>netA<2></code>, ..., <code>netA<10></code> nets. New nets are created if the ones specified do not already exist. If you start creating a bus from existing pins, the net names are automatically picked from the net of the pins. If you create the bus in free space without specifying any net names, the bus created is not assigned to any net and therefore, does not have any connectivity. Environment variable: <u>netName</u>

Width

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Number of Bits</i>	Specifies the number of wires in an unassigned bus. The default value is 2. ■ In the <i>Nets</i> field, if you specify net names as a bundle name or by using the bus notation, the value in the <i>Number of bits</i> cyclic field automatically updates to match the number of nets specified. If net names are specified in the <i>Nets</i> field and you manually update the value in the <i>Number of bits</i> cyclic field, the net names specified in the <i>Nets</i> field are removed. ■ In the <i>Nets</i> field, if you specify the net names by using the bus base name, you can set up the cardinality of the bus by using the <i>Number of bits</i> cyclic field. The bus base name expands to display individual net names in the <i>Net Name</i> field only after you start creating the bus. If you update the <i>Number of bits</i> cyclic field after starting bus creation, the net names are removed from the <i>Net Name</i> field. ■ If you change the number of bits from the default value, which is 2, to any other value, and then specify the net name, the number of bits does not get reset to default value 2. The last changed value is retained in the <i>Number of bits</i> cyclic field.

Environment variable: weBusNumBits

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Width Mode</i>	Sets the width mode to one of the following: ■ <i>Same for all layers</i>: Specifies to use the same wire width for all layers. The width value is determined by the value specified in the <i>Width</i> field of the Create Wire form. The constraint overrides for all layers are available in the Wire Assistant table. ■ <i>Overrides by layer</i>: Sets the wire width the same as the result of the constraint lookup for <code>minWidth</code> in the technology file when the layer is not used. Else, the width is determined by the last width used for the layer. The width value from the Wire Assistant table can also be used for the wires on each layer. You can also use the value specified in the <i>Width</i> field for the wires on each layer. ■ <i>Default Constraints</i>: Sets the wire width same as the result of the constraint lookup for <code>minWidth</code> in the technology file or in any applicable constraint group defined at the design or object level. ■ <i>Source Objects</i>: This is the default option. The source object can be pins or wires. It is specified to use the maximum width determined by the width of the tapped object. Selecting the <i>Source Objects</i> option for the first segment of a wire ensures that <i>Tap Wires Width</i> or <i>Tap Pins Width</i> is selected, depending on the source object. You can also select <i>Source Objects</i> even if the first segments of the wires does not have the source object width. In such a case, the <i>Tap Wires Width</i> and <i>Tap Pins Width</i> option does not change. ■ <i>Target Objects</i>: Sets each bus wire to match the width of the same edge of each respective target pin. If the pins are vertically aligned to bus wires, the bus wire widths match the width of the vertical edges of the respective target pins. Similarly, if the pins are horizontally aligned to bus wires, the bus wire widths match the width of the horizontal edges of the respective target pins. The horizontal and vertical edges might correspond to narrow edges for some target pins and wide edges for other target pins. This option cannot be selected and you need to use the <i>Target Pin Narrow Edge</i> or <i>Target Pin Wide Edge</i> options depending on the edge whose width you want. The narrow and wide edges for target pins is only applicable in case of single wire creation.

Automatic Insertion

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Via on WSP Crossing</i>	Enables automatic insertion of via while crossing a WSP on a layer above or below.
<i>Trim</i>	Controls the interactive and post-edit insertion of trim shapes in interactive routing.
Via	
<i>minNumCuts</i>	Specifies the minimum number of cuts that can be added to a via. You can override the <code>minNumCut</code> rule value for creating and editing wires. If the override value is less than either the technology rule or the current constraint group on the net or on the design, then the value is reset to the minimum value.
<i>Row/col</i>	Specifies the via cuts as the number of rows or the number of columns. By doing this, you can override the <code>minNumCut</code> rule with a more explicit configuration for creating and editing wires. The increase and decrease in the number of via row and column is dependent on the minimum number of via cuts. <code>number of via rows * number of via columns >= minimum number of via cuts.</code>
<i>Maximize cuts</i>	Lets you get a larger cut size depending on the added metal.
Snapping & Tapping	
<i>Smart Snapping</i>	Controls how the cursor snaps when you create the wire.
<i>Tap Layer</i>	Taps the routing layer. Layers that can be tapped are determined from the technology file.
<i>Tap Width Enable</i>	Enables width tapping for objects: wire, via, and shapes and pins.
<i>Wire</i>	Taps an object's layers and attributes automatically when digitizing the first coordinate of a wire.
<i>Via</i>	Taps the width on the starting object for new wire when the starting object is a via. The edge of the via determines the width of the started wire.
<i>Shape and pin</i>	Taps a top-level shape, a top-level pin, or a level-1 instance layer and width attributes. The objects type is either a rectangle, a polygon, or a path. All valid routing layers can be tapped.

