Preparation for use with Quick Cells

Creating a New Library to Use With QCells

Setting Units Correctly for QCells

Using an Existing Library

Inserting a QCell Device

Using Layout Optimization

Using Properties

Property Definitions

Preparing Pins

Extra Pins in the Symbol or Layout Views

External Connections

4

Setting Up the Virtuoso XL Environment

Setting Up Your Desktop

Customizing Your Desktop Layout

Using Multiple Cellviews

Printing to the Command Interpreter Window

Changing Display Colors

Editing Entry Layers

Using Bindkeys

Displaying Bindkeys

Loading Bindkeys

Setting Environment Variables

Using the Layout XL Options Form

Using Setup Files or the CIW

Online Forms

Layout XL Options

5

Device Abutment

Requirements for Abutment

Setting Up Cells for Abutment

Snapping Instances in the Direction Perpendicular to the Abutment

Multiple Pins
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Pins on Ignored Instances</td>
<td>74</td>
</tr>
<tr>
<td>Steps in Automatic Abutment</td>
<td>75</td>
</tr>
<tr>
<td>Sample Parameterized Cells Set Up for Abutment</td>
<td>75</td>
</tr>
<tr>
<td>Creating CMOS Pcells to Use with Abutment</td>
<td>78</td>
</tr>
<tr>
<td>MOS Abutment without a User-Defined abutFunction</td>
<td>81</td>
</tr>
<tr>
<td>Automatic Abutment Properties</td>
<td>81</td>
</tr>
<tr>
<td>Manual Device Abutment</td>
<td>81</td>
</tr>
<tr>
<td>Chaining Transistors Interactively</td>
<td>82</td>
</tr>
<tr>
<td>Chaining Transistors during Pick From Schematic</td>
<td>85</td>
</tr>
<tr>
<td>Moving Chained Transistors</td>
<td>86</td>
</tr>
<tr>
<td>Automatic Device Abutment</td>
<td>88</td>
</tr>
<tr>
<td>Chaining Transistors Automatically during Layout Generation</td>
<td>89</td>
</tr>
<tr>
<td>Chaining Transistors Automatically during Update Components and Nets</td>
<td>89</td>
</tr>
<tr>
<td>Abutting Parameterized Cells and Quick Cells</td>
<td>90</td>
</tr>
</tbody>
</table>

6 Generating Your Layout

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naming Conventions</td>
<td>91</td>
</tr>
<tr>
<td>Starting Virtuoso XL from the Schematic</td>
<td>94</td>
</tr>
<tr>
<td>Importing a Netlist for a Connectivity Reference</td>
<td>96</td>
</tr>
<tr>
<td>Starting Virtuoso XL from the Layout View</td>
<td>98</td>
</tr>
<tr>
<td>Connectivity Reference as a Netlist</td>
<td>100</td>
</tr>
<tr>
<td>Virtuoso XL Map File Structure</td>
<td>102</td>
</tr>
<tr>
<td>Using a Map File in a Schematic Driven Flow</td>
<td>104</td>
</tr>
<tr>
<td>Defining the Design Boundary</td>
<td>105</td>
</tr>
<tr>
<td>Automatic Area Estimation</td>
<td>106</td>
</tr>
<tr>
<td>User-Defined Area Estimation Functions</td>
<td>107</td>
</tr>
<tr>
<td>Working with Template Files</td>
<td>109</td>
</tr>
<tr>
<td>Saving Form Contents</td>
<td>110</td>
</tr>
<tr>
<td>Loading Template Files</td>
<td>110</td>
</tr>
<tr>
<td>Modifying Template Files</td>
<td>111</td>
</tr>
<tr>
<td>Loading Template Files</td>
<td>111</td>
</tr>
<tr>
<td>Creating Template Files</td>
<td>112</td>
</tr>
<tr>
<td>Template File Syntax</td>
<td>113</td>
</tr>
<tr>
<td>General Syntax Rules</td>
<td>114</td>
</tr>
</tbody>
</table>
Virtuoso XL Layout Editor User Guide

7 Editing Your Layout

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open File</td>
<td>167</td>
</tr>
<tr>
<td>Pick from Schematic</td>
<td>168</td>
</tr>
<tr>
<td>Remove Correspondence Components</td>
<td>170</td>
</tr>
<tr>
<td>Set Pin Label Text Style</td>
<td>170</td>
</tr>
<tr>
<td>Startup Option</td>
<td>172</td>
</tr>
<tr>
<td>Template File</td>
<td>172</td>
</tr>
<tr>
<td>Identifying Incomplete Nets</td>
<td>174</td>
</tr>
<tr>
<td>Moving Objects Manually</td>
<td>177</td>
</tr>
<tr>
<td>Moving Objects Using Move Options</td>
<td>178</td>
</tr>
<tr>
<td>Setting the Move Form to Appear Automatically</td>
<td>182</td>
</tr>
<tr>
<td>Swapping Components</td>
<td>182</td>
</tr>
<tr>
<td>Permuting Component Pins</td>
<td>183</td>
</tr>
<tr>
<td>Permuting Pins Manually</td>
<td>185</td>
</tr>
<tr>
<td>Checking Permutation Information</td>
<td>186</td>
</tr>
<tr>
<td>Updating Permuted Pins</td>
<td>187</td>
</tr>
<tr>
<td>Using Device Locking</td>
<td>187</td>
</tr>
<tr>
<td>Component Types</td>
<td>188</td>
</tr>
<tr>
<td>MOS Transistor Chaining and Folding Parameters</td>
<td>188</td>
</tr>
<tr>
<td>Defining a Component Type</td>
<td>190</td>
</tr>
<tr>
<td>Modifying a Component Type</td>
<td>193</td>
</tr>
<tr>
<td>Using Transistor Folding</td>
<td>193</td>
</tr>
<tr>
<td>Folding Transistors</td>
<td>194</td>
</tr>
<tr>
<td>Adding Instances to a Layout</td>
<td>197</td>
</tr>
<tr>
<td>Adding Pins to a Layout</td>
<td>199</td>
</tr>
<tr>
<td>Assigning Pins to a Net</td>
<td>201</td>
</tr>
<tr>
<td>Using Edit-In-Place</td>
<td>202</td>
</tr>
<tr>
<td>Setting the Default Application</td>
<td>203</td>
</tr>
<tr>
<td>Online Forms</td>
<td>203</td>
</tr>
<tr>
<td>Assign Nets</td>
<td>204</td>
</tr>
<tr>
<td>Edit Component Types</td>
<td>204</td>
</tr>
<tr>
<td>Move</td>
<td>206</td>
</tr>
<tr>
<td>Set Transistor Folding</td>
<td>207</td>
</tr>
</tbody>
</table>
8
Using the Placer

Main Features ......................................................... 213
Place Menu Command Summary ................................. 214
Other Commands Used with the Placer ......................... 215
Placement Styles ....................................................... 215
Setting Up Virtuoso XL for Placement ......................... 217
Identifying the Placement Translation Rules ................. 217
Setting Cadence Design Framework II Environment Variables ........................................... 218
Setting MOS Chaining and Folding Parameters ............... 219
Defining the Boundary Layer (Placement Region) ............ 219
Abutting Standard Cells ............................................. 219
Using Automatic Abutment During Placement ................. 220
Placement Constraints ................................................. 221
Constraint Manager Geometric Constraints .................... 221
Pin Placement Constraints ......................................... 226
Constraint Limitations ................................................. 227
Placement Parameters and Component Types .................. 227
MOS Transistor Chaining and Folding Parameters .......... 229
Defining Standard Cell Substrate Contacts .................... 229
Pin Placement .......................................................... 230
Placing a Pin on a Boundary Edge ............................... 231
Placing Ordered Pins on a Boundary Edge ..................... 232
Placing a Pin in a Fixed Position not on the Boundary ....... 233
Placing a Pin on a Lower-level Instance Terminal .......... 233
Assigning Spacing Between Pins .................................. 234
Converting a Pin into a Rail ......................................... 234
Converting a Rail into a Pin ......................................... 235
Loading a Template File ............................................. 236
Saving Pin Placement to a Template File ....................... 236
Partitioning the Design ............................................... 237
Creating a Partition ................................................. 238
Loading Information from a Template File ................. 240
Saving Partitions to a Template File ......................... 241
Setting Placement Planning ...................................... 241
  Assisted CMOS Placement .................................... 241
  Using Assisted Standard Cell Mode ......................... 251
  Manual User-Defined Row-Based Placement ................. 255
  Defining User-Programmed Styles .......................... 257
  Choose Component Types Form .............................. 264
Running the Placer ................................................ 265
  Prerequisites to Placement .................................. 266
  Running the Placer: Initial Placement ....................... 266
  Stopping the Placer ........................................... 269
  Running Load Balancing Service (LBS) ....................... 269
  Troubleshooting Placement Results ......................... 269
  Refining Placement Results ................................ 269
  Running the Placer: Detailed Placement .................... 270
Showing Congestions .............................................. 270
Troubleshooting Placement ..................................... 271
  Error Messages ............................................... 271
Online Forms ....................................................... 271
  Auto Placer .................................................... 272
  Partitioning .................................................... 275
  Choose Component Types ................................... 277
  Pin Placement .................................................. 277
  Load Template File ........................................... 280
  Placement Planning .......................................... 280
  Placement Planning (Assisted Standard-Cell) .............. 284
  Placement Planning (Assisted Mixed CMOS/Standard-Cell) 287
  Placement Planning (Manual User-Defined) .................. 292

9
Preparing Your Design for Routing ................................ 295
  Connectivity Extraction ...................................... 296
    Top-Level Design Extraction .............................. 297
## 10 Wire Editing

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>330</td>
</tr>
<tr>
<td><strong>Main Features</strong></td>
<td>330</td>
</tr>
<tr>
<td>Wire Editing Commands</td>
<td>330</td>
</tr>
</tbody>
</table>

---

**Virtuoso XL Connectivity Extraction**

- Hierarchical Extraction .................................................. 298
- Via Mosaic Support (CDB only) .......................... 302

**Pseudo-Parallel Connections** ............................................ 302
- What is a Pseudo-Parallel Connection .................. 303
- Examples of Pseudo-Parallel Connections ............ 304
- Defining a Pseudo Parallel Connection ............... 307
- Resetting a Pseudo-Parallel Connection ............. 308
- Generating a Layout without Chaining and Folding on Pseudo-Parallel Nets .................. 309
- Generating a Layout with Chaining and Folding on Pseudo-Parallel Nets ................. 311
- Using Pick From Schematic with Pseudo-Parallel Nets ........................................ 312

**Layer Selection** ............................................................ 312
- Selecting a Layer in the Layer Selection Window .... 312
- Selecting a Layer Automatically when Creating a Rectangle, Polygon, or Path .......... 313

**Connecting Nets** ........................................................... 314
- Creating Paths .............................................................. 315
- Connecting Nets with Path Stitching .................... 317
- Connecting Nets with Design Shapes ...................... 318

**Checking Connectivity with Flight Lines** .................. 319

**Checking Connectivity with Markers** .................. 320
- Finding Markers ............................................................. 320
- Explaining Markers ....................................................... 320
- Deleting Single Markers .............................................. 321
- Deleting All Markers .................................................... 321

**Physical Vias** ............................................................. 322
- Defining a Physical Via ................................................ 322
- Using Via Mosaics ........................................................ 325

**Using the Virtuoso Compactor on a Routed Design** .......... 326

**Online Forms** .............................................................. 327
- Virtuoso XL Connectivity Extraction .................... 327

**Online Forms** .............................................................. 327
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtuoso Router to Virtuoso XL Command Mapping</td>
<td>331</td>
</tr>
<tr>
<td>Requirements</td>
<td>332</td>
</tr>
<tr>
<td>Rule Information</td>
<td>332</td>
</tr>
<tr>
<td>Net Connectivity Information</td>
<td>333</td>
</tr>
<tr>
<td>Routing Area Boundary</td>
<td>333</td>
</tr>
<tr>
<td>The Wire Editing Environment</td>
<td>333</td>
</tr>
<tr>
<td>Valid Connectivity Objects</td>
<td>334</td>
</tr>
<tr>
<td>Types of Routing Objects</td>
<td>334</td>
</tr>
<tr>
<td>Net Assignment</td>
<td>335</td>
</tr>
<tr>
<td>Preview Wires and Routing Aids</td>
<td>336</td>
</tr>
<tr>
<td>Mouse Button Behavior</td>
<td>336</td>
</tr>
<tr>
<td>Grid Controls</td>
<td>337</td>
</tr>
<tr>
<td>Using Environment Variables</td>
<td>338</td>
</tr>
<tr>
<td>Enabling Wire Editing</td>
<td>338</td>
</tr>
<tr>
<td>Virtuoso XL Autorouting</td>
<td>338</td>
</tr>
<tr>
<td>Loading ASCII Rules Files</td>
<td>338</td>
</tr>
<tr>
<td>Routing Wires</td>
<td>340</td>
</tr>
<tr>
<td>Routing a Single Wire</td>
<td>340</td>
</tr>
<tr>
<td>Routing Multiple Wires</td>
<td>342</td>
</tr>
<tr>
<td>Routing Floating Wires</td>
<td>344</td>
</tr>
<tr>
<td>Preventing and Checking Design Rule Violations</td>
<td>345</td>
</tr>
<tr>
<td>Interactive Checking</td>
<td>345</td>
</tr>
<tr>
<td>Same Net Checking</td>
<td>346</td>
</tr>
<tr>
<td>Checking Regions</td>
<td>346</td>
</tr>
<tr>
<td>Checking Route and Pin Violations</td>
<td>347</td>
</tr>
<tr>
<td>Routing Options and Styles</td>
<td>349</td>
</tr>
<tr>
<td>Matching Wire Width and Pin Widths</td>
<td>349</td>
</tr>
<tr>
<td>Matching Wire Width and Pin Widths for Multiple Wires</td>
<td>352</td>
</tr>
<tr>
<td>Gathering Bus Wires</td>
<td>353</td>
</tr>
<tr>
<td>Spacing for Gathered Bus Wires</td>
<td>353</td>
</tr>
<tr>
<td>Overriding Bus Spacing</td>
<td>355</td>
</tr>
<tr>
<td>Rotating the Bus Cursor</td>
<td>356</td>
</tr>
<tr>
<td>Cycling the Control Wire</td>
<td>356</td>
</tr>
<tr>
<td>Working with Insufficient Routing Space</td>
<td>356</td>
</tr>
<tr>
<td>Allowing Redundant Wiring</td>
<td>357</td>
</tr>
<tr>
<td>Allowing Orthogonal Jogs</td>
<td>358</td>
</tr>
</tbody>
</table>
Route To Cursor ......................................................... 359
Using Alternate Views ............................................... 362
Connecting Multiple Component Pins ......................... 363
Pushing Routes and Components ................................... 363
Routing Shielded Nets ................................................ 364
Routing Tandem Layer Pairs ...................................... 365
Showing Length Rules Constraints ............................... 366
Using Vias .............................................................. 369
Changing Layers and Adding Vias ............................... 369
Using Vias Patterns on Multiple Wires ......................... 370
Adding Vias to Gathered Bus Wires ............................. 371
Legal Via Sites ......................................................... 372
Rotating Vias .......................................................... 372
Minimum Number of Cuts ......................................... 373
Pseudo Vias .............................................................. 373
Using Default Vias .................................................... 373
Stacking Vias ........................................................... 374
Editing Routed Connections ....................................... 374
Stretching Wires and Vias ......................................... 375
Splitting and Stretching Wires ................................... 375
Copying Routes ......................................................... 378
Removing Notches and Extra Wire Bends ...................... 379
Compacting Wires Using Pull .................................... 381
Reporting ................................................................. 382
Displaying Routing Status Reports ............................ 383
Displaying Network Reports ..................................... 384
Displaying Component Reports .................................. 385
Displaying Net Reports ............................................. 387
Creating Rules Reports ............................................ 389
Search Reports ......................................................... 391
Saving Reports ........................................................ 392
Setting Constraints .................................................... 392
Using the Virtuoso Constraint Manager ....................... 393
Using .do Files ........................................................ 393
Online Forms .......................................................... 396
Add Via ................................................................. 396
11 Checking Design Data ................................................. 413
  Finding Design Elements ........................................ 414
    Probing Hierarchical Designs ................................. 417
    Removing Probes ................................................ 420
    Exiting the Probe Command .................................. 420
    Showing the Options Form ..................................... 420
  Checking Shorts and Opens ...................................... 420
  Comparing Design Elements (Check Against Source) .......... 422
    Using CDF Callbacks with Check Against Source .......... 423
  Online Forms .................................................... 423
    Probe Options .................................................. 423

12 Updating Design Data ............................................. 425
  Updating Components and Nets (Engineering Change Order Mode) ........... 426
Updating Layout Parameters ........................................... 432
Updating Schematic Parameters ...................................... 434
Updating Device Correspondence .................................... 436
  Updating Connectivity from Schematic to Layout ................. 437
  One-to-Many Assignment with Update Device Correspondence .. 438
  Using Many-to-Many or Many-to-One Mapping ..................... 441
  Modifying Many-to-Many or Many-to-One Mapping between Components .... 443
  Deleting Many-to-Many or Many-to-One Mapping between Components . 443
Creating Device Correspondence .................................... 444
  Needed Mode ................................................................ 444
  Computer Aided Mode .................................................. 448
Updating the Connectivity Reference ................................. 449
Changing the Device (Instance) View ................................. 450
Online Forms ................................................................ 451
  Change Instance View ................................................... 451
  Change Inst View Device List Form / Update Layout Device List Form / Update Schematic Device List Form .......... 452
  Create Device Correspondence ...................................... 453

13
Troubleshooting ............................................................ 454
  Problems with the Interface ........................................... 454
  Problems with Editing .................................................. 454
  Problems with Connectivity .......................................... 454
  Problems with Cloning .................................................. 455
  Problems with Pick From Schematic ................................. 455
  Problems with Parameter Evaluation ................................. 455
Problems with the Interface ........................................... 455
  Invalid Markers from Previous Software Versions ................. 455
  Options Form Does Not Appear .................................... 456
  Virtuoso XL Performance Is Slow ................................... 457
Problems with Editing .................................................. 457
  Components Move Slowly .............................................. 457
  Extra Probes Appear .................................................... 457
  Layout Generation Options Form Does Not Keep Values from the Last Entry .... 458
Parameters Not Updated ............................................................. 458
Schematic Not Editable ............................................................... 458
Warning to Update Your Design Appears at Startup ....................... 458

Problems with Connectivity ......................................................... 459
Connections Not Made ................................................................ 459
Incomplete Nets Command Does Not Recognize Connected Pins and Nets 459
Markers for Nonexistent Overlaps and Shorts Appear ....................... 460
Path Ends Not Accepted ............................................................... 460
Placement and Routing Do Not Run ................................................. 461
Virtuoso XL Does Not Recognize Physical Vias ............................... 461
Moving Software Executables To a New Location ............................ 461

Problems with Cloning ................................................................. 461
Structure Already Exists in the Layout ............................................. 462
Connectivity Structure is Different ................................................ 463
Master Cells are Different ............................................................ 463
More Source Components are Selected than Target Components ......... 464
Properties are Different ............................................................... 465

Problems with Pick From Schematic .............................................. 466
Hierarchical Designs ................................................................ 467
Global Terminals and Implicit Inherited Connection Terminals ........... 467

Problems with Parameter Evaluation ............................................. 467
Netlisting Mode ........................................................................ 467
Evaluating CDF Callbacks by Default ............................................ 468

A

Environment Variables .................................................................. 470
Inherited Environment Variable Settings ........................................ 470

Virtuoso XL Layout Editor ............................................................ 471
abutPerpSnapOn .................................................................... 471
allowRotation ........................................................................ 472
autoAbutment .......................................................................... 473
autoArrange ............................................................................. 474
autoExtractEnabled ................................................................. 475
autoExtractSaveEnabled ............................................................ 476
autoSpace ................................................................................. 477
checkTimeStamps .................................................. 478
checkOldIgnoredParamsProps ................................ 479
ciwWindow ........................................................ 480
cloningDoExactMatch ........................................... 481
colorDraglines .................................................... 482
compTypeRefLibs ............................................... 483
constraintAssistedMode ...................................... 484
createBoundaryLabel .......................................... 485
crossSelect ....................................................... 486
deleteUnmatchedInsts .......................................... 487
deleteUnmatchedPins ........................................... 488
drdUseNetName .................................................. 489
extractEnabled .................................................. 490
extractShowMustConnectMarkers .............................. 491
extractShowUnimplementedInstTerms ......................... 492
extractShowWeakMarkers ...................................... 493
extractStopLevel ................................................. 494
flightLineEnable ................................................. 495
globalPlacement .................................................. 496
hideDraglinesForGlobalNets ................................... 497
ignoreNames ........................................................ 498
IgnoredParams ..................................................... 499
IgnoreParamsForCAS ............................................ 500
incNetCycleHilite ............................................... 501
incNetHiliteLayer ............................................... 502
infoWindow ........................................................ 503
initAspectRatio .................................................... 504
initAspectRatioOption .......................................... 505
initBoundaryLayer ............................................... 506
initCreateBoundary .............................................. 507
initCreateInstances ............................................ 508
initCreateMTM ..................................................... 509
initCreatePins .................................................... 510
initDoFolding ..................................................... 511
initDoStacking .................................................... 512
initEstimateArea .................................................. 513
initGlobalNetPins .................................................. 514
initIOLabelType .................................................. 515
initIOPinLayer .................................................. 516
initIOPinName .................................................. 517
initPinHeight ................................................... 518
initPinMultiplicity ............................................. 519
initPinWidth ................................................... 520
initPrBoundaryH .................................................. 521
initPrBoundaryW .................................................. 522
initSymbolicPins .................................................. 523
initUtilization .................................................. 524
layoutWindow .................................................... 525
IswWindow ......................................................... 526
lxAllowPseudoParallelNets ..................................... 527
lxDeltaWidth .................................................... 528
lxEvalCDFCallbacks ............................................. 529
lxFingeringNames ............................................... 530
lxGenerationOrientation ....................................... 531
lxGenerationTopLevelOnly ..................................... 532
lxGetSignifDigits ................................................ 533
lxGroundNetNames .............................................. 534
lxInitResetSource ............................................... 535
lxLocalAbutment ............................................... 536
lxSchExtractTopLevelOnly ..................................... 537
lxStackMinimalFolding .......................................... 538
lxStackPartitionParameters .................................... 539
lxSupplyNetNames ............................................... 540
lxToggleLocalAbutment ......................................... 541
lxUseLibList ..................................................... 542
lxWidthTolerance ............................................... 543
maintainConnections ........................................... 544
mfactorNames ................................................... 545
mfactorSplit .................................................... 546
mfactorSplitParamNames ....................................... 547
moveAsGroup .................................................... 548
openWindowTitle ................................................ 549
Virtuoso XL Layout Editor User Guide

schematicWindow .................................................. 588
setPPConn ............................................................ 589
sfactorNames .......................................................... 590
sfactorParam .......................................................... 591
shapeSwitchLayer ...................................................... 592
showDraglinesForDistantConns ..................................... 593
showIncNetEnable ..................................................... 594
stopList ................................................................. 595
templateFileName ..................................................... 596
draglineMixedHierarchies ........................................... 597
updateReplacesMasters .............................................. 598
updateWithMarkers ................................................... 599
cellboundaryLPPs ...................................................... 600
cellConductorDepth ................................................... 601
cellGlobalPathScript .................................................. 602
keepoutDepth ........................................................... 603
cellVerboseLevel ....................................................... 604
writeToCIW ............................................................. 605
viewList ................................................................. 606

Wire Editor ............................................................... 607
allowJogs ................................................................. 607
allowRedundantWiring ............................................... 608
allowSameNetPicking ............................................... 609
allowViaOnCurrentLayer ............................................ 610
alternateViews ........................................................ 611
autoAdjustLength ...................................................... 612
autoShield ............................................................... 613
busOverride ............................................................. 614
busOverrideValue ...................................................... 615
busWireSpacing ........................................................ 616
busWireSpacingType .................................................... 617
checkConflict .......................................................... 618
checkCornerCorner .................................................... 619
checkCrosstalk ........................................................ 620
checkLength ............................................................ 621
checkLimitWay ........................................................ 622
checkMaxProcessWireWidth ........................................ 623
checkMaxStackViaDepth .............................................. 624
checkMaxTotalVia .................................................. 625
checkMinMaskEdgeLength ........................................... 626
checkMinProcessWireWidth ......................................... 627
checkMiter .......................................................... 628
checkNetOrder ........................................................ 629
checkOffManGridPin .................................................. 630
checkOffWireGridPin .................................................. 631
checkPinSpacing ....................................................... 632
checkPolygonWire ..................................................... 633
checkProtected ........................................................ 634
checkReentrantPath ................................................... 635
checkRegion ........................................................... 636
checkSameNet .......................................................... 637
checkSegment ........................................................... 638
checkStub ............................................................... 639
checkUseLayers ......................................................... 640
checkUseVias ........................................................... 641
checkWireExtension ..................................................... 642
doFile ................................................................. 643
enableBusRouting ..................................................... 644
enableTandemPair ...................................................... 645
fitViaPattern ........................................................... 646
gatherBusWires .......................................................... 647
inaccessiblePin ........................................................ 648
interactiveChecking .................................................... 649
matchPinWidth .......................................................... 650
matchPinWidthValue .................................................... 651
matchTargetPinWidth ................................................... 652
matchTargetPinWidthValue ........................................... 653
matchWidthSticky ...................................................... 654
matchWireWidth ........................................................ 655
multiplePinsConnection ............................................... 656
pinLargerMaxProcessWidth ......................................... 657
pinSmallerMinProcessWidth ......................................... 658
Virtuoso XL Layout Editor User Guide

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>pushComponent</td>
<td>659</td>
</tr>
<tr>
<td>pushRouting</td>
<td>660</td>
</tr>
<tr>
<td>routeAsManyAsPossible</td>
<td>661</td>
</tr>
<tr>
<td>routeToCursor</td>
<td>662</td>
</tr>
<tr>
<td>routeToCursorStyle</td>
<td>663</td>
</tr>
<tr>
<td>sameNetChecking</td>
<td>664</td>
</tr>
<tr>
<td>showTimingMeter</td>
<td>665</td>
</tr>
<tr>
<td>showTimingOctagon</td>
<td>666</td>
</tr>
<tr>
<td>snapToPinOrigin</td>
<td>667</td>
</tr>
<tr>
<td>useDisplayGrid</td>
<td>668</td>
</tr>
<tr>
<td>useDoFile</td>
<td>669</td>
</tr>
<tr>
<td>useRulesFile</td>
<td>670</td>
</tr>
<tr>
<td>viaAssistance</td>
<td>671</td>
</tr>
<tr>
<td>viaPattern</td>
<td>672</td>
</tr>
</tbody>
</table>

B
Command Quick Reference 674

C
Using Spice and CDL for Netlist-Driven Layout Generation 678

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>678</td>
</tr>
<tr>
<td>Specifying Spice Designs</td>
<td>678</td>
</tr>
<tr>
<td>Cell Creation Rules</td>
<td>678</td>
</tr>
<tr>
<td>Character Considerations</td>
<td>680</td>
</tr>
<tr>
<td>Case</td>
<td>680</td>
</tr>
<tr>
<td>The * Character</td>
<td>680</td>
</tr>
<tr>
<td>The $ Character</td>
<td>680</td>
</tr>
<tr>
<td>Characters in Names</td>
<td>681</td>
</tr>
<tr>
<td>Spice Statements</td>
<td>681</td>
</tr>
<tr>
<td>Conventions for Format Descriptions</td>
<td>681</td>
</tr>
<tr>
<td>File Level Statements</td>
<td>682</td>
</tr>
<tr>
<td>Statements Allowed at File Level or within a Subckt Cell or a Top Level Cell</td>
<td>685</td>
</tr>
<tr>
<td>Statements Allowed within a Subckt Cell or a Top Level Cell</td>
<td>685</td>
</tr>
<tr>
<td>Spice Design Example</td>
<td>689</td>
</tr>
<tr>
<td>CDL Design Example</td>
<td>693</td>
</tr>
</tbody>
</table>
Parameter Resolution .......................................................... 696
  Specifying Parameters .................................................. 696
Parameter Levels ............................................................. 697
Resolving Parameters ...................................................... 697
Sample Rules ..................................................................... 699
Parameter Scaling .............................................................. 703
Complete ibuf Example Results ........................................... 705

D
.do File Commands ............................................................. 728
Rule Hierarchy ................................................................. 728
Commands ......................................................................... 730
circuit .............................................................................. 730
define/forget .................................................................... 731
do ................................................................................... 732
limit .................................................................................. 733
rule .................................................................................. 734
set ................................................................................... 735

E
Properties ............................................................................ 736
  abutAccessDir .................................................................. 738
  abutClass ....................................................................... 739
  abutCondInclusion ......................................................... 740
  abutFunction .................................................................. 741
  abutGateNet .................................................................... 744
  abutMosStretchMat ....................................................... 745
  abutStretchMat ............................................................ 748
  abutOffset ..................................................................... 750
  ignore ............................................................................ 752
  lxAutoAbut .................................................................... 753
  lxAutoSpace ................................................................... 754
  lxBlockExtractCheck .................................................... 755
  lxBlockOverlapCheck .................................................... 756
  lxCombination ............................................................. 757
| **lxIgnoredParams** | 760 |
| **lxIgnoreParamsForCAS** | 761 |
| **lxMfactorSplit** | 762 |
| **lxNetNamePrefix** | 763 |
| **lxParamsToIgnore** | 765 |
| **lxParamsToIgnoreForCheck** | 767 |
| **lxRemoveDevice** | 768 |
| **lxRounding** | 771 |
| **lxSeriesTerms** | 773 |
| **lxStickyNet** | 774 |
| **lxStopList** | 776 |
| **lxUseCell** | 778 |
| **lxVcpSubContMaxSpacing** | 780 |
| **lxVcpSubContMinSpacing** | 782 |
| **lxViewList** | 784 |
| **mfactor** | 786 |
| **permuteRule** | 789 |
| **sfactor** | 792 |
| **vxlnsSpacingDir** | 795 |
| **vxlnsSpacingRule** | 797 |
Preface

This guide describes how to use the Virtuoso® XL Layout Editor. It is for developers and designers of integrated circuits and assumes that you are familiar with

- The Cadence® Design Framework II environment and the application infrastructure mechanisms designed to support consistent operations between all Cadence tools.
- The DFII on OpenAccess technology file.
- The Component Description Format, which lets you create and describe your own components for use with Virtuoso XL.
- The design and use of Quick Cells and other types of parameterized cells.
- The applications used to design and develop integrated circuits in the DFII environment, notably the Virtuoso Layout Editor.
- The procedures used to translate designs for routing with the Virtuoso Chip Assembly Router.

Related Documents

The following documents provide more information about the topics discussed in this user guide.

Installation, Environment, and Infrastructure

- For information on installing Cadence products, see the Cadence Installation Guide.
- For information on the DFII environment, see the Cadence Design Framework II User Guide.
- For information on database SKILL functions, including data access functions, see the Design Framework II SKILL Functions Reference.
- For information on library structure, the library definitions file, and name mapping for data shared by multiple Cadence tools, see the Cadence Application Infrastructure User Guide.
Technology Information

- For information on how to create and maintain a technology file and display resource file, see the *Technology File and Display Resource File User Guide* and the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

- For information on how to access the technology file using SKILL functions, see the *Technology File and Display Resource File SKILL Reference Manual*.

Virtuoso Tools

- For information on how to perform design tasks with the Virtuoso layout editor, see the *Virtuoso Layout Editor User Guide*.

- For information on Quick Cells and the design-rule-driven (DRD) editing functionality, see the *Virtuoso Layout Editor Turbo User Guide*.

- For information on creating parameterized cells using the graphical user interface or low-level SKILL functions, see the *Virtuoso Parameterized Cell Reference* and the *Cell Design Tutorial*.

- For information on Component Description Format, see the *Component Description Format User Guide*.

- For information on how to route a design, see the *Virtuoso Chip Assembly Router Guide*.

- For information on how to stream mask data, see the *Design Data Translator’s Reference*.

- For information on custom layout SKILL functions, see the *Custom Layout SKILL Functions Reference*.

Relative Object Design and Inherited Connections

- For information on using relative object design (ROD) functions, see the *Virtuoso Relative Object Design User Guide*.

- For information on using inherited connections and net expressions with various Cadence tools, see *Inherited Connections Flow Guide*.

- For information on connectivity and naming conventions for inherited connections, and how to add and edit net expressions in a schematic or symbol cellview, see *Virtuoso Schematic Editor User Guide*.
Typographic and Syntax Conventions

This section describes typographic and syntax conventions used in this manual.

text Indicates text you must type exactly as it is presented.

\texttt{z_argument} Indicates text that you must replace with an appropriate argument. The prefix (in this case, \texttt{z_}) indicates the data type the argument can accept. Do not type the data type or underscore.

\[] Denotes optional arguments. When used with vertical bars, they enclose a list of choices from which you can choose one.

\{} Used with vertical bars and encloses a list of choices from which you must choose one.

| Separates a choice of options.

... Indicates that you can repeat the previous argument.

=> Precedes the values returned by a Cadence® SKILL language function.

/ Separates the possible values that can be returned by a Cadence SKILL language function.

text Indicates names of manuals, menu commands, form buttons, and form fields.

\textbf{Important}

The SKILL language requires many characters not included in the preceding list. You must type these characters exactly as they are shown in the syntax.
Introduction

The Virtuoso® XL Layout Editor is a connectivity-based editing tool that automates each stage of the design task, from component generation through automatic and interactive routing.

When used as part of an automated custom physical design methodology, Virtuoso XL lets you generate custom layouts from schematics or netlists and edit existing layouts that have defined connectivity. It continuously monitors connections of components in the layout and compares them with connections in the schematic. You can use Virtuoso XL to view incomplete nets, shorts, invalid connections, and overlaps to help you wire your design.

The following chapters contain more information on these topics.

- Editing Your Technology File
- Preparing Your Connectivity Source
- Setting Up the Virtuoso XL Environment
- Device Abutment
- Generating Your Layout
- Editing Your Layout
- Using the Placer
- Preparing Your Design for Routing
- Wire Editing
- Checking Design Data
- Updating Design Data
- Troubleshooting

For information on Virtuoso XL SKILL functions, see the Custom Layout SKILL Functions Reference Manual.
Editing Your Technology File

This chapter explains how to edit the technology file for your design so that you can generate, place, and route a layout using the Virtuoso® XL Layout Editor (Virtuoso XL). It contains specific information about the elements you need to prepare and discusses the following topics.

- Technology File Requirements on page 31
  - Layer Rules on page 32
  - Devices on page 33
  - Physical Rules on page 34
  - Virtuoso XL Rules (lxRules) on page 34
  - Compactor Rules on page 36
- Sample Technology File on page 38

The Virtuoso layout editor and other Cadence® layout applications require technology-specific information about your design to be stored in a technology file for the design library. You can use Cadence SKILL language functions to query or update the technology file.

- For information on creating and editing technology files, see the Technology File and Display Resource File User Guide.
- For information on using SKILL functions to query and update technology files, see the Technology File and Display Resource File SKILL Reference Manual.

The layout editor software includes a sample technology file that you can use or adapt for the needs of your design.

Technology File Requirements

The technology file information you need to run Virtuoso XL includes additions to the following sections of the technology file:
Layer information in the Layer Rules class

Symbolic contact information in the Devices class

Device spacing information in the Physical Rules class

Connectivity information in the Virtuoso XL Rules (lxRules) class

Compactor information in the Compactor Rules class

Note: If you have information in the Device-Level Editor (DLE) Rules class in your technology file, you do not have to rewrite it; the Virtuoso XL software automatically translates dleRules into lxRules.

Layer Rules

In the layerRules class, Virtuoso XL requires the equivalentLayers subclass to specify layers that have equivalent connectivity and the viaLayers subclass to define layers used for vias and the layers they connect. This software uses this information to identify short and open circuits.

- **equivalentLayers** (called leEquivalent in version 4.3.4 and earlier) defines layers that are electrically equivalent. In the following example, when the metal1 layer touches the vddmetal1 layer, the two layers form a connection.

  ```
  layerRules(
      equivalentLayers(
        ( metal1 vddmetal1 )
      );equivalentLayers
  );layerRules
  ```

- **viaLayers** (called leEquivalent in version 4.3.4 and earlier) define layers used for vias and the layers they connect. The layers must be named in the correct sequence. In the following example, the layers in the first column (metal1 and poly1) are the lower layers and the layers in the rhid column (metal2 and metal1) are the upper layers.

  ```
  layerRules(
      viaLayers(
        ;( layer1 viaLayer layer2 )
        ;( ------- --------- ------- )
        ( metal1 via metal2 )
        ( poly1 contact metal1 )
      );viaLayers
  );layerRules
  ```
If you have contact and via layers defined as equivalent to their corresponding interconnect layers, you can flatten contacts or draw new shapes to maintain or create connectivity.

⚠️ **Important**

If you use devices from different libraries for your design, the layer and connectivity information in the technology file for each library must be compatible.

💡 **Tip**

Cadence recommends that you define and use symbolic vias for all of the required cut and layer combinations, including base layers. Doing so means that you do not need viaLayer definitions for these cuts and layers. Use viaLayer definitions only if you want to connect flat shapes on polysilicon, contact or metal1 layers without using a symbolic via. For more information on defining symbolic vias, see Devices.

### Devices

In the Devices class, Virtuoso XL requires symbolic contact (symContactDevice) definitions for instances used as vias and for path stitching.

```plaintext
devices

symContactDevice(
  ;( name viaLayer viaPurpose layer1 purpose1 layer2 purpose2
  ; w 1 ( row column xPitch yPitch xBias yBias ) encByLayer1 encByLayer2
  ; legalRegion )

  ( M2_M1 via drawing metal1 drawing metal2 drawing
    .6 .6 ( 1 1 1.8 1.8 center center ) .6 .6
    _NA_ )
)
);symContact Device

);devicesdevice
```

You need contact definitions if you want to

- Place contacts using the **Create – Contact** command.
- Perform path stitching (which automatically places contacts) using the **Create – Path** command.
- Perform path stitching with the routing commands.

**Note:** The mpu.tf file has sample contact definitions in the devices class.
Physical Rules

The *Create – Path* command requires that in the `minWidth` parameter is defined in the `spacingRules` subclass of the `physicalRules` class. The `minWidth` parameter defines the minimum width of objects and the default width of paths drawn on a particular layer.

```
physicalRules(
    spacingRules(
        ; ( rule layer value )
        ; ( ---- ----- ----- )
        ( minWidth cont 0.60 )
    )
);spacingRules
);physicalRules
```

Virtuoso XL Rules (lxRules)

In the `lxRules` class, Virtuoso XL requires information about layer-purposes in the following subclasses.

- `lxExtractLayers` (`leConducting` in versions 4.3.4 and earlier) defines conducting layers.
- `lxNoOverlapLayers` (`leOverlap` in versions 4.3.4 and earlier) defines layers that must not overlap.

**Note:** In previous releases, the `lxRules` class was called the `dleRules` class.

`lxExtractLayers`

**Note:** All purposes within a layer are electrically equivalent unless they are defined separately in the `layerRules` class. Sequence is not important.

```
Tip
To optimize the performance of your applications, define only the layer name in the `lxExtractLayers` section.
```

You must specify layers used for connectivity with `lxExtractLayers`. In this example, all `metal1` and `metal2` layers connect components. The `lxExtractLayers` list must be on a single line, with no end-of-line or return characters.
Virtuoso XL Layout Editor User Guide
Editing Your Technology File

lxRules(
  lxExtractLayers(
    ( metal1 metal2 )
  );lxExtractLayers
);
xRules

You must set lxExtractLayers for a layer before you can set lxNoOverlapLayers and equivalentLayers for that layer.

⚠️ Important
In order for the abutment, folding, and chaining features to work correctly, you must define the diffusion layers in the lxExtractLayers section.

Defining lxExtractLayers with Layer Purposes

⚠️ Important
You should specify a layer-purpose pair in the lxExtractLayer section only when you need that layer-purpose pair to be treated differently from the other purposes for that layer.

If you specify only the layer name in the lxExtractLayers section, then all purposes on that layer are treated the same by the extractor. Consider the example below.

lxExtractLayers(
  ( layer1 layer2 )
);lxExtractLayers

equivalentLayers(
  ( ( layer1 purpose1 ) layer2 )
);equivalentLayers

In this example, layer1 purpose1 is equivalent to layer2 and layer1 purpose2 is also equivalent to layer2. layer1 purpose2 is not listed in the equivalentLayers section, but because layer1 is defined in the lxExtractLayers section, all purposes associated with layer1 are used.

If you define the layer-purpose pair in the lxExtractLayers section, each layer-purpose pair is treated differently by the extractor. You define the layer-purpose pairs in the lxExtractLayers section as shown below.

lxExtractLayers(
  ( ( layer1 purpose1 ) ( layer1 purpose2 ) layer2 )
);lxExtractLayers
equivalentLayers(
    ( ( layer1  purpose1 )  layer2 )
);equivalentLayers

**Note:** This works the same with `lxNoOverlapLayers`.

### lxNoOverlapLayers

`lxNoOverlapLayers` specifies that two layers that must not overlap. In this example, when the `poly` layer touches the `diff` layer, Virtuoso XL reports an error.

```
lxRules(
    lxNoOverlapLayers(
      poly  diff
    );lxNoOverlapLayers
  );lxRules
```

**Note:** If you are working with a hierarchical design, you can set properties on instances at the lower levels of the design to avoid false overlap and short markers.

The `lxBlockOverlapCheck` property, defined on a shape, instance, or instance master, tells the extractor not to check whether a nonoverlap layer of this shape or instance is touching a nonoverlap layer at the current cellview level.

For example, if you define the `lxBlockOverlapCheck` property on a shape in cellview “A”, you will not see any markers when overlapping the shape with an instance. However, if cellview “A” is placed inside cellview “B”, and a `noOverlap` layer is drawn over the shape in cellview “A”, then a marker is displayed. This happens even though the shape in cellview “A” has the `lxBlockOverlapCheck` property defined.

The `lxBlockExtractCheck` property, defined on a shape, instance, or instance master, tells the extractor not to check the connectivity between this shape and others it touches.

For example, when you move a floating shape (i.e., a shape stamped on a net but with no `lxStickyNet` property or with `lxStickyNet` set to false) with the `lxBlockExtractCheck` property, the shape remains on its net after the move.

### Compactor Rules

To use the Virtuoso compactor to compact your design, you need to include information about layer-purposes in the `compactorRules` class.

- The `compactorLayers` subclass defines the layer names and purposes used in the design.
The `symWires` subclass specifies symbolic wires and how the compactor processes them when it compacts a design.

```
symWires(
  ;( name layer [(impLayer impSpacing)] [(default min max)]
  ;[(legalRegion regionLayer)] [WLM] )
  ;( ----------------------------------------------- )
    ("poly1" ("poly1" "drawing") nil (.6 nil nil)
      nil 5.0 )
    ("pdiff" ("diff" "drawing") ("pimplant" "drawing") .3) (.6 nil nil)
      ("outside" "pwell") 100.0 )
    ("ndiff" ("diff" "drawing") nil (.6 nil nil)
      ("inside" "pwell") 100.0 )
    ("metal1" ("metal1" "drawing") nil (.6 nil nil)
      nil 0.04 )
    ("metal2" ("metal2" "drawing") nil (.6 nil nil)
      nil 0.02 )
    ("metal3" ("metal3" "drawing") nil (1.2 nil nil)
      nil 0.02 )
  );symWires
```

The `symRules` subclass checks enclosures between objects on different layers and spacings between objects on the same layer. The `drc` functions specify that rules apply only to particular devices or objects on the same or different nets.

```
symRules(
  ; list of rules
    drc("cont" "drawing" "M1_POLY1") ("diff" "drawing" "NTR") (sep < .6))
    drc("cont" "drawing" "M1_POLY1") ("diff" "drawing" "PTR") (sep < .6))
  );symRules
For more information on the compactorRules and symWires subclasses, see the Technology File and Display Resource File SKILL Reference Manual.

For more information on using the Virtuoso compactor, see the Virtuoso Compactor Reference Manual.

Sample Technology File

Virtuoso XL includes a sample technology file called mpu.tf that you can use as is or change to meet the requirements of your design. The file contains data for a two-layer metal CMOS process and information about Virtuoso XL technology file requirements. To see the contents of mpu.tf, type the following in a terminal window.

```
  cd your_install_dir/tools/dfII/samples/techfile
  more mpu.tf
```

where your_install_dir is the directory in which you store the Cadence software.

The mpu.tf technology file covers the following areas.

- Differences between the pre-4.4 versions and later versions of the technology file
- Layer definitions (layer numbers, purposes, priorities, and connectivity)
- Symbolic devices (including symbolic contacts)
- Physical rules, such as spacing rules (including minWidth) and ordered spacing rules
- Electrical rules
- Virtuoso XL-specific rules
- Other application-specific rules

You must define this information in the technology files referenced by the design.
Preparing Your Connectivity Source

This chapter explains how to prepare a schematic or netlist connectivity source for your design for generating, placing, and routing a layout with the Virtuoso® XL Layout Editor (Virtuoso XL).

This chapter covers the following topics.

- Placing Design Elements on page 39
- Using Design Variables on page 40
- Using One-to-Many Mapping on page 42
- Preparing a Library for use with Quick Cells on page 43
- Using Properties on page 49
- Preparing Pins on page 51

Placing Design Elements

You can use any schematic to generate a layout with Virtuoso XL. However, the process is easier if you follow these guidelines when you create your schematic:

- Place symbols of devices and pins in the schematic approximately where you want them to be in the layout. The Virtuoso XL Edit – Place As In Schematic command lets you place layout devices and create pins in positions that correspond to the positions of their schematic symbols.

- Use consistent parameter defaults for comparable symbols

Virtuoso XL applies the parameters of the symbols to the layout devices.
Important

Virtuoso XL does not pass CDF parameters that are not evaluated (are not AEL expressions) and have `cdfParamRec->storeDefault=nil`. These are master parameters and are expected to match for both schematic and layout master. Virtuoso XL does not pass them to the layout instance even when `lxUseCell` is used. If you require the parameters to be passed, set `cdfParamRec->storeDefault` to `t` for the schematic master parameters.

Note: When transferring information from the schematic to the layout, Virtuoso XL flattens the schematic (expands schematic symbols into corresponding devices) if you provide a view list or a stop list to show Virtuoso XL which view of the lower-level instances to use.

Virtuoso XL also supports the use of

- Design variables
- One-to-many mapping
- Many-to-many and many-to-one mapping

If you are preparing a hierarchical design, you also need to make sure the symbol view of each top-level design element is mapped to the correct layout view of the corresponding layout element for generating the layout. For more information, see “Hierarchical Extraction” on page 298.

Using Design Variables

You can use the following design variables with Virtuoso XL.

- Netlist Processor Expressions
- Analog Expression Language Expressions
- Simulation Design Variables

Netlist Processor Expressions

Netlist Processor (NLP) expressions are properties that specify parameter values. These expressions are used by the Open Simulation System (OSS) in netlisting.

For more information, see the *Open Simulation System Reference*. 
Analog Expression Language Expressions

Analog Expression Language (AEL) expressions, such as \texttt{iPar} and \texttt{pPar}, define the value of a parameter as a function of other instance parameters or parameters passed from other levels of hierarchy. If you specify the value of a parameter using an AEL expression, the parameter

- Must be defined in the component description format (CDF) for the cell of which the symbol is a view.
- Must be a string for which \texttt{parseAsNumber} and \texttt{parseAsCEL} properties are set to \texttt{t}.
- Must not have a CDF callback (because the evaluation of the expression does not trigger the execution of the callback).

If Layout XL detects a parameter value defined with \texttt{iPar}, \texttt{pPar}, or other AEL expressions not defined in the CDF, you see a warning in a message box.

Netlisting Mode

To ensure that CDF parameters are always evaluated correctly, make sure that the \texttt{CDS\_Netlisting\_Mode} shell environment variable is set to \texttt{Analog}.

To set \texttt{CDS\_Netlisting\_Mode},

\begin{itemize}
  \item Type the following commands in the CIW.
    \begin{verbatim}
    setShellEnvVar("CDS\_Netlisting\_Mode=Analog")
    cdsSetNetlistMode()
    \end{verbatim}
\end{itemize}

To check which mode is currently set,

\begin{itemize}
  \item Type the following command in the CIW.
    \begin{verbatim}
    cdsGetNetlistMode()
    \end{verbatim}
\end{itemize}

For more information on \texttt{CDS\_Netlisting\_Mode}, see “\texttt{Customizing the Simulation Environment}” in the \texttt{Open Simulation System Reference}.

For more information on AEL expressions, see “\texttt{Scope of Parameters}” in Chapter 3 of the \texttt{Cadence Analog Circuit Design Environment User Guide}.
Simulation Design Variables

When you use simulation design variables to specify the value of a parameter in the circuit, Virtuoso XL uses the value last saved during the simulation of the circuit as the value for the layout implementation.

For more information about simulation design variables, see “Design Variables and Simulation” in Chapter 3 of the Cadence Analog Circuit Design Environment User Guide.

Using One-to-Many Mapping

You can implement one-to-many mapping in Virtuoso XL designs using

- Iterated Instances and Bus Pins
- The Multiplication Factor (mfactor)
- The Series-connected Factor (sfactor)
- One-to-Many Assignment with Update Device Correspondence

Iterated Instances and Bus Pins

You can use iterated instances and bus pins in a schematic with Virtuoso XL. Use of iterated instances enables one-to-many mapping between pins and devices in the schematic and multiple instances of the pins and devices in the layout.

Note: For more information on iterated instances, see “Adding Instances Using an Iterative Expression” in the Virtuoso Schematic Editor User Guide.

For example, the figure below shows

- Bus pin SEL<0:1> in the schematic mapped to bus pins SEL<0> and SEL<1> in the layout.
Iterated instance $K^{0:1}$ mapped to instances $IK(0) \mid P0$, $IK(0) \mid N0$, $IK(1) \mid P0$, and $IK(1) \mid N0$ in the layout.

Preparing a Library for use with Quick Cells

Quick Cells (QCells) are parameterized cells (currently MOS devices, substrate/well ties, cdsVias, and guard rings) that are installed from the Install QCell form, which lets you specify names, design rules, stretch handles, and layer definitions for devices.

The installation program prompts you graphically for process design rule values that are not already defined in the technology file. After the first devices are created, it becomes much easier to create slightly different variants because you need to supply only process rule values that do not already exist in the technology file.

For more information about QCells, see “Quick Cells,” in the Virtuoso Layout Editor Turbo User Guide.
Creating a New Library to Use With QCells

Creating the Library

1. Choose the technology file to be loaded into the new library.
   - Use the Tools – Technology File Manager command to dump out a technology file from an existing library.
   - Use an existing ASCII technology file.

2. Remove from the technology file any reference to layout cellviews from another library.

   **Caution**

   *When you install Quick Cells into a new or existing technology library, the software either creates a new layout view or overwrites an existing layout. To preserve the original versions, rename the existing NMOS and PMOS layout cellviews to a different view name. For example, rename nmoslayout to nmoslayout.orig.*

3. In the CIW, select Tools – Technology File Manager and click the New button.

4. Specify a name for the new technology library, turn on the Load ASCII Technology File option, and specify an ASCII technology file.

5. Click OK to create the new library.

Copying Symbols

1. Decide which NMOS and PMOS symbols you want to copy from an existing library.

2. In the CIW, select Tools – Library Manager and select the symbols that you want to copy into the new library.

3. In the Library Manager choose Edit – Copy and change the To Library field to your new library name.

4. Deselect the Copy All Views checkbox in order to make the Views To Copy field editable. If there are any layout views listed in this field, remove them. You will create your layouts by installing QCells instead.

5. The Copy Problems dialog box appears. Click Fix Errors and select Update Instances. Next select New Copy then click OK. The cell is copied to your new library.

6. Repeat the steps for the other symbols.
Remastering Symbol Instances

The names of the schematic symbols must match the names of the MOS devices you create using Quick Cells. If you know the names of the MOS devices you will create, complete the following steps.

⚠️ **Important**

If the symbol name does not match the corresponding MOS device, then you must add the `lxUseCell` property to the symbol. For more information, see “Setting the `lxUseCell` Property” on page 45.

2. Choose the name of the schematic you have copied.
4. In the Search For field select `instName` from the cyclic field and type in the name of the current symbol.
5. In the Replace With field select `instName` from the cyclic field and type in the name of the MOS device to be created.
6. Click Replace.

Setting the `lxUseCell` Property

Virtuoso XL places the layout with the same cell name as the symbol in the schematic. If you do not want the name to be the same, you must add the `lxUseCell` property to the corresponding symbol in the schematic. You do this using the Edit Object Properties form.

⚠️ **Caution**

*Cadence recommends that you do NOT set the `lxUseCell` property for mixed-signal designs. For digital designs, the cells can be in different libraries. For mixed-signal designs, you must use a CDF to define the parameters for the devices. Therefore, the symbol and layout must be in the same library and cell.*

1. In the schematic, select the symbol for which you want to set the property.
2. Choose Edit – Properties – Objects.
3. Click the Add button to display the Add Property form.
4. Specify the Name of the cell, the Type as string, and the Value as the library and cell name.

**Installing Qcells**

1. Choose Tools – Technology File Manager and click QCell.
2. Select the technology library from the pull-down list.
3. Specify the name of the MOS device in the Name field.
4. Specify Diffusion, Gate Poly, Contact, and Metal layers.
5. Click Rules to display the Rules Browser.
   
   The Rules Browser provides a graphical representation of a generic device, and the layers that you specified in the Layers section of the Install QCell form.
6. The rules are taken from the technology file attached to your library. If there are any rules missing, add them in the Rules section of the Install QCell form.
   
   When you position the cursor in the rules fields, the Rules Browser window displays an arrow to show the physical area where the corresponding rule applies.
7. Choose Stretch Handles and turn on the Select All option or just the stretch handles that you want to use.
8. Choose Parameter Defaults. The form is partially grayed out until you specify the Component Class.
9. Set the Abutment Class to either NMOS or PMOS depending on whether you are creating an NMOS or PMOS device. Alternatively, you can specify a user-defined abutment class.
   
   **Note:** The abutment class is the device name.
10. Set the Component Class to either NMOS or PMOS.
   
   The information from the Component Class field updates the component type for that device in the Edit Component Types form. You can verify that the device has been updated in the Component Type form after you click Apply or OK.
   
   **Note:** The purpose of component types is to identify the NMOS and PMOS transistor cells and set their parameters for device chaining and folding.
11. Specify the MOS Type. The entire form is now editable.
   
   The CDF Parameters section updates with default CDF parameters or defined CDF if present.
Important

You can edit the Parameter Defaults or units in the Install QCell form, but you cannot change the parameter types once they have been applied. Use Tools - CDF - Edit command to change the component parameters.

12. Set the Unit cyclic field to either meter or micron. The CDF unit meter = lengthMetric and micron = don’t use.

Caution

You cannot reset or edit the Unit field.

13. Set the values for w, l, and fingers as needed (e.g., w=2u, l=.18u, fingers=1).
14. Click Apply to create the QCell.
15. Repeat the step for the other device (NMOS or PMOS) that needs to be created.
16. Save the new technology library.

Setting Units Correctly for QCells

If the symbol view corresponding to the QCell have dimension units defaulting to lengthMetric (i.e., meter rather than micron), you must enter the values in the schematic with the appropriate extension (e.g., 1u or 1000n or 1e-06 instead of simply 1, which would be interpreted as 1M).

The same applies to setting the folding threshold in the various Virtuoso XL forms. However, using the Property Editor form to edit QCells offers you the flexibility to enter values both with and without the unit extension.

Note: Typically most PDKs have callbacks to prevent you from entering a value such as this erroneously in the schematic.

Using an Existing Library

If you are using an existing technology library, consider the following.

- The symbols must contain CDF parameters.
- In reference libraries, the component type is not updated unless the reference library is added to the compTypeRefLibs environment variable in the .cdsenv file. In your .cdsenv file add
layoutXL compTypeRefLibs string (refLibName)

- Rename existing NMOS and PMOS layout cellviews to a different view name. For example, rename nmoslayout to nmoslayout.orig. When you install the QCells, they do not overwrite the existing layout cellview correctly. You can also use the lxUseCell property. For more information, see “Setting the lxUseCell Property” on page 45.

- If layout devices are being created directly from the technology file, remove or rename them.

**Inserting a QCell Device**

When you insert a QCell device into your design, you must run Connectivity – Update – Device Correspondence to update the connectivity. If you do not update the device correspondence, the QCell you inserted will have no connectivity.

1. Choose Create – Device and click on the icons to specify the type of device.

2. Move the cursor into the layout window to place the device in the design.

3. Choose Connectivity – Update – Device Correspondence. The schematic window prompts you to click on one or more symbols.

   For more information on this command, see “Updating Device Correspondence” on page 436.

4. In a schematic window, click on the symbols whose connectivity you want to update in the layout.

5. With the cursor in the schematic window, press Return.

   The layout window prompts you to select one or more corresponding objects in the layout window.

6. In the layout window, click on the object or objects to associate with the symbol.

7. With the cursor in the layout window, press Return.

\[\text{Important}\]

The correspondence is defined only when you press Return.

A message similar to the one below is displayed in the CIW saying that the connection has been made.

*Virtuoso XL: Instance ‘I1’ in cellView ‘overview amp1 layout’ is renamed ‘|R21’.*
Using Layout Optimization

The Optimize tool facilitates the hand layout of cells. For more information, see “Layout Optimize”, in the Virtuoso Layout Editor Turbo User Guide.

Using Properties

Before Virtuoso XL can create a layout from a schematic, you must create a layout device for every symbol in the schematic. The layout master cell of a device or contact can be a fixed cell, a parameterized cell (pCell), or a device or contact defined in the technology file.

PCells are often the most effective because you can assign the dimensions of the device at layout generation and vary the sizes of a contact each time you place the cell. For more information about pCells, see the Virtuoso Parameterized Cell Reference Manual.

Note: The default values for any given property must be the same in the schematic and the layout. If the default value of a property differs between the two cellviews, and the property has a value of storeDefault=nil, then when you invoke the Design – Gen From Source command, the layout instance is not updated (i.e., the CDF default of the layout is used). This can lead to a parameter mismatch between the layout and the schematic.

Property Definitions

The properties used by Virtuoso XL are described below.

lxUseCell

Add to the symbols of device cells to specify which cell (or library and cell, if you want to use a cell from different library) to use for the layout instance.

lvsIgnore, ignore, ignoreNames, nlAction

Add to the symbols in the schematic to prevent them from being placed in the layout.

lxParamsToIgnore

Blocks unwanted parameters.

lxRemoveDevice

Ignores parasitic devices.

lxViewList, lxStopList
Add to the symbols of device cells in hierarchical designs to specify the layout views to use.

**Note:** The `lxParamsToIgnore`, `lxViewList`, and `lxStopList` properties have default values that are used when properties are not found. The corresponding environment variable names are `paramsToIgnore`, `viewList`, and `stopList`. Their values are all strings containing words separated by spaces.

To use Virtuoso XL to create complex devices and to generate your own internal net prefixes, add the following properties.

**lxCombination**

Builds a complex set of devices consisting of series (`sfactor`) and parallel (`mfactor`) connections.

**Note:** The `lxCombination` property supports a maximum of two terminal devices.

**lxNetNamePrefix**

Adds a prefix to internal nets in the layout.

**lxBlockExtractCheck**

Blocks the connectivity of shapes that touch.

**lxBlockOverlapCheck**

Prevents the extractor from checking whether a nonoverlap layer of this shape or instance is touching a nonoverlap layer at the current cellview level.

**lxMfactorSplit**

Controls whether Virtuoso XL places a schematic device with the `mfactor` property as multiple devices in the layout.

**lxRounding**

Prevents folded devices from becoming off-grid.

**lxSeriesTerms**

Add the property to understand the connectivity of the devices with `sfactor` > 1 and instance terminals > 2.

**mfactor**
Preparing Pins

For connectivity assignment tracing and cross-probing to work correctly in Virtuoso XL, the pins and pin names in the layout cellview of a device must match those in the corresponding schematic symbol.

**Preparring Pins**

Defines a one-to-many parallel relationship between a device in a schematic and multiple devices in the layout.

**sfactor**

Defines a one-to-many relationship between a device in a schematic and multiple instances in a series connection in the layout.

**permuteRule**

Makes the instance pins or terminals of a device permutable.

**vxlInstSpacingDir**

Assigns automatic spacing to pins of instances.

**vxlInstSpacingRule**

Defines a spacing offset when pins are on a different net.
Extra Pins in the Symbol or Layout Views

Layout XL cannot maintain connectivity for any extra pins in the symbol view because there is no corresponding pin for the device in the layout view.

Layout XL does maintain connectivity for extra pins whose names are global nets (for example, vdd!) in the layout view. It also maintains connectivity for any extra pins in the layout whose connectivity is defined by inherited connections. The inherited connection can be defined relative to the layout instance itself or relative to the schematic hierarchy that ends with the schematic instance corresponding to that layout instance.

To determine which net to connect to such a pin, Layout XL first tries to resolve the inherited net expression on the layout pin. If there is no net expression, the software looks for a property named either sub, sub_inh, or bn on the layout master, uses the value of the property as the substrate net name, and connects the extra terminal to that net.

Note: Do not place pins where you do not want to make a connection; for example, on a poly layer that covers the gate area of a FET. For more information, see “Adding Pins to a Layout” on page 199.

External Connections

You can also define pins to be connected externally to the design.

Note: The commands in the Connectivity – Define Pins submenu let you connect pins in four different ways.

- **Must Connect** connects selected pins in a net externally at a higher level of the hierarchy.
- **Strongly Connected** connects selected pins within the device. By default, pins are connected internally (strongly).
- **Weakly Connected** connects selected pins in a limited external connection to avoid specific internal connections (typically ones with high-resistance paths).
- **Pseudo Parallel Connect** connects selected instance terminals on the same net within an instance as though they were connected externally; that is, they are defined as a connection but need never be physically connected.

For more information about must-connect pins, strongly connected pins, weakly connected pins, and pseudo-parallel connected pins, see the “Using Connectivity” section of the Virtuoso Layout Editor User Guide.

To permute pins see “permuteRule” on page 789.
If you are using parameterized cells (PCells) and want to give them the capability for abutment (overlapping pins to create a connection), see “Device Abutment” on page 71.
Setting Up the Virtuoso XL Environment

This chapter explains how to set up your Virtuoso® XL Layout Editor (Virtuoso XL) design environment to suit your preferences and work habits. You can use design variables to change the value of many aspects of your design environment either for an individual design session or permanently (until you change the value of a variable).

This chapter discusses the following topics,

- Setting Up Your Desktop on page 56
  - Customizing Your Desktop Layout on page 56
  - Using Multiple Cellviews on page 57
  - Printing to the Command Interpreter Window on page 57
- Changing Display Colors on page 58
- Using Bindkeys on page 61
  - Displaying Bindkeys on page 61
  - Loading Bindkeys on page 62
- Setting Environment Variables on page 62
- Using Setup Files or the CIW on page 64

For information on Virtuoso XL forms, see “Online Forms” on page 65.
Setting Up Your Desktop

After you start Virtuoso XL, you see “Virtuoso® XL Layout...” in the banner above the layout window (on the right) and you see a schematic window (on the left) that corresponds to the layout window. This is the default configuration.

You can move, resize, and iconify the Command Interpreter Window (CIW), schematic window, Layer Selection Window (LSW), and layout window to suit your needs.

Customizing Your Desktop Layout

You can save and reuse your customized window environment in one of the following ways.

- Using the Options – Save Defaults command from the CIW.

When you use the CIW Options – Save Defaults command to save the size and position of the schematic and layout windows as you have customized them, Virtuoso XL saves this environment information to the layout cellview as a property. When you open the layout and schematic cellviews the next time, they appear in the same configuration.
as they had when you saved them. The command sends the LSW and the CIW window size and location information to the .cdsenv file.

For information on this command, see “Saving and Recalling Default Values” in the Cadence Design Framework II User Guide.

- By adding the appropriate environment variables to your .cdsenv file; for example, as shown in the example below.

To prevent Virtuoso XL from rearranging or resizing windows from the way you have arranged them, add the following line to the .cdsenv file:

```plaintext
envSetVal("layoutXL" "autoArrange" 'boolean nil)
```

For more information, see “Environment Variables” on page 470.

**Note:** If you save your current window positions and form settings to a file other than .cdsenv (for example, .envFileName), you can restore them by adding the following command to your .cdsinit file or typing it in the CIW.

```plaintext
envLoadFile("layoutXL" ".envFileName")
```

### Using Multiple Cellviews

If you open multiple cellviews in Virtuoso XL, the second and subsequent sets of windows are not automatically configured in the same way as the first set.

You can open more than one pair of schematic-layout windows in Virtuoso XL and work on each pair independently.

- If you open a schematic and two copies of the same layout, the Check, Probe, and Update commands in the Connectivity submenu apply to both layouts.

- If you open a schematic and two different layouts, the Check, Probe, and Update commands in the Connectivity submenu apply only to the layout from which you selected the command.

### Printing to the Command Interpreter Window

To print process information to the CIW instead of to a separate Virtuoso XL Info window, add the following command to your .cdsenv file or type it in the CIW

```plaintext
envSetVal("layoutXL" "infoWindow" 'boolean nil)
```
Changing Display Colors

You can change the colors and characteristics of the entry layers that Virtuoso XL commands use for display purposes. You can change the display layers Virtuoso XL uses for the following commands:

- The *Show Incomplete Nets* command uses the *y0* through *y9* entry layers to show flight lines.
- The *Connectivity – Check – Against Source* command uses the *hilite drawing9* entry layer to identify missing devices.
- The *Connectivity – Probe* command uses the *hilite drawing* through *hilite drawing9* entry layers to probe pins, nets, and devices.
- The *Highlight* command for placement constraints uses the *annotate drawing3* entry layer to highlight in the layout the components bound by a placement constraint.
- The *Pick from Schematic* command uses the *hilite drawing1* layer to highlight schematic symbols.

Editing Entry Layers

Although you can edit these layers from the LSW, it is not recommended. To edit these layers,

1. Add the layers to the LSW as valid layers. Valid layers are those you use for creating layout shapes.
2. In the CIW, choose *Tools – Display Resource Manager*.
3. In the Display Resources Tool Box, click *Edit*.
   - The *Display Resource Editor* window appears.
4. Click *All*, so that you can see all of the available layers (not just the ones you have defined as drawing layers).

5. In the *Layers* column, choose the name of the layer to change.

6. Turn on the *Fill Style*, *Fill Color*, *Outline Color*, *Stipple*, and *Line Style* options you want to use for that layer.

7. Click *Apply*.

The next time you use a command that displays the layer you changed, the color or pattern changes in the design window.
Highlighting Incomplete Nets

To highlight incomplete connections with flight lines, Virtuoso XL cycles through the y0 – y9 layers. For more information, see “Identifying Incomplete Nets” on page 174.

Virtuoso XL displays the flight lines of each net in a different color. If there are more than 10 incomplete nets, the colors repeat unless you have specified layers for specific nets. For more information, see “Assigning Colors to Incomplete Nets” on page 176.

If you assign new colors to the flight lines,

- Use colors that do not resemble each other or resemble the colors you use for paths and components.
- Avoid solid fill so you can see what is underneath.

Highlighting Probes

Virtuoso XL uses the hilite drawing through hilite drawing9 entry layers for probes, which identify equivalent design elements on the schematic and the layout.

Virtuoso XL uses the hilite drawing9 layer to identify components in the schematic that are not in the layout (and vice versa) with the Connectivity – Check – Against Source command.
If you change the colors of the *hilite drawing* layers,

- Use thick lines so you can see device pins.
- Use bright colors to make probes and highlights easy to find.

**Tip**

Do not make the *hilite drawing* layer and the *hilite drawing2* layer solid fill because these layers are used to indicate selected components and to manipulate shapes.

**Using Bindkeys**

Bindkeys, also called function keys, are keyboard macros that assign a menu command to a key you choose from the keyboard. When your cursor is in the layout window, you can use all of the bindkeys that are loaded for the applications you are running.

**Displaying Bindkeys**

To display the list of bindkeys for Virtuoso XL, follow these steps.

1. From the CIW, choose *Options – Bindkey*.
   
   The Key or Mouse Binding form is displayed.

2. In the *Application Type Prefix* cyclic field, choose *Layout*.

3. Click *Show Bind Keys*.
   
   A text window displays the layout editor bindkeys and commands.
Loading Bindkeys

You can find a list of default bindkeys for Virtuoso XL at the following location:

```
your_install_dir/samples/local/1xBindKeys.il
```

where `your_install_dir` is the top directory in which you store Cadence® software.

To load this file of bindkey definitions every time you run Virtuoso XL, add the following line to your `.cdsinit` file.

```
load(prependInstallPath("samples/local/1xBindKeys.il"))
```

Setting Environment Variables

**Using the Layout XL Options Form**

Environment variables control the values of Virtuoso XL options. You can set some Virtuoso XL environment variables in the individual options forms for Virtuoso XL commands. You can set other Virtuoso XL environment variables using the Layout XL Options form.

You must set the options relating to initialization (*Show All Incomplete Nets* and *Auto Arrange Windows*) on the Layout XL Options form before you start your design session. The values you enter for these options in the current session take effect the next time you start Virtuoso XL. Changes made to other values take effect as soon as you click *OK* or *Apply*. To avoid having to restart the session, set these values in your `.cdsenv` or `.cdsinit` file. For more information, see “Using Setup Files or the CIW” on page 64.

Not all the Virtuoso XL environment variables are available through forms. You can set the value of environment variables not available on forms in your `.cdsenv` or `.cdsinit` file.

⚠️ **Important**

For a list of all the Virtuoso XL environment variables and their values, see “Environment Variables” on page 470.

To set environment variables using the Layout XL Options form, follow these steps.

1. From the layout window, choose *Options – Layout XL.*
The Layout XL Options form is displayed.
2. In the *Edit* section at the top of the form, click *Global Options* if you want the information you enter in this form apply to the your whole design environment.

   Click *Cellview Options* if you want the information you enter in this form apply to only one cellview. If you choose *Cellview Options*, several of the options on this form are grayed out because they cannot be localized to individual cellviews.

3. In the main body of the form, choose the layout options you want to apply to your design.

4. In the *Load/Save* section at the bottom of the form, set the options according to whether you want to load the information from a file, save the information to a file, or delete the information in the form from the current cellview, the design library, the technology library, or a file.

   If you click the *File* option, the text field is enabled, letting you type in a path and name of a file.

   - To save your design options to a technology library, or to a cell or library other than the one from which you opened the form, type the name of a file and click *Save to*.
   - To load previously saved options into this form, type the name of a file and click *Load from*.
   - To delete these values from a cell or a library, click *Delete from*.

   **Note:** You cannot delete environment values from a file.

5. Click *OK*.

   The environment variable values you choose are enforced for all subsequent design sessions until you change the value of the variables again.

**Using Setup Files or the CIW**

If you use any environment variable values consistently and do not want to set these values each time you use a command, you can set the variables to the value you normally use for a single session or you can set them permanently in setup files such as the *.cdsenv* file and the *.cdsinit* file.
To set environment variables for a single session, do one of the following.

- Include `envSetVal( )` in any other Cadence SKILL file you load.
- Type `envSetVal( )` in the CIW.

For more information on using `envSetVal( )`, see “About the Command Interpreter Window” in the *Virtuoso Layout Editor User Guide*.

To set environment variables permanently, do one of the following.

- Include the environment variables in the `.cdsenv` file in your home directory; for example,
  
  ```
  layoutXL alignApplySeparation boolean t
  ```

- Include `envSetVal( )` in your `.cdsinit` file

For example, to set the Virtuoso XL `alignApplySeparation` variable (which forces Virtuoso XL to separate aligned components by a separation factor you supply), type the following in the CIW or include it in a setup file.

```
envSetVal("layoutXL" "alignApplySeparation" 'boolean t)
```

To determine the current value of any Virtuoso XL environment variable, type the following in the CIW.