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Snap To</i>	Controls the snapping of wires to a grid. ■ <i>Off Grid</i> segments and vias can be created or moved anywhere without snapping. ■ <i>Manufacturing Grid</i>, snaps the wires to the manufacturing grid. This is the default option. ■ <i>Routing Grid</i> snaps the segments and vias other than the ones directly connecting to the off-grid pins to the routing grid. ■ <i>Track Pattern</i> snaps the segments and vias other than the ones directly connecting to the off-grid pins to track. The track patterns are saved on the design and so each design can have its own set of non-uniform track patterns. ■ <i>Snap Pattern</i> enables snapping of wires to the nearest snap pattern track.
<i>Pin center</i>	Snaps wires to the center of rectangular pins or of pins that are created as polygons and have rectangular shapes. If this check box is selected, wires snap to the starting and ending pins. The <i>Pin Center</i> check box is deselected by default.
<i>Via center</i>	Snaps wires to the center of vias. If this check box is selected and you click a via, the starting wire starts from the clicked point and two small segments are added to connect to the via origin. With this check box selected, tapping a via and a wire gives preference to the via. This check box is deselected by default.
<i>Show hints</i>	Controls the display of the <i>Interactive Routing Bindkey</i> hint box in Layout EXL when you start the <i>Create Wire</i>, <i>Create Bus</i>, or <i>Create Stranded Wire</i> commands. In Layout XL, the hint box is displayed only for the <i>Create Stranded Wire</i> command. The <i>Interactive Routing Bindkey</i> hint box describes the useful bindkeys and shortcuts for the interactive routing commands. This option is selected by default.
<i>Alignment Markers</i>	Dynamically displays an alignment arrow and an alignment marker. The arrow points to the closest object on the active flightline. The marker snaps the cursor to the target object edges and center as you move the cursor. Environment variable: <u>showAlignmentMarkers</u>

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Controlled wire only</i>	Displays an alignment arrow and an alignment marker only if the control wire is aligned with its target pin. For more information about control wires, see Using the Control Wire . This checkbox is selected by default. Environment Variable: weShowCtrlWireAlignMarkerOnly
<i>Flightline</i>	Controls the display of flightlines during wire creation. You can hide the display of flightlines by setting it to <code>off</code> or display only three flightlines for each bus by setting it to <code>minimum</code> . The default is <code>on</code> .

CopyRoute

The following table describes the fields available on the *CopyRoute* tab of the Routing assistant for routing in interactive mode.

Field	Description
<i>Mode</i>	Describes the selected reference wire state. The target wires can have the following modes. ■ <i>No ref wire</i>: the reference wire is not yet selected. ■ <i>Connected to Pins</i>: the reference wire is either pin-to-space or pin-to-pin. You can switch to <i>In Space</i> mode and the result would be generated from space-to-space. ■ <i>In Space</i>: the reference wire is space-to-space.
<i>Number of Copies</i>	Defines the number of copies to be created when the first and last transitions of route copies are not connected to pins. Environment variable: copyRouteNumCopiesInSpace
<i>Copies Allow without matching pins</i>	Selects and creates copies of the unpaired net pin. Environment variable: copyRouteAllowCopiesWithoutMatchingPins
<i>Copies Remove last transition</i>	Removes the last dangling section of copy route. Environment variable: copyRouteTrimmingFixer
Width	Lets you specify the width settings for the routes being copied.

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Intermediate Sections</i>	Defines the copy width mode for intermediate sections. Intermediate sections are the sections which are not connected to pins. ■ <i>Same as reference</i>: the copy width is exactly the same as the reference width. This is the default option. ■ <i>From tech</i>: uses <code>minWidth</code> value that is defined in the technology file. ■ <i>From user</i>: uses the value defined for <code>copyRouteWidthOverrideAllLayers</code>. ■ <i>Same as pin</i>: uses the pin width if the wire is starting from a pin. Environment variable: <code>copyRouteWidthMode</code>
<i>User</i>	Specifies the width override value to be used for intermediate sections that are not connected to pins. This field is enabled only when the <i>From user</i> option is selected from the <i>Intermediate Sections</i> drop-down list. Environment variable: <code>copyRouteWidthOverrideAllLayers</code>
<i>Enable</i>	Specifies whether the widths of the first and last sections are independent of the intermediate sections width. When this option is not selected, the widths of the first and last sections is set to the width of the intermediate sections. Environment variable: <code>copyRouteEnableWidthAtEnds</code>

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>First/last Sections</i>	Specifies the width of the connected pin sections. ■ <i>Ref only</i>: the width of the copies is exactly the same as the reference width. This is the default option. ■ <i>Ref or pin</i>: if the reference wire matches the pin width, then copies also match that pin width. Else the reference width is considered. ■ <i>Tech only</i>: the copies use the <code>minWidth</code> value that is defined in the technology file. ■ <i>Tech or pin</i>: if the reference wire matches the pin width, then copies also match that pin width. Else the <code>minWidth</code> defined in the technology file is considered. ■ <i>User only</i>: the copies use the value defined for <code>copyRouteWidthOverrideAllLayersAtEnds</code>. ■ <i>User or pin</i>: if the reference wire matches the pin width, then copies also match that pin width. Else, the copies use the value defined for <code>copyRouteWidthOverrideAllLayersAtEnds</code>. ■ <i>Same as pin</i>: the copies use the same width value as that of the pin. ■ Environment variable: <code>copyRouteWidthModeAtEnds</code>
<i>User</i>	Specifies the width value of the first and last sections. This field is enabled only when the <i>User only</i> option is selected from the <i>First/last Sections</i> drop-down list. Environment variable: <code>copyRouteWidthOverrideAllLayersAtEnds</code>
Spacing	Lets you specify the spacing settings for the routes being copied.
<i>Intermediate Sections</i>	Defines the copy spacing mode for the intermediate sections. ■ <i>From tech</i>: uses <code>minSpacing</code> value defined in technology file. This is the default option. ■ <i>From user</i>: uses the value specified for <code>copyRouteSpacingOverrideAllLayers</code>. ■ <i>From pins</i>: follows the pin spacing for copy route. Environment variable: <code>copyRouteSpacingMode</code>

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>User</i>	Defines the spacing override value to be used for intermediate sections that are not connected to pins. This field is enabled only when the <i>From user</i> option is selected from the <i>Intermediate Sections</i> drop-down list. Environment variable: <code>copyRouteSpacingOverrideAllLayers</code>
<i>Enable</i>	Enables spacing at the first and last sections. Environment variable: <code>copyRouteEnableSpacingAtEnds</code>
<i>First/last Sections</i>	Specifies the spacing at the first and last sections that are not connected to pins. ■ <i>From tech</i>: uses <code>minSpacing</code> value defined in technology file. This is the default option. ■ <i>From user</i>: uses the value specified for <code>copyRouteSpacingOverrideAllLayersAtEnds</code>. Environment variable: <code>copyRouteSpacingModeAtEnds</code>
<i>User</i>	Specifies the spacing value to be used at the first and last sections. This field is enabled only when the <i>From user</i> option is selected from the <i>First/last Sections</i> drop-down list. Environment variable: <code>copyRouteSpacingOverrideAllLayersAtEnds</code>
Vias	Lets you specify the via settings for copy routes.
<i>Preserve cut classes</i>	Generates copies with the same <code>cutClass</code> as that of the reference wire. Environment variable: <code>copyRoutePreserveCutClasses</code>
Ellipses button	Displays the Select Via Options form. This button is enabled only when the <i>Preserve cut classes</i> option is deselected.
Layers	Lets you specify the layer settings for copy routes.