```
envGetVal("layoutXL" "alignApplySeparation" 'boolean)
```

For more information on the `.cdsenv` and `.cdsinit` files, see “Setting Layout Editor Defaults” in the *Virtuoso Layout Editor User Guide*.

**Online Forms**

**Layout XL Options**

**Edit**

- **Global Options** applies the values specified in the form to the cellview currently being edited and all other views of the current cell.

- **Cellview Options** applies the values specified in the form only to the cellview currently being edited.

**Connectivity** lets you specify whether the connectivity extractor is enabled and what markers it displays in the layout window. The settings in this section control only the current top-level cellview; they do not control lower level cellviews.
Connectivity Extractor switches on connectivity extraction. When the extractor is turned off, the Connectivity – Show Incomplete Nets command is disabled.

Extract Connectivity to Level sets the level at which hierarchical extraction stops.

Show Weak-Connect Violations toggles the display of top-level weak-connect violation markers in the current cellview. Note that the extractor shows only violations at the current top level, irrespective of the value you set for Extract Connectivity to Level.

Show Must-Connect Violations toggles the display of must-connect violation markers in the current cellview. Note that the extractor shows only violations at the current top level, irrespective of the value you set for Extract Connectivity to Level.

Important

If you change the Extract Connectivity to Level, Show Weak-Connect Violations, or Show Must-Connect Violations options, connectivity extraction is triggered right away and any changes are reflected immediately in the layout window.

Cross Selection turns on cross-selection between the layout and schematic. When you select a component in the layout, the corresponding component is selected in the schematic and vice versa.

Auto Abutment turns on automatic abutment, which abuts prepared transistors so that they can share pins.

Auto Space turns on automatic spacing, which allows components with the properties vxlInstSpacingDir and vxlInstSpacingRule to be spaced automatically according to the values specified in the properties.

Auto Permute Pins turns on automatic pin permutation during manual routing or editing. For more information, see “permuteRule” on page 789.

Constraint Assisted Move And Stretch turns on the constraint-assisted move and stretch capability.

Verification

Show Parameters to Ignore lists the parameters that are ignored by the Update – Check Against Source command.

The following parameters are always ignored.
Show Properties Used To Ignore Objects lists the properties that cause pins and instances to be ignored during the Connectivity – Check – Against Source command. Any object with one of the listed properties set to t is ignored during these operations.

The following properties are always ignored.

- ignore
- lvsIgnore
- nlAction

Add lets you add to the list a parameter or property name you have typed into the text field.

Remove lets you remove parameters or properties selected in the list box.

Inherited from Generation lists the parameters or properties that are inherited from the list defined for generation. Parameters and properties ignored for generation are always ignored for verification. To change this list, you need to edit it in the Generation group box.

Tolerance specifies the relative tolerance the Connectivity – Update – Layout Parameters command uses to compare values between the layout and the schematic.

If \(| (a/b) - 1.0 | < e \) then the parameters are considered equal, where a is source parameter value, b is the layout parameter value, and e is the Tolerance. The default of e is 1e-6.

Ref Libs lists the reference libraries for which component types can be defined in the Edit Component Types form.

Device Folding

Generate Minimal Folding creates the minimum number of folded devices. If this option is turned off, then an odd number of folded devices is created. For example, it adds one finger if the device width divided by finger maximum width yields an even number.

Retain Instance Orientation preserves the device orientation when running the Edit – Folding or Design – Gen From Source commands with only folding switched on. When the Transistor Chaining option is also turned on, the chain orientation is always with the gates in a vertical position.

pCell fingering Names specifies parameter names that define pcell gate fingering. Folding and instance fingers are incompatible and this option is used to filter out those
instances when folding. If you do not set the names correctly, it might result in abutment failures for even-fingered devices and folding.

**Update Parameter** specifies how the *Connectivity – Update – Layout Parameters* command updates folded devices.

- **No Change** folded devices will not be updated.
- **Equalize** uses the updated schematic width to produce fingers of equal width. The number of fingers does not change, only the width of each finger.
- **Distribute** uses the updated schematic width and distributes the width amongst all the fingers. The number of fingers does not change, only the width of each finger.

**Generation**

- **Show Traverse Hierarchy Views** lists the views included in the view list when descending a hierarchical design.
- **Show Layout Instance Views** lists the views included in the stop list when descending a hierarchical design.
- **Show Parameters to Ignore** lists the parameters that are ignored by the following Virtuoso XL update and check commands: *Gen From Source, Update Components and Nets, Update Layout Parameters, Update Schematic Parameters, Pick From Schematic, Clone, and Update – Check Against Source*.

The following parameters are always ignored.

- `dleIgnoredParams` `dleSchExtractPath` `dleStopList` `dleUseCell`
- `instancesLastChanged` `itemNamePrefix`
- `lxIgnoreParams` `lxIgnoreParamsForCAS` `lxMFactorNum`
- `lxParamsToIgnore` `lxParamsToIgnoreForCheck`
- `lxPlacementStatus` `lxStopList` `lxTimeStamp` `lxUseCell`
- `maskLayoutViewName` `pin#` `posi`

**Show Properties Used To Ignore Objects** lists the properties that cause pins and instances to be ignored during the generate and update commands listed above. Any object with one of the listed properties set to `t` is ignored during these operations.

The following properties are always ignored.

- `ignore` `lvsIgnore` `nlAction`

**Add** lets you add to the list a parameter or property name you have typed into the text field.

**Remove** lets you remove parameters or properties selected in the list box.

**Update Layout Instances** replaces an existing instance that uses an incorrect master with a new instance that uses the correct master during the *Connectivity – Update –*
Device Correspondence command. When switched off, Virtuoso XL forces the binding without changing either the name or the master of the instance in the layout.

For more information, see “updateReplacesMasters” on page 598.

Multiple Instances

mfactor Names lists the names of properties used in the schematic to specify the multiplication factor (mfactor) for transistors. The default is mM. The mfactor property value can be expressed as an integer or an expression.

mfactor Split Param Names lists the names of the device parameters to be calculated or checked during Design – Gen From Source, Connectivity – Check – Against Source, Connectivity – Update – Layout Parameters. Virtuoso XL checks each instance for the listed parameter names and changes or updates the value of the matching parameters such that

number of mfactored instances * mfactorSplit value = source parameter value

The default value is w.

Important

Virtuoso XL currently checks only for width parameters.

sfactor Names lists the names of properties used in the schematic to specify the number of series-connected (sfactor) devices to be generated in the layout. Virtuoso XL checks each instance for a property matching one of the names on the list and uses the value of that property to generate the appropriate number of series-connected devices. The default is s S.

sfactor Split Param Names lists the names of the device parameters to be split among generated series-connected devices during Design – Gen From Source, Connectivity – Check – Against Source, Connectivity – Update – Layout Parameters. Virtuoso XL checks each instance for the listed parameter names and changes or updates the values of the matching parameters. The default list includes names for resistor, capacitor, and inductor parameters, r R c C l L.

Generate Multiple Instances controls whether Virtuoso XL places schematic devices with the mfactor property as multiple devices in the layout.

Display

Show All Incomplete Nets enables the Show Incomplete Nets command the next time you start Virtuoso XL. To see incomplete nets immediately, use Connectivity – Show Incomplete Nets.
**Auto Arrange Windows** rearranges the positions of the design windows the next time you start Virtuoso XL.

**Draglines** controls the display of draglines when using the *Create – Pick From Schematic*, *Create – Clone*, *Edit – Move*, and *Edit – Stretch* commands.

**Show Distant Connections** displays draglines for all of the connections to the instance that is being manipulated. When switched off, draglines are shown only for the connections closest to the current instance.

**Hide Global Nets** hides the draglines for global nets and for power and ground nets specified using the `lxGroundNetNames` and `lxSupplyNetNames` environment variables.

**Use Colors** shows each dragline in a different color. The color cycle is defined in the *Show Incomplete Nets* form.

**Load/Save**

**Cellview** lets you load, save, or delete the values specified in the form to or from the cellview from which you opened the Layout XL Options form.

**Library** lets you load, save, or delete the values specified in the form to or from the library that contains the cellview from which you opened the Layout XL Options form.

**Tech Library** lets you load, save, or delete the values specified in the form to or from the technology library of the cellview from which you chose the Layout XL Options form.

**File** lets you load, save, or delete the values specified in this form to or from your `.cdsENV` file (the default) or another file specified in the text field.
Device Abutment

This chapter describes the Virtuoso® XL abutment capability. Abutment allows cells to be automatically aligned or overlapped and electrically connected.

This chapter discusses the following topics.

- Requirements for Abutment on page 72
- Setting Up Cells for Abutment on page 73
- Sample Parameterized Cells Set Up for Abutment on page 75
- Creating CMOS Pcells to Use with Abutment on page 78
- Manual Device Abutment on page 81
- Automatic Device Abutment on page 88

The Virtuoso XL abutment capability lets you create a connection between two cells overlapping each other without introducing a design rule violation or connectivity error. The two sets of shapes must include pins that are connected to the same net.
Abutment reduces both the area occupied by the circuit and the length of the interconnect wiring. You can use abutment during interactive layout generation.

**Requirements for Abutment**

Devices can be abutted only between pins of different instances. Abutment requires the following conditions to be satisfied:

- MOS transistors sharing a diffusion
- Devices can be abutted only between pins of different instances.
  - A. Same-size terminals with external connections on the same net
  - B. Same-size terminals with no other connections on the same net
  - C. Different-sized terminals with external connections on the same net
  - D. Wells sharing part of their surface and bias contacts
  - E. Resistors sharing terminals
  - F. Multiple abutment of different-sized terminals with external connections on the same net
Both instances must be pcells set up for abutment or cells with the appropriate abutment properties set. For more information, see “Setting Up Cells for Abutment” on page 73.

Both instances must have either the abutClass property set to the same value OR the same master (if the abutClass property is not present).

- The software first checks that the abutClass property set to the same value on the pin of each instance. If it is, the instances can be abutted; if it is not, the abutment is rejected.

- If the abutClass property is not set on either instance, but both instances have the same master, the instances can be abutted.

For more information, see “About Pcell Super and Submaster Cells” in the Custom Layout SKILL Functions Reference.

Both instance pins must be connected to the same net.

Both instance pins must be defined on shapes on the same layer or on layers that are defined as equivalent layers in the technology file.

Both instance pins can have the same abutment direction if the rotation of one (but not both) of the pins is either R180 or MY, or any other rotation that transforms the left-hand side to the right-hand side.

During placement, the Connectivity Extractor and Auto Abutment options in the Virtuoso XL Options form must be turned on.

When the Auto Abutment option is turned on, devices that have not been abutted but are overlapping are abutted.

Abutted devices can share diffusion, contacts, metal tabs, or any shape combined in an instance pin.

Setting Up Cells for Abutment

You can set up both regular cells and pcells for abutment. To set up a regular cell so that it can be abutted, add the following properties on the pins of the cellview.

- abutFunction
- abutOffset
- abutAccessDir
- abutClass
You can also add these properties to a cellview in order to set up abutment for all the pins in the cellview. If any pin in the cellview has properties that differ from the properties set for the cellview, the properties on the pin override the properties on the cellview.

For examples of how to set abutment properties, see the sample pcell libraries at the following locations.

- `your_install_dir/tools/dfII/samples/ROD/rodPcells/components/mos/mos.il`
- `your_install_dir/tools/dfII/samples/ROD/rodPcells/components/mos/sample_mos.il`

**Snapping Instances in the Direction Perpendicular to the Abutment**

The layout environment variable `abutPerpSnapOn` controls whether Virtuoso XL snaps an instance in the direction perpendicular to the abutment. When `abutPerpSnapOn` is set to `true`, Virtuoso XL automatically snaps the moving instance vertically such that the lower-left y coordinates of the two abutting pins are the same.

**Multiple Pins**

Virtuoso XL supports the abutment of more than one pin to a wider single pin.

Multiple pins on the edge of a cell can also be abutted. Any pair of pins with the correct properties can trigger automatic abutment, but once automatic abutment has been triggered, other pins on that edge that touch pins on the cell it is abutted to will not trigger automatic abutment again. If those pins do not connect, auto-permute tries to resolve the conflict. If the conflict cannot be resolved, a connection violation is flagged.

**Note:** Make sure that you define abutment in such a way that no Design Rule Checker (DRC) or connectivity violations are introduced by abutment between any selected pair of pins.

**Additional Pins on Ignored Instances**

The abutment engine ignores instance terminals belonging to instances with the schematic properties `lvsIgnore` or `ignore`. This means that additional pins on ignored instances are also ignored, allowing abutment to take place where it otherwise would not and enabling a tighter placement of abutted components.

For example, a schematic has two MOS devices connected in series (one’s source to the other’s drain). The schematic also places a parasitic capacitor between these MOS devices.
The capacitor has the lvsignore property attached to it. When generating the layout, Layout XL correctly ignores the capacitor and its pin, and abuts the two MOS devices.

**Steps in Automatic Abutment**

1. Overlapping pins trigger automatic abutment.
2. Automatic abutment identifies cells for abutment by master name or class.
3. Automatic abutment calls abutFunction.
4. abutFunction adjusts the parameters of pcells and calculates reference edge offsets of conventional cells.
5. Pcells calculate a new configuration based on any parameters changed by abutFunction.
6. If the cells can be abutted, the abutment connection condition is 1 or 2 and the cells are abutted to the reference edges and the pins are aligned in the direction to the direction of abutment.
   
   If the cells cannot be abutted, the abutment connection condition is 3 and the cells remain in their original configuration.
   
   For information on abutment connection conditions, see abutFunction.

**Sample Parameterized Cells Set Up for Abutment**

**Example 1**

The example below shows what each of the different parts does.

```plaintext
; Function: abutFunction(iA iB pA pB pASide connection event
; @optional (group nil))
;
; Inputs: group - the abutment group that these two cells belong to.
;
; iA = Instance Id of cell that will move during abutment
;
; iB = Instance Id of cell being abutted to
;
; pA = Overlapping Pin Fig of iA
```
Virtuoso XL Layout Editor User Guide
Device Abutment

; pB = Overlapping Pin Fig of iB
; pASide = Abutting pin access direction

; connection = an integer value of 1 or 2 that indicates:
; 1. pins are connected to the same net and do not connect to any other pin.
; 2. pins are connected to the same net and the net connects to other pins

; event = integer that represents abutment event:
; 1. compute abutment offset
; 2. pcell parameter adjustment for abutment
; 3. pcell parameter adjustment for unabutment

; group = abutment group pointer available to events 2 and 3

; Outputs: depends
; Side effects:
procedure( abutFunction(iA iB pA pB pASide connection event @optional (group nil))
  prog((result)
    case(event
      (1 ; Compute offset
        result = getAbutmentOffset(iA iB pA pB pASide connection)
      )
      (2 ; Adjust pcell parameters
        result = setAbutmentParams(iA iB pA pB pASide connection group)
      )
      (3 ; Adjust pcell parameters back to default resetAbutmentParams(group iA iB)
        result = t
      )
      (t ; Anything else return a nil
        result = nil
      )
    )
    return(result)
  )
)

Example 2

The example below shows how to add abutment properties to pins in an inverter pcell.
; *** the following 4 pins are on metal1 and are on ***
; *** the power and ground rails. This will allow ***
; *** abutment to other standard cells ***

obj = leftVddPin~>dbId
dbReplaceProp(obj "abutAccessDir" "list" list("left"))
dbReplaceProp(obj "abutClass" "string" "stdcell")
dbReplaceProp(obj "abutFunction" "string" "stdCellFunc")

; *** the following property is a technique for passing ***
; *** information to the user defined function. In this ***
; *** case the minimum cell height is dependent on many ***
; *** factors including design rules and is originally ***
; *** calculated in the pcell code. This avoids ***
; *** duplicating calculations in the user-defined ***
; *** function ***

dbReplaceProp(obj "minCellHeight" "float" minH)

obj = rightVddPin~>dbId
dbReplaceProp(obj "abutAccessDir" "list" list("right"))
dbReplaceProp(obj "abutClass" "string" "stdcell")
dbReplaceProp(obj "abutFunction" "string" "stdCellFunc")
dbReplaceProp(obj "minCellHeight" "float" minH)

obj = leftGndPin~>dbId
dbReplaceProp(obj "abutAccessDir" "list" list("left"))
dbReplaceProp(obj "abutClass" "string" "stdcell")
dbReplaceProp(obj "abutFunction" "string" "stdCellFunc")
dbReplaceProp(obj "minCellHeight" "float" minH)

obj = rightGndPin~>dbId
dbReplaceProp(obj "abutAccessDir" "list" list("right"))
dbReplaceProp(obj "abutClass" "string" "stdcell")
dbReplaceProp(obj "abutFunction" "string" "stdCellFunc")
dbReplaceProp(obj "minCellHeight" "float" minH)

; *** The following 4 pins are the diffusion pins ***
; *** of the individual MOS devices in the inverter ***
; *** this will allow abutment of other individual ***
; *** MOS devices to the inverter ***

obj = POutPin~>dbId
dbReplaceProp(obj "abutAccessDir" "list" list("right"))
dbReplaceProp(obj "abutClass" "string" "ptran")
dbReplaceProp(obj "abutFunction" "string" "mosAbutFunc")
dbReplaceProp(obj "contactParam" "string" "POutCnts")
dbReplaceProp(obj "w" "float" pw)
; *** the automatic spacing properties are used if abutment fails ***
dbReplaceProp(obj "vxlInstSpacingDir" "list" list("right"))
dbReplaceProp(obj "vxlInstSpacingRule" "float" .35 )

obj = NOutPin~>dbId
dbReplaceProp(obj "abutAccessDir" "list" list("right"))
dbReplaceProp(obj "abutClass" "string" "ntran")
dbReplaceProp(obj "abutFunction" "string" "mosAbutFunc")
dbReplaceProp(obj "contactParam" "string" "NOutCnts")
dbReplaceProp(obj "w" "float" nw)
dbReplaceProp(obj "vxlInstSpacingDir" "list" list("right"))
dbReplaceProp(obj "vxlInstSpacingRule" "float" .35 )
Creating CMOS Pcells to Use with Abutment

To create a pcell that has a built-in abutment capability,

1. Create a pcell specifically for abutment.

   Editing an existing pcell for abutment is much more difficult than creating a new one for the purpose. For information on creating pcells, see the Virtuoso Parameterized Cell Reference.

2. Specify the conditional inclusion or exclusion of contacts. For information, see abutCondInclusion.
3. Identify gate pins. For information, see `abutGateNet`.

4. Specify the access direction. For information, see `abutAccessDir`.

5. Specify stretchable material. For information, see `abutStretchMat`.
   - The first element in the `abutStretchMat` list is `drainStretch`.
- The second element in the `abutStretchMat` list is `abutMinExt`.

- The third element in the `abutStretchMat` list is `abutRule1Ext`.

- The fourth element in the `abutStretchMat` list is `abutRule2Ext`.
The fifth element in the `abutStretchMat` list is `abutContactExt`.

**MOS Abutment without a User-Defined `abutFunction`**

Another way to process abutment and unabutment without creating an `abutFunction` callback is to use the `abutMosStretchMat` property. For more information on this property, see `abutMosStretchMat`.

This section describes the automatic abutment properties that you must add to pcells to use automatic abutment on standard MOS pcells without having to define your own `abutFunction`.

**Automatic Abutment Properties**

As well as creating specific parameters in the pcells and named diffusion shape pins, you also have to add the following properties on the shapes the pins are created on. The automatic abutment properties specify if the pcells can abut.

- `abutClass` - Pins of the same `abutClass` are able to abut.
- `abutAccessDir` = Defines the directions in which the pins are able to abut.
- `abutGateNet` = Defines the name of the net the gate shapes are on.
- `abutCondInclusion` = Defines the name of the boolean property that adds or removes contact shapes over the diffusion pin.

**Manual Device Abutment**

Use the following commands to abut devices manually.
Edit – Transistor Chaining lets you abut devices interactively while you are editing a layout. For more information, see “Chaining Transistors Interactively” on page 82.

Create – Pick from Schematic lets you abut devices while using the Pick From Schematic command to generate a layout. For more information, see “Chaining Transistors during Pick From Schematic” on page 85.

Edit – Move lets you control abutment while moving devices. For more information, see “Moving Chained Transistors” on page 86.

Chaining Transistors Interactively

Transistor chaining is the process by which a list of MOS transistors or the fingers of folded transistors are abutted in a specific order. Chaining helps reduce layout area and capacitance.

Prerequisites

The transistors to be chained

- Must be set up for abutment. For more information, see “Setting Up Cells for Abutment” on page 73.
- Must have the lxComponentType property set on the cell or library to which they belong. You set this property using the Design – Component Types. For more information, see “Component Types” on page 188.

All parameter values are retained after chaining. The chaining engine always creates pairs of P and N chains with the P chain on top (because the power rail is typically at the top of a cell) and the N chain at the bottom (because the ground rail is typically at the bottom).

To chain transistors,

1. From the layout window menu bar, choose Edit – Transistor Chaining.

The Transistor Chaining form is displayed.
2. In the layout window, select the transistors to chain or click *Add Cell ListBox* to see the list of transistors available for chaining.

The list box appears in the Transistor Chaining form.

![Transistor Chaining](image)

**Note:** You can also select transistors in the layout window before you choose the *Transistor Chaining* command. The transistors you selected are highlighted in the list when you open the Transistor Chaining form.

3. Select the transistors to chain from the list box and click *Apply*.

The selected transistors are highlighted and you are prompted to select a destination point in the layout window.

4. Move the cursor into the layout window.

The selected transistors appear as the image of a transistor chain that follows the cursor.

Folded transistors are chained with the number of fingers specified in the Set Transistor Folding form. For more information, see “Using Transistor Folding” on page 193.

Chains that comprise only NMOS or only PMOS transistors are aligned by the bottom edge. If you use the *Gen From Source* command to create a cluster comprising both
an NMOS and a PMOS transistor, the PMOS is placed on the top aligned to the lower edge, and the NMOS is placed below it aligned to the upper edge.

![PMOS Chain](image1)

![NMOS Chain](image2)

To rotate the instances, click the right mouse button. To mirror the instances, do Shift-right-click.

**Note:** You must have the Virtuoso XL bindkeys loaded in order to use the right mouse button to rotate and mirror instances.

To change the vertical orientation of the chains, click the *Upside Down* button located at the bottom of the Move form.

5. Click on the layout window where you want to place the chained transistors.

The transistors are placed where you click.

![Before transistor chaining](image3)

![After transistor chaining](image4)

**Removing a Transistor from a Chain**

To remove a transistor from a chain,

1. In the layout window, select the transistor you want to remove from a chain.
2. Choose *Edit – Move*.
3. Click on the layout window where you want to place the selected transistor.
The transistor is placed where you click.

Before removing a transistor from chain

After removing a transistor from chain

### Chaining Transistors during Pick From Schematic

Virtuoso XL can automatically abut devices during *Pick From Schematic* command if you turn on the *Group As In Schematic* and *Transistor Chaining* options in the *Pick from Schematic* form.

The *Place Individually* option does not let you automatically abut devices. To abut devices during *Pick from Schematic* with the *Place Individually* option set,

1. From the layout window menu bar, choose *Options – Virtuoso XL*.
   
   The *Layout XL Options* form is displayed.

2. Turn on the *Connectivity Extractor* and *Auto Abutment* options and click *OK*.

3. Set the two environment variables to control how automatic abutment works in your design.

4. From the layout window menu bar, choose *Create – Pick from Schematic*.
   
   The *Pick from Schematic* form is displayed. For more information on using this command, see “Moving Components from the Schematic into the Layout (Pick from Schematic)” on page 135.

5. In the schematic, select the instances you want to place.

6. Click on the layout where you want to place the instances.
   
   The software places the selected instances where you click in the order that you selected them.

7. Where flight lines indicate abutment is possible, place devices so that the pins overlap.
   
   The devices are abutted automatically.
Moving Chained Transistors

Move chained devices by setting one of the following options in the Move form.

- **Chain Mode**
  
  For more information, see “Moving Chained Transistors in Chain Mode” on page 86.

- **Constraint Assisted Control**
  
  Virtuoso XL can set and maintain constraints added to instances. If some of the instances being moved belong to a satisfied constraint, the abutment system must maintain the constraint. If abutment would cause a constraint to be broken, the abutment system will not abut the two cells.

  For more information, see “Moving Chained Transistors using Constraint Assisted Control” on page 88 and “constraintAssistedMode” on page 484.

Moving Chained Transistors in Chain Mode

To move chained devices while maintaining the chains,

1. From the layout window menu bar, choose *Edit – Move*.
   
   The *Move* form is displayed.
2. Select a chained device and change the Chain Mode to All.

3. Click on the reference point for the move.

   You see that all of the devices that are chained together are highlighted.

4. Click in the layout where you want to place the devices.

5. Change the Chain Mode option to Selected and click on the selected device.

   The selected device is moved and the device is no longer a part of the chain.

   **Note:** To move the device back into the chain select Edit – Undo.

6. Select a device in the center of the chain and change the Chain Mode to Selected Plus Right.

7. Click on the reference point for the move.

   You see that all of the devices to the right of the selected device remain chained as you move the devices.
8. Click in the layout where you want to place the devices.

**Note:** The *Selected Plus Left* option selects all chained devices to the left of the selected device.

### Moving Chained Transistors using Constraint Assisted Control

To move chained transistors with constraint assisted control,

1. From the layout window, choose *Edit – Move*.
   
The *Move* form is displayed.

2. Under *Constraint Assisted Control*, turn on *together* or *individually*.
   
   You can also activate constraint enforcement with the *Constraint Assisted Mode* option in the Virtuoso XL Options form.
   
   If you do not want constraints to have any influence on interactive moves, turn on *ignore*.
   
   For more information on this form, see “Setting Environment Variables” on page 62.

3. Move an instance so that one of its pins touches the pin of another instance.
   
   - If the two pins are on the same net, the instances are automatically abutted.
   
   - If the two pins are on different nets but automatic pin permutation of one or both of the pins will bring two pins in the same net into contact, the pins are automatically permuted and the two instances abutted.

   After abutment, the cell is modified in accordance with the pcell abutment parameters you specified.

   The system moves only one device to abut two cells. Of two instances that can be abutted, it chooses to move

   - **a.** A selected cell over a cell that is not selected.
   
   - **b.** A cell that is not already abutted over a cell that is already abutted

### Automatic Device Abutment

Use the following commands to automatically abut devices.

- Layout generation automatically abuts when the *Transistor Chaining* option is turned on. For more information, see “Chaining Transistors Automatically during Layout Generation” on page 89.
The *Update Components and Nets* command automatically abuts devices when the *Transistor Chaining* option is turned on. For more information, see “Chaining Transistors Automatically during Update Components and Nets” on page 89.

### Chaining Transistors Automatically during Layout Generation

To chain transistors automatically during layout generation,

1. From the layout window menu bar, choose *Design – Gen From Source*.
   
   For more information, see “Generating a Layout with Components Not Placed (Gen From Source)” on page 123.

2. In the *Layout Generation* section, turn on the *Transistor Chaining* option.

3. When you click *OK*, the software chains transistors automatically during layout generation.

   **Tip**
   
   You can also set the environment variable `initDoStacking` to turn on transistor chaining from the *Layout Generation Options* form.

### Chaining Transistors Automatically during Update Components and Nets

To chain transistors automatically when updating components and nets,

1. From the layout window menu bar, choose *Connectivity – Update – Components and Nets*.
   
   If the schematic has any devices that are missing from the layout, the Layout Generation Options form is displayed.

2. In the *Layout Generation* section, turn on the *Transistor Chaining* option.

3. When you click *OK*, the software chains transistors automatically during layout generation.
Important

Update Components and Nets chains only new devices after running the Layout Generation Options with Transistor Chaining turned on. New chains are separate and do not attach to old chains. Devices that had been chained prior to running Update Components and Nets are not moved from their existing positions.

For more information on this command, see “Updating Components and Nets (Engineering Change Order Mode)” on page 426.

Abutting Parameterized Cells and Quick Cells

For information on how to code parameterized cells to automatically abut to Quick Cells, see Appendix A, “Pcell and QCell Abutment,” in the Virtuoso Layout Editor Turbo User Guide.
Generating Your Layout

This chapter shows you how to use the Virtuoso® XL Layout Editor (Virtuoso XL) to generate a layout and make an initial placement of components in it.

This chapter covers the following topics:

- **Naming Conventions** on page 91
- **Starting Virtuoso XL from the Schematic** on page 94
- **Importing a Netlist for a Connectivity Reference** on page 96
- **Starting Virtuoso XL from the Layout View** on page 98
- **Defining the Design Boundary** on page 105
- **Creating Template Files** on page 112
- **Generating a Layout with Components Not Placed (Gen From Source)** on page 123
- **Placing Components in a Layout in the Same Relative Position as in the Schematic (Place As In Schematic)** on page 134
- **Moving Components from the Schematic into the Layout (Pick from Schematic)** on page 135
- **Cloning** on page 144

For information on Virtuoso XL forms, see “Online Forms” on page 158.

**Naming Conventions**

When generating and manipulating components in the layout view, Layout XL uses the naming conventions outlined in this section.
Important

Layout XL relies on these conventions when establishing correspondence between the schematic and layout views. If you manually change a system-generated name, or if you rename an instance such that its new name conflicts with a system-generated name, the correspondence between schematic and layout might be lost.

General Convention

The OR bar (|) is used to delimit hierarchy and to prefix the names of all layout instances generated by Layout XL.

<table>
<thead>
<tr>
<th>Schematic</th>
<th>Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>P0</td>
</tr>
<tr>
<td>NAN0/P0</td>
<td>NAND0/P0</td>
</tr>
</tbody>
</table>

Mfactored Devices and Folded Devices

Devices specified using the mfactor property in the schematic and those that are folded in the layout are named instName.integer in the layout.

For example, if you have a schematic instance named P0 with

mfactor = 3

the layout instances generated are named as follows.

<table>
<thead>
<tr>
<th>Schematic</th>
<th>Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>P0.1</td>
</tr>
<tr>
<td></td>
<td>P0.2</td>
</tr>
<tr>
<td></td>
<td>P0.3</td>
</tr>
</tbody>
</table>

Series-Connected Devices

Devices specified using the sfactor property in the schematic are named instName.sinteger in the layout.

For example, if you have a schematic instance named R0 with

sfactor = 3
the layout instances generated are named as follows.

<table>
<thead>
<tr>
<th>Schematic</th>
<th>Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R0.s1</td>
</tr>
<tr>
<td></td>
<td>R0.s2</td>
</tr>
<tr>
<td></td>
<td>R0.s3</td>
</tr>
</tbody>
</table>

**lxCombination Devices**

Devices defined using the `lxCombination` property in the schematic are named `instName.ms{integer}` in the layout.

For example, if you have a schematic instance named `R0` with

```
  lxCombination = 2*10K + 1*100K
```

the layout instances generated are named as follows.

<table>
<thead>
<tr>
<th>Schematic</th>
<th>Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R0.ms1</td>
</tr>
<tr>
<td></td>
<td>R0.ms2</td>
</tr>
<tr>
<td></td>
<td>R0.ms3</td>
</tr>
</tbody>
</table>
Starting Virtuoso XL from the Schematic

To start Virtuoso XL from a schematic window, follow these steps.

1. From the Command Interpreter Window (CIW), choose File – Open.

   The Open File form is displayed. For more information on this form, see “Open File Form” in the Cadence Design Framework II User Guide.

2. In the Library Name cyclic field, choose a library name.

3. In the Cell Name field, type a cell name.

4. In the View Name cyclic field, choose a schematic view.

5. Click OK.

   The schematic view you specified appears.

6. From the schematic window, choose Tools – Design Synthesis – Layout XL.

   **Note:** Layout XL does not appear in the Tools menu unless the Virtuoso XL software is installed. If you do not see Layout XL in the Tools – Design Synthesis menu, check with your system administrator.
The **Startup Option** form is displayed.

7. Choose whether you want to create a new cellview or open an existing one.

   **Note:** To set the connectivity source without opening the layout graphically, use the `lxSetConnRef` SKILL function. For more information, see “*lxSetConnRef*” in the *Custom Layout SKILL Functions Reference*.

8. Click **OK**.

   If you choose *Create New*, the Create New File form is displayed. The default view is a new layout.

   - In the **Library Name** cyclic field, choose a library name.
   - In the **Cell Name** field, type a cell name.
   - In the **View Name** field, type a view name.
   - Click **OK**.

   If you choose *Open Existing*, the Open File form is displayed.

   - In the **Library Name** cyclic field, choose a library name.
   - In the **Cell Name** field, type a cell name.
   - In the **View Name** field, type a view name.
   - Click **OK**.

The default configuration of Virtuoso XL appears: the schematic window, the layout window, the CIW, and the LSW (Layer Selection Window).
Importing a Netlist for a Connectivity Reference

Important
Support for the netlist-driven flow described in this section has been discontinued in this release. If you encounter a problem using the File – Import – XL Netlist command, use File – Import – CDL instead. Import CDL generates a schematic from your netlist. You can then use that schematic as the connectivity source for your design. For more information on importing CDL, see “Translating CDL Files,” in the Design Data Translator’s Reference.

To import a netlist to use as the connectivity source for a Virtuoso XL design, follow these steps.

Important
The netlist syntax must comply with the guidelines defined in “Using Spice and CDL for Netlist-Driven Layout Generation” on page 678. If you do not follow these guidelines, then the layout might not reflect the full intent of the netlist.

The netlist may contain user defined properties and generic parameters that can be passed to the layout instance generated from Virtuoso XL. Any parameter of type String, Boolean, int, and float is supported.

1. From the CIW, choose File – Import – XL Netlist.
The Import XL Netlist form is displayed.

2. In the Netlist File field, type the name of the input file (the netlist you want to import).

To see the names of your directories and files in the file browser, put the cursor in the Netlist File field and click Browse.

Note: Netlists in circuit description language (CDL) and SPICE (3f4) formats can be imported.

For information on importing CDL files, see “Translating CDL Files” in the Design Data Translator’s Reference.

3. Choose whether you want to copy the imported netlist file to the destination library, link the imported netlist file to the destination library, or move the imported netlist file to the destination library.

4. In the Library field, type the name of the library in which you want to store the netlist.
5. In the *Cell* field, type the name of the cell in which you want to store the netlist.

6. In the *View* field, type the name of the view you want to give the netlist.

   To see the libraries, cells, and views in the Library Browser, put the cursor in the *Library* or *Cell* or *View* fields and click *Browse*.

7. Click *OK*.

   Virtuoso XL imports the netlist to the library, cell, and view name you specified. You can now use this netlist as a connectivity reference for a Virtuoso XL design.

## Starting Virtuoso XL from the Layout View

To start the Virtuoso XL from a layout window, follow these steps.

1. From the CIW, choose *File – Open*.

   The Open File form is displayed. For more information on this form, see “Open File Form” in the *Cadence Design Framework II User Guide*.

![Open File Form](image)
2. In the Library Name cyclic field, choose a library name.

3. In the Cell Name field, type a cell name.

4. In the View Name cyclic field, choose a layout view.

5. Click OK.

   The layout view you specified appears.

6. From the layout window, choose Tools – Layout XL.

   The default configuration of Virtuoso XL appears: the schematic window, the layout window, the CIW, and the LSW.

   **Note:** Layout XL does not appear in the Tools menu unless the Virtuoso XL software is installed. If you do not see Layout XL in the Tools menu, check with your system administrator.

   If the layout from which you chose the Layout XL command has no connectivity reference (schematic or netlist) associated with it, the Define Connectivity Reference form is displayed.

   ![Define Connectivity Reference](image)

7. Choose one of the following as the connectivity source type.

   - If there is no connectivity reference for the layout, choose None.

     Virtuoso XL is installed, but Virtuoso XL commands requiring information from a connectivity source are disabled. You can define a connectivity reference at any time by choosing Connectivity – Update – Source.
When you choose *None* (no connectivity reference), the connectivity model requires that the nets of pins and instance terminals cannot be changed, but the nets of all other shapes can be changed if they connect to a pin or instance terminal.

- If the connectivity reference is a schematic, choose *Schematic* and do one of the following:
  - Type the name of the library, cell, and view into *Library*, *Cell*, and *View* fields.
  - Click *Browse* to see what cells are available in the Library Browser, and click on a cell name in the Library Browser.
  - Click *Sel by Cursor* and then click on an open schematic.

Click *Update layout cellviews hierarchically* to update the library setting for any for any lower-level cellviews in a hierarchical cellview.

You can click *Browse* to open the Library Browser to find a netlist and click on the netlist name to enter it in the Define Connectivity Reference form.

8. Click *OK*.

Virtuoso XL opens a schematic window showing the schematic you specified.

**Connectivity Reference as a Netlist**

⚠️ *Important*

The netlist syntax must comply with the guidelines defined in “Using Spice and CDL for Netlist-Driven Layout Generation” on page 678. If you do not follow these guidelines, then the layout might not reflect the full intent of the netlist.

The netlist can contain user defined properties and generic parameters that can be passed to the layout instance generated from Virtuoso XL. Any parameter of type `String`, `Boolean`, `int`, and `float` is supported.

To specify a netlist as a connectivity reference,

1. Choose *Netlist* in the Define Connectivity Reference form.
The form changes to let you enter the name of the *Map File* and the *Top Cell* of the netlist.

![Define Connectivity Reference](image)

2. Type the name of a Virtuoso XL map file to import or click *Browse* to open the Library Browser to find a netlist and click on the netlist name to enter it in the Define Connectivity Reference form.

The Virtuoso XL map file lists model, cell (subcircuit), and instance properties and maps them to the names they have in the layout.

You can also use the Virtuoso XL map file to map terminal names in the netlist (such as resistor terminal names *Plus* and *Minus*) to terminal names in the layout (such as resistor terminal names *A* and *B*), and parameter names in the netlist (such as the resistance value *r*) to parameter names in the layout (such as the resistance value *R*).

The Connectivity – Update – Source command loads the information in the Virtuoso XL map file into the cellview you specify on the form, overwriting any previously entered information. The Virtuoso XL map file structure, with an example, is shown in “Virtuoso XL Map File Structure” on page 102.
Virtuoso XL Map File Structure

The Virtuoso XL map file consists of a model, subcircuit, or cell mapping section, `lxNetlistCellMap`, which is mandatory when using a netlist source, and an instance mapping section, `lxNetlistInstMap`;

```
lxNetlistCellMap(
  (device_class
    (device_type
      (property)
      (property)
      ...
    )
    (property)
    (property)
    ...
  )
)
```

```
lxNetlistInstMap(
  (hier_inst
    (property)
    (property)
    ...
  )
  (hier_inst
    (property)
    (property)
    ...
  )
)
```

- `device_class` is the model card name (for example, M, R or C) specified in quotes. It applies only to the netlist-driven flow, however, it still needs to be specified as a blank string in schematic map file.

- `device_type` identifies the model, subcircuit, or cell in the source. It contains a keyword, a separator, and a name. The keyword can be `model` or `subckt` or `cell`; the separator is the equals character (=); and the name is a quoted string specifying the model, subcircuit, or library and cell name.

- `hier_inst` is a hierarchical instance in the source specified as a quoted string with forward slashes as the hierarchical delimiter. For example, “/X1/X22/M3” would represent an instance M3 in subcircuit X22 which is in turn instantiated in subcircuit X1.

- `property` can be one or more of the supported properties listed below. You can define these properties for either `device_class` or `device_type` under `lxNetlistCellMap`, and for `hier_inst` under `lxNetlistInstMap`.

- `useCell “libName cellName”` specifies the layout cell to be used to implement the device.
paramNameMap list defines the mapping between device parameters in the source schematic and the parameters of the layout cell, in the form (“device_param” “cell_param”).

pinNameMap “list” defines the mapping between the device terminals in the source schematic and the terminals of the layout cell, in the form (“device_pin” “cell_pin”).

paramSet “list” sets the parameters listed to the given values and types, in the form (“name” “type” “value”).

stopList “list” lists the view names that stop hierarchy traversal for the cell or the instance.

ignorePins “list” specifies names of pins to be ignored during layout generation and check against source.

ignoreParams “list” specifies the names of parameters to be ignored during layout generation and check against source.

ignoreNames “list” defines the names of instances to be ignored during layout generation and check against source. For information on how to set this property, see Examples.

ignoreCells “list” defines the names of cells to be ignored during layout generation and check against source. For information on how to set this property, see Examples.

Note: Properties defined at a higher level apply to all devices defined at that level or lower, except where they are locally overwritten. The order in which devices and properties are defined in the Virtuoso XL map file is irrelevant.

Examples

The following example illustrates a map file suitable for use in a netlist-driven flow.

```
lxNetlistCellMap
  {
    ("M"
      ("MODEL" "=" "P"
        (useCell "refLib pmos"
          (paramNameMap ("WP" "w") ("wp" "w") ("LP" "l") ("lp" "l"))
        )
      )
    )
    ("MODEL" "=" "N"
      (useCell "refLib nmos"
        (paramNameMap ("WN" "w") ("wn" "w") ("LN" "l") ("ln" "l"))
      )
    )
    (paramNameMap ("W" "w") ("L" "l") ("M" "m") ("C" "c"))
    (pinNameMap ("s" "S") ("g" "G") ("d" "D") ("b" "B"))
  )
```
To ignore a specific cell during layout generation or check against source, use the following.

```plaintext
lxNetlistCellMap
{
   ("myLib myCell"
    (ignoreCells "myLib myCell")
   )
}
```

To ignore an instance of a particular cell during layout generation or check against source, use one of the following.

```plaintext
lxNetlistCellMap
{
   ("myLib myCell"
    (ignoreNames "myInstance")
   )
}
```

Or

```plaintext
lxNetlistInstMap
{
   ("myInstance"
    (ignoreCells "myLib myCell")
   )
}
```

To ignore an instance during layout generation or check against source, use the following.

```plaintext
lxNetlistInstMap
{
   ("myInstance"
    (ignoreNames "myInstance")
   )
}
```

### Using a Map File in a Schematic Driven Flow

In a schematic-driven flow, the Virtuoso XL map file lets you use a different set of layout cells to map to the schematic masters (similar to the `lxUseCell` property). This might be necessary
if you do not have permission to write to the schematic and therefore cannot add the necessary \texttt{lxUseCell} properties.

For Virtuoso XL to search for any instance in a schematic flow, the \texttt{device_class} must be empty which means that you cannot specify M, R, C, and so on. Additionally, you should use the library and cell name instead of the model name. Below is an example of a map file to use in a schematic driven flow.

\textbf{Note}: Properties defined at a higher level apply to all devices defined at that level or lower, except where they are locally overwritten. The order in which devices and properties are defined in the Virtuoso XL map file is irrelevant.

\section*{Example}

The following example illustrates a map file suitable for use in a schematic-driven flow. For information on how to use the different ignore properties, see \texttt{Examples}.

\begin{verbatim}
lxNetlistCellMap
 {
  (""
   ("model" == "gpdk nmos")
    (useCell "myGPDK nmos")
    (stopList "myLayout")
   )
   ("model" == "gpdk pmos")
    (useCell "myGPDK pmos")
    (stopList "myLayout")
  )
  ( paramNameMap ("w1" "w") ("l1" "l") )
 )
 (""
   ("model" == "gpdk nmos_hv")
    (useCell "myGPDK nmos_hv")
    (stopList "myLayout")
   )
   ("model" == "gpdk pmos_hv")
    (useCell "myGPDK pmos_hv")
    (stopList "myLayout")
  )
  ( paramNameMap ("w2" "w") ("l2" "l") )
 )
}
\end{verbatim}

\section*{Defining the Design Boundary}

When you initialize a layout, Virtuoso XL provides a design boundary.

To reset the default size for the design boundary, choose the layout window \textit{Design – Gen From Source} command and set the options in the \textit{Boundary} section. For information on
how to do this, see “Generating a Layout with Components Not Placed (Gen From Source)” on page 123.

To stretch, move, or delete the design boundary, use the layout editor Edit commands. For more information on these commands, see “Editing Objects” in the Virtuoso Layout Editor User Guide.

The design boundary must be a valid layer in the LSW and must be layer prBoundary or layer cellBoundary, or else it must have the purpose boundary (by).

To place all objects and pins inside the boundary, use the layout window’s Edit – Place from Schematic command. For more information, see “Placing Components in a Layout in the Same Relative Position as in the Schematic (Place As In Schematic)” on page 134.

To draw a new design boundary, follow these steps.

1. If you are using the Layout Generation Options form to generate a new layout, turn off Boundary.

2. In the LSW, click on the prBoundary layer.

   The boundary layer appears as the current layer at the top of the LSW. You can also use the layer cellBoundary from the Layout Generation Options form Boundary section Layer cyclic field.

3. From the layout window, choose Create – Rectangle or Create – Polygon.

   For information on these commands, see “Using the Placer” on page 213.

4. Draw the design boundary in the layout window where you want it placed.

   For a rectangular boundary, click to place one corner of the new boundary, drag the mouse to place the opposite corner, and release the mouse button.

   For a polygonal boundary, use the cursor to click on the first point, each corner point, and end point (identical to the first point) of the polygon.

   The new boundary appears in the layout window.

   Note: if there are multiple shapes on the prBoundary or cellBoundary layers, a message box appears (only once) asking you to select which one is the top-level boundary.

**Automatic Area Estimation**

When you specify a rectangular boundary in the Layout Generation Options form, you can specify that Virtuoso XL estimates the area required to accommodate the design.
You specify the percentage of the area within the cell boundary that you want the components to fill, the aspect ratio of the design boundary, and an Area Calculation function, which controls the shapes that are used to calculate the contents of the design and, in turn, the area required to accommodate them.

There are two predefined area calculation functions, which are described below.

- **PRBoundary Based** uses the place and route boundaries of the individual lower-level cells to calculate the area. This provides more accurate results and more compact layouts, especially when working with rectilinear blocks.

- **BBox Based** uses the bounding boxes of cellview masters to calculate the area. The derived bounding box is typically larger than the place and route boundary in standard cells because of metal overhangs and presence of text display objects and labels. This can result in pessimistic area estimates with values much higher than would be expected for a compact layout.

For more information on these functions, see “Layout Generation Options” on page 163.

Alternatively, you can define your own area estimation function. For more information on how to do this, see User-Defined Area Estimation Functions.

**User-Defined Area Estimation Functions**

**Syntax**

```
function_name(
    dbCellViewID
)
=> x_area
```

**Description**

Returns an estimate of the area required to accommodate the design represented by the specified cellview ID. If the value returned is negative, its absolute value is taken.

The value returned is divided by the Utilization factor and the boundary shape determined using the Aspect Ratio specified in the Layout Generation Options form.
Important

When the Transistor Chaining and Transistor Folding options are switched on, the Utilization is applied only after chaining and folding is complete in order to calculate the size of the PR boundary accurately.

Arguments

cellviewID

The database ID of the cellview whose area you want to estimate.

Value Returned

x_area

The estimated area required to accommodate the specified cellview expressed as either a float or an integer.

Example

You can define your own area estimation function as follows.

```plaintext
procedure(myFunc(cvId)
    let ( (area inst)
        area = 0.0
        foreach( inst cvId~>instances
            x1 = caar(inst~>bbox)
            x2 = caadr(inst~>bbox)
            y1 = cadar(inst~>bbox)
            y2 = cadadr(inst~>bbox)
            printf("%L\n" inst~>bbox)
            printf("%f %f %f %f\n" x1 y1 x2 y2)
            area = area + abs(x1-x2)*abs(y1-y2)
            )
        area
    )
)
```

This function

- Traverses all the instances defined in the cellview cvId
- Calculates the sum of the areas of the bounding boxes of each instance
- Returns the area value

You specify this area estimation function in the Boundary Area Estimation group box in the Layout Generation Options form.
1. From the Area Calculation cyclic field, choose User Defined.

2. Type the name of the function into the text field.

   For example, to use the function above, you would type \texttt{myFunc} into the text field.

   \textbf{Note:} Type in only the name of the function. The cellview ID is the currently open cellview.

**Working with Template Files**

Options from the Pin Placement, Partitioning, Placement Planning, Layout Generation, and Update Components and Nets commands are saved to the cellview. These options can be copied from the cellview into template files. The template files can then be used to update or redo sections of your design.

Use the Design – Save To Template command to create template files. You can chose to save:

- Boundary information
- I/O pin layers, multiplicity, sizes, and types.
- Pin locations
- Pin constraints
- Partitioning information
- Row information

Templates can contain all of the available sections or just a subset of the available sections. For example, you might use separate templates for your I/O pin constraints and row information.

**Table 6-1 Template Sections and Contents**

<table>
<thead>
<tr>
<th>Template Section</th>
<th>Scope of the Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boundary</td>
<td>The boundary shape and layer</td>
</tr>
<tr>
<td></td>
<td>There are a variety of ways to define the shape of a boundary in the Layout Generation Options form.</td>
</tr>
<tr>
<td>I/O Pins Location</td>
<td>The location info from the Pin Placement form.</td>
</tr>
<tr>
<td>I/O Pins Constraints</td>
<td>The edge, etc. info from the Pin Placement form.</td>
</tr>
</tbody>
</table>
Table 6-1 Template Sections and Contents, continued

<table>
<thead>
<tr>
<th>Template Section</th>
<th>Scope of the Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows</td>
<td>The row information from the Placement Planning form.</td>
</tr>
<tr>
<td>Partitions</td>
<td>The partition information from the Partitioning form.</td>
</tr>
</tbody>
</table>

**Saving Form Contents**

Use the *Design – Save to Template* command to save the form information to an ASCII file. The template file is used to modify or delete the design data or create new data in the design. To be able to reuse this information, you must create template files. Specify the information that you want to save in the Template File form and that information is written from the cellview to the template file.

**Note:** If you exit Design Framework II and do not save the design, then the form settings are not saved to the cellview. Use the *Design – Save* command to save the form settings.

All the information on the forms is updated to the cellview when you click the *OK* or *Apply* buttons. You can then save the information to a template file using the *Design – Save To Template* command.

The template file name is stored in the cellview. The template file name is then used by default when opening the form.

**Loading Template Files**

You can load template files while working with several different command forms. The type of template information you can save or load depends on what is appropriate to the current task.

Table 6-2 Commands and Template Sections

<table>
<thead>
<tr>
<th>This Command...</th>
<th>Lets You Save and Load This Template Information...</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Design – Gen From Source</em></td>
<td>Pin and Boundary</td>
</tr>
<tr>
<td><em>Connectivity – Update – Components and Nets</em></td>
<td>Pin and Boundary</td>
</tr>
<tr>
<td><em>Place – Pin Placement</em></td>
<td>Constraints</td>
</tr>
<tr>
<td><em>Place – Pin Placement</em></td>
<td>Pin Location and Constraints</td>
</tr>
<tr>
<td><em>Place – Partitioning</em></td>
<td>Partitions</td>
</tr>
</tbody>
</table>
Modifying Template Files

To modify templates, edit the file with a text editor. Follow the syntax rules described in “Template File Syntax” on page 113.

Modifying a template file using VI has no effect on any cellview until you load it into that cellview.

If you modify the form then that information is read into the cellview.

Loading Template Files

To load existing information about pins, the design boundary, partitions, or rows from a template file into the corresponding form, follow these steps.

1. From the layout window, select either Place – Pin Placement, Place – Partitioning, Place – Placement Planning, Design – Gen From Source, or Connectivity – Update – Components and Nets.

2. Type in the template name in the Template field.

   **Note:** In the Pin Placement form click the Template file button to choose the template file from the Load Template File form.

3. If you would like a different template select Browse.
The Open File form is displayed.

![Open File Form]

**Note:** You can also load a template as part of
- Layout Generation. For information, see “Generating a Layout with Components Not Placed (Gen From Source)” on page 123
- Update Components and Nets. For more information, see “Updating Components and Nets (Engineering Change Order Mode)” on page 426.

**Creating Template Files**

There are two ways to create a template.

- From the *Design – Save To Template* form
- Using a text editor
To save current information about pins and design boundary from the cellview to a template file, follow these steps.

1. From the layout window, choose Design – Save To Template.

   The Template File form is displayed.

2. In the Template file name field type a template file name, \texttt{ex.row.lxt}.

3. Select the sections of the template file you want to save (\textit{I/O Pin Locations}, \textit{I/O Pin Constraints}, \textit{Boundary}, \textit{Rows}, and \textit{Partitions}). If you want to save all selections, select \textit{All}.

4. Click \textit{OK}. The data you selected is saved in the template file.

\textbf{Template File Syntax}

Template files generally follow the syntax rules for technology files, but they are not technology file fragments.

Templates contain these sections:

- Boundaries Section
- I/O Pins Section
Partition Section

A template can contain some or all of these sections.

General Syntax Rules

The following rules apply throughout the template file.

Comments

Begin comment lines with a semicolon (;).

; This is a comment

Layer Names

Specify layer names as either a layer name string or as a parenthesized list containing the layer name and purpose.

metal1
metal2
(metal1 drawing)

Numeric Values

Specify numbers as floating-point or integer values or as expressions based on design rules in the technology file.

(minWidth metal1)
(minSpacing poly)

Layer names can be omitted if the layer is defined and unambiguous. When you refer to the minWidth and minSpacing rules without specifying a layer, they must be enclosed in parentheses even though you are not expressing a list.

(minWidth)
(minSpacing)

Points

Specify points as a pair of coordinates, either numbers or expressions. The syntax is

(x y) for example,

((minWidth metal1) (minWidth metal1))
Offset Points

Specify offset coordinates in a sequence of points as a relative shift from another point. The keyword \texttt{D} identifies offset coordinates. The syntax is

\begin{verbatim}
(D x y)
\end{verbatim}

For example:

\begin{verbatim}
(D 3 5)
(D (minWidth met1) 7)
\end{verbatim}

Sequences of Points

Specify a sequence of points as a list of points or offsets.

\begin{verbatim}
(0 0) (D 10 0) (D 0 20) (D -10 0)
(D 10 0) (D 0 10) (D -10 0)
\end{verbatim}

If the first item is an offset, the entire list must be composed of offsets.

Polygons

Specify polygons using the \texttt{polygon} keyword followed by a sequence of points. The syntax is

\begin{verbatim}
(polygon sequence_of_points)
\end{verbatim}

The last point of the sequence is always connected by an edge to the first one, even if such an edge does not appear in the sequence. Clockwise and counterclockwise orders are both acceptable. Polygons characterized by a sequence of offsets (no fixed points) are called floating polygons. If a floating polygon is encountered in a context where a non-floating polygon is expected, the first point is implicitly set to \begin{verbatim}(0 0)\end{verbatim}.  

The following examples all describe the same polygon

\begin{verbatim}
(polygon (0 0) (D 10 0) (D 0 20) (D -10 0))
(polygon (0 0) (D 10 0) (10 20) (D -10 0))
(polygon (0 20) (0 0) (10 0) (10 20))
\end{verbatim}

Rectangles

Specify rectangles using the \texttt{rectangle} keyword and the options in the table below. The syntax is
(rectangle options)

### Table 6-3 Rectangle Specification Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>aspect_ratio X</td>
<td>Ratio between width and height. X must be a number.</td>
</tr>
<tr>
<td>height H</td>
<td>Absolute height of the rectangle. H can be a number or a design rule.</td>
</tr>
<tr>
<td>width W</td>
<td>Absolute width of the rectangle. W can be a number or a design rule.</td>
</tr>
<tr>
<td>lowerLeft point</td>
<td>Position of the lower-left corner.</td>
</tr>
<tr>
<td>upperRight point</td>
<td>Position of the upper-right corner.</td>
</tr>
<tr>
<td>left X</td>
<td>Position of the left side.</td>
</tr>
<tr>
<td>right X</td>
<td>Position of the right side.</td>
</tr>
<tr>
<td>bottom Y</td>
<td>Position of the bottom edge.</td>
</tr>
<tr>
<td>top Y</td>
<td>Position of the top edge.</td>
</tr>
<tr>
<td>center point</td>
<td>Position of the center of the rectangle.</td>
</tr>
</tbody>
</table>

No option can appear more than once.

Rectangles you define without using any of the center, upperRight, or lowerLeft options are floating rectangles. The shape of floating rectangles is defined, but not their absolute position. If a floating rectangle is encountered in a context where a nonfloating one is expected, its lower-left corner is implicitly set to (0 0).

Below are some examples.

(rectangle aspect_ratio 1.1 height 300 lowerLeft (0 0))
(rectangle width 30 height 20 upperRight (375 66))
(rectangle width 100 aspect_ratio 1) floating rectangle

### Shapes

A shape is a polygon or rectangle on a particular layer.

(shape (layer layer_name) (rectangle options))
(shape (layer layer_name) (polygon sequence_of_points))

A shape that is a floating rectangle or a floating polygon is called a floating shape.
Layers can also be defined externally, within the scope of the option using a shape for its own
definition. A layer defined as part of the shape definition overwrites all external layer
definitions.

**Header Section**

The header section is used by the parser. Do not edit or remove the template header
comments.

```plaintext
(lxTemplatefileHeader
 ; (version 1.1)
 ; (creator "Template File Writer")
 ; (design "VCPLIB" "DFF" "layout" )
 ; (timeGenerated "May 20 11:13:41 2003")
```

**Boundaries Section**

The boundaries section defines the shape of the boundary. The syntax is

```plaintext
boundary_section(
   [(boundary options)]
)
```

**Boundaries Options**

The options are

- **shape**
  
The shape outlining the boundary. By default, the shape is a
  rectangle with the aspect ratio defined by `initAspectRatio`
in the `.cdsenv` file. By default, or if the shape is floating, the
  lower left corner is placed at `(0 0)`.

- **layer layer_name**
  
The boundary layer name. The default is `prBoundary`.

- **utilization N**
The area utilization factor.

This factor is used to compute the boundary size for rectangular boundaries that do not fully define the size in the shape options. It is applied only after any chaining and folding is complete.

The value is ignored for polygonal boundaries and for rectangular boundaries whose shape options fully specify the dimensions. The default is the value of initUtilization in the .cdsenv file.

Below is an example boundary definition:

```plaintext
boundary_section(
    (boundary
        (shape (layer prBoundary) (rectangle aspect_ratio 1.1))
        (utilization 0.25)
    )
)
```

Below is a second boundary example:

```plaintext
boundary_section(
    (boundary
        (layer ("prBoundary" "boundary"))
        (utilization 0.250000)
        (shape (rectangle aspect_ratio 1.000000 lowerLeft (0.0 0.0)))
    )
)
```

**I/O Pins Section**

The I/O pins section defines the shape and layer of the I/O pins for the new layout. The syntax is

```plaintext
IO_section(
    [global_pin_options]...
    [(pin {pin_name | ( pin_name1 pin_name2 ... )} pin_options)]...
    [changes to global pin options]
    [more pins]...
)
```

Every pin entry has one or more pin names, corresponding to net names for which pins are defined. Pins have one or more options. Global options appear outside of a pin entry and are overridden by corresponding pin-specific options. Changes to global options apply to subsequent pins (unless the pins each set a value for the corresponding option).

The global options in effect at the end of the I/O pins section apply to all pins not defined in the template.
Pin Options

The pin options and global pin options are identical, except that you cannot use expressions that evaluate to pin-specific values (such as pin width values based on `minWidth`) in the global options. The following options are available.

```
type pin_type

The pin type. Valid values are geometric or symbolic. The default is geometric.
```

```
multiplicity "integer"

Specifies the number of pins for the given nets. This setting is required. If the value is greater than 1, all the pins are identical.
```

```
net "name"

Specifies the net name.
```

```
term name

Specifies the terminal name for inherited connection nets.
```

```
layer ("layerName" "purposeName")

Specifies the layer-purpose pair for these pins. This option is required but only with geometric pins.
```

```
symbolic "pinName"

Symbolic pin name. This option can be used only with symbolic pin types.
```

```
shape (shape width float height float)

The pin shape, a floating rectangle. The default shape is a floating rectangle with size equal to the minimum width for the pin layer. Size can be expressed using device rules, including the implicit ones, because the layer specification is always provided with pins. With symbolic pins, it is an error to specify anything but the width.
```

```
position pos_spec
```
The pin position. By default, the pin is placed in the same relative position as in the schematic. Valid values for `pos_spec` are `(left), (right), (top), (bottom), (edgenumber num), (order num).

You can specify the position as `left`, `right`, `top`, or `bottom` for a rectangular region, or as an edge number for any shape of region. The edgenumber `num` is the lower index of the edge’s two end points relative to the point ordering when the polygon was defined (in the case of rectangles, edge 0 is the left, edge 1 the top, edge 2 the right, and edge 3 the bottom). The `order` specification indicates the ordering along the appropriate edge (0 being left or bottom).

fixed (x) (y)

The pin is fixed to a certain location. The selected pins are fixed at the placed location for good. Neither the `Edit-Move` command nor the placer can move the pin from its current location. Valid values for `fixed` are any floating point number.

usermovable (x) (y)

The pin can be moved even after setting it to a location. After placing and protecting a pin you may move that pin to another location and the placer would consider it fixed at the new location. Valid values for `usermovable` are any floating point number.

Pin Definition Examples

Below are some pin definition syntax examples.

Pin Shape and Layer Syntax

```lisp
(pin "net1"
    (net "net1")
    (layer metall)
    (shape (rectangle (minWidth)))
)

(pin "Vdd"
    (net "Vdd")
    (layer metall)
    (shape (rectangle width 3.0 height 3.0))
)

(pin "Gnd"
    (net "Gnd")
    (layer metal2)
)
Fixed Pin Syntax

Where pin A is a fixed pin.

(pin "A"
  (net "A")
  (layer ("metal1" "drawing"))
  (shape (rectangle width 0.350000 height 0.350000))
  (fixed (x 0.175000) (y 10.775000))
  (position (left) (order -1))
)

Usermovable Pin Syntax

Where pin B is a usermovable pin.

(pin "B"
  (net "B")
  (layer ("metal1" "drawing"))
  (shape (rectangle width 0.350000 height 0.350000))
  (usermovable (x 21.325000) (y 10.725000))
  (position (right) (order -1))
)

Edgenumber Pin Syntax

(pin "A"
  (net "A")
  (layer ("metal1" "drawing"))
  (shape (rectangle width 0.350000 height 0.350000))
  (position (edgenumber 1) (order -1))
)

I/O Section Examples

In this example, MET1 is the default layer for pins A, B, and C. MET2 is the default layer for D and E. All other pins, whether or not they are named in the template, will have MET2 (the most recent choice) as the default layer when you open the Layout Generation Options form.

(IO_section
  (layer MET1)
  (pin A ...)
  (pin B ...)
  (pin C ...)
  (layer MET2)
  (pin D ...)
)
(pin E ...)
)

In the next example, pins A, B, and C are on layer MET1. Their size is the minimum width of that layer. Pins Vdd and Gnd are on layer MET2. Their size is 10x10.

All the remaining pins in the cell (one for every net with an I/O terminal) are on POLY. Their size is the minimum width of POLY.

(IO_section
  (layer MET1) (minWidth)
  (pin (A B C ) )
  (layer MET2)
  (pin "Vdd" width 10 height 10 )
  (pin "Gnd" width 10 height 10 )
  (layer POLY)
)

Partitioning Section Examples

partitioning_section( sectionContents )

Where sectionContents is designed to consist of the data.

The data section describes 0 or more partitions. Each partition looks as follows,

(partition partitionName
  (attributeName attributeName)
  ...
)

An attribute can be one of,

(shape rectangle ((XL YB) (W H)))
(shape polygon ((X1 Y1) (X2 Y2) (X3 Y3) (X4 Y4) ...))
(cell cellName (cellAttribute cellAttributeValue) ...)

Note: The shape attribute is optional.

Sample Template

boundary_section(
  (boundary
    (layer ("cellBoundary" "boundary"))
    (utilization 0.300000)
    (shape (rectangle width 100.000000 lowerLeft (0.0 0.0)))
  )
)

IO_section(
  (type "geometric")
)
Generating a Layout with Components Not Placed (Gen From Source)

You invoke the layout generation function using the Design – Gen From Source command. Use this command to generate layout representations of schematic design components.

⚠️ Important

Gen From Source deletes any existing components in the layout view and generates everything from scratch.

Layout Generation Using SKILL

You can also use the public SKILL API to run Gen From Source using either of the following methods.

- The SKILL function `lxGenFromSource`, which runs the functions with reference to a template file specifying the individual settings.

  For more information, see `lxGenFromSource` in the Custom Layout SKILL Functions Reference.
A combination of the following SKILL functions, which let you specify what is generated directly in SKILL instead of referencing an external template file.

- lxGenerateFinish
- lxGenerateGetAvailableBoundaryLPPs
- lxGenerateGetAvailablePinLPPs
- lxGenerateGetPinNets
- lxGenerateSetBoundaryOptions
- lxGenerateSetAreaEstimationOptions
- lxGenerateSetGenerateOptions
- lxGenerateSetNetPinSpecs
- lxGenerateStart

For more information, see the sections on the above functions in the Custom Layout SKILL Functions Reference.

Generating a Layout from Source

To place design elements from the schematic in an empty layout window or to clear the layout window design area so you can start over, follow these steps.

1. From the layout window, choose Design – Gen From Source.

The Layout Generation Options form is displayed. This form displays default values each time it opens, it does not carry over values from the previous entry. To use values from a previous entry, you must save the values to a template and use the Load option to load the template into the form.
Layout generation does not maintain the value of the \texttt{lxMaxWidth} parameter unless all device widths and all \texttt{lxDeviceWidth} values are specified consistently; for example, all in meters (10 \textmu m is 1-e-5) or all in user units (10 \textmu m is 10).
2. In the Layout Generation section at the top of the form, do the following:

   a. If you want to generate pins, instances, or a boundary as specified in the schematic or in a template file, turn on the options labeled *I/O Pins*, *Instances*, and *Boundary*.

   *Gen From Source* automatically snaps the pins it generates to the placement grid. Iterated pins placed in the schematic are placed in order (i.e. Q1, Q2, Q3) in the layout.

   **Note:** If you want a label specifying the library, cell and view of your design just inside the center top edge of the boundary, set the environment variable `createBoundaryLabel`.

   b. If you want to *create* the abutment into chains of MOS transistors or fingers of folded MOS transistors that were changed since you created them with the *Gen From Source* command, turn on the *Transistor Chaining* option.

   The chains are not moved from their existing positions; missing devices are added in their proper position in the chain and not under the PR boundary.

   For more information, see “Chaining Transistors Interactively” on page 82 and “MOS Transistor Chaining and Folding Parameters” on page 188.

   c. If there are MOS transistors in the design that you want to *divide into fingers*, turn on the *Transistor Folding* option. This option also recreates folded transistors as they were after *Gen From Source*. For more information, see “MOS Transistor Chaining and Folding Parameters” on page 188.

   d. If you want to preserve existing many-to-many mapping of devices between the connectivity source and the layout, turn on *Preserve Mappings*. For more information, see “Using Many-to-Many or Many-to-One Mapping” on page 441.

   **Note:** The *Preserve Mappings* functionality does not report missing devices or shapes within a mapped group.

3. In the *I/O Pins* section, where the names and layers of existing I/O pins are shown (based on information in the technology file), specify any pin values you want to add or change.

   **Important**

   Power and ground pins defined at a lower level of the design hierarchy are not listed in the *I/O Pins* section but are nevertheless generated in the layout view. The system issues an information message to tell you what has happened.
Virtuoso XL uses the same naming convention used by Composer. This allows different names to be assigned to terminals vs. nets (wires). In the case where terminal and net (wire) names are different in the schematic, Virtuoso XL creates pins with the same terminal name as in the schematic, even through the actual net name associated with the pin might be different.

Patchcords (aliasing nets together) are supported and understood by Virtuoso XL so that the correct connectivity is propagated to nets that are connected via patchcords.

a. For geometric I/O pins, choose the routing layer for pins from the Layer/Master cyclic field. If no routing layers (lxExtractLayers) are defined in the technology file, a message box appears telling you so. For more information, see “Virtuoso XL Rules (lxRules)” on page 34.

b. For symbolic I/O pins, choose the name of a symbolic pin from the Pin Name list of valid symbolic pins defined in the technology file. If no symbolic pins are defined in the technology file, the symbolic option does not appear on the Pin Type cyclic field.

c. If you do not want a pin to appear on the layout, turn off its Create button.

d. To add a new pin click Add A Pin and the Add A New Pin form is displayed. Type in a pin name and select OK. If the pin to be added has the same attributes as any of the existing pins in the pin list (pin name, pin layer, and/or purpose), then the added pin is highlighted. You can change the multiplicity of the highlighted pin in order to populate the number of the added pin.

e. To generate name labels for each design element, turn on Pin Label Shape – Label. (You must first turn on the Display Pin Name option on the form associated with the layout editor Create – Pin command.)

   Note: Virtuoso XL does not generate labels for symbolic pins.

f. To set the style of the pin labels, click Pin Label Options
The Set Pin Label Text Style form is displayed.

g. Choose the options you want from the Set Pin Label Text Style form and click OK.

**Note:** The corresponding environment variables are listed below.

- `labelHeight`
- `labelFontStyle`
- `labelOverbar`
- `labelDrafting`
- `labelJustify`
- `pinTextLayer`
- `pinTextPurpose`
- `pinTextSameLayer`
- `pinTextSamePurpose`
- `orientation`
4. In the Boundary section of the Layout Generation Options form, choose from the Shape cyclic field whether to create a rectangular or polygonal design boundary.

- To create a rectangular boundary,
  
  a. Choose Rectangle from the Shape cyclic field.

  ![Boundary Form](image)

  b. From the Layer cyclic field, choose the layer you want to use for the boundary.

  c. Use the Left and Bottom fields to specify the origin of the boundary.

  d. In the Boundary Area Estimation group box, specify the Utilization and Aspect Ratio for the boundary.

}\textbf{Important}\n
When the Transistor Chaining and Transistor Folding options are switched on, the utilization is applied only after chaining and folding is complete in order to calculate the size of the PR boundary accurately.

- From the Area Calculation cyclic field, choose the method used to derive the contents of the design and, in turn, the area required to accommodate them.

  - \textit{PRBoundary Based} uses the place and route boundaries of the individual lower-level cells to calculate the area.

  - \textit{BBox Based} uses the bounding boxes of cellview masters to calculate the area.
User Defined lets you specify your own boundary area estimation function in the text field provided.

For more information on these controls, see Layout Generation Options on page 163.

To create a polygonal boundary,

a. Choose Polygon from the Shape cyclic field.

b. Type the X and Y coordinates of each corner of the polygon in the Points field in the following format. For example, for a rectangle, type the following.

\[(0\ 0)\ (0\ 10)\ (10\ 8)\ (8\ 0)\ (0\ 0)\]

When you move the cursor in the layout window, the X and Y coordinates of the location of the cursor are displayed in the Status Line at the top of the layout window.

5. In the Template File section, to load a template file (an ASCII file containing the information entered in this form in a previous session and saved to a file), follow these steps.

a. Select the Load Template File for Layout Generation option.

b. Click Browse.

You can set an environmental variable, templateFileName, to always load the name of a template file which you specify as the value of the variable.
The Open File form is displayed.

c. Click on the directories in the left list box to descend into your file hierarchy until you reach the name of the template file to use.

d. Click on the name of the template file in the right list box to enter it in the Name field.

The left and right arrows at the right side of the form let you go up and down in the file hierarchy.

e. Click OK.

The file name is displayed in the text field at the bottom of the Layout Generation Options form.
f. Select *Load* to load the template file.

**Caution**

*The pin list is refreshed with the pin information from the template file. If there is no pin information saved in a template file from a previous session, then the pin list is empty. To update the pin list invoke the Connectivity – Update – Components and Nets command.*

6. To save the template file information stored in the cellview to an ASCII file select *Design – Save To Template.*

The Template File form is displayed.

7. In the Template File form turn on *Boundary* and *I/O Pins.*

8. In the *Template file name* text field specify a user-defined template file name.

9. Click *OK.*

**Note:** When you use the *Gen From Source* command, constraints entered for layout devices are preserved. When you use the Virtuoso layout editor commands *Design – Discard Edits* and *Design – Open,* however, constraints entered for layout devices are lost unless you first save all changes made to the design.
Caution

Before you click OK in the Layout Generation Options form, remember that clicking OK in this form deletes everything in the layout window.

10. Click OK in the Layout Generation Options form.

   If you already have design elements in the layout window, a message box warns you that this command deletes all layout instances and wiring in the design.

11. Click Yes on the message box.

   Virtuoso XL places the device equivalents of the schematic symbols and the pin equivalents of the schematic terminals below the new design boundary.

   Instances and pins are placed in the same relative position as in the schematic. Iterated pins are sorted and placed together in the layout.

   Virtuoso XL uses a compact placement for the design elements that corresponds to the schematic placement.

   If there are no layout instances for components in the schematic, the missing components are reported in a Virtuoso XL Info text window.

   Note: When Gen From Source is not able to find a valid view as specified in the stopList and the layout was not placed as a result, the following error message appears.

   \w *WARNING* Could not find master cellView 'test,nmos_2v,layout compacted symbolic'.
   \w *WARNING* Could not find master cellView 'test,nmos_2v,layout compacted symbolic'.
   \w *WARNING* There were errors during Gen From Source.
   \w *WARNING* Run Check Against Source for details.

   Note: If you do not have layout pins for global nets in the schematic and you want to create these pins in the layout, set the initGlobalNetPins environment variable to t in your .cdsenv file before you start Virtuoso XL. For more information, see “Setting Up Your Environment” in the Virtuoso Layout Editor User Guide.

   Note: If you are using inherited connections in your design to assign more than one value to a global net, remember that the netSet properties on schematic instances, which specify the new value of a global signal, are not copied over with the instance from the schematic to the layout when you use the Gen From Source, Pick from Schematic, or Update Layout Parameters command.

   Note: If you are using inherited connections, the netExpression pin properties which have not been set are copied to the layout terminals. If multiple netExpression properties exist
for the same terminal name, then you are prompted to select at most one property to be preserved on the terminal.

For more information on inherited connections, see “Overview of Inherited Connections” in the Inherited Connections Flow Guide.

Mismatched Pins

If a symbol in a schematic has more pins than the corresponding device on the layout, Virtuoso XL generates a warning about mismatched pins.

If you want Virtuoso XL to ignore a pin on an instance in the schematic during layout and not generate a warning about mismatched pins for it, follow these steps.

1. Descend into the symbol of the instance and add a Boolean property, named whatever you choose (for example, unusedPin) and set to t, on the pin.

2. Add that property name (for example, unusedPin) as a value in the propsUsedToIgnoreObjs environmental variable in your .cdsenv file

   layoutXL propsUsedToIgnoreObjs string "unusedPin"

   Or type it in the CIW

   envSetVal("layoutXL" "propsUsedToIgnoreObjs" 'string "unusedPin")

   The software does not issue warnings about mismatched or missing pins during layout generation or when you run the Check – Against Source or Update – Components and Nets commands.

Using CDF Callbacks with Layout Generation

Layout generation applies all callbacks before any instance is created. The entire list of parameters found on the connectivity source instance (including the adjusted folding, sfactor, or lxCombination parameters) are used to call the CDF; the resulting list is passed to the layout instance.

Placing Components in a Layout in the Same Relative Position as in the Schematic (Place As In Schematic)

To place all components (or all components you select) inside the design boundary of the layout window in the same relative position as they are in the schematic, follow these steps.

Note: Iterated instances are sorted and placed together in the layout.
From the layout window, choose Edit – Place As In Schematic.

Virtuoso XL automatically moves all the components into the design boundary according to the relative locations of the symbols in the schematic.

*Place As In Schematic* moves all components or, if a selected set exists, only the components in that selected set.

**Note:** *Place As In Schematic* does not attempt to fit all the components inside the place and route boundary. If the schematic is hierarchical or is sparsely drawn, then some or all of the components are placed outside the boundary.

*Place As In Schematic* will arrange *mFactor* devices in a row when the *mFactor* objects are not abutted together. Iterated instances are placed in a column when not abutted together.

If some objects in the design, such as pins, are much smaller in size than other objects, such as components, the small objects might not appear in the layout, even though they have been moved. You need to zoom in to see these smaller objects.

If there is no design boundary, Virtuoso XL places the devices within a square the size of the default boundary (25% utilization).

I/O pins do not move. To move the I/O pins so that they align to the edges of the design boundary use the Placer – Pin Placement command.

**Note:** When placing components within the design boundary, the *Place As In Schematic* command might cause the components to overlap, resulting in shorts. When you create a short, Virtuoso XL displays a marker over the short.

Virtuoso XL does not compact the layout, so the placement of components might not meet Design Rule Checker (DRC) or other design requirements.

Constraints entered using the Virtuoso® Constraint Manager are not observed by the *Place As In Schematic* command. For more information, see the *Virtuoso Constraint Manager User Guide*.

---

**Moving Components from the Schematic into the Layout (Pick from Schematic)**

To place devices and/or pins directly from the schematic into the layout use the Create – *Pick from Schematic* command. If your connectivity reference is not a schematic, you cannot use this command.
Placing a Group of Schematic Elements Together

Use the Group As In Schematic option to place the instance/pins in the layout with the same relative position as in the schematic.

1. From the layout window, choose Create – Pick from Schematic.

   The first time you open a new design and use this command, the Define Connectivity Reference form is displayed.

   If the Define Connectivity Reference form is displayed, confirm the library, cell, and view name of the schematic from which you want to pick elements to place in the layout and click OK.

2. In the Pick from Schematic form, turn on Group As In Schematic.

   In this mode, you cannot change the parameters of the components. For example, you cannot change the entry layer after you have started the command.

   Note: You can set pseudo-parallel nets when in Group As In Schematic mode.

3. Select a group of components (or all the components) in the schematic.

   Note: Iterated pins placed in the schematic are placed in order (i.e. Q1, Q2, Q3) in the layout.

4. If some components in the schematic are already present in the layout, you cannot select them; only those components not yet placed can be selected.
The **Pick from Schematic** form is displayed.

5. Turn on the options you want.

- **Draglines** displays rubberbanding lines that connect pins of the object you are moving to pins of the nearest objects. For more information, see “Changing the Appearance of Draglines” on page 181.