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>Intermediate Sections</i>	Specifies the layer mode for copy route. ■ <i>Same as reference</i>: This is a single layer mode that does not let you select pins whose layer is different from the reference pin. This is the default option. ■ <i>Based on pins</i>: Similar to <i>Follow Reference Layer Pattern Except Pins</i>, but the intermediate section layer is adjusted based on the pin layer difference. ■ <i>Based on pattern</i>: specifies the user-defined pattern. Environment variable: <u>copyRouteLayersMode</u>
<i>Pattern</i>	Specifies the layer pattern to be followed for copy route. The layer pattern is a list of integers. Environment variable: <u>copyRouteLayerPattern</u>
Preview	Lets you specify the preview settings for copy routes.
<i>Show preview</i>	Displays the preview of the copy route result. Environment variable: <u>copyRoutePreviewMode</u>
<i>Effort</i>	You can select the effort as <i>Low</i> or <i>High</i> for the displayed preview. ■ <i>Low</i>: Displays a preview. The preview computation is faster than <i>High</i> but less accurate. ■ <i>High</i>: Displays a preview. The preview computation is more accurate than <i>Low</i> but slower. Environment variable: <u>copyRoutePreviewMode</u>
<i>Clear preview when cmd cancelled</i>	Automatically clears the route preview when the copy route command is canceled. Environment variable: <u>copyRouteClearPreviewWhenEsc</u>
Copies	Lets you specify the grouping options for copy routes.
<i>In group</i>	Creates the route copies in a fig group. Environment variable: <u>copyRouteCreateFigGroup</u>

Virtuoso Interactive and Assisted Routing User Guide

Routing Assistant - Interactive Mode

Field	Description
<i>With Errors</i>	Selects one of the following options for the copy route results with violations: ■ <i>None</i>: generates the result without any violations. The copy route is stopped as soon as there is a single violation. ■ <i>All</i>: generates the result no matter how many violations are present. This is the default option. ■ <i>First Only</i>: generates the result with any violation. The copy route is stopped as soon as there is a violation, but the first copy route with violations is kept. Environment variable: <u>copyRouteGenerateCopyWithErrors</u>
Annotate Copies	Lets you specify the annotation settings for copy routes.
<i>Out of layer range</i>	Annotates layers that are outside the top and bottom layer range.
<i>Top</i>	Specifies the top layer for copy route. Environment variable: <u>copyRouteTopLayer</u>
<i>Bottom</i>	Specifies the bottom layer for copy route. Environment variable: <u>copyRouteBottomLayer</u>
<i>Outside routing area</i>	Annotates layers that are outside the user-defined layer-purpose pair area. Environment variable: <u>copyRouteEnableRoutingChannel</u>
<i>Layer</i>	Lets you select the user-defined routing area layer-purpose pair area. Environment variable: <u>copyRouteRoutingChannelLPP</u>

Routing Assistant Command Buttons

The following table lists the functions of the different command buttons on the Routing assistant for interactive mode.

Icon	Command	Description
	<i>Create Wire</i>	The <i>Create Wire</i> icon on the <i>Wire</i> tab lets you create pathSegs and vias that are in routes.
	<i>SDR Checker Mode: Enforce</i>	Estimates the current or resistance in the edited wire and vias according to the EAD settings (dataset, temperature, and current scaling) and automatically calculates the wire width and via cuts to avoid EM violations.
	<i>Automatically Connect Twigs</i>	Toggles the twig modes between ON and OFF.
	<i>Create Bus</i>	The <i>Create Bus</i> icon on the <i>Bus</i> tab lets you create a bus. All wires of the bus, referred to as bus bits, are created on the same layer.
	<i>Start Copy Route</i>	Starts the copy route command.
	<i>Copy</i>	Runs the copy route command. It is similar to pressing the <code>Enter</code> key.
	<i>Clear preview</i>	Clears the preview of the copied routes.

Related Topics

[Copy Route Form](#)

[Setting Copy Route Options](#)

[Copy Route Command](#)

[Copying Pin-to-Pin Routes](#)

[Variables to Customize the Settings for Copy Route](#)