- **Constraint Assisted** allows objects to be moved only in ways that satisfy the constraints entered using the Constraint Manager, once the object has satisfied the constraint.

- **Transistor Chaining** allows MOS transistors (or fingers of folded transistors) to be automatically abutted to form a chain.

- **Transistor Folding** allows MOS transistors to be automatically divided into fingers for optimum area usage.

- **NMOS Fold Threshold** lets you type in the maximum size of the folded NMOS transistor. This value defaults to the $lxMaxWidth$ value set on the connectivity source component.

Changing this value overwrites the $lxMaxWidth$ value you entered in the Edit Component Type form.
PMOS Fold Threshold lets you type in the maximum size of the folded PMOS transistor. This value defaults to the lxMaxWidth value set on the connectivity source component.

Changing this value overwrites the lxMaxWidth value you entered in the Edit Component Type form.

6. To see a list of the components in the schematic that are not yet placed in the layout, click Unplaced.

The Pick from Schematic Instance/Pin List appears.

7. In the list box, click on the names of the devices you want to place and click Apply or OK.

8. Move the cursor into the layout window and click.

The group of components selected are placed in the layout in the same relative position as in the schematic.
Only components not yet present in the layout can be placed.

**Placing Individual Components**

To place individual components from the schematic in the layout, follow these steps.

1. In the Pick from Schematic form, turn on *Place Individually*.

   The *Pick from Schematic* form changes to let you select individual instances.
Note: The Place Individually option does not allow you to chain devices. Transistor Chaining is available only with the Group As In Schematic option.

2. In the schematic, click on the component or group of components you want to place in the layout.

   To select multiple components, click on the first object to select it and Shift-click on the other objects or click and drag over the objects. You can deselect objects using Control-click.

   Selected objects in the schematic remain highlighted until they are deselected or placed in the layout.

   You cannot select objects in the schematic if objects with the same name are already present in the layout.

3. To automatically fold MOS devices into divided fingers for optimum are usage turn on the Transistor Folding option.

   NMOS Fold Threshold lets you type in the maximum size of the folded NMOS transistor. This value defaults to the $l_{x\text{MaxWidth}}$ value set on the connectivity source component.

   PMOS Fold Threshold lets you type in the maximum size of the folded PMOS transistor. This value defaults to the $l_{x\text{MaxWidth}}$ value set on the connectivity source component.

   Important
   When the Transistor Folding option is turned on the device properties do not appear at the bottom of the Pick from Schematic form. If you need to update any of the device properties turn off Transistor Folding.

4. Select Unplaced to view the Pick from Schematic Instance/Pin List. Select the pin or instance and click Apply. The pin or instance is selected in the schematic window.

5. Move the cursor into the layout window and the Pick from Schematic form updates with the Library, Cell, View, and InstName.

   The Pick from Schematic form updates to show the parameters associated with the component currently being placed.

6. The component to be placed first is highlighted in the schematic in the color assigned to the hilite drawing layer. Selected instances that have not yet been placed are highlighted in the color assigned to the hilite drawing layer.

   The outline of the component follows the cursor in the layout. Draglines showing connectivity to already placed components follow the component outline.
7. Turn on the options you want.

8. Click in the layout window to place the highlighted instance or pin.

**Generating Pins**

When you select a pin to place in your design the pin options are updated on the Pick From Schematic form.

1. If the components you selected are pins, the Pick from Schematic form changes to include pin options (only if you have turned on the *Place Individually* option) you can choose.

   For more information, see “Pin Options” on page 119.

   ![Pin Options](image)

2. Click in the layout where you want to place the pin.

   The pin is created where the image was when you clicked and has the connectivity indicated in the schematic.

3. If you selected more than one pin, without moving the cursor back to the schematic, click to place the other pins in the layout.

   The pins are placed in the layout at the location where you click. The pin currently being placed in the layout is always highlighted in the schematic in the color assigned to the *hilite drawing2* layer. Selected instances that have not yet been placed are highlighted in the color assigned to the *hilite drawing layer*.

   You can use the *Next* and *Previous* buttons at the top of the Pick from Schematic form to move ahead or back in the order of the list of components you picked.
Viewing Unplaced Instances/Pins

1. To see a list of all components in the schematic for which there is no corresponding layout instance or pin, click *Unplaced*.

   Wires, labels, text, instances with *ignore* properties, and instances that have already been placed in the layout are not considered available components and are not shown in this list.

2. Double click on the name of each device in the list and move the cursor to the layout to place it.

   The Pick from Schematic form changes to show available information about that component.

   Constraints entered using the Virtuoso Constraint Manager are observed while you are using the *Pick from Schematic* command.
3. To exit this command, click *Cancel* on the Pick from Schematic form.

**Note:** If you are using inherited connections in your design to assign more than one value to a global net, remember that the netSet properties on schematic instances, which specify the new value of a global signal, are not copied over with the instance from the schematic to the layout when you use the *Gen From Source, Pick from Schematic,* or *Update Layout Parameters* command.

For more information on inherited connections, see “Overview of Inherited Connections” in the *Inherited Connections Flow Guide*.

**Viewing in Place**

1. Select *Edit – Pick From Schematic.*

2. Move your cursor into the schematic window and select *Design – Hierarchy – Descend Read.*

3. In the schematic select a device to descend into.

4. Select *Hide* in the Descend form.

5. In the Descend form change the *View Name* to *schematic* and click *OK.*

   The window banner updates in the schematic window.

6. In the Pick From Schematic form select *Unplaced.*

7. Select an instance or pin in the Pick From Unplaced Instance/Pin form.

8. Click *Apply.*

9. Place the instance/pin in the layout window.

**Using CDF Callbacks with Pick From Schematic**

*Pick From Schematic* applies all callbacks before any instance is created. The entire list of parameters found on the connectivity source instance (including the adjusted folding, *sfactor,* or *lxCombination* parameter) are used to call the CDF; the resulting list is passed to the layout instance.
Cloning

Cloning is the ability to replicate a section of the layout that is associated with a section of the schematic in such a way that the new piece of layout material can be placed at more than one location, with each part preserving the hierarchical structure of the design.

Cloning can help you lay out

- Datapath bit slices, datapath and memory control blocks with identical logic cones, programmable logic structures, and other arrayed and iterated structures in device and gate-level custom digital design.
- Groups of analog/mixed signal components with identical constraints (for example, current mirrors, cascode stages, differential pairs, and other isomorphic structures).
- Instances of the same symbol master – or instances with the same logic but with differing parameters and hence a different symbol master – that have a flat layout representation

Cloning differs from copying in that the cloned structure maintains its schematic correspondence and inherits connectivity information.

The section of the schematic or layout that is used as a template for the clone is called the *clone source*. The section of the schematic that is to be implemented by replicating the source is the *clone target*. The result of replicating the clone source to implement the clone target is the clone.

You can use multiple connectivity sources and multiple layout cellviews to create clones using devices, pins, and (if selected from the layout window) interconnect structures such as wires and path shapes. You can also clone components that are implemented as one-to-many relationships defined using the *mfactor* property on a schematic instance. For more information, see “Cloning Mfactored Components” on page 145.

**Note:** You cannot use structures in the layout that use many-to-one mapping as the connectivity source for cloning. If you select a many-to-one structure to use as a connectivity source, a warning appears in the CIW explaining the error and the structure is not selected.

**Exact and Non-Exact Matches**

To clone components, you select the source structure to be replicated and the software searches for target structures which match that source. You can specify whether potential targets must match the source exactly or whether the match can be non-exact.

- Exact matches must match in terms of instance master, instance parameters and topological structure.
Non-exact matches can tolerate differences in pin permutation, parameters, and connectivity between the target and the source.

For information on how the cloning engine finds matching targets, see the Cloning form.

For information on troubleshooting cloning, see “Problems with Cloning” on page 455.

**Cloning Mfactored Components**

The cloning functionality handles one-to-many relationships defined using the mfactor property on a schematic instance.

For example, if you have a schematic component with an mfactor of 100, you can choose ten of these components, hook them up as required in the layout window, and then clone the resulting structure ten times to generate all of the mfactored component instances.

You can clone mfactored components both partially and in their entirety across multiple hierarchy levels, including source structures that include folded and abutted devices. When you select a folded instance, all of the other folded legs of the regular device or mfactored device must also be part of the selection.

**Important**

When cloning mfactored devices that are partially implemented in the layout, you must select the clone source from the layout window.

The examples below illustrate what the cloning functionality considers to be valid targets for a number of different mfactored source structures.

**Example 1: Mfactored component instance**

If the clone source is an mfactored component instance (for example, an NMOS instance with m=2), then a valid clone target is also an mfactored component instance; for example, another nmos instance also with m=2.

<table>
<thead>
<tr>
<th>Schematic</th>
<th>Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Schematic Diagram]</td>
<td>![Layout Diagram]</td>
</tr>
</tbody>
</table>
Example 2: Mfactored component instance partially realized in the layout

If the clone source is one or more of an mfactored component instance that is partially realized in the layout, then a valid clone target is the remaining components in the mfactored instance.

Important

You must select the clone source in the layout window because of the one-to-many relationship.
Example 3: One or more of multiple mfactored instances partially realized in the layout

If the clone source is one or more of multiple mfactored component instances that are partially realized in the layout, a valid clone target is the remaining mfactored component instances that follow the same pattern.

Important

You must select the clone source in the layout window because of the one-to-many relationship.

<table>
<thead>
<tr>
<th>Schematic</th>
<th>Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone Source</td>
<td><img src="image" alt="Clone Source Diagram" /></td>
</tr>
<tr>
<td>MN1 [m=4]</td>
<td><img src="image" alt="Clone Target Diagram" /></td>
</tr>
<tr>
<td>MN2 [m=4]</td>
<td>MN1.1</td>
</tr>
<tr>
<td>MN1.1</td>
<td>MN2.1.1</td>
</tr>
<tr>
<td>MN1.2</td>
<td>MN2.2</td>
</tr>
<tr>
<td>MN1.3</td>
<td>MN2.3</td>
</tr>
</tbody>
</table>

And other such targets
Example 4: Mfactored component instance with other component instances

If the clone source is an mfactored component instance along with other component instances, a valid clone target is any other identical configuration with mfactored component instances which have the same mfactor value.

Example 5: One or more of an mfactored component instance partially implemented in the layout along with other component instances

If the clone source is one or more of an mfactored component instance that is partially implemented in the layout along with other component instances, valid clone targets are any
configuration with the same topology and the same mfactor values but which only partially implement the mfactored components in the same way as the clone source.

**Important**

You must select the clone source in the layout window because of the one-to-many relationship.

<table>
<thead>
<tr>
<th>Schematic</th>
<th>Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Diagram of Clone Source]</td>
<td>[Diagram of Clone Target]</td>
</tr>
</tbody>
</table>

**Cloning a Component**

To select a section of a design from either the schematic or the layout to be used as the source for a cloned structure elsewhere in the layout,

1. In the schematic or layout window, select the devices you want to clone.
Important

You can select the connectivity source from either the layout cellview or the schematic or another Virtuoso XL source cellview using single or area selection. If the structure to be cloned has been routed, you must select the structure from the layout view so that all of the required components and interconnects are selected. Selecting the source from the schematic clones only the components and their relative placement, and not the routing or interconnects. If the structure you want to clone is a partially implemented mfactored component, then you must select it in the layout window.

2. In the layout window menu bar, select Create – Clone.

   The CIW prompts you to select a connectivity source.

3. In the schematic or layout window, press Return to confirm your selection.
The Cloning form is displayed.

4. Set the Target Schematic and the Target Layout to the correct cellviews.

Note: The target structure (components and interconnects) must not exist in the layout view at the time of cloning, otherwise the cloning command generates a message stating that the target structure is incorrectly or partially implemented. If the target structure has already been generated in the layout, delete it prior to cloning.
5. Choose whether you want to search for matches in the *Entire Target Schematic* or only the *Selected Set*.

6. Choose the criteria you want to use to find matches from the *Find Match Targets using* section.

   For more information on these options, see the *Cloning* form.

7. Click *Find Matching Targets*.

   Targets that match the specified criteria are listed in the *Unplaced* pane.

8. Select one or more devices from the *Unplaced* list box and click *Clone*.

9. Choose the placement options (*Rotate*, *Flip Horizontal*, *Flip Vertical*, and *Draglines*) that affect the placement or appearance of the clone.

   For more information on draglines, see “Changing the Appearance of Draglines” on page 181.

10. Click in the layout window to place the cloned structure where you want it.

    The cloned structure is placed in the layout.

11. If you selected more than one target structure, click *Clone* on the Cloning form and move the cursor to the layout window again to place the next target.

    The outline of the layout structure follows the cursor. When you click, the cloned structure is placed in the layout. The *Placed* list box lists all of the cloned devices.

**Modifying the Correspondence**

When searching for matching targets during cloning, Virtuoso XL does not consider the relative placement of instances in the source structure as long as swapping these instances does not change the source topology. Consequently, the relative placement of instances in a target can differ significantly from the source.

The Modify Correspondence form lets you change the relative placement of instances in the target structure manually by changing the correspondence of the individual instances in the source and the target. When you modify the correspondence manually, Virtuoso XL checks that the connectivity of the structure remains valid before allowing you to place the clone.

To modify the correspondence of source and target instances,

1. Create a list of matching targets by following steps 1-7 described in “Cloning a Component” on page 149.
2. Select an entry in the *Unplaced* list box and click *Modify*.

The *Modify Correspondence* form is displayed.

The instances in the source structure are listed under *Connectivity Source*. The list is read-only. The corresponding instances in the target structure are listed under *Connectivity Target*.

When you select an instance in the target structure, the corresponding instance in the source structure is highlighted.

3. To change the correspondence of a single instance, select the instance in the *Connectivity Target* list and use the arrow buttons to move it up or down in the list.

4. To swap the correspondence of two instances, select the instances and click *Swap*.

5. When you have finished modifying the correspondence, click *OK*.

Before accepting the new correspondence, Virtuoso XL checks that the connectivity of the structure remains valid.

6. In the Cloning form, click *Clone* and choose the placement options (*Rotate*, *Flip Horizontal*, *Flip Vertical*, and *Draglines*) that affect the placement or appearance of the clone.

7. Move the cursor into the layout window to view the clone.

- If you are happy with the structure, click to place it in the layout.
- If you want to modify the structure further, click *Modify* in the Cloning form and repeat steps 2-6.
Important

You must always click *Clone* in the Cloning form before placing a clone. This ensures that the structure placed reflects all of the changes you made in the Modify Correspondence form.

Cloning a Group of Components

To clone a group of components,

- Create a cell containing all the devices and then clone the cell using the procedure in “Cloning a Component” on page 149.

  This ensures that all the cells or like circuit sections have the same layout parasitics associated with them.

Cloning between Multiple Cellviews

You can place a clone in a different layout from the one in which you selected the source as long as

- The second layout has the same connectivity source as the first layout
- You select all of the source objects from the layout from which you started the *Clone* command (or the corresponding schematic)

To clone an object from one layout cellview into another layout cellview that has the same connectivity source,

1. From the layout window menu bar, choose *Create – Clone*.

2. In the layout window from which you started the *Clone* command (or the corresponding schematic), select the structure you want to replicate (the connectivity source) and press Return.

   The Cloning form is displayed.

3. Open the target layout window and select *Tools – Layout XL*.

4. From the Virtuoso XL layout window menu bar, choose *Connectivity – Update – Source*.

   The Define Connectivity Reference form is displayed.

5. In the Define Connectivity Reference form, specify the schematic that you want to use as the connectivity source for the new layout and click *OK*. 
6. In the Cloning form, change the Target Layout to the second layout window.

7. Click Find Matching Targets.

8. Select devices from the Unplaced list box.

9. Click Clone.

10. Click in the layout to place the cloned structure.

Creating Correspondence Points

Correspondence points are matching pairs of locations in the schematic. You define correspondence points to help you locate the structures you want to clone. Once you define the correspondence points, they are highlighted in bright colors.

You select pairs of items – for example, R1 and R7 – to be stored in a correspondence points file, a standard text file containing net and device names. You can create any number of files with different names and then specify in the Correspondence Points form which one you want.

To create a correspondence points file,

1. From the layout window menu bar, choose Create – Clone.

   The Cloning form is displayed.

2. Turn on Correspondence File.

   The Correspondence File field becomes active.

3. Type the name of the correspondence file you want to create and click Create.

   The Correspondence Pairs form is displayed. The Working File field displays the name of the correspondence points file displayed in the Cloning form.

4. Click Add.
The Add Correspondence Pairs form is displayed.

![Add Correspondence Pairs form]

5. Type the component names or click Set By Cursor and select in the schematic the names of the two components you want to correspond.

6. Click Apply.

   The information is added to the end of the correspondence file you specified. Information at the end of the file is the most recent information and overrides information earlier in the file.

**Displaying Correspondence Points**

To display a specific correspondence point defined in the correspondence points file,

1. From the layout window menu bar, choose Create – Clone.

   The Cloning form is displayed.

2. Turn on Correspondence File.

   The Correspondence File field becomes active.

3. Type the name of the correspondence file you want to display.

4. Click Display.
The **Display Specific Correspondence Components** form is displayed.

![Display Specific Correspondence Components form](image)

5. Turn on **Source** or **Target** to choose the point you want to display.

6. Type the name of the point you want to display or click on it in the schematic or layout.

7. Click **Apply**.

    The correspondence points you specified are highlighted in the schematic.

To display all the points in the correspondence points file,

➤ In the Correspondence Pairs form, click **Display All Pts**.

    All the correspondence points in the correspondence points file are highlighted in the schematic.

To remove the highlighting on all the points in the correspondence point file

➤ In the Correspondence Pairs form, click **Clear All Pts**.

    The highlights are removed from correspondence points in the schematic.

**Removing Correspondence Points**

To remove correspondence points,

1. From the layout window menu bar, choose **Create – Clone**.

    The Cloning form is displayed.

2. Turn on **Correspondence File**.

    The **Correspondence File** field becomes active.

3. Type the name of the correspondence file you want to change.

4. Click **Remove**.
The Remove Correspondence Components form is displayed.

5. Choose whether you want to remove the *Source* or *Target* point.

6. Type the name of one of the correspondence points in the *Name Of Point* field.

7. Click *Apply*.

   The correspondence points are removed from the bottom of the correspondence file.

   If a correspondence point has several corresponding points, all of its correspondence points are removed.

**Online Forms**

- Add Correspondence Pairs on page 159
- Cloning on page 159
- Correspondence Pairs on page 161
- Define Connectivity Reference on page 161
- Display Specific Correspondence Components on page 162
- Import XL Netlist on page 162
- Layout Generation Options on page 163
- Open File on page 167
- Pick from Schematic on page 168
- Remove Correspondence Components on page 170
- Set Pin Label Text Style on page 170
- Startup Option on page 172
Add Correspondence Pairs

**Name in Connectivity Source** lets you type the name of the connectivity point you want to be the source when you click *Apply*.

**Name in Connectivity Target** lets you type the name of the connectivity point you want to be the target when you click *Apply*.

**Set By Cursor** lets you indicate by clicking on a point with the cursor the name of the point you want added when you click *Apply*.

### Cloning

**Options**

**Draglines** controls whether draglines are displayed during the *Clone* command. By default, the draglines indicate connections from the pins of the object you are moving to pins of the nearest objects.

For information on how to change the display of draglines, see the *Draglines* options described in “Layout XL Options” on page 65.

**Correspondence File** enables the text field where you can specify the name of the correspondence file.

**Create** opens the *Correspondence Pairs* form, which lets you edit correspondence pairs.

**Target Schematic** contains the circuit to be replicated by the *Clone* command. The default value for this field is the connectivity source from the cellview pair which launched the *Clone* command.

**Target Layout** is the cellview where the clone is to be placed. The default value for this field is the layout from which the *Clone* command was launched.

**Find Matching Targets on**

**Entire Target Schematic** searches the entire schematic to find matching targets.

**Selected Set on Target Schematic** searches only the selected set to find matching targets.

**Find Matching Targets using**
Permutation means that pin permutability is taken into account when searching for matching targets. If permuting the pins on a device results in a match, then that match is reported.

Exact Parameter Match requires that the parameter values on the target components match exactly the parameter values on the source components.

Exact Connectivity Match requires that the connectivity of the target components matches exactly the connectivity of the source components. If you switch off this option, the software also considers non-exact connectivity matches. You can control the quality and number of non-exact connectivity matches reported using the following options.

Maximum partial net matches sets the maximum number of source nets which can be unmatched in a target structure. If a target structure has more than the specified number of unmatched nets, it is not reported.

Tip
If you set the value for Maximum partial net matches too high, it can prevent the software from finding better target structures with fewer partial nets. To avoid this situation, either lower the Maximum partial net matches value or limit the search to a selected set of instances.

Allowed partial net names specifies the names of source nets that are permitted to be unmatched in the target structure. You can use regular expressions to specify multiple net names with common elements. If a source net name is not listed (or does not match one of the regular expressions specified), then the net must be matched exactly in the target structure.

Maximum found matches sets the maximum number of non-exact target matches reported.

Find Matching Targets invokes the pattern matching.

Unplaced lists the components or structures chosen as target structures for cloning but which are not placed in the target layout window.

Placed lists all the target structures that have been cloned and placed in the target layout window.

Modify brings up the Modify Correspondence form where you can change the correspondence between the instances in source and target structures.

Clone lets you place in the layout window the structure selected in the Unplaced pane.
**Rotate** rotates the component or structure to be placed through 90 degrees counterclockwise.

**Flip Horizontal** mirrors the component or structure to be placed on its y axis.

**Flip Vertical** mirrors the component or structure to be placed on its x axis.

**Correspondence Pairs**

**Working File** lets you accept the default name of the correspondence points file (lvs_corr_file) or type a different name.

**Add** opens a form that lets you add correspondence points to the correspondence points file.

**Remove** opens form that lets you delete correspondence points from the correspondence points file.

**Display** opens a form that lets you display individual correspondence points from the correspondence points file.

**Display All Pts** opens a list box that shows all the correspondence points in the correspondence points file.

**Clear All Pts** clears all correspondence points from the correspondence points file.

**Define Connectivity Reference**

**Source** specifies the connectivity source for the design: *Schematic, Netlist, or None*.

**Library** specifies the name of the library containing the connectivity reference.

**Update layout cellviews hierarchically** descends into a hierarchical layout cellview and updates the library setting for any lower-level cellviews in accordance with the library name specified for the top-level cell. The system traverses the entire hierarchy accessible from the current cellview.

**Note:** Lower-level cellviews updated in this way are not saved until you confirm that you want to do so at the end of the session.

This option is available only when **Source** is set to *Schematic*.

**Cell** lets you type in the name of the cell you want to use as a connectivity reference.

**View** defaults to *schematic*. You can type in a different view name.
Browse displays the Library Browser, which lets you click on names to fill in the form. Even if the Library Browser is already open, you must click Browse if you want the names you click on to appear in the form.

Sel by Cursor lets you specify the schematic or layout to use by clicking in the open cellview window.

Top Cell (appears only if you choose Netlist as the source) lets you enter the top cell of the netlist.

Display Specific Correspondence Components

Identify Connectivity

Source sets the software to display the connectivity point identified as the source when you click Apply.

Target sets the software to display the connectivity point identified as the target when you click Apply.

Name Of Point lets you type the name of the point you want displayed when you click Apply.

Set By Cursor lets you indicate by clicking on a point with the cursor the name of the point you want displayed when you click Apply.

Import XL Netlist

Netlist File specifies the name of the file to use as a netlist.

Copy copies the imported netlist file to the destination library location.

Link links the imported netlist file to the destination library location.

Move moves the imported netlist file to the destination library location.

Library specifies the name of the library in which you want to put the netlist.

Cell specifies the name of the cell in which you want to put the netlist.

View specifies the view name in which you want to store the netlist.

Browse opens the Open File form if your cursor is in the Netlist File text field. From the Open File form, you can choose any file to which you have access. If your cursor is in the Library or Cell or View fields, the Browse button opens the Library Browser form, from which you can choose a library, cell, and view.
Format is either CDL or SPICE.

Check allows you to check the netlist for any errors before generating your layout. Any errors are displayed in the CIW, along with a message stating that there are syntax errors which may be the result of unsupported syntaxes.

Layout Generation Options

Generate

I/O Pins generates all the pins specified in the I/O Pins section of the form. The generated pins are automatically snapped to the placement grid

Instances generates all the instances in the schematic that do not have “ignore” properties attached to them.

Boundary generates a design boundary using the layer and size information you specify in the Boundary section of this form.

Transistor Chaining enables the abutment of ordered lists of MOS transistors into chains and generates them in the design.

Transistor Folding lets you split devices into fingers which keeps the gate width from going beyond the manufacturing foundry specification size.

Preserve Mappings, when switched on, preserves previous many-to-many, many-to-one, and one-to-many mappings. By default these are ignored.

I/O Pins

Apply applies the specified Defaults to all of the pins displayed in the list box.

Term Name is the schematic terminal name. Where terminal and net (wire) names are different in the schematic, Virtuoso XL creates pins in the layout that have the same name as the schematic terminal but the net associated with the pin in the layout is the name of the wire attached to the terminal in the schematic. If there is no explicit wire label in the schematic, both the pin and the net name in the layout are the same as the schematic terminal (default behavior).

Net Name is the net associated with the pin in the layout that is the name of the wire attached to the terminal in the schematic. If there is no explicit wire label in the schematic, both the pin name and the net name in the layout are the same as the schematic terminal.

Pin Type specifies the type of pin to be generated.
Geometric specifies that a geometric pin is to be generated. You must also choose the layer on which to create the pin from the Layer/Master cyclic field.

Symbolic specifies that a symbolic pin is to be generated. You must also choose the name of a symbolic pin master from the Layer/Master cyclic field.

Note: If you choose this option, the Height field is disabled because symbolic pins are square. If there are no symbolic pins defined in the technology file, this option is not available.

Layer/Master lets you choose the layer when generating geometric pins or the master cellview when generating symbolic pins.

For geometric pins, the default is the current drawing layer if it is defined as lxExtractLayer in the technology file. Otherwise, the default is the first layer defined as lxExtractLayer, and the cyclic field offers only the conducting layers. If no layers have been defined as lxExtractLayer, the cyclic field shows all valid layout layers.

For symbolic pins, the field displays the symbolic pin masters defined in the technology file. If none are shown, it means there are no symbolic pins defined in the technology file.

Width specifies the width for each pin. The default is the minWidth property value set for the current layer in your technology file.

Height specifies the height for each pin. The default is the minWidth property value set for the current layer in your technology file.

Num specifies how many instances of this pin to create. If you type 0 in this field, the pin is not created.

Create, when switched on, specifies which of the pins listed on the form (based on pins shown in the schematic) are to be generated in the layout.

Select lets you type in a pin name to select the pin in the list box.

Number Selected shows you the number of pins that are currently selected in the list box.

Add a Pin adds a pin to the list box after you click OK.

Update changes the Pin Type, Layer/Master, Width, Height, Num, and Create settings for one or more selected pins.

Pin Type specifies the type of pin to be updated.

Geometric specifies that a geometric pin is to be updated. You must also choose the layer on which to update the pin from the Layer/Master cyclic field.
Symbolic specifies that a symbolic pin is to be generated. You must also choose the name of a symbolic pin master from the Layer/Master cyclic field.

If you choose this option, the Height field is disabled because symbolic pins are square. If there are no symbolic pins defined in the technology file, this option is not available.

Layer/Master lets you choose the layer when updating geometric pins or the master cellview when updating symbolic pins.

- For geometric pins, the default is the current drawing layer if it is defined as lxExtractLayer in the technology file. Otherwise, the default is the first layer defined as lxExtractLayer, and the cyclic field offers only the conducting layers. If no layers have been defined as lxExtractLayer, the cyclic field shows all valid layout layers.

- For symbolic pins, the field displays the symbolic pin masters defined in the technology file. If none are shown, it means there are no symbolic pins defined in the technology file.

Width specifies the width for each updated pin. The default is the minWidth property value set for the current layer in your technology file.

Height specifies the height for each updated pin. The default is the minWidth property value set for the current layer in your technology file.

Num specifies how many instances of this pin to update. If you type 0 in this field, no pins are updated.

Create, when switched on, specifies which of the pins listed on the form (based on pins shown in the schematic) are to be updated in the layout.

Pin Label Shape defines the type of label generated when you create a pin. This setting is honored by the Gen From Source and Pick From Schematic commands.

- Label creates a label on the layer specified on the Set Pin Label Text Style form.

- Text creates text on the layer specified on the Set Pin Label Text Style form. If the text is not visible, turn on the Pin Names option in the Display Options form.

- None does not create a label or text for the pins.

Pin Label Options opens the Set Pin Label Text Style form, which lets you set the size, font, style, justification and orientation of the label lettering and the drawing or pin layer on which the labels are shown.

Boundary
Layer specifies the layer on which you draw the cell boundary. The default is the entry layer defined by the `initBoundaryLayer` environment variable.

Shape lets you set the shape to Rectangle or Polygon.

- If you set Shape to Rectangle, if you do not want the design boundary’s left and bottom sides located at the default position, you can enter new values in the Left and Bottom fields.

  Left lets you enter a value on the x axis to situate the left side of the rectangle.

  Bottom lets you enter a value on the y axis to situate the bottom of the rectangle.

Additionally, with Shape set to Rectangle, the Boundary Area Estimation controls are also displayed. Use these to control how Virtuoso XL estimates the area required for the design.

  Utilization (%) specifies the percentage of area within the cell boundary that you want to fill. The default is 25%.

Important

When the Transistor Chaining and Transistor Folding options are switched on, the utilization is applied only after chaining and folding is complete in order to calculate the size of the PR boundary accurately.

  Aspect Ratio W/H is the width-to-height ratio of the design boundary. For example, a value of 1 specifies a square boundary, 0.5 specifies a boundary twice as high as it is wide, and 2 specifies a boundary twice as wide as it is high.

Area Calculation controls which shapes are used to calculate the contents of the design and, in turn, the area required to accommodate them.

- PRBoundary Based uses the place and route boundaries of the individual lower-level cells to calculate the area. The place and route boundary is a polygon derived by merging the areas of all of the shapes found on layer-purpose pair (prBoundary boundary). If there are no shapes on (prBoundary boundary), the estimator looks at (prBoundary drawing); if there are no shapes on either of those, the estimator looks at (instance drawing).

- BBox Based uses the bounding boxes of cellview masters to calculate the area. The derived bounding box is typically larger than the place and route boundary in standard cells because of metal overhangs and presence of text display objects and labels. This can result in pessimistic area estimates with values much higher than would be expected for a compact layout.
User Defined lets you specify your own boundary area estimation function in the text field provided. Type in only the name of the function; the cellview ID is picked up automatically from the currently-open cellview.

For more information on defining your own area estimation function, see “User-Defined Area Estimation Functions” on page 107.

If you set Shape to Polygon, the form changes to let you define the number and length of the sides.

Points is a field that lets you enter the points where the sides meet. For example, the format for a rectangle is:

(0 0) (10 0) (10 10) (0 10)

Load Template File for Layout Generation allows you to load an ASCII template file into the Layout Generation Options form.

Browse opens the Open File form, which allows you to specify a file that contains stored layout generation options.

Load loads the stored layout generation options into the Layout Generation Options form.

You can set an environment variable, templateFileName, to always load the name of a template you specify.

Modify Correspondence

Connectivity Source lists the instances in the source structure. The list is read-only.

Connectivity Target lists the instances in the target structure. When you select an instance in the target structure, the corresponding instance in the source structure is highlighted in the Connectivity Source list.

You change the correspondence of an instance by selecting it in the Connectivity Target list and moving it up or down in the list using the arrow buttons. You can also swap the correspondence of a pair of target instances using the Swap button.

When you click OK, Virtuoso XL checks that connectivity is still correct before accepting the new correspondence.

Open File

Name lets you specify the name of a template file.
Pick from Schematic

Placement

**Group As In Schematic** lets you move a group of components together into the layout in the same relative position as in the schematic. In this mode, you cannot change the parameters on the components.

**Place Individually** lets you move each component individually.

**Note:** The chaining option is not available when using the Place Individually option. To chain devices interactively, use the *Edit – Transistor Chaining* command.

Options

**Draglines** controls whether draglines are displayed during the *Pick From Schematic* command. By default, the draglines indicate connections form the pins of the object you are moving to pins of the nearest objects.

For information on how to change the display of draglines, see the *Draglines* options described in “Layout XL Options” on page 65.

**Constraint Assisted**, when active, allows objects to be moved only in ways that satisfy the constraints entered using the Constraint Manager, once the object has satisfied the constraint.

**Transistor Chaining** is active only with the *Group As In Schematic* option. Transistor Chaining allows a set of MOS transistors (or fingers of folded transistors) to be automatically abutted to form a chain. Transistors to be abutted must be set up for abutment first. For more information, see “Setting Up Cells for Abutment” on page 73.

**Transistor Folding**, when active, allows MOS transistors to be automatically divided into fingers for optimum area usage. The *Group As In Schematic* option makes the Transistor Chaining and Transistor Folding options available. **Transistor Folding** operates independently from **Transistor Chaining**. The **Place Individually** option makes available only the **Transistor Folding** option.

**NMOS Fold Threshold**, when **Transistor Folding** is active, lets you type the maximum size of the folded NMOS transistor. This value defaults to the \( lxMaxWidth \) value set on the connectivity source component.

Changing this value overwrites the \( lxMaxWidth \) value you entered in the Edit Component Type form associated with the *Place – Component Types* command.
PMOS Fold Threshold, when Transistor Folding is active, lets you type the maximum size of the folded PMOS transistor. This value defaults to the lxMaxWidth value set on the connectivity source component.

Changing this value overwrites the lxMaxWidth value you entered in the Edit Component Type form associated with the Place – Component Types command.

Unplaced opens the Pick from Schematic Instance/Pin List form displaying a list of each component in the schematic for which there is no corresponding instance or pin in the layout.

When the Place Individually option is active, the form adds the following four fields to the Pick from Schematic form displaying information about the components you select for placement.

LibName displays the name of the library of the component you select from the list of unplaced components or from the schematic.

CellName displays the cell name of the component you select from the list of unplaced components or from the schematic.

ViewName displays the view name of the component you select from the list of unplaced components or from the schematic.

InstName displays the instance name of the component you select from the list of unplaced components or from the schematic.

When a component is selected, the form adds additional fields, depending on the component and its attributes.

Pin Options

When a pin is selected, the form displays the following pin creation options. Generated pins are automatically snapped to the placement grid in the layout view.

Terminal Name automatically displays the name of the pin first in line to be placed.

Type lets you choose the type of pin to place.

Rectangle Pin lets you create a rectangular pin.

Symbolic Pin changes this form to let you create symbolic pins.

Layer allows you to change the layer purpose pair of the pin that has been selected.

Width allows you to change the width of the pin.
Height allows you to change the height of the pin.

Create Label attaches a label, showing the terminal name, to the pin.

Display Pin Name Options opens the Set Text Style form, which lets you set the font, height, justification, and orientation of the pin name.

I/O Type assigns a property used by routers to identify the direction of the signal into or out of this pin. The signal can be input, output, inputOutput (bidirectional), switch (carries data either in or out, but not simultaneously), or jumper (passes data through this pin).

Access Direction assigns a property used to identify the part of the pin to which routers can connect routing. The access direction can be top, bottom, left, right, any, or none.

When a device is selected, the form displays properties set for that device.

Rotate rotates the component 90 degrees counterclockwise.

Sideways mirrors the component on the y axis (flips it horizontally).

Upside Down mirrors the component on the x axis (flips it vertically).

Remove Correspondence Components

Identify Connectivity

Source sets the software to remove the connectivity point identified as the source when you click Apply.

Target sets the software to remove the connectivity point identified as the target when you click Apply.

Name Of Point lets you type the name of the point you want removed when you click Apply.

Set By Cursor lets you indicate by clicking on a point with the cursor the point you want removed when you click Apply.

Set Pin Label Text Style

Height sets the height of the label in user units (usually microns).

Font sets the text style of the label. The choices are euroStyle, gothic, math, roman, script, stick, and swedish.
Text Options

 Drafting prevents the label from being rotated more than 90 degrees.

 Overbar is a display option that determines how text strings containing underscore characters are displayed in a layout window.

 When the overbar is switched off (default), the software displays underscore characters (_ ) as part of the text string. When the overbar is switched on, the software interprets underscore characters in the text string name as toggle switches that control where overbars begin and end. Overbars appear above the text string, as shown in the examples below.

<table>
<thead>
<tr>
<th>Text string</th>
<th>Appears in design window as</th>
</tr>
</thead>
<tbody>
<tr>
<td>_abcde</td>
<td>abcde</td>
</tr>
<tr>
<td>ab_cde</td>
<td>abcde</td>
</tr>
<tr>
<td>_abc_de</td>
<td>abcde</td>
</tr>
<tr>
<td>ab_cd_ef_gh_ij</td>
<td>abcdefghij</td>
</tr>
</tbody>
</table>

Layer sets the drawing layer on which the labels appear.

 Pin Layer sets the pin layer on which the labels appear.

 Justification sets the location of the label origin. The origin appears as a small square on the label when you place or select it. The choices are lowerLeft, centerLeft, upperLeft, lowerCenter, centerCenter, upperCenter, lowerRight, centerRight, and upperRight.

 Orientation lets you specify the orientation of the labels. The choices are

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>No rotation</td>
</tr>
<tr>
<td>R90</td>
<td>Rotate 90 degrees</td>
</tr>
<tr>
<td>R180</td>
<td>Rotate 180 degrees</td>
</tr>
<tr>
<td>R270</td>
<td>Rotate 270 degrees</td>
</tr>
<tr>
<td>MX</td>
<td>Flip upside down</td>
</tr>
<tr>
<td>MY</td>
<td>Flip sideways</td>
</tr>
<tr>
<td>MXR90</td>
<td>Flip upside down and rotate 90 degrees</td>
</tr>
</tbody>
</table>
### Virtuoso XL Layout Editor User Guide

**Generating Your Layout**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MYR90</td>
<td>Flip sideways and rotate 90 degrees</td>
</tr>
</tbody>
</table>

**Startup Option**

- **Create New** opens the Create New File form to let you type the name of a new layout view for your Virtuoso XL design.

- **Open Existing** opens the Library Browser to let you choose the name of an existing layout view for your Virtuoso XL design.

**Template File**

- **All** allows you to select all options so that the boundary, pin, partition, and row information can be saved to a template file.

- **Boundary** allows you to save boundary information to a template file.

- **I/O Pins**
  - **Locations** allows you to save the location of the pins from the Pin Placement form.
  - **Constraints** allows you to save the edge, order, and pitch information from the Pin Placement form.

- **Rows** allows you to save the row information from the Placement Planning form.

- **Partitions** allows you to save the partition information from the Partitioning form.

- **Template file name** allows you to type in a name for the template file, `ex.row.lxt`. 
Editing Your Layout

This chapter shows you how to use the Virtuoso® XL Layout Editor to edit the placement of components in the layout and how to add additional components. This chapter discusses the following topics.

- Identifying Incomplete Nets on page 174
- Moving Objects Manually on page 177
- Swapping Components on page 182
- Permuting Component Pins on page 183
- Using Device Locking on page 187
- Component Types on page 188
- Using Transistor Folding on page 193
- Adding Pins to a Layout on page 199
- Assigning Pins to a Net on page 201
- Using Edit-In-Place on page 202

For information on defining pin groups for external connections (must-connect groups), see “Using Connectivity” in the Virtuoso Layout Editor User Guide.

For information about using constraints to restrict the placement of objects, see the Virtuoso Constraint Manager User Guide.

For information on Virtuoso XL forms, see “Online Forms” on page 203.
Identifying Incomplete Nets

To view flight lines (lines that show uncompleted electrical connections between component terminals, pins, and shapes in each net), follow these steps.

**Note:** Turn off the connectivity extractor by

- Turning off the *Connectivity Extractor* option in the *Layout XL Options*.
- Setting the *extractEnabled* environment variable to *nil* in the *.cdsenv* file.

You will not be able to see flight lines.

1. From the layout window, choose *Connectivity – Show Incomplete Nets*.

   The *Show Incomplete Nets* form is displayed.
By default all incomplete nets are not displayed with flight lines until you select either a single net, a group of nets, or all nets and click Apply in the Show Incomplete Nets form.

**Note:** You can set an environment variable, `showIncNetEnable`, that causes flight lines to appear as soon as the design is opened. The default setting for this variable is `nil`, so flight lines showing incomplete nets do not appear until you choose the Show Incomplete Nets command, select the incomplete nets and click Apply.

The Show Incomplete Nets form lists all of the incomplete nets in the design. This includes

- Unrouted instance terminals with no pins
- Unrouted instance terminals with pins that are not extractable
- Instance terminals with dangling must-join instance pins.

Use the Show Incomplete Nets form to display flight lines only for the nets you specify or to filter by net name the names of the nets shown.

2. Select the nets to be displayed by clicking on their names in the Incomplete list of nets or by typing the names in the text field. You can click on multiple net names or deselect selected names using Control-click. You can select a range of names in the list by clicking on the top name and then clicking on the bottom name with Shift-click.

   You can display nets that have been put into a class. To create a class use the Tools – Constraint Manager form. In the Constraint Manager form change the Select Entity to Support Entities. Click the Classes button and select Create. In the Create Class form, specify the nets to be put into classes and a class name. The class name appears in the list box.

3. Click Apply.

   Flight lines for the nets whose names are highlighted in the list of nets are displayed in the layout. Where possible, a flight line is drawn between shapes in the layout window. When a flight line starts from a logical object, the line is drawn to the center of the instance.

   The Show Incomplete Nets form reports how many incomplete nets there are in the design and how many are displayed.

   **Note:** The Show Incomplete Nets command does not highlight nets that are complete. To highlight a complete net, use the Connectivity – XL Probe command. If you have selected a complete net and the net becomes incomplete by deleting a path segment, then the net is highlighted.
Zooming In on Flight Lines

1. To zoom in on the flight lines that are displayed, click *Zoom*.

   The flight lines and their surroundings are shown enlarged as much as possible depending on how many flight lines are shown.

2. To zoom in on each flight line in turn, click *Individual Flight Line Segments*.

   Arrows appear on the form to let you cycle forward or backward through the displayed flight lines.

Virtuoso XL updates flight lines automatically each time you move a component and make or remove a connection.

Assigning Colors to Incomplete Nets

By default, Virtuoso XL uses the colors you assigned to the Layer Selection Window (LSW) entry layers $y0$ through $y9$ to show flight lines. If there are more than ten incomplete nets in
the design, the colors recycle. When you choose *Show Incomplete Nets*, the colors used are the ones assigned when the command was last used or when the design was last saved.

To assign colors other than the default colors to the flight lines you specify, follow these steps.

1. In the *Incomplete* nets list box, select the names of the nets whose color you want to change.
2. In the color choice cyclic field, select the color you want the nets to be.
3. Click *Set Color*.

   The color of the net changes to the color you specified.

To turn off flight lines in the layout window,

- From the layout window, choose the *Connectivity – Hide Incomplete Nets* command.

Tip

If editing commands run slowly while the *Show Incomplete Nets* command is active, use the Incomplete Nets form to turn off the display of incomplete nets on all the nets except the ones you are working on.

**Moving Objects Manually**

To move objects (devices, pins, or shapes) in the layout window, follow these steps.

1. From the layout window, choose *Edit – Move*.

   The layout window prompts you to select an object to move.

2. Click on the object you want to move in either the layout or the schematic.
The object is highlighted in both the schematic and layout windows. An outline of the object appears in the layout window and follows the cursor until you click in the layout window to place the object.

Draglines show connections from pins of the selected object to pins of the nearest instance or the nearest I/O pin.

For more information, see “Changing the Appearance of Draglines.”

The layout window prompts you to click on the new location in the layout where you want to place the selected object.

3. Click on the location where you want to place the object.

The object moves to the new location.

**Note:** If you select a component in the layout or the schematic before you choose the Move command, Virtuoso XL highlights the corresponding component in the schematic or layout.

4. Click on the next object you want to move or press **Escape** to cancel the command.

**Note:** Using the *Move* command cancels any existing commands in the current or in other windows.

**Moving Objects Using Move Options**

When you move objects, you can use the Move form to change

- The layer of the object you select.
- The angles at which you can move the object.
- The presence of draglines on the object as you move it.
- The orientation of the object you select.
- The use of placement constraints. For more information, see the *Virtuoso Constraint Manager User Guide*.
- The ability to move members of constrained groups individually or together.

To use the *Move* options form to place components, follow these steps.

1. From the layout window, choose *Edit – Move*.
2. Press **F3**.
The Move form is displayed. The *Display Draglines*, *Constraint Assisted Control*, and *Chain Mode* options appear on the Move form only when you are running Virtuoso XL.

3. If you want to change the angle at which you can move an object (*any angle, diagonal, orthogonal, horizontal, vertical*), choose the angle you want from the *Snap Mode* cyclic field.

4. If you want to change the layer on which a shape is drawn, turn on *Change To Layer* and choose a new layer from the cyclic field.

   The layer is updated when you move the cursor into the layout window.

5. If you want to keep chained devices together as you move them select either *All*, *Selected*, *Selected Plus Left*, or *Selected Plus Right*. 
6. If you want to turn the draglines off or on for an object you move, turn *Display Draglines* off or on.

   For more information, see “Changing the Appearance of Draglines” on page 181.

   **Note:** Disabling *Display Draglines* can speed up the action of the *Move* command.

7. To move groups of components bound by placement constraints as a unit or individually, turn on *together* or *individually*.

   If a group of components you move is affected by placement constraints, those constraints are obeyed during the duration of the *Move* command.

   If a single component you move is subject to an unsatisfied Distance, Fixed, or Grouping (with a fence) constraint, and you move it across a coordinate that satisfies the constraint, the component snaps to that coordinate.

   Components restricted by Alignment or Symmetry constraints do not snap to coordinates when you move them, but if their constraints have already been satisfied, you can move them only in ways that do not break the constraints.

   In the case of a group constraint with a fence, the *Move* command lets you move the device until it is inside the fence; after that, you cannot move it outside the fence.

8. Use the *Delta X/Y* fields to type in the coordinates of the selected devices instead of using your cursor to move devices. In the *Delta X* and *Y* text fields type the amount of coordinates to move the devices, and then click *Apply XY*.

9. To turn off the use of placement constraints for the components you move, turn on *ignore*.

10. To change the orientation of the object, click on the button describing the orientation you want (*Rotate*, *Sideways*, or *Upside Down*).

    The orientation of the object changes when you move your cursor into the layout window.

    **Note:** When you use the *Edit – Stretch* command with Virtuoso XL running, the *Display Draglines* and the *Constraint Assisted Control* options also appear on the
Stretch form and work the same way.

![Stretch Form Illustration]

### Changing the Appearance of Draglines

When using the *Create – Pick From Schematic, Create – Clone, Edit – Move, and Edit – Stretch* commands, Virtuoso XL displays draglines which connect the pins on the object to the nearest pins on another object on the net. When you move the object closer to a different object on the same net, the lines reconnect to the new object.

**Note:** This option is available only in Virtuoso XL.

If you need to know more about the potential connections for the instance you are working on, you can change both the number and the appearance of the draglines using the options under *Draglines* in the *Layout XL Options* form.

You can display draglines for all of the connections to the instance you are working on, hide the draglines for global nets and for power and ground nets, and have each dragline shown in a different color.
Tip

Disabling Display Draglines can speed up the action of the Move command.

Setting the Move Form to Appear Automatically

To set the Move form, or other options forms, to appear each time you use the Move command (without having to press F3) follow these steps.

1. From the CIW, choose Options – User Preferences.
   
   The User Preferences form is displayed.

2. Click Options Displayed When Commands Start.

3. Click OK.

   The Move form is displayed each time you choose the Move command.

Swapping Components

To swap the location of two components in the layout window, follow the steps below.

1. From the layout window, choose Edit – Other – Swap Components.

2. Click on the first component in the layout window.

   Virtuoso XL prompts you to select a second component.

3. Click on the second component in the layout window.

   Virtuoso XL switches the locations of the components you selected, but retains the orientation of the component associated with the position.

4. To undo the swap, from the layout window choose Edit – Undo.

5. To swap another two components, choose Edit – Other – Swap Components again.
Virtuoso XL does not move the connections to the components. If swapping components causes shorts, markers show where the shorts are. If swapping components causes open connections, flight lines show the open connections.

The *Swap Components* command does not allow you to swap components that are part of a satisfied constraint (unless the swap maintains the constraint).

**Permuting Component Pins**

Virtuoso XL lets you permute (exchange the connectivity or net connections of) the pins of a component.

**Pins to be permuted**

- Must belong to different nets.
- Must first be defined as permutable terminals. For more information, see “permuteRule” on page 789.

When the *Auto Permute Pins* option is active (the default setting) in the Virtuoso XL Options form, Virtuoso XL automatically permutes pins in a component if doing so corrects a short caused by routing when the routing was done interactively (using automatic abutment) or done while moving or stretching a shape (or using any operation that changes connectivity).

To change the setting of the *Auto Permute Pins* option, follow these steps.

1. From the layout window, choose *Options – Layout XL.*
The Layout XL Options form is displayed. For more information, see “Layout XL Options” on page 65.

2. Turn on *Auto Permute Pins*.

3. Click *OK*.

Virtuoso XL automatically permutes the pins of a single component when doing so removes a short, as shown below. In the diagram below, two pins belonging to different nets (A and B) create a short when they are touching each other. If these pins are permutable, and reversing the pins of one component connects the two pins that are on the same net (B), Virtuoso XL permutes them.

Virtuoso XL also automatically permutes the pins of two components at the same time when doing so removes a short, as shown below. In the diagram below, two pins belonging to different nets (A and C) create a short when they are touching each other.
If these pins are permutable, and reversing the pins of both components connects the two pins that are on the same net (B), Virtuoso XL permutes them.

Permuting Pins Manually

To manually permute pins within a component, follow these steps.

1. From the layout window, choose Connectivity – Permute Pins.
   Virtuoso XL prompts you to select a pin.

2. Click on the first pin in the layout window.
   Virtuoso XL prompts you to select a second pin.

3. Click on the second pin in the layout window.
   Virtuoso XL switches the net connections of the pins you selected. If flight lines are displayed, the flight lines change to show the new nets assigned to the pins.

Note: This command is modal; that is, you can continue to select sets of pins to permute without going back to the command again each time.

4. To undo the permutation, from the layout window, choose Edit – Undo.
The pins return to their original net connection in the layout.

5. To cancel the command, press `Escape`.

**Note:** Any pin previously connected to a path, polygon, rectangle, or component pin cannot be permuted.

### Checking Permutation Information

To see information about permuted pins, follow these steps.

1. From the layout window, choose *Connectivity – Permute Pins*.
2. Press `F3`.

The Permutation Information form is displayed.

If the pins you select do not permute, this form also tells you why.

![Permutation Information Form](image)

**Caution**

*It is not recommended to set the Probe Options form to “Pins” when probing pins for permuted instance. Instead set the Probe Options form to “Nets” to view the correct information.*
### Updating Permutated Pins

Permutated pins are not preserved by the *Update Components and Nets* command. If any component in the design needs to be updated, the command resets the permutation status of all of the cells to that originally stored in the schematic.

### Using Device Locking

To lock devices in their current positions in the layout (set temporary fixed constraints), so that they cannot be moved during interactive or automatic placement, follow these steps.

1. In the layout window, select the devices to lock.
2. From the layout menu, choose *Edit – Other – Lock Selected.*

   The locked devices cannot be moved when the Constraint Assisted option is switched on in the *Move* or *Stretch* commands.

   If the selected devices are constrained by other fixed constraints, a message window asks you to approve overwriting those constraints.

   Fixed constraints for the selected devices appear in the constraint manager and the Constraint Status Browser Window if they are open. For more information, see the *Virtuoso Constraint Manager User Guide.*

   When you save the design, a dialog box asks you if you want to convert the locked devices into permanent fixed constraints.

   Click *OK* to convert the locks to fixed constraints.

To delete the locks on the devices you locked, follow these steps.

1. In the layout window, select the devices to unlock.
2. From the layout menu, choose *Edit – Other – Unlock Selected.*

   The temporary fixed constraints are removed from the locked devices and they can then be moved by the *Move* or *Stretch* commands.

You can also remove these temporary locks using the Constraint Manager by following these steps.

1. In the Constraint Manager window, click on the *Fixed* category to show the individual constraints.
2. Middle click and hold over the name of the device to unlock.
3. From the pop-up menu that appears, choose *Delete*.

   The locked devices are no longer locked in position. The names of the locked devices disappear from the *Fixed* constraint category in the Constraint Manager window and in the Constraint Browser.

**Component Types**

A component type is part of the library or cell information for a cell in which components are placed. Its purpose is to identify NMOS and PMOS transistor cells and set the parameters for device chaining and folding. The component type also controls how components are assigned to rows.

You can define a component type at the library level if all of the cells in the library share the same types. You can define different component types for individual cells that will contain the placed components.

You set the component type using the *Design – Component Types* command.

**Caution**

> *If you edit component types in IC 5.0.0 or later, the data is saved in a new format that is incompatible with IC 4.4.6.*

**MOS Transistor Chaining and Folding Parameters**

MOS transistors that are candidates for chaining and folding need to be identified in the Edit Component Types form and must have certain parameter values set before you generate a layout. For more information, see “Defining a Component Type” on page 190.

Use the parameters described below in NMOS or PMOS type definitions to control device chaining and folding during placement.

**Note:** You must set the *Active Layer, Width Par Name* and *Fold Threshold* parameters for NMOS and PMOS type definitions; the other parameters are optional. Terminal names are not case sensitive.

*Active Layer* (*lxActiveLayer*)
Specifies the diffusion layer for NMOS and PMOS devices.

This parameter must be set for NMOS or PMOS component type definitions to control device chaining and folding during placement. If the Component Class is set to either PMOS or NMOS and there is no lxActiveLayer set, a message appears in the CIW informing you that the lxActiveLayer must be set.

The Placement Planning form uses the layer-purpose pair to look up in the technology file the spacing rules for the specified layer and uses these rules to seed the defined spacing type in fields on the Assisted CMOS form.

The Calculate Estimates command for Assisted CMOS placement uses the spacing rule between adjacent MOS device chains to achieve more accurate estimates.

*Width Par Name* (lxDeviceWidth)

Specifies the name of the transistor width parameter on the device master cell. The default name is \( w \). If you use a different name for the transistor width parameter, type that name into the text field.

**Note:** You must set this parameter even if the devices will not be folded.

*Fold Threshold* (lxMaxWidth)

Specifies the maximum width beyond which NMOS and PMOS must be automatically folded. You must set this parameter, even if the devices will not be folded.

This value applies globally to all devices within the scope of the component type definitions (i.e., the library containing the current cell, or just the current cell). You can override this value using the layout generation commands *Gen from Source* and *Pick from Schematic*.

You can override the *Fold Threshold* by setting *PMOS Width Threshold* and *NMOS Width Threshold* in the Placement Planning form.

**Note:** Virtuoso XL cannot enforce the *Fold Threshold* value unless all device widths and all values are specified consistently; for example, all in meters (10um is 1-e-5) or in all in user units (10um is 10).
**Component Class** (lxMOSDeviceType)

Specifies the device type. This parameter must be set to either NMOS or PMOS to allow Virtuoso XL and the placer to identify device types for folding, chaining, and assisted MOS operations.

**Valid Values:** Undefined, PMOS, NMOS, STDCELL, STDSUBCONT, FILLER

**Source Terminal** (lxSourceName)

Lists the instance terminal names corresponding to the MOS transistor source terminal. The default names are source and S. If you use different a different name, type it into the text field.

**Drain Terminal** (lxDrainName)

Lists the drain terminal names. The default names are drain and D. If you use different a different name, type it into the text field.

**Gate Terminal** (lxGateName)

List the gate terminal names. The default names are gate and G. If you use different a different name, type it into the text field.

**Bulk Terminal** (lxBulkName)

Lists the bulk terminal names. The default names are bulk and B. If you use different a different name, type it into the text field.

**View Name** (lxViewName)

Specifies the view name to be used when defining a filler cell, standard cell, or standard cell substrate contact.

When the **Component Class** is set to STDCELL, STDSUBCONT, or FILLER, this is the only parameter that is valid.

**Defining a Component Type**

To define a component type and assign cells to it,

1. In the layout window menu bar, choose Design – Component Types.
The Edit Component Types form is displayed.

2. Choose the **Scope** for the component type you want to create.

   - **Current** means that the component type applies only to the current cell.
   - **Lib** means that the component type applies to the current library.
   - **Ref Libs** means that the component type applies for one of the reference libraries. Choose the library you want from the cyclic field.
     
     You specify the list of reference libraries in the Layout XL Options form.

**Important**

When it searches for values, Virtuoso XL reads the **Ref Libs** settings first. If there are no **Ref Libs** settings, it reads the **Library** settings next. **Current** settings override both **Library** and **Ref Libs** settings.

The software updates the **Cells in Selected Scope** pane.
3. Create a new component type by typing a name into the text field and clicking Add.

   The new component type name is listed in the Type cyclic field.

4. To populate the Components in type list, do one of the following.

   □ Choose the cells you want to assign to this component type in the Cells in Selected Scope list and click >>> to move them into the Components in Type list.

   □ Select the cells in the layout window design area and click Add Selected Components to move them into the Components in Type list.

   **Note:** You cannot add instances in component types.

   If you have not already generated a layout for the design, and the Cells in Selected Scope list is incomplete or empty,

   a. Type the library and cell name in the Cells in Selected Scope field. Separate the names with a space and enclose the list in quotation marks; for example, "MYLib Mycell".

   b. Click >>> to move the cell into the Components in Type list.

   c. Repeat steps a and b as required.

5. Use the cyclic field at the bottom of the form to choose the Component Class for this component type. The component class is required if you want the cells to be included in the placer’s component-assisted row generation.

   □ Choose PMOS or NMOS if you are using the placer’s MOS component-assisted row generation.

   □ Choose STDCELL if you are using the placer’s Standard Cell component-assisted row generation.

   □ Choose either if you are using a combination of placement modes.

   □ Choose Undefined if the device is none of the others.

6. Specify values for the placement parameters in the Edit Parameters fields. Select the parameter from the cyclic field and type its value in the text field.

   □ For PMOS and NMOS component types, specify values for the Width Par Name, Fold Threshold, and Active Layer parameters. This enables the devices to be chained and folded.

   □ (Optional) For PMOS and NMOS component types, supply values for the following parameters if they are different from the defaults: Source Terminal, Drain Terminal, Gate Terminal, Bulk Terminal.
For more information on these parameters, see “MOS Transistor Chaining and Folding Parameters” on page 188.

7. Click OK or Apply at the top of the form to complete the component type definition.

**Note:** Setting the component types in this form sets the `lxComponentType` parameter for the cells in question.

### Modifying a Component Type

When modifying a component type, be sure that the *Scope* of the component type you are editing (library or cell) is appropriate. It is possible to modify a component type with a narrower scope than the one with which it was created (changing library-wide type definitions for just one cell, for example). This might not yield the result you want.

If you change the *Fold Threshold*, you will need to re-run `Design – Gen From Source` for the changes to take effect.

To modify a component type,

1. In the layout window menu bar, select `Design – Component Types`.
   
   The `Edit Component Types` form is displayed.

2. Choose the *Scope* for the changes you want to make.

3. Choose the component type you want to change from the *Type* cyclic field.

4. Change the component type by
   
   - Adding or removing cells to or from the *Components in Type* list.
   
   - Selecting a parameter from the `Edit Parameters` cyclic field and typing a different value into the text box.

5. Check that *Component Class* is set to the correct value.

6. Click OK or Apply to save your changes.

### Using Transistor Folding

You can use the *Transistor Folding* command to interactively divide a transistor or a transistor chain (an abutted group of MOS transistors) into two or more fingers, layout instances with terminals all connected in parallel to the same nets. Folding transistors and chaining them allows you to change their aspect ratio for design efficiency.
Transistors to be folded must have a property named `lxComponentType` to store their width attributes set on the cell. You must use the Edit Component Types form associated with the Design – Component Types command to set this property on the cell. Using this form, you can define `lxDeviceWidth`, the MOS device gate width parameter used to edit the transistor folding (required), and `lxMaxWidth`, the maximum transistor width (optional).

For more information about setting these parameters, see “Component Types” on page 188.

You must have the `mfactorSplit` environment variable set to `t` in a setup file for the design, or the Fold Transistors command does not work. For more information, see “mfactorSplit” on page 546.

Folded devices are named `instName.integer` in the layout. For example, if you have an instance named `I0` and you fold it into three legs, the legs are named as follows.

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Folded Instance Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>I0</td>
<td>I0.1</td>
</tr>
<tr>
<td></td>
<td>I0.2</td>
</tr>
<tr>
<td></td>
<td>I0.3</td>
</tr>
</tbody>
</table>

**Note:** You can keep folded devices from becoming off-grid by setting the `lxRounding` property. For more information, see “`lxRounding`” on page 771.

**Folding Transistors**

To fold one or more transistors, follow these steps.

**Note:** You can preselect the transistors you want to fold in the layout or schematic window. If you select a transistor that has already been folded, you must select it from the schematic or select all of its fingers at the same time in the layout.

1. Choose *Edit – Transistor Folding.*
The Set Transistor Folding form is displayed.

If you did not preselect any transistors, no transistor-specific information is shown in the form until you select at least one.

If you preselected transistors to fold, the name of first one you selected is shown in the Transistor Name field.

The width of the transistor, as specified in the schematic, is shown in the Transistor Width field.

2. For the selected transistor, type in the Number of Fingers field the number of fingers into which you want it divided.

   The Set Finger Widths option button appears on the form.

3. To set the finger widths to unequal widths, click Set Finger Widths.
The form changes to let you enter the finger widths.

You can add new fingers with the Add Finger button, delete fingers with the Delete button opposite each finger, and assign the same width (dividing the total width of the transistor by the number of fingers) to all the fingers with the Same Width button.

Note: Interactive folding might overwrite the default $\text{lxMaxWidth}$ parameter; by using the specifications you supply. For example, if you fold the transistor in too few fingers, the fingers might end up longer than the $\text{lxMaxWidth}$ parameter you must specify using the Edit Component Types form associated with the Design – Component Types command. Automatic folding always respects the $\text{lxMaxWidth}$ parameter.

4. Type the finger widths you want.
5. If you do not want the fingers that are created to be chained, turn off *Chain Fingers*.

6. Click *Apply*.

7. Move the cursor into the layout window.
   
   A representation of the fingers of the transistor follows the cursor in the layout window.

8. Click in the layout where you want to place the transistor fingers.
   
   The transistor fingers are placed where you click.
   
   If you have selected more than one transistor, you can press the *Next or Previous* buttons to move to another transistor. You must click *Apply* to save the changes to a transistor before clicking *Next or Previous*.

**Notes**

- When the *Fold Transistors* command is active, you cannot select any pcell that has a fingering value set to greater than 1.

- If you enter an lxMaxWidth parameter that would yield a device with more than 999 fingers (for example, a device width of 2 and lxMaxWidth of 1 µ, which could result in a device folded in 2 M fingers), the Command Interpreter Window (CIW) prints a warning and does not fold the device. You can still fold such a large device manually.

- Interactive folding always considers the parameter that sets the device width in the source, even if you have specified that this parameter is to be ignored in the IgnoreParamsForCAS or paramsToIgnoreForCheck environment variables.

**Adding Instances to a Layout**

To add an instance not present in the schematic to the layout, you can add the instance to the schematic and connect it to a net, as explained below, or you can add it to the layout using the layout editor *Create Instance* command and connect it with the Virtuoso XL *Update Components and Nets* command.

**Note:** The schematic must be opened in edit mode.

To add an instance not present in the schematic to the layout, follow these steps.

1. In the schematic window, choose *Add – Instance*. 
The Add Instance form is displayed. For more information on this form, see “Add Instance Form” in the *Virtuoso Schematic Editor User Guide*.

2. In the *Library* field, type the library name of the new instance.
3. In the *Cell* field, type the cell name of the new instance.
4. In the *View* field, type the view name of the new instance.
5. In the *Names* field, type the instance names (if any).
6. Move the cursor to the schematic. The outline of the instance follows the cursor.
7. To rotate the instance press the right mouse button. Press Shift-right-click to mirror the instance.
   
   **Note:** You must have the Virtuoso XL bindkeys loaded in order to use the right mouse button for rotation and mirroring.
8. Click in the schematic where you want to place the instance. The instance is placed in the schematic.
9. To connect the instance to a net, use the schematic window *Add – Wire* command.
10. When you have finished adding instances to the schematic, from the layout window, choose *Connectivity – Update – Components and Nets*. A dialog box asks whether you want to extract the schematic.
11. Click Yes.

Virtuoso XL extracts the schematic and adds the instance to the layout.

**Note:** If you have a large design and are only adding a few instances, it saves time to use the *Connectivity – Update – Device Correspondence* to update each instance rather than using *Connectivity – Update – Components and Nets* to update the whole design.

**Note:** If the instance does not need to be connected to a net, add it directly to the layout and add the `ignore` property to it. For more information, see “ignore” on page 752.

**Adding Pins to a Layout**

Pins occurring in both the schematic and the layout must have identical names. If you add pins that are not in the schematic to the layout, Virtuoso XL maintains their connectivity in the layout. To add a pin to the layout, follow these steps.

1. In the Layer Selection Window (LSW), choose the layer on which you want to create the pin.

2. From the layout window, choose *Create – Pin*.
The Create Symbolic Pin form is displayed. For more information, see “About the Create Symbolic Pin Form” in the *Virtuoso Layout Editor User Guide*.

3. In the *Terminal Names* field, type the terminal name for the pin.
   
   The pin name must match the name of an existing net in the schematic. You can create more than one pin with the same name.

4. In the *I/O Type* section, choose the appropriate I/O type.
   
   **Note:** To create a pin for a net that is not on the schematic (a feedthrough, for example), give the pin a new name.

5. In the layout window, click to place one corner of the pin; then click to place the opposite corner of the pin.
   
   The pin appears on the layout window.
   
   **Note:** Do not place pins where you do not want to make a connection. For example, do
not create a pin on poly that covers the gate area of a FET.

If *Show Incomplete Nets* is on, flight lines show the pin connected to the net.

**Note:** You can also add pins defined with shapes. You can use any layer purpose for pin shapes. Pins made as large as possible make routing easier.

### Assigning Pins to a Net

If you place instances in a layout that does not have a connectivity source (a schematic or netlist), the software does not connect them to any net unless you specifically assign the instance pins to a net using the *Assign Nets* command.

You can also use this command to connect a new instance to an existing net in a design that has a connectivity source.

To assign the pins of an instance that you place in a layout to a net that you specify, follow these steps.

1. Place the new instances that you need in the layout using *Create – Instance*.
2. Choose *Connectivity – Show Incomplete Nets*.
3. Choose *Connectivity – Assign Nets*.
   
   The layout window and the CIW prompt you to select a pin to add to a net.
4. Click on a pin in one of the new instances you have placed.
   
   You can use Shift-click or click and drag to select multiple pins to be added to the same net. You can use Control-click or click and drag to deselect multiple pins you have selected in this step.
   
   The pin you select is highlighted. The layout window and the CIW prompt you to select a net to add the pin to.
5. Click on a pin that is connected to the existing net you want to add the pin to.
   
   The flight lines representing that net are extended to show that the pin is added to the net. The CIW lists the name of the pin and the name of the net it was connected to.
   
   The layout window and the CIW prompt you to select another pin to add to a net.
6. Display a list of all the nets in the layout by pressing F3.
The Assign Nets form is displayed listing all the nets in the layout.

7. To add a pin you selected in the layout to a net listed in this form, click on the name of the net in the form and then click Apply.

The flight lines representing that net are extended to show that the pin is added to the net.

8. To exit this command, press Escape.

Using Edit-In-Place

To edit a placed instance in your design and maintain the connectivity use the Design – Hierarchy – Edit In Place command. You can also use the Design – Hierarchy – Descend command to descend into an instance.
Important

You cannot edit parameterized cells in place.

To edit a placed instance in your design,


2. Click on the instance whose cell you want to edit.
   The banner at the top of the window changes to show that you are now editing the master cell for that instance.

3. Choose Window – Fit Edit to fit the design to the window.
   Notice the instance is highlighted along the border.

4. After you have edited the design choose Design – Hierarchy – Return to stop editing in place, and to return you to the next level of the hierarchy.
   You can also specify the level of hierarchy to return to by using the Design – Hierarchy – Return To Level command.

Setting the Default Application

If the new master cellview opens in Virtuoso Layout Editor rather Virtuoso XL, it means that you have specified Virtuoso Layout Editor as the default application to use when opening cellviews of this type.

You can change this by choosing Design – Set Default Application from the layout window and setting the default application to Virtuoso XL instead.

For more information, see Chapter 4, “Setting Up Your Environment,” in the Virtuoso Layout Editor User Guide.

Online Forms

- Assign Nets on page 204
- Edit Component Types on page 204
- Move on page 206
- Set Transistor Folding on page 207
- Show Incomplete Nets on page 208
Assign Nets

Create enters in the list of net names the new net name you type in the entry field.

Delete deletes from the list of net names the name you type in the entry field or select in the list of net names if no objects are attached to that net.

The entry field above the list of net names has a search mechanism that highlights in the list box the net name you type in the field.

Edit Component Types

The Edit Component Types form lets you create and modify placement component types for libraries or cells.

Scope specifies the scope for the component types you are defining. Components that you place in the cells encompassed by the scope share the component type definitions you create in the lower part of the form.

- Lib assigns these type definitions to the library containing the current cellview.
- Current assigns these type definitions to the current cellview only.
- Ref Libs assigns these type definitions to the specified reference library from the cyclic field. The values for the cyclic field are set in the Layout XL Options form.

Type is the name of a component type. Enter a new type name in this field or choose an existing type from the cyclic field.

Add creates a new type with the name shown in the Type field.

Delete removes the type named in the Type field from the types defined for the current scope. If you delete a type accidentally, click Cancel to discard all of the changes you have made since you last clicked OK or Apply.
Important

You cannot use this command to define chaining and folding parameters for instances. All instances of a master must share the same parameter values, and you must define those values on the master cell (by including the master in the Components in Type list).

Cells in Selected Scope lists all the cells in the current Scope.

If the scope is Current, all the cells in the current design are listed; if the scope is Ref Libs, all the cells in the specified reference library are listed; if the scope is Lib, all the cells in the library and all the cells in the current design are listed (if there are no overlaps).

For example, the current library, A, has two cells; cell1 and cell2. The design uses cell1 from library A. The display shows A cell1 and A cell2 (instead of A cell1, A cell1, A cell2). A cell1 is the overlap between the cells in the current library and the cells in the current design.

Another example uses cell1 from library B. The cells displayed would be B cell1, A cell1, and A cell2.

You can highlight the cells by clicking on them, or typing a quoted library and cell name in the type-in field. The syntax is "myLib myCell".

Browse opens the Library Browser – Edit Component Types window where you can search for particular types of cells.

Report tells you the component types to which a selected cell is assigned. The information is printed in the CIW window.

<table>
<thead>
<tr>
<th>Cell: MOSLIB FC6</th>
<th>Current View</th>
<th>Library</th>
<th>refLib: acpd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FC</td>
<td>&lt;- FC</td>
<td></td>
</tr>
</tbody>
</table>

If you click Report with no cell selected, all cells are reported for all component types in all scopes. Component types that are overloaded have a <- next to the component type name.

>>> moves the selected components to the Components in Type list.

Components in Type lists the masters and instances belonging to the current Type.

Delete unassigns the highlighted master or instance from the current Type.

Select selects the highlighted cells in the layout and schematic windows.
Unassigned Cells

Report lists in the CIW the cells that are not assigned to a component type.

Select in Layout automatically selects the unassigned cells in the layout and schematic windows.

Edit parameters is a cyclic field containing the names of the required and optional placement parameters; the name of a parameter shared by the cells in a type. Enter new names in this field, or click on parameter names in the list box to select them for editing.

For information about each parameter, see “MOS Transistor Chaining and Folding Parameters” on page 188.

Component Class lets you identify MOS transistors suitable for chaining, folding, and the Assisted MOS placement style. For MOS transistors, set this field to NMOS or PMOS as appropriate or set STDCELL, STDSUBCONT or FILLER for those design components. For all other devices, set this field to Undefined.

You can have multiple devices assigned to the NMOS and PMOS component types. This would typically be the case for multiple voltage designs.

Move

Snap Mode controls the direction in which you can move the object.

Change To Layer, when on, lets you move a shape to another layer. Choose the new layer from the cyclic field, which displays all the available valid layers.

Chain Mode allows you to specify how to move chained devices.

All moves the selected device and the other devices that are chained together.

Selected moves only the selected device out of the chain.
**Selected Plus Left** moves the selected device and all of the other devices to the left of the chain.

**Selected Plus Right** moves the selected device and all of the other devices to the right of the chain.

**Delta X/Y** allows you to specify the distance to move the selected objects in the X and/or Y direction.

**Apply XY** invokes the distance specified in the **Delta X/Y** fields.

**Display Draglines** controls whether draglines are displayed during the **Move** command. By default, the draglines indicate connections from the pins of the object you are moving to pins of the nearest objects.

For information on how to change the display of draglines, see the **Draglines** options described in “Layout XL Options” on page 65.

**Constraint Assisted Control** allows objects to be moved only in ways that satisfy the constraints entered using the constraint manager.

- **together** moves preselected objects in the layout as a group.
- **individually** moves preselected objects in the layout individually.
- **ignore** specifies that constraints have no effect on objects in the layout when they are moved.

**Rotate** rotates the component 90 degrees counterclockwise.

**Sideways** mirrors the component on the y axis (flips it horizontally).

**Upside Down** mirrors the component on the x axis (flips it vertically).

**Set Transistor Folding**

**Transistor Name** displays the name of the first transistor you selected.

**Transistor width as specified in the schematic** displays the value of the width property of the transistor in the schematic.

**Number of Fingers** lets you enter the number of fingers (divisions) into which you want to fold the transistor. Entering a value in the **Number of Fingers** field causes the appearance of the **Set Finger Widths** option button.
**Set Finger Widths**, when turned on, changes the Set Transistor Folding form to include new fields that let you enter the width of each finger.

**Chain Fingers**, when active, allows the software to abut the fingers into chains.

**Total Width** displays the total value of the all widths you enter in the Width fields.

**Add Finger** adds another Width field to the form to let you enter the width of an additional finger.

**Same Width** sets all the values in the Width fields to the same width (totaling the same value as specified in the schematic, unless you override that value in the entries you make in the Width fields).

**Width** lets you enter a separate width for each finger.

**Delete** removes the Width field and decreases the number in the Number of Fingers field by 1.

---

**Show Incomplete Nets**

**Cellview** displays the library, cell, and view name of the layout window on which the Show Incomplete Nets form is focused.

**Class** displays class names in the list boxes and prepares the search fields for class names. Classes are created in the Constraint Manager form.

**Net** displays net names in the list boxes and prepares the search fields for net names.

Color choice cyclic field:

- **Cycle** assigns the next entry layer in the color choice cyclic field to the next flight line you choose.

  **Note:** You can assign one of the y1 through y9 entry layers as a single color for all the flight lines. You cannot assign the y0 layer as a single color for all flight lines because it is the default layer.

- **As Is** keeps the colors currently assigned from the y1–y9 entry layers in the color choice cyclic field.

**Set Color** sets the color of all incomplete nets highlighted in the Incomplete list box and highlighted in the layout window to the drawing layer color shown in the color choice cyclic field.
Search lets you type one or more net or net class names (separated by a space) to be highlighted in the Incomplete or Complete list boxes and in the layout window. You cannot use the wildcard character ( * ) in this field, but typing the initial characters of net names highlights all names in both lists beginning with the specified characters.

Incomplete

The type-in field lets you type partial or complete names of nets to be highlighted in the Incomplete list box and displayed in the layout window (as long as they are incomplete) when you click Apply.

The list box lets you select the names of nets you want to display or to which you want to assign a color. You can use Control-click to select more than one or to deselect selected net names.

Select All displays flight lines for all incomplete nets in the design when you click Apply.

Select from Display lets you click on pins and instances in the layout window to select them and displays them when you click Apply.

Less truncates the form so that it does not display the Complete nets list box or the Zoom features; the button label changes to More.

More expands the truncated version of the form to include the Complete nets list box and the Zoom features; the button label changes to Less.

Complete

The type-in field lets you type in partial or complete names of nets to be highlighted in the Complete list box and displayed in the layout window (only if they become incomplete) when you click Apply. This capability lets you mark nets of critical importance so that they show up if their connectivity is lost.

The list box lets you select the names of nets you want to display or to which you want to assign a color. You can use Control-click to select more than one or to deselect selected net names.

Clear All Selections removes the display of flight lines for all incomplete nets in the design when you click Apply.

Zoom enlarges the flight lines and the area around them.

All Displayed Flight Lines enlarges as much of an area as possible around all the flight lines.
**Individual Flight Line Segments** enlarges as much of an area as possible around individual flight lines. This option displays the zoom arrows.

**Zoom arrows** (--> and <--) display each net individually in the order listed in the *Display* field.

To zoom out again after viewing individual flight line segments, in the *Zoom* section, select *All Displayed Flight Lines*, then in the *Incomplete* section, click *Select All*, and then click *Zoom*.

The list box at the bottom of the form tells how many incomplete nets are in the design and how many incomplete nets are displayed.

**Stretch**

**Snap Mode** controls the direction in which you can move the object.

- **anyAngle** (default)
- diagonal
- orthogonal
- horizontal
- vertical

**Lock Angles**, when on, prevents you from changing the angle of a corner or edge as you stretch it.

**Delta X/Y** allows you to specify the distance to stretch the selected objects in the X and/or Y direction.

**Apply XY** invokes the distance specified in the *Delta X/Y* fields.

**Display Draglines** controls whether draglines are displayed during the *Stretch* command. By default, the draglines indicate connections from the pins of the object you are moving to pins of the nearest objects.

For information on how to change the display of draglines, see the *Draglines* options described in “Layout XL Options” on page 65.

**Use “ignore” for Stretchable Pcells** reminds you to choose the *ignore* option in the Constraint Assisted Control section when you are working with stretchable pcells because, if Constraint Assisted Controls are active, you cannot use the stretch handle to stretch pcells.
Constraint Assisted Control allows objects to be moved only in ways that satisfy the constraints entered using the constraint manager.

- **together** moves preselected objects in the layout as a group.
- **individually** moves preselected objects in the layout individually.
- **ignore** specifies that constraints have no effect on objects in the layout when they are moved.

Alignment Form

**Selection Mode** allows you to align components that are preselected or postselected.

- **Set reference to align preselected objects** works on the preselected set of components in the design.
- **Set reference and select objects to be aligned** allows you to set the options on the Alignment form and then selects the components to be aligned.

**Alignment Direction** defines the alignment direction.

- **horizontal** aligns objects in the X direction.
- **vertical** aligns objects in the Y direction.

**Align Using** indicates how objects are aligned as per the Selection Mode. For preselected objects you can use the following options.

- **Component BBox** aligns the *Reference Point* of the component’s bounding box
- **Component Origin** aligns the origin of the component.
- **Layer BBox** aligns the *Reference Point* of the specified Layer.

For Post selection use the following available options. Notice that the *Reference Point* options do not pertain to post selection.

- **Component Center** aligns the centers of components.
- **Component Edge** aligns the left, right, top, or bottom edges of the component.
- **Component Origin** aligns the origins of the objects.
- **Layer Center** determines the bounding box of all of the shapes on the current layer and takes the center of that bounding box as the reference point. Abutted devices are treated as a single component when determining the layer center.
**Layer Edge** determines the bounding box of all of the shapes on the current layer and then uses either the left, right, top, or bottom edge (depending on which edge is closest to the point you clicked) of the bounding box as the reference point. Abutted devices are treated as a single component when determining the layer edge.

**Layer** is a cyclic field showing the layers defined in your technology file. Use this field to specify the layer to use for aligning by layer edge, center, or BBox.

**Reference Point** is used with preselection mode options *Component BBox* and *Layer BBox*. Specify *upperLeft*, *centerLeft*, *lowerLeft*, *upperCenter*, *centerCenter*, *lowerCenter*, *upperRight*, *centerRight*, or *lowerRight*.

**Apply Spacings** specifies the spacing between the aligned components.

- **None** does not put any space between the components. This option can cause errors in your design.

- **Component Pitch** is used only in preselect mode. The distance (in user units) for center to center spacing between aligned shapes. You can set this value to a negative number to overlap instance.

- **Component Space** specifies the distance (in user units) to maintain between aligned shapes. Set this value to a negative number to overlap instances (and abut them automatically if *Auto Abutment* is switched on in the Layout XL Options form).

- **Pitches** is the value when using the *Component Pitch* option.

- **Spacing** is the value when using the *Component Space* command.

**Sort Components By** allows you to change how the selected objects are sorted in relationship to the reference object. This option only works with preselected objects.

- **Align Direction Order** sorts the objects by the alignment direction order.

- **Horizontal Order** sorts the objects aligned horizontally.

- **Vertical Order** sorts the objects aligned vertically.

- **Order Selected** sorts the objects in the order that they were selected.

- **Instance Name** sorts the objects aligned by instance name.

- **Net Name** sorts the objects aligned by net name.

**Reverse Sort** allows you to reverse the sorted objects.

**Set New Reference** specifies the reference for the selected components to be aligned to.
Using the Placer

This chapter shows you how to use the placer to automatically place transistors, devices, and cells in both block and cell designs.

This chapter discusses the following topics.

- Setting Up Virtuoso XL for Placement on page 217
- Placement Constraints on page 221
- Placement Parameters and Component Types on page 227
- Pin Placement on page 230
- Partitioning the Design on page 237
- Setting Placement Planning on page 241
- Running the Placer on page 265
- Showing Congestions on page 270

For information on Virtuoso XL placer forms, see “Online Forms” on page 271.

The placer is a generic placer that can be used for a variety of placement styles. You control placement using topological and geometric constraints.

You begin placement by generating layout connectivity and circuit components with the Virtuoso® XL Layout Editor (Virtuoso XL) layout generation commands. The connectivity source can be either a Virtuoso schematic composer design or a CDL netlist.

Main Features

- Re-entrant automated batch placement capability within the Virtuoso environment
- Connectivity- and constraint-driven placement to achieve overall shortest wiring length
- Automatic support for a wide variety of placement styles
Row-based MOS
Row-based standard cell
Area-based analog
Any combination of the above placement styles concurrently

- Manual capabilities to partition placement
- Accelerated engineering change orders
- Support for the following placement constraints
  - Distance
  - Alignment
  - Grouping
  - Symmetry
  - Fixed

- Automatic row generation for both device-level MOS and standard cells designs
- Automated pin placement positioning user interface, with support for
  - Assigning pins to any edge of placement boundary
  - Ordered and unordered pins
  - Specifying fixed pin positions
  - Placing groups of pins at any given spacing
  - Multiple pins of the same net

**Place Menu Command Summary**

The *Place* menu contains these commands.

- *Pin Placement*, which opens the *Pin Placement* form.
  
  Use this command to specify pin location constraints. This command can also be used to preplace pins independently from the placer.

- *Partitioning*, which opens the Partitioning form.
Use this command to cluster components together. Partitions are used in Placement Planning.

- **Placement Planning**, which opens the Placement Planning form.

Use this command to create/modify/edit rows.

- **Placer**, which opens the Auto Placer form.

Use this command to run the automatic placer.

- **Show Congestions and Hide Congestions**, which toggle the display of congested areas on the layout. For more information, see “Showing Congestions” on page 270.


Use this command to open the placement rules from an ASCII file or the technology file.


Use this command to create new placement rules.

**Other Commands Used with the Placer**

- **Design – Component Types**, which opens the Edit ComponentTypes form

  Use this command to update your library with information to guide the placement of components in rows, and to define parameters for MOS device chaining and folding.

**Placement Styles**

For row-based MOS transistor-level placement:

- Diffusion sharing is optimized through both automatic and interactive (with the Pick from Schematic command) transistor chaining during layout generation. Maximizing diffusion sharing minimizes the diffusion gaps in the generated layout.

For standard cell placement:

- You can place custom and standard cells in single or multiple rows. Rows can be either horizontal or vertical.
There are two modes of row definition: user-defined (manual) and component-assisted.

- In user-defined mode, you define the geometry of a series of rows and specify how to align and orient components in the rows.

- In component-assisted mode, the application defines row geometries for you based on area utilization goals and other design goals you specify. This allows the user to adjust various aspects of the design to explore different scenarios.

**Note:** Row-based is the most structured of all the placement styles. You define specific component types to only be placed with these rows. The devices are constrained within the row to certain orientations and alignments.

Area-based is the least restrictive of the placement styles. Area placement does not require component types or other constraints. The placer places devices within the placement boundary and reduces the overall wire length between the devices. At the same time, individual constraints that have been placed upon the devices are honored.

With area-based placements the placer also has the freedom to place devices anywhere within the placement boundary unless devices have been locked down outside of the placement boundary. Without individual constraints on devices or groups of devices, the placer's only objective is to reduce overall wire length and to achieve a balanced placement. Devices will tend to spread out using all the area allowed for placement.

The placement task for row-based designs typically involves these steps. For more information, see “Setting Placement Planning” on page 241.

1. Create component types to use in assigning devices to rows.

2. For MOS device-level designs, set the parameters for device chaining and folding.

   For more information, see “MOS Transistor Chaining and Folding Parameters” on page 188.
3. (Optional) Generate layout data from the schematic for critical components using the 
_Pick from Schematic_ command. Make a preliminary placement of the critical 
components using the object editing commands. If critical components are hand-placed 
in their final positions, they can be locked in place during automatic placement to 
preserve the preplacement.

4. Generate layout and a preliminary placement for the remainder of the design using _Gen 
From Source_ if you did not perform Step 3; otherwise use _Update Components and 
Nets_.

5. Define rows for placement.
   You can use component assisted mode only, a combination of assisted and user-defined, 
or just user-defined.

6. Assign component types to the appropriate rows.

7. Align objects to their respective rows.
   Part of defining rows is to check that the component type, alignment, and orientation are 
set properly.

8. Specify the orientation of components with rows.

9. (Optional) Set confinement, grouping, and other constraints.

10. Place and constrain pins using the _Place – Pin Placement_ command.

11. Run the automatic placer with the _Place – Placer_ command to generate a placement.

12. (Optional) Update constraints and run _Place – Placer_ again.

13. Repeat Step 12 as needed to further refine the placement.

**Setting Up Virtuoso XL for Placement**

This section describes some variables you might want to set and some design style decisions 
you should make before you begin working with the placer.

**Identifying the Placement Translation Rules**

The placer and the Virtuoso custom router both use a rules file to define how data should be 
translated between the Cadence® Design Framework II (DFII) environment and the 
placement and routing environment. In general, the router requires more elaborate translation 
rules than the placer because the router needs to know more about the design and use of 
routing resources). Rules you create for the router should work unmodified with the placer.
Tip

Cadence recommends that you have a separate translation rules file for placement purposes to minimize the amount of data to be translated. Additional data that is not required by the placer will adversely affect the placer’s performance, especially with mixed design styles.

Note: For CMOS device-level designs, diffusion layers must be translated as routing layers in order for the placer to chain devices.

Placement translation rules can be stored in the technology file or in a separate ASCII file.

rulesFile

The name of the rules file. If you do not set this variable, the translator uses the rules in the technology file. If the rules are not present in the technology file a warning message appears indicating that you will have to provide a rules file.

Setting Cadence Design Framework II Environment Variables

To permit the placer to properly handle permutable pins, type the following in a shell window.

    setenv CDS_Netlisting_Mode Analog

or

    setenv CDS_Netlisting_Mode Compatibility

Note: Do not set the netlisting mode to Compatibility if you intend to use chaining and folding for your devices.

If you do not have permutable pins in your design, and make no other use of CDF data, you can get faster netlisting performance by typing the following.

    setenv CDS_Netlisting_Mode Digital

Important

Several placement environment variables have equivalent variables to control the Virtuoso custom router. Placer variables always override their router equivalents during placement.
Setting MOS Chaining and Folding Parameters

You must set the following .cdsenv parameters to control MOS device folding and chaining.

- lxDeltaWidth
- lxWidthTolerance
- lxAllowPseudoParallelNets
- lxStackPartitionParameters

Defining the Boundary Layer (Placement Region)

The boundary shape defines the placement region. You specify the name of the boundary layer-purpose pair at several points in the placement and routing design flow. It is important that you identify this layer-purpose pair consistently.

You identify the boundary layer in the boundary translation rules for the Virtuoso custom router, and you create the boundary shape either with the layout generation commands or by drawing a shape. When you draw the boundary layer shape, or when you define it during layout generation, be sure to use the same layer-purpose pair that you specified in the translation rules.

⚠️ Important

If the iccLayers are not defined in the technology file you will receive the following error messages in the Placement Status window:

*ERROR* No boundary layers defined in placer rules, placement cannot continue.

The boundary layer shape must be a single rectangle or polygon. If the boundary layer consists of more than one shape, or a nonorthogonal shape, the translator exports the enclosing bounding box as the boundary.

Note: The default layer-purpose pair in the translation Rules Editor is prBoundary drawing. The default layer-purpose pair during Virtuoso XL layout generation is also prBoundary drawing.

Abutting Standard Cells

Standard cells are typically defined with a component class type of STDCELL. During automatic placement, standard cells are treated somewhat differently than regular devices. Because of this, there are up-front design considerations for proper standard cell abutment.
To achieve optimal placements quickly, standard cells are considered to be correct by construction. That is, all standard cells are designed to be abutted without creating design rule errors between each standard cell. The placer does not spend countless cycles on design rule checking between internal objects of one standard cell to another, just between the boundaries of adjacent standard cells. Using this method, the placer can use the boundary of each cell as the abutment edge for standard cells.

**Note:** Full design rule checking does occur between STDCELL component classes and all other devices.

The placer determines the cell boundary for each standard cell by the following precedence:

1. The list of layer-purpose pairs defined in the `vcpCellBoundaryLPPs` environment variable.
2. The `(prboundary drawing)` layer-purpose pair.
3. The `(prboundary boundary)` layer-purpose pair.
4. The `(instance drawing)` layer-purpose pair.

If none of the above layer-purpose pairs exists in the cell, a boundary is derived equal in size to all of the objects within the cell. Due to the nature of standard cell design and the desire to share/overlap objects between adjacent standard cells, defining a boundary is the preferred method to ensure proper cell abutment.

**Using Automatic Abutment During Placement**

If switched on in Virtuoso XL, automatic abutment is performed during layout generation for assisted MOS. To use automatic abutment,

- Your components (including your MOS device parameterized cells) need to be set up correctly with abutment properties.

  For an example of such a parameterized cell, see the spcnmos and spcppmos devices in the sample parameterized cells library. For more information, see the [Sample Parameterized Cells Installation and Reference](#).

- Automatic abutment needs to be switched on in the Layout XL Options form

For more information, see “Setting Up Cells for Abutment” on page 73.
Placement Constraints

You can create geometric constraints using the constraint manager or the constraint manager SKILL functions to constrain devices or pins. You can also create constraints through the Pin Placement form to constrain pins.

- Geometric Constraints

  Geometric constraints are used to control the placement of objects. There are various types of geometric constraints that can be created through the constraint manager.

  - For information about the types of constraints and how they are supported by Virtuoso XL and the placer, see “Constraint Manager Geometric Constraints” on page 221.

  - For information about setting constraints through the constraint manager, see the *Virtuoso Constraint Manager User Guide*.

  - For information about setting constraints using the constraint manager SKILL functions, refer to the *Custom Layout SKILL Functions Reference*.

- Pin Placement Constraints

  Creating pin constraints through the Pin Placement form lets you place different types of constraints on multiple pins at one time and automatically moves the pins to their assigned locations. Conversely, when creating pin constraints through the constraint manager, different types of pin constraints must be created separately and placement is not automatic.

  For more information about pin constraints, see “Pin Placement Constraints” on page 226.

Some of the procedures in this chapter suggest that you set certain types of constraints at certain points in the design flow, but generally this is not a strict requirement. You can constrain any device in the layout at any time, except while the placer is running. For example, you can use the placer iteratively, setting increasingly strict constraints as you refine the placement.

Constraint Manager Geometric Constraints

You can create the following constraints in the constraint manager for placement of objects

- Distance
- Alignment
Grouping
Symmetry
Fixed

Caution

The placer does not recognize constraint weights for alignment, fixed, symmetry, and distance constraints. Only grouping constraint weights with a fence is recognized by the placer.

Distance Constraints

The placer honors distance constraints between two devices. When using the nearest edge option, two devices or more can be constrained. Devices can be constrained in the X or Y direction but not both. In addition, the placer aligns the devices in the orthogonal direction. The placer interprets the distance constraint between a pair of objects.

Table 8-1 Distance Constraint Handling in Virtuoso XL and the Placer

<table>
<thead>
<tr>
<th>Distance Constraint Option</th>
<th>Virtuoso XL</th>
<th>Placer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max X</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Min X</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Max Y</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Min Y</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Reference X</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Reference Y</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Support for more than two objects</td>
<td>Between any pair in the set</td>
<td>Two or more objects are supported for Nearest Edge reference.</td>
</tr>
</tbody>
</table>
Alignment Constraints

The placer supports all alignment references on any layer or the instance bounding box. The first object is the alignment reference object and the rest of the objects are aligned to that reference. Relative orientation is also supported.

### Table 8-2 Alignment Constraint Handling in Virtuoso XL and the Placer for Two or More Objects

<table>
<thead>
<tr>
<th>Alignment Constraint Option</th>
<th>Virtuoso XL</th>
<th>Placer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative Orientation</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Align</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Reference Layer or Bounding Box</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ordering</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Grouping Constraints

A grouping constraint restricts a group of components to a rectilinear region (“fence”). A fence can optionally exclude all other, or specified, components.

The placer moves instances as necessary to implement confinement constraints even if they are not properly positioned in the preplaced view.

### Table 8-3 Grouping Constraint Handling in Virtuoso XL and the Placer

<table>
<thead>
<tr>
<th>Grouping Constraint Option</th>
<th>Virtuoso XL</th>
<th>Placer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preserve Relative Position</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Fence</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Exclude</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Note:** A grouping constraint with a fence that has a weight of 255 is translated as a hard fence. Any other weight between 1 and 254 is translated as a soft fence.
**Firm Grouping Constraints**

If you are creating constraints in the schematic view and you want to preserve the relative position of components in the layout view, you can use the *Preserve Relative Position* option to create a firm grouping constraint (also called a firm group).

To create a firm group,

➤ Use the constraint manager to create a grouping constraint and choose *Preserve Relative Position*.

**Soft Grouping Constraints**

A soft grouping constraint keeps a group of components close together within the cell boundary, but it does not constrain the group.

The placer moves instances as necessary to implement soft grouping constraints even if they are not properly positioned in the preplaced view.

To create a soft grouping constraint,

➤ Use the constraint manager to create a grouping constraint with the cell boundary as the fence.
Symmetry Constraints

The placer implies an order between the pairs involved in the constraint. For example, if Q1 and Q2 are constrained along their horizontal line, then the constraint implies that Q1 is below the line and Q2 is above.

The placer has the ability of defining a line of symmetry, self-symmetries, and for a given pair (pairwise symmetries). The symmetry axis can be floating as well as fixed.

Table 8-4  Symmetry Constraint Handling in Virtuoso XL and the Placer

<table>
<thead>
<tr>
<th>Symmetry Constraint Option</th>
<th>Virtuoso XL</th>
<th>Placer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating line</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Fixed line</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Horizontal/vertical direction</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Note: The placer interprets a symmetry constraint on a pair of objects such that the first object must be placed below a horizontal line of symmetry or to the left of a vertical line of symmetry. The other object would be placed on the opposite side of the line of symmetry.

Self-symmetry Only on fixed axis Yes, bounding box center

Fixed Constraints

You can define a fixed constraint with an allowable set of orientations on one or more named components to a required x and y location. The User Movable option in the Create Fixed Constraint form can affect the description of pin position constraints. Components can also
be temporarily locked to a fixed location and orientation through the *Edit – Other – Lock Selected* command.

**Table 8-5  Fixed Constraint Handling in Virtuoso XL and the Placer**

<table>
<thead>
<tr>
<th>Fixed Constraint Option</th>
<th>Virtuoso XL</th>
<th>Placer</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Y</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Orientation</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Reference</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Edit-Other-Lock Selected</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Pin Placement Constraints**

There are three methods of setting constraints on pins. You can use the Pin Placement form, the constraint manager, or the constraint manager SKILL functions.

**Pin Constraints Created Using the Pin Placement Form**

When creating constraints through the Pin Placement form, three categories of constraints are used:

- **Alignment**
- **Distance**
- **Fixed**

You can assign pins to a fixed location or to a particular edge and optionally ordered along each edge according to your specification. You can also specify fixed spacing for a group of pins ordered and aligned to an edge.

The Pin Placement form lets you apply all three types of constraints; distance, alignment and fixed on multiple pins at the same time. Once the *Apply* button is pressed, the pins are automatically moved to the constraint position.

For information about the Pin Placement form, see “Pin Placement” on page 230.
Pin Constraints Created in the Constraint Manager

The placer honors distance, alignment, grouping, symmetry, and fixed pin constraints created through the constraint manager or constraint manager SKILL functions. For information about these types of constraints and how they are supported by Virtuoso XL and the placer, see “Constraint Manager Geometric Constraints” on page 221.

When more than one type of constraint is required, each constraint must be created separately. For example, if you want to constrain pins to an edge of a bounding box with a specified pitch, an alignment and a distance constraint must be created. Pins constrained through the constraint manager are moved to their assigned positions in the layout view when the placer is run.

Important

Using mixed methods to create constraints is not recommended. To avoid problems resulting in conflicting and redundant constraints, the creation of pin constraints may be performed in either of two ways:

- Use the Pin Placement form exclusively
- or
- Use the constraint manager and constraint manager SKILL functions exclusively

For information about setting constraints using the constraint manager or the constraint manager SKILL functions, see the Virtuoso Constraint Manager User Guide or the Custom Layout SKILL Functions Reference.

Constraint Limitations

- The placer does not support Virtuoso constraint manager net-based constraints.
- Constraints must be entered at the top level schematic or layout.
- The constraint manager and the Pin Placement form, if used at the same time, can create conflicting constraints.

Placement Parameters and Component Types

You control how components are assigned to rows, and their chaining and folding parameters, by defining component types.
You set the component type using the Design – Component Types command. For more information, see “Component Types” on page 188 and “Defining a Component Type” on page 190.

**Caution**

_If you edit component types created in IC 4.4.6, using an editor in IC 5.0.0 or later, the data is saved in a new format that is incompatible with IC 4.4.6._

There are two reasons you need to define component types.

- To identify PMOS and NMOS devices to enable chaining and folding
- To identify groups of devices for row-based placements

**Important**

You must assign component types for devices that are to be placed within rows. Currently there is no need to define component types for devices that are placed by area.

Component types are part of the library, cell, or reference library information for cells in which components are placed. You can define component types at the library level, if all cells in the library share the same types, or you can define component types for the individual cells that will contain the placed components. This lets you define a set of types common to all designs in a library and override these for a selected set of designs.

Devices are placed within the place and route boundary regardless of their component type unless

- The devices have been locked in place
- The _Place Selected Objects Only_ option is specified in the Auto Placer form

**Important**

If the placer does not place a component in the rows, verify that the `1xComponentType` parameter has been assigned to the component and that the row definition allows for that particular component type to be placed.
MOS Transistor Chaining and Folding Parameters

MOS transistors that are candidates for chaining and folding need to have the \texttt{lxComponentType} parameter set with certain values. To set the \texttt{lxComponentType} parameter, use the Edit Component Types form.

For more information, see “MOS Transistor Chaining and Folding Parameters” on page 188.

Defining Standard Cell Substrate Contacts

A standard cell substrate contact is neither a contact nor a via; it is a standard cell with a component type of \texttt{STDSUBCONT} and two properties defined - \texttt{lxVcpSubContMaxSpacing} and \texttt{lxVcpSubContMinSpacing} - which define the maximum and minimum separation between two adjacent standard cell substrate contacts.

To define and place a standard cell substrate contact,

1. Copy a filler cell to use as the basis for the standard cell substrate contact.
2. Add substrate and well contacts in the cell. Whether these are \texttt{oaVia}, \texttt{via} instance, or polygons is immaterial.
3. Add the \texttt{lxVcpSubContMaxSpacing} and \texttt{lxVcpSubContMinSpacing} properties to the cell.
   
   a. From the CIW, choose \textit{Tools – CDF – Edit}.
      
      The Edit Component CDF form is displayed.
   
   b. Specify the library and cell names and click \textit{Apply}.
      
      The form is populated with the options you can set.
   
   c. In the \textit{Component Parameters} section, click \textit{Add}.
      
      The Add CDF Parameter form is displayed.
   
   d. Add the properties and accept the changes.
4. In the Edit Component Types form, define the cell as \texttt{STDSUBCONT}.
5. Run theplacer with the \textit{Insert Standard Cell Substrate Contacts} option switched on.

The placer fills any gaps between placed cells with the specified standard cell substrate contacts.
Pin Placement

Use the *Place – Pin Placement* command to

- Place pins in fixed positions on the boundary. For more information, see “Placing a Pin on a Boundary Edge” on page 231 and “Placing Ordered Pins on a Boundary Edge” on page 232.

- Place pins in fixed position that are not on the boundary. For more information, see “Placing a Pin in a Fixed Position not on the Boundary” on page 233.

- Place pins directly on lower-level instance terminals on the same layer. For more information, see “Placing a Pin on a Lower-level Instance Terminal” on page 233.

- Place pin groups with any given pin pitch spacing. For more information, see “Assigning Spacing Between Pins” on page 234.

- Create vertical and horizontal rails. For more information, see “Converting a Pin into a Rail” on page 234.

- Load the pin constraints from a template file. For more information, see “Loading a Template File” on page 236.

- Report pins and their assigned locations; expand and collapse iterated bus pins \((Q^{7:0} = Q^7, Q^6, Q^5, \text{ and so on.})\); and handle multiple pins on the same net, gnd.1 gnd.2 gnd.3.

You can use the *Place – Pin Placement* command only if there is a place and route boundary present in the layout view. To generate a boundary, use the *Design – Gen From Source* command.

The Pin Placement form lists all of the pins currently in the design. When you click *Apply*, all of the pins – other than those that have been confined to a fence – are moved to the specified locations. Pins that are not assigned to an edge or a specific location are placed anywhere on the place and route boundary. When you click *Defaults*, all of the pins are returned to their unplaced positions below the design boundary.

⚠️ **Important**

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

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

1. From the layout window menu bar, choose Place – Pin Placement.

   The Pin Placement form is displayed.

2. Choose Unplaced from the Display cyclic field and select the pin from the Pin Name list box.

   You can expand iterated bus pins (for example, A<7:0>) using the Expand button. This expands the bus into individual pins, which you can place individually with differing constraints.
Tip

If you click Link to Layout and select the pin you require in the layout window, the pin name is highlighted automatically in the list box.

3. Do one of the following.

- From the Edge cyclic field, choose either Left, Top, Right, Bottom, Any, or As is.
  
  **Note:** If the boundary is a polygon, the edges are listed as Edge 0, Edge 1, Edge 2, and so on. The As is setting leaves the edge settings of pins intact when changing other pin properties.

- Click Select edge from layout and select the edge you want in the layout window. The Edge cyclic field displays the name of the edge you selected.

4. Choose Fix after Placing from the Location cyclic field.

  *Fix after Placing* places the pin at a location on the boundary and then fixes the pin at that location, meaning that it cannot be moved during placement. (This option changes to *Fix at Placed Location* if the selected pin is placed.)

  Pins that are User-movable after Placing (User-movable at Placed Location for placed pins) can be moved by the user to another location; pins that are Floating can be moved by the placer to another location before being fixed. The As is option specifies that the status of the pin is not to be changed after placing.

5. Click Apply.

  The pin is aligned to the specified boundary edge in the layout. The Location cyclic field changes to show the coordinates of the center of the pin.

6. Repeat steps 3 through 5 for each pin you want to place on the boundary.

### Placing Ordered Pins on a Boundary Edge

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

1. From the layout window menu bar, choose Place – Move.

   The Pin Placement form is displayed.

2. Select one or more pins from the Pin Name list box.

3. Choose the edge you want from the Edge cyclic field.

4. Type a number in the Order field.
You can change the order of the pins by selecting a pin in the list and clicking *Move pins*. Use the *Swap order* option to swap the position of two selected pins.

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

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

---

**Placing a Pin in a Fixed Position not on the Boundary**

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

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

   The *Pin Placement* form is displayed.
4. Select the pin from the *Pin Name* list box.
5. Choose *Fix after Placing* from the *Location* cyclic field.
6. Click *Apply*.

   The pin is fixed at its current location and is not aligned with any edge.

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

---

**Placing a Pin on a Lower-level Instance Terminal**

To place a pin directly onto its corresponding instance terminal at a level below in the hierarchy,

1. From the layout window menu bar, choose *Place – Pin Placement*.

   The *Pin Placement* form is displayed.
2. Select the pin from the *Pin Name* list box.
3. Choose *None* from the *Edge* cyclic field.
4. Choose *Floating* from the *Location* cyclic field.
5. Click Apply.
   
   The pin you selected is placed over its corresponding instance terminal in the layout window.

Assigning Spacing Between Pins

To set the exact distance between two or more ordered pins or between the pins of a bus pin,

1. From the layout window menu bar, choose Place – Pin Placement.
   
   The Pin Placement form is displayed.

2. Filter the list of pin names as required using the Display cyclic field.

3. Select the pins from the Pin Name list box.

4. Type the center-to-center spacing for the pins in the Pitch field.
   
   **Note:** You can set a Pitch value only for ordered pins. The Pitch field becomes active only when you select a set of ordered and placed pins in the list box, or you select a set of unplaced pins and enter a value in the Order field.

5. Press the Tab key to activate the Apply button.
   
   The pitch is valid if the pins are on the same boundary edge.

6. Click Apply.
   
   The pins move to the positions specified.

Notes

- Pitch values are enforced only between adjacent pins on an edge, even if the spacing is set between two non-adjacent pins.

- If you specify a spacing between a pin and an iterated pin, the software applies the spacing between the last pin of the iterated pin and the individual pin.

- If the specified pitch cannot be satisfied, you see a warning in the CIW and the pins are not moved.

Converting a Pin into a Rail

To convert a pin into a rail,
1. From the layout window menu bar, choose Place – Pin Placement.
   The Pin Placement form is displayed.

2. Choose Unplaced from the Display cyclic field.

3. Select a pin from the Pin Name list box.

4. Choose an edge from the Edge cyclic field.

5. Click Apply.
   The pin is placed on the boundary.

6. Select the pin in the layout do one of the following.
   - Click HRail to convert it into a horizontal rail.
   - Click VRail to convert it into a vertical rail.

   When a pin is changed into a rail, it becomes aligned with two edges of the boundary. For a horizontal rail, the alignment is to the left and right edges. For a vertical rail, the alignment is to the bottom and top edges. The rail stretches with the boundary in a constrained editing mode.

   **Important**
   The Pin Placement form does not distinguish layer information when the pins are placed on the boundary edges. For example, you cannot convert a metal2 pin to a rail if that pin occupies the same location as a metal1 pin.

7. (Optional) To view the alignment constraints of the rail,
   a. From the CIW choose Tools – Constraint Manager.
   b. Change the Select Entity to Geometric Constraints.
   c. Click Alignment. The pin placement is displayed. The pin constraints created from the Pin Placement form can be viewed but not edited from the Constraint Manager form.

**Converting a Rail into a Pin**

To convert a rail back into a pin,

1. From the layout window menu bar, choose Place – Pin Placement.
   The Pin Placement form is displayed.
2. In the layout window, select the rail you want to change.

3. In the Pin Placement form, click unHRail or unVRail as required.
   The rail is converted back into a pin and returned to its original size and edge, regardless of the railing direction.

**Loading a Template File**

To load information from a template file into the Pin Placement form,

1. From the layout window menu bar, choose Place – Pin Placement.
   The Pin Placement form is displayed.

2. Click Template file.
   The Load Template File form is displayed.

3. In the left list box, click on the directories to descend into your file hierarchy until you reach the name of the template file to use.
   The left and right arrows at the right side of the form let you go up and down in the file hierarchy.

4. In the right list box, click on the name of the template file. The name is entered in the Name field.

5. In the Load sections area, turn on the sections of the template file you want to load (I/O Pins).

6. In the Existing data area, do one of the following.
   - To merge the information from the template with the placement data already in virtual memory, turn on Merge.
   - To replace the placement data in virtual memory with different placement data in a template file, turn on Replace.

7. Click OK.

**Saving Pin Placement to a Template File**

To save the pin location and constraint information to a template file, use the Design – Save To Template command.

**Note:** Template files from the 4.4.6 release are loaded correctly into the 5.1.41 release.
You can save the following information from the pin placement form.

- **Pin location**
  - Floating
  - Fixed at placed location
  - As Is

- **Constraints**
  - The boundary edge on which a pin is placed
  - The order in which a pin has been placed on an edge boundary
  - If the pin is fixed to a certain location
  - The required distance between a pin and its neighboring pins
  - Pins that are either vertical or horizontal rails

### Partitioning the Design

To plan the layout more efficiently, cluster components together and assign them to specific areas of the layout. Such areas are called partitions, and they let you monitor the placement more closely.

Use the *Place – Partitioning* command to assign components to a partition. A partition defines an area in the design to which you can confine components.

- All placements have to be made inside the design boundary
- The design boundary is the default partition:
  - If no partition exists, all the components belong to the design boundary
  - If a component is not assigned to any partition, it belongs to the design boundary
  - If no partition is selected, the design boundary becomes the target partition
  - The keyword “boundary” is a reserved name that cannot be used as a partition name
- A component should not be placed in a partition to which it is not assigned
- A partition can be created without having a shape giving you the ability to partition a netlist or schematic
Partitions can be made in terms of the connectivity source name space or the layout name space.

All members of a one-to-many, many-to-one, or many-to-many group must be assigned to the same partition.

Important

To view engineering changes in the Partitioning form, you must close and re-open the Partitioning form.

Creating a Partition

1. Create a shape to be attached to the partition.

2. In the Layer Selection Window, choose Edit – Set Valid Layers and add the softFence drawing layer-purpose pair to the list of valid layers.

3. From the layout window, choose Create – Rectangle and draw a rectangle on the softFence layer around the devices that you want to keep together during placement.

4. Choose Place – Partitioning.
The **Partitioning** form is displayed.

5. Turn on *Partition*. The *Link to Layout* button is enabled.
6. Click *Link to Layout*.

7. Click anywhere in the layout cellview to link the Partitioning form to the layout.
   
   In the CIW, a message appears telling you that the Partitioning Form is linked to the design.

8. To choose a new partition name, in the *Choose or type a Partition name* field, do one of the following:
   
   - Type the name of the partition you want to create.
   - Select an existing partition name from the drop-down list.

9. Click *Create Partition*.

   The new partition name with a question mark appears in the cyclic field. The question mark denotes that the partition does not have a shape attached to it.

10. Click *Attach Shape* and click to select the *softFence* rectangle.

    The rectangle is added to the partition.

11. In the *Target Partition* field, type the new partition name.

12. In the *Component Name* list, choose the devices to be added to the partition.

13. Click *Move*.

    The devices move into the partition. Notice that the components and the partition are highlighted.

**Loading Information from a Template File**

1. Click on the *Browse* button.

    The Open File form is displayed.

2. Do one of the following:

   a. If you know the name of the template file, type the name in the *Template* text field.

   b. Click on the directories in the left list box to descend into your file hierarchy until you reach the name of the template file to use. Click on the name of the template file in the right list box to enter it in the *Name* field.

    The left and right arrows at the right side of the form let you go up and down in the file hierarchy.
3. Click *OK*.

**Saving Partitions to a Template File**

You can save the partition information from the cellview to a template file using the *Design – Save To Template* command. Below is an example of the partition information that is saved to the template file.

```plaintext
partition_section(
    connectivity "VCPLIB DFF schematic"
    (partition partition1
        (shape rectangle ((5.300000 3.050000) (13.500000 10.600000)))
        (cell /P4)
        (cell /P6)
        (cell /P7)
        (cell /P2)
        (cell /P12)
    )
)
```

**Setting Placement Planning**

Placement planning lets you define where in the layout you want to place transistors and other design elements.

Before you can set the placement planning, you or the library developer must first define the component types for placement. Components that have types assigned to rows can only be placed in these rows. For more information, see “Defining a Component Type” on page 190.

There are two row definition modes for the Placement Planning form: Assisted and User-Defined.

- The assisted mode automates typical styles of MOS and standard cell placement
- The user-defined mode offers greater flexibility but less automation and supports both transistor- and cell-level placement

You set the mode in the *Style* field at the top of the Placement Planning form.

**Assisted CMOS Placement**

The Assisted CMOS row definition style helps you define rows that fit your area budget. You can change the independent design variables that constrain the row shapes and observe their effect on area utilization and folded transistor widths.
Limitations of Assisted CMOS Placement

The Assisted CMOS placement style restricts the placement to the most typical arrangements. There are several limitations that are not present in the user-defined row definition mode.

- All rows must be horizontal or vertical
- All rows have the same width and span the entire placement region
  
  **Note:** Row width can vary if the region is polygonal.
- Each row contains either NMOS or PMOS devices but not both and not any other component types
- The number of NMOS and PMOS rows is the same
- There is a regular arrangement of alternating N and P rows that follows one of these patterns:
  
  - N-P
  - P-N
  - N-P-P-N
  - P-N-N-P
- All rows have the same device alignment relative to the devices in adjacent rows: inside, outside, or center
- Device rotation depends on the row and pattern
- The rows can extend outside of the boundary/partition
  
  **Note:** The placer can place only within the boundary/partition

If any of these limitations is not acceptable, you can either

- Define the rows in the user-defined mode
- Define the rows as completely as possible using the component-assisted mode and then refine the definitions in the user-defined mode
- Use assisted mode on either standard cell or MOS, but not both at the same time.
About Calculated Values

For both NMOS and PMOS transistors, the system calculates the number of transistors and the minimum, maximum, and average transistor widths. To update the displayed values after altering variables, click *Calculate Estimates*.

The system also calculates the normalized percentage of intra-row and inter-row area utilization for the placement region. A value of 100% in the *Intra Row* field means that the entire width of the placement region is used by the devices. If there are too many devices to fit within the placement region, rows may overlap or extend outside the placement region. In this case, one or both of the *Region Utilized* values might be greater than 100%.

Planning Placement with the Assisted CMOS Style

To use the *Assisted CMOS* row definition style,

1. Choose *Place – Placement Planning*. 


The Placement Planning form is displayed.

2. In the **Partition** field, specify a partition name.

   You can select a partition by clicking on **Select From Layout** and then clicking in the layout window to select the partition.
3. Change the Style to Assisted CMOS.

4. Use the controls in the Regions tab to create a placement region within the partition.

The placement region should be contained within the partition. For example, you might restrict assisted placement to the lower part of the partition, leaving the upper part available for manual placement. Or, you might leave space around the periphery for routing.

- Type the dimensions of the drawing region in the Width and Height fields.
- Type the X and Y coordinates of the location where you want the row in the Origin fields.
  You can see the X and Y coordinates of the cursor position in the status line at the top of the layout window. If you do not type any coordinates, the new row is placed at the lower left corner of the design boundary.
- Specify the placement region manually in the layout window, select Draw and you are prompted to point at the first corner of the region’s rectangle.
Important

When targeting a partition for assisted row generation, the estimator uses the entire partition area in its calculation. You can optionally specify a region within the partition by using the Draw option. If a region is defined the estimator will use the region area instead of the partition area.

5. Click the Rows tab to update the row properties.

![Rows section of the Placement Planning form]

Note: If rows are already defined, change the Style to Manual User-Defined and the row information is displayed at the bottom of the Placement Planning form. Click on the row that you want to edit. The row properties are updated in the Row Name field.

6. Set the direction of the rows.

For more information on the other fields in the Rows section, see Placement Planning on page 280.
7. Click the *Rails* tab to define the rails.

- Choose a *Layer* and type values in the *Width* and *Net* fields for power and ground. The layers are valid layers that are not used as vias or contacts.
- In the *Rail To Device Spacing* field, specify the minimum distance between the power rail and PMOS and NMOS devices.
- In the *Pattern* cyclic field, choose a pattern for the alternating NMOS and PMOS rows.
- In the *Position* cyclic field, choose the position of the rail relative to the row (*Inside*, *Outside*, or *Center*).

**Caution**

*Do not add a u at the end of a value for rail width and spacing rules because the value will be converted to 0.000001. For example, 12u is converted to 0.012.*

To create non-rectangular regions,

1. Choose *Place – Partitioning* and create a partition for the components.
2. Choose *Place – Placement Planning*.
3. In the *Style* cyclic field, choose *Assisted CMOS*.
4. From the *Partition* cyclic field, choose a partition name.
5. In the *Regions* tab, click *Draw* and you are prompted to draw a region. Create the region inside of the partition.

6. Click *Calculate Estimates*.

   The rows are created and the row statistics are displayed at the bottom of the Placement Planning form.

7. Draw another separate region that does not overlap the previous region (area now contains rows) using the *Draw* option.

8. Click *Calculate Estimates*.

   The rows are created and the row statistics are displayed at the bottom of the Placement Planning form.

   **Note:** The final utilization must be calculated by hand using the row statistics from both regions.

9. Click the *Components* tab.
Caution

Do not add a u at the end of a value for spacing rules because the value will be converted to 0.000001. For example, 12u is converted to 0.012.

10. From the Align Components cyclic field, choose Inside, Outside, or Center.

Note: Alignment can be either top, bottom, or center for horizontal rows or right, left, or center for vertical rows.

11. In the Width Threshold field, type the minimum width and in the Diffusion Spacing field, type the required diffusion spacing for the PMOS and NMOS transistors within the row.

12. To place only the devices for the selected component below the prBoundary, click Regenerate All.

The Update Layout option at the bottom of the Placement Planning form updates the current layout with the components and deletes existing components. This option is used on a selective basis based on the selected partition and style.
13. Click *Calculate Estimates* to calculate the row count, area utilization, device count, and device widths.

![Diagram of Virtuoso XL Layout Editor User Guide](image)

**Caution**

*The estimator designs the rows so as to avoid any preplaced components even if they are a part of the estimator computation. You should move these components out of the placement region before invoking the estimator.*

The system regenerates folded transistors using the new values. The section at the bottom of the form is updated with the percentage of the region utilized and width information about the PMOS and NMOS components.

14. Evaluate the device counts against the need to generate equal numbers of P and N rows. Evaluate the area utilization based on routability considerations. If necessary, change the *Number of Rows* or *PMOS Width Threshold* or *NMOS Width Threshold* for folding.

15. Click *Calculate Estimates* to update the calculated estimates.

If necessary, redefine the placement region and begin again with Step 4.

**Note:** After you click *Calculate Estimates*, if you make extensive changes to device folding and chaining choices (either through automatic layout generation or interactive device folding), the estimates you calculate in step 13 may become invalid.
Using Assisted Standard Cell Mode

To use the Assisted Standard Cell row definition mode

1. Select Place – Placement Planning.

   The Placement Planning form is displayed.

2. Change the Style to Assisted Standard-Cell.

3. Click the Regions tab.

4. Do one of the following.

   a. Type the dimensions of the row(s) in the Width and Height fields and the X and Y coordinates of the location where you want the row in the Origin field.

      You can see the X and Y coordinates of the cursor position in the status line at the top of the layout window. If you do not type in any coordinates, the new row is placed at the lower left corner of the design boundary.

   b. To draw the placement region manually in the layout window, click Draw.

      The layout window prompts you to point at the first corner of the region’s rectangle.
5. Click the *Rows* tab.

6. In the *Row Name* field, type the names of the rows to be designed and set the direction of the rows.

7. Switch on *Use Filler Cells* to insert filler cells between the other cells in the row. Choose the cells by doing one of the following.

   - Type names into the text field to select filler cells you defined in the Edit Component Types form.
Click Choose Types and choose the names of the filler cells from the list that appears in the Choose Component Types form. Use the Search field to filter the selection.

You must also specify a type when using the Use Substrate Contacts option. Use STDSUBCONT as the Component Class.

8. Click the Rails tab.

Choose a Layer and type values in the Width and Net fields for power and ground. The layers are valid layers that are not used as vias or contacts.
In the *Pattern* cyclic field, choose a pattern for the alternating power and ground rails.

- In the *Rail to Rail Spacing* field, type a value for the spacing between power and ground rails for the same row.

9. Click *Calculate Estimates* to calculate the row count, area utilization, device count, and device widths.

The section at the bottom of the form is updated with the percentage of the region utilized.

If you choose to use the *Regenerate All* option all standard cells are regenerated once the *Update Layout* option is invoked. Or you can use the *Generate Only Missing Components* option and the missing components are created. Generation/regeneration only affects standard cells within the chosen partition.

**Note:** To modify the size of a rail, change the Rail information and then click *Calculate Estimates*. 
10. Evaluate the device counts against the need to generate equal numbers of PG to GP rows. Evaluate the area utilization based on routability considerations. If necessary, change the Width or Height fields or the Width Threshold for folding. Click Calculate Estimates to update the calculated estimates.

**Note:** After you click Calculate Estimates, if you make any changes to row definitions (either through automatic layout generation or interactive device folding), the estimates you calculate in step 10 are rendered invalid.

11. Click Apply to save your row definitions.

**Manual User-Defined Row-Based Placement**

The Manual User-Defined mode for row definition lets you define row boundaries flexibly. For example, you can draw rectangles to delimit individual rows. Use the manual mode for placements above the device level or for MOS device placement styles that are not accommodated in Assisted MOS mode.

The picture below shows an example of a placement region with four rows. Rows 1 and 2, the lower rows, span only half the placement region. Rows 3 and 4 span the entire placement region.

**Figure 8-2 Examples of Rows**

![Region](image)

**Defining Rows Manually**

To define rows or columns manually,

1. From the layout window, choose Place – Placement Planning.
The Placement Planning form is displayed.

2. Change the *Style* to *Manual User-Defined*.

3. Click *General* to define one or more rows.
   
   a. Type the name of the rows to be designed in the *Row Name* field.
   
   b. Set the direction of the rows.
   
   c. Type the X and Y coordinates of the location where you want the first row in the *Origin* field.
      
      You can see the X and Y coordinates of the cursor position in the status line at the top of the layout window. If you do not type in any coordinates, the new row is placed at the lower left corner of the design boundary.
      
   d. Type the dimensions of the rows in the *Width* and *Height* fields.
   
   e. Click *Draw* and draw the shadow of the row. The row is not created until you press the *Create New* button.
   
   f. In the *Spacing Between Rows* field, type the spacing you want between rows.
   
   g. Type in the number of rows you want in the *Number of Rows to Create* field.
      
      If *Number of Rows to Create* is greater than 1, then this will create an array of rows with the same devices and attributes at a fixed spacing between adjacent rows.

4. Click the *Types* tab.
   
   a. Click *Edit*. The Choose Component Types form is displayed.
   
   b. Select the component type and click *OK*. The selected component type appears in the *Component Type(s)* field.
   
   c. Specify a *Component Grid, Reference Point For Type, Reference Point For Row, Offset, and Allowed Orientations*.
   
   d. Click *Create New* and the component type is displayed in the *Types* field box.
   
   e. To change any of the component types properties select the component type in the *Types* field and change the properties on the form. Click *Update* and the component type properties are updated.

5. In the *Rails* section of the form,
a. Specify the Layer, Width, and Net name of the rail.

b. Change the Reference Point for Rail and Reference Point For Row to be Bottom, Top, or Center as needed.

c. Specify a Rail Offset from Row.

d. Click Add. The rail is added and displayed in the Rails field.

e. To change any of the rail properties, select the rail in the Rails field and change the properties on the form. Click Update to update the rail with the new properties.

Defining User-Programmed Styles

The Style File text field on the Placement Planning form lets you specify a SKILL file that can correspond to one or more styles.

The style definition should contain the following information;

- The style name.
- The required fields in the form.
  - You are allowed to program the style-specific parameter and statistics sections from the form. Other than label fields, the name of the field is used as the common reference for the field value. A hash table is passed to the callbacks with the extracted values. Field names should be unique to the form.
- Names of files that contain style-specific callbacks.

Creating the Style File

The file structure of a style-definition consists of a header and contents. The header is as follows

```latex
(1xPlcStyleHeader
 (version versionValue)
 (creator "name or tool")
 (timeLastChanged "YYYY:MM:DD HH:MM:SS")
 ...
 )
```

The function 1xPlcStyleHeader must be the first item in the file aside from comments. The version indicates if the style can be loaded. If the version is too old or too new, you must update the file or use a different file. The creator and timeLastChanged fields are for reporting purposes only.
The style definition function is as follows;

```
(lxPlcDefineStyle
 (name "nameString")
 (paramGUICreationCB "functionName")
 (statGUICreationCB "functionName")
 (estimatorCB "functionName")
 (updateLayoutCB "functionName")
 (loadFile "fileName"...)
 ...)
```

The function, lxPlcLoadStyle loads the style definition files. This function should verify the version from the header. If the version is correct the file is loaded. The loadFile function refers to one or more files containing the SKILL callbacks that can be loaded using SKILL load.

**Note:** You can use implicit load to load sections that are common to many styles

### Loading the Style File

When you select the *Calculate Estimates* button, the style file is loaded and then checks for the selected style along with

- Identifying the components based on the chosen partition
- Obtaining the region information from the form
- Clearing the region from any rows
- Setting all the necessary parameters
- The style-specific estimation function returns row specifications
  - If the estimation function returns any row specifications, these specifications are passed to a function that generates rows from specifications.

### Style File Example

The following is a standard cell example.

```
;First write a file, mySTDCell.stl, that declares the style as follows,

(lxPlcStyleHeader
 (version 1.0)
 (creator "customer A")
 (timeLastChanged "2001:1:24 13:00:00")
 )

(lxPlcDefineStyle
 (name "mySTDCellStyle")
```
When the system loads the mySTDCell.stl file, which will search for the file called mySTDCellFun.il and load it. The file looks like

```
(defun mySTDCellParamGUICreationCB (cvId x y other)
  (let (field1 field2...)
    (setq field1 (hiCreate???Field...))
    (setq field2 (hiCreate???Field...))
    ...
    (list
      (list
        (list field1 ?:? ?:? ?)
        (list field2 ?:? ?:? ?)
      ...
      )
    )
    nil
  )
)
```

The program can take the list of fields and add them to the form.

```
(defun mySTDCellStatGUICreationCB (cvId x y other)
  (let (field1 field2 ...)
    (setq field1 (hiCreate???Field...))
    (setq field2 (hiCreate???Field...))
    ...
    (list
      (list
        (list field1 ?:? :
        (list field2 ?:? :
      ...
      )
    )
    nil
  )
)
(dfun mySTDCellEstimationCB (cvId paramTable)
  ; The param table is a table of agreed-upon parameters
  ; and their corresponding values.
  ; gather information about the current partition, and its
  ; contents.
  ; generate/update the layout for the partition in
  ; the scratch cellview.
  ; select the components that correspond to the style.
  ; estimate the space needed for the components.
  ; estimate the rows needed for the components.
  ; return the list of row specifications.
)
```
Drawing Rows

You can define rows by drawing them one at a time in the layout window. Select Draw and define the placement region for assisted MOS or cells.

Redefining a Row

Rows are actual database objects you can move and stretch graphically. To graphically edit rows make sure that the row drawing layer-purpose pair is a valid layer and made selectable in the Layer Selection Window (LSW).

You can delete a row by changing the Style to Manual User-Defined mode. The rows are displayed at the bottom of the Placement Planning form. You can select a row and use the Delete Selected option to delete the selected rows, or select the row in the layout window and then delete it.

Setting Up Power and Ground Rails

You can define arrays of multiple rows in one step to create alternating rows with the same definitions. This arrangement facilitates the layout of power lines. Figure 8-3 on page 261 shows a portion of a buffer array with alternating NMOS and PMOS transistors in columns.
To define the placement style for these eight rows, you define two arrays of components. To define the PMOS rows, follow these steps:

1. Choose *Place – Placement Planning*.
3. Create the N row as the bottom row.
4. Create the P row above the N row.
5. Create another P row.
6. Create the N row which is on the top.
7. Set *Num Of Rows to Create* to 2.

The row setup is now complete.
For other arrangements of power lines between rows, you might, for example, need to define arrays of every fourth or eighth row in order to achieve the desired mirroring pattern.

**About Horizontal and Vertical Rows**

You can define both horizontal and vertical rows for placement. To do this, set the desired direction as you define each row or set of rows.

Figure 8-4 on page 262 shows how the *Width* and *Height* parameters should be set for horizontal and vertical rows. As this figure illustrates, the *Width* is measured along the direction of the rows.

**Figure 8-4 Setting Width and Height for Horizontal and Vertical Rows**

![Diagram](image)

1. Row direction is *Vertical*, *Width* is 10, *Height* is 4, *Spacing* is 2.
2. Row direction is *Horizontal*, *Width* is 10, *Height* is 4, *Spacing* is 2.

**About Device Orientation**

There is no implied rotation of devices in vertical rows. The *Orientation* value you specify in the Placement Planning form is always relative to the master cell coordinates. Figure 8-5 on page 263 shows the undesired result of placing the same device with orientation R0 (unrotated) in both a horizontal and vertical row.
About Device Abutment and Orientation

For MOS row placement, diffusion would be abutted along the row direction as shown in Figure 8-6 on page 263.

The permitted orientations in the example in Figure 8-6 on page 263 would depend on the orientation of the transistor in its master cell. Assuming that the polysilicon shape is vertical in the master cell for the transistor, for a horizontal row the reasonable orientations would be R0 (unrotated), R180 (rotated 180 degrees), MX (mirrored along the x axis) and MY (mirrored along the y axis). R90 would not be an advisable orientation.

For standard cells, the power and ground rails would run parallel to the row direction, as shown in Figure 8-7 on page 264.
Figure 8-7 Standard Cells with Power Rails Parallel to the Row Direction

About Space at the End of a Row and Row Compaction

The placer can abut MOS devices so that rails meet each other. However, when placing standard or other cells that are not set up for device abutment, the placer does not compact the rows to ensure that the rails abut.

The placer never extends rails to the row boundary. If the cells do not entirely fill the row, you may need to extend the rails or use filler cells.

Figure 8-8 on page 264 shows standard cells placed within a row. The cells do not abut each other and the power rails do not extend to the right side row boundary.

Figure 8-8 Row with Extra Space

Choose Component Types Form

From the Placement Planning form, in Manual User-Defined mode, the Edit button in the Types section opens the Choose Component Types form. Use this form to identify the component types to place in the currently active rows.
Below the *Search* button is a list of component types in the current design. Use the mouse to highlight the types you wish to include in the *Component Types* field of the Placement Style form, and then click *OK*.

![Choose Component Types](image)

*Search* scrolls the list to the component type name you type into the field. The name must match exactly. You can then highlight the name. This field is useful if you have a long list of types.

### Running the Placer

You launch the Virtuoso custom placer with the Auto Placer form. The placer runs in foreground, background, or Load Balancing Service (LBS) mode, reporting its status in the Virtuoso XL Placement Status window.

**Note:** While the placer is running on a cellview, the cellview is switched to read-only mode and you cannot perform edits on it but you can edit any other cellview. When the placer has finished, the cellview is switched back to edit mode.

After the placer finishes, the placed cellview appears (if you are updating the source view with the placed view, or if you are creating a new view and chose the *Open Window* option).

After placement, you can iteratively

- Lock the position of components whose placement is good.
Refine the constraints on other components to better guide the placer.

Rerun the placer as needed to improve the placement results.

**Prerequisites to Placement**

To prepare for placement

1. Set any necessary UNIX and DFII environment variables before launching the Cadence software.

   For more information on setting variables, see “Setting Up Virtuoso XL for Placement” on page 217.

2. If you only want to place some of the components, select the components to be placed in the cellview window and turn on *Place Selected Objects Only*.

**Running the Placer: Initial Placement**

To start the automatic placer and perform an initial placement of the selected components

1. Choose *Place – Placer*. 
The **Auto Placer** form is displayed.

2. Turn on *Global Placement*.

3. Set the other placement options on the left side of the form as needed.

   For information on these options, see “Auto Placer” on page 272.

4. To save the placed cellview with a different view name,
   a. Switch on the *Save As* option.
   b. Fill in the *Library*, *Cell*, and *View* fields. By default the view name is *layout.plc*.

   You can also change the library or cell name, or browse to choose the cellview name.

**Caution**

*Do not use the view name “placerScratch”. This name is reserved by the placer.*

5. Click *OK* or *Apply*.

**Note:** The *layout.plc* cellview is placed on top of the *layout* cellview.
If the iccLayers are not defined in the technology file or the rules file, you will receive the following error messages in the Placement Status window:

*ERROR* No boundary layers defined in placer rules, placement cannot continue.

Refer to the Virtuoso custom placement and routing preparation guide for more information in defining the iccLayers.

After the placer initializes, the Virtuoso XL Placement Status window appears. You can specify the level of detail that appears in the status window by setting the `vcpVerboseLevel` environmental variable.

---

**Important**

The text columns in the Virtuoso XL Placement Status form do not align if a variable font is used. You must use a fixed text font. The default text font is a fixed size text font. You can change the text font by using Options – User Preference from the CIW.

The placer moves pins that were not assigned to placement information inside the design boundary and aligns to any boundary edge.

You can use the Edit – Undo command to discard the placement results.
Stopping the Placer

To stop the placer before it has completed all placement passes,

1. Select *Place- Placer*.
2. Set the options in the Auto Placer form and click *OK*.

   The XL Placement Status form is displayed.
3. In the Virtuoso XL Placement Status form select *Process – Stop Placer*.

Running Load Balancing Service (LBS)

LBS lets the placer support load sharing capability. The procedure for running LBS is as follows.

1. Set either the Cadence default LBS queue or an LSF queue.
2. In the latter case, set the `LBS_BASE_SYSTEM` environmental variable to `LBS_LSF`.
3. Set the search path to include the path to any load sharing service scripts.

   Note: You set the path using the `vcpGlobalPathScript` environmental variable.
4. Activate the load sharing queue daemons.
5. Disk directories must have been exported to enable load sharing service to work across the network.
6. Password restrictions have been eliminated to enable batch automatic login from one machine to the other.

Troubleshooting Placement Results

This section discusses common placement problems and some possible solutions.

**Problem:** Components were placed with overlaps.

**Solution:** The boundary region or the rows are too small. Redraw the boundary or rows.

Refining Placement Results

To refine the placement results
Set grouping and fixed constraints by choosing Tools – Constraint Manager in the CIW.

Redefine the row boundaries.

For MOS transistor-level designs, modify the layout for one or more transistors with the Edit – Transistor Chaining and Transistor Folding commands.

Running the Placer: Detailed Placement

To perform a detailed placement after refining the constraints

1. Choose Place – Placer. The Auto Placer form is displayed.

2. Deselect Global Placement.

3. Choose Optimize Placement.

4. Continue as with initial placement, beginning with step 4.

Showing Congestions

After you have run placement, you can use the Place – Show Congestions command to highlight areas where design elements are so close together that routing might be a problem.

To show congested areas in your placed design

➤ Choose Place – Show Congestions.

The system surrounds congested areas with tiles on different highlight layers showing degrees of congestion.

The layer purpose pairs designFlow drawing through designFlow drawing9 are used to show congestion. Each layer purpose pair represents different regions of routing resource utilization.

For example, the designFlow drawing layer purpose pair represents utilization from 0 to 11.1111111%. The designFlow drawing1 layer purpose pair represents utilization from 11.111112 to 22.222222%, until layer designFlow drawing8 which represents utilization of 88.888888% to 100%. Layer purpose pair designFlow drawing9 represents utilization over 100%.

The legend bar appears displaying the percentage of utilization for each designFlow layer.

The command name changes to Hide Congestions.
To remove the highlighting around congested areas in your placed design

➤ Choose Place – Hide Congestions.

The highlighting surrounding the congested areas disappears.

The command name changes to Show Congestions.

**Troubleshooting Placement**

**Error Messages**

<table>
<thead>
<tr>
<th>Message:</th>
<th>location: (&quot;lib&quot; &quot;cell&quot; &quot;view&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>reason:</td>
<td>VCP: half_spacing_violation (component name)(left_boundary)</td>
</tr>
<tr>
<td>Description:</td>
<td>Generated by the placer whenever the distance between one edge of the place and route boundary and the nearest component violates the half-maximum-spacing rule.</td>
</tr>
</tbody>
</table>

The half-maximum-spacing rule is a distance equal to half of all of the spacing rules for the layers specified in the iccLayer section of the rules file.

This happens typically if the place and route boundary's width, length, or both are smaller than expected.

Troubleshooting: To resolve this violation, turn on either intra or inter compaction which will adjust boundary in the appropriate direction.

**Online Forms**

- Auto Placer
- Partitioning
- Choose Component Types
- Pin Placement
- Load Template File
- Placement Planning
Auto Placer

**Group CMOS Pairs** is for CMOS designs that have been generated with chaining.

*Note:* This option does not work with devices that have been manually-abutted because clustering needs chaining information.

**Preserve Abutment Chains** specifies that abutment chains that were generated automatically during *Gen From Source* or *Placement Planning* are preserved during placement. When this option is switched off, such chains are dissolved leaving the placer free to share diffusion on individual transistors, which can reduce the wire length of polysilicon gate connections and result in better alignment.

*Important*

Cadence recommends that you leave this option switched on and inspect the results obtained by the automatic chaining function to see if they are satisfactory. If they are not, you can switch off the option and let the placer try to generate a better result.

*Note:* When disabling this option, consider the following:

- It is more efficient for the placer to place transistor chains generated by *Gen From Source* or during *Placement Planning* than to place individual transistors. This is especially true when placing a large number of transistors.
- Diffusion sharing during automatic placement is not guaranteed because the placer is geared primarily towards reducing overall wire length. If the area within the rows is limited, the placer attempts to share diffusion; if there is excess area within the rows, it does not.

**Group M factor Devices** automatically adds a grouping constraint for complimentary MOS devices that have a multiplication factor from the schematic.

**Place Selected Objects Only** places only the objects in the selected set. When switched off, the placer places all of the objects in the cellview.

**Allow Rotation** rotates the components as a part of optimization. When switched off, the placer can move components but not rotate them.
ECO Mode places any components that were left unplaced outside the design boundary, but it does not touch the placement of components that are already placed inside the design boundary.

The placer keeps placed components close to their original placed locations and inserts unplaced components into the design. In some cases, in order to minimize wire length, it might be necessary to move already-placed components; for example, if you want to add a buffer next to each register, but do not want to change the overall relative positions of the placed components.

If you want to place unfixed components, use the Place Selected Objects Only option or the Edit – Other – Lock Selected command.

In ECO mode, all filler cells are deleted before the placement and reinserted after placement.

Global Placement places components without regard to any initial placement. The placer optimizes global objectives such as total wire length and overlap violations between components might be ignored. You would typically use this option the first time you place a design or to discard previous unsatisfactory placement results.

Optimize Placement performs a detailed placement. This option lets the placer run until the objective cost is achieved. The default value is off.

- **Runtime** controls how long the optimizer tries to optimize to achieve the best possible placement. Choices are *quick*, *moderate*, and *optimized*.

Run Spacer within Rows adds or removes space within each row.

Run Spacer between Rows adds or removes space between each row.

- **Note:** The boundary is adjustable when you run the spacer option.

Reserve Space For Routing (%) lets you specify the percentage of extra space you want added, above what the routing channels between rows should occupy, to improve routability. Use this option with the Run Spacer between Rows.

Adjust Boundary automatically recalculates the boundary when you click Update Layout.

Insert Filler Cells fills empty spaces in the rows with the filler cells you defined in the Component Types form.

Insert Substrate Contacts inserts standard cell substrate contacts in the rows based on a specified minimum and maximum contact spacing value. This option is honored only when using the Assisted Standard Cell and Assisted Mixed CMOS/Standard Cell placement styles. For more information on substrate contacts, see “Defining Standard Cell Substrate Contacts” on page 229.
Save As lets you specify a new library, cell, and view name for the placed cellview. By default, this option is switched off and the current and placed names are the same and these fields are grayed out.

**Important**

Be sure to save the placed cellview to disk before translation if you want to preserve any edits that are still in memory.

Edits to the output cellview are not saved to disk automatically before placement. To discard the changes the placer makes, do not save the output cellview. However, this also discards any edits in memory.

- When you let the placer update the source layout cellview.
- When you specify a different cell or view name, but that cellview happens to be open and modified in another window.

  **Library** sets the new library for the placed cellview.

  **Cell** sets the new cell name for the placed cellview.

  **View** sets the new view name for the placed cellview.

  **Browse** displays a library browser to let you choose the new name.

**Open Window** displays the placed view in a new window after placement, if it is not already displayed. Otherwise, the system updates the output view in an existing window.

**Rules** lets you type the name of a rules file.

  **Set File** opens the Open File window, which displays the files on your system so that you can choose a rules file.

**Show More Options** displays more placement options

  **Job Host**

  **Foreground** runs the placer within the same process as the current session.

  **Background** runs the placer in a child process on the same machine.

  **LBS** submits the placement job to the LBS (Load Balancing Service) which schedules the placement job according to your settings and the availability of computing resources. The final placement session will run on the selected host using the dfII working directory for data read/write.
Offset From Boundary lets you control the minimum spacing between the boundary edge and the nearest component. The default value is half of the maximum of all the spacing rules for the layers specified in iccLayer section of translator rules.

Show Placement Progress displays a graphical representation of the placer's progress.

Place Un-Assigned Components In Mixed Mode when switched on allows unassigned (floating) components to be moved when using the Assisted Mixed CMOS/Standard Cell placement mode. When switched off, unassigned components are not moved.

Note: Enabling this option can adversely affect the performance of the placer.

Partition lets you place components by partition instead of placing the entire cellview. From the cyclic field, select the partition you want to place.

Only Check Cell Boundary, when switched on, specifies that the placer use the cell boundary to check for out-of-design violations. When switched off, the placer uses the physical boundary to check out-of-design violations.

Note: The physical boundary is instance boundary including every layer of its master. The cell boundary is the boundary enclosing only active and polysilicon layers. Enabling this option checks that the outlines of all of the components are inside the boundary, but allows the contents of the components to be outside the boundary.

Partitioning

Select lets you set up the partitioning environment.

Components lets you select a component in the layout window and have the component name highlighted in the Partitioning form.

Partition lets you select a partition in the layout window and have the partition name highlighted in the Partitioning form. This option must be turned on when you are attaching a shape on a soft fence to an existing partition.

Note: You must click on the Link to Design button and then click inside the layout window to highlight the selected component/partition in the Partitioning form.

Link to Design links the Partitioning form to the cellview in the context of partitioning. For example, the Link to Design option lets you select components or partitions in order to highlight them in the Partitioning form.
**Choose or type a Partition name** adds the name of a new partition or lets you choose from a list of existing partition names. If the Partitioning form is linked and any shape is already attached to the partition, the partition is highlighted in the layout window.

- **Create Partition** adds a new partition with the name in the *Choose or type a Partition name* text field.
- **Delete Partition** deletes the partition in the *Choose or type a Partition name* field. If there are any components assigned to this partition, they are reassigned to the boundary by default.
- **Attach Shape** attaches a shape to the current partition. The shape must already exist in the layout and must exist only on the soft fence layer. If the shape is already assigned to another partition, then it is detached from the other partition and attached to the current partition.
- **Detach Shape** detaches the selected shape from the current partition.

**Template** lets you specify a template file name. Typing a name in the text field enables the load button and the template file is loaded to the form.

- **Browse** lets you browse/specify a template file name using the Open File form.
- **Load** loads the partition information from the template file specified in the Template text field in to the Partitioning form.

**Display by** defines what is displayed in the component browser.

- **Flat Instances** displays instances that have no hierarchy.
- **Hierarchical Instances** displays instances at the top level of the hierarchy.

**Note:** If your design is flat, then the display will appear the same for both *Flat Instances* and *Hierarchical Instances* options. This is because the *Hierarchical Instances* option displays top level instances and *Flat Instances* displays leaf instances. Top level and leaf instances in a flat design are the same.

**Component Types** displays the component types used with the instances.

- **Source master** displays the names of all the master symbols that exist in the layout.
- **Target master** displays the names of all the master layouts that exist in the layout.
- **All** displays all partitions and their components.
- **Boundary** displays components that are placed in the boundary.
Sort by sorts information by component Name or Partition. The partitions themselves are sorted first, followed by the components in each partition. Click on either the Component Name or Partition Name table header to sort the table.

Select lets you type in a search pattern to select components in the component browser.

Filter lets you filter the components displayed in the component browser.

- **Reset set** clears the highlighted set.
- **Add to set** lets you add to a new selected set to an already existing set. For example the existing highlighted set is aX aY aZ aP aQ. You select b* in the component browser, then select Add to set and click OK. The new highlighted set is all of the components including a* and b*.
- **Remove from set** lets you remove any selected elements from the selected set. For example, if you type in the Select field aP and then click Ok the new highlighted set is aX aY aZ aQ.
- **Limit to set** lets you highlight a subset of components from the highlighted set. For example if you type in the Select field aY and click OK the new highlighted set is aY.

Ok invokes the action based on correct Filter options.

Expand/Collapse lets you expand (or collapse) a selected hierarchical node in the component browser. If the selected node is not hierarchical, the button is grayed out.

Target Partition is the name of the partition to which the components are to be assigned.

Move assigns the selected instances from the component browser to the current partition.

Choose Component Types

Search scrolls the list to the component type name you type into the field. The name must match exactly. This field is useful if you have a long list of types. To select a device or devices on which you want to perform an action, click on the device you want to select, or hold down Control and click to select more than one device.

Pin Placement

Display Control specify what pins are displayed in the Pin Placement form.
**Display** lets you specify from a combination of two cyclic fields which pins are displayed. The choices are *All, Placed, Unplaced, Left, Top, Right, and Bottom* and *All, Fixed, User-movable, Ordered, Unordered, I, O, IO* pins.

**Select edge from layout** displays the pins on the selected edge of the boundary.

**Sort** *By name* lists the pins in alphabetical order. *By order* sorts each pin entry per edge, ordered pins, in their order, first and unordered pins after them. *For Pitch* lists placed pins in a special manner related to their possible Pitch values with each other.

**Link to Layout** lets you select pins in the layout window and have them highlighted in the Pin Name list box. You can also select pins in the Pin Placement form and they are selected in the layout. Click **Link to Layout** and click anywhere in the layout window to link the Pin Placement form to the layout window, then select the pins that you want highlighted in the Pin Placement form.

**Unlink from layout** unlinks the Pin Placement form from the linked layout window.

**Select** lets you type in a pin name and press the **Tab** key to select the pin in the Pin Name list box. You may use wildcard characters ‘?’ and ‘*’ to select more than one pin. The wild character ‘?’ replaces any single character and ‘*’ replaces any substring of characters, including a single character in the pin names.

**Pin Name** is the name of the pin.

**Edge** is the edge of the boundary that the pin is placed on.

**Order** defines how the pins are ordered.

**Pitch** the distance between neighboring pins.

**Note:** Press the **Tab** key to activate the **Apply** button.

**Status** indicates the status of the pin. For example, *Fixed* tells you that the pin is fixed and *M* indicates that the pin is fixed but also user-movable in the location that the pin is fixed at.

**Type** I, O, or IO.

**Layer** the layer the pin has been created on.

**Edit selected pin or select more pins from list** lets you edit pin settings including placement of them on a boundary edge.

**Edge** determines the edge that the pin is placed on. The options are *Left, Top, Right, Bottom, Any, None, and As is*. The *As is* setting leaves the edge settings of pins intact when changing other pin properties.
Order specifies the order of the selected pins. To place the pins as unordered pins, enter “-“ or a blank space. To specify an order for the pins, enter a digit.

Pitch assigns a specific distance between two or more neighboring pins. The pitch value may be set for ordered pins, unordered pins, or a combination of the two.

Location defines the final placed positions of the pins.
- Fixed at Placed Location commits the selected pins to their current positions.
- User-movable at Placed Location lets you move the pins to a new location. This option is similar to a Fixed constraint, but you may use the Edit – Move command to move the pin to another location. The pin will automatically be fixed to the new location. If you move a fixed pin to a different location, the placer will put the pin back to its original fixed location.
- Floating changes previously set fixed pins or user-movable pins to be unfixed or non-user-movable.
- As is option does not constrain pins to a certain location.

Select edge from layout lets you select the edge from the layout canvas. This is best used for polygonal boundaries.

Iterated Pins lets you expand or collapse the listing of iterated (bus) pins.
- Interleave lets you interleave the bits of any two selected buses.
- Expand expands the pin lists to show all bits of one or more iterated pins.
- Collapse collapses the pin lists to show bus notation for one or more iterated pins.
- All performs the expand or collapse function on all the pins in the list box.
- Selected performs the expand or collapse function only on highlighted pins in the list box.

Change Order lets you change the order of ordered pins.
- Arrow to the left/down moves the selected pins down in the list box.
- Arrow to the right/up moves the selected pins up in the list box.
- Swap inverts the order of two selected pins in the list box.

Place pins as in specifies where the pin placement information comes from.
- Schematic view takes a snapshot of the relative position of the pins in the schematic and updates the pin placement in the layout window accordingly.
Template file displays the Load Template File form which lets you specify a template file containing information to be loaded into the pin placement form. For more information, see Load Template File.

HRail elongates the selected pins from one end of the boundary to another horizontally.

VRail elongates the selected pins from one end of the boundary to another vertically.

**Load Template File**

**Name** is the user-defined name of the template file.

**Filter** lets you specify a letter or keyword to filter the template file names.

**Load sections** loads the I/O Pins section of the template file.

**Existing data** either merges or replaces the template file.

**Placement Planning**

**Partition** lets you select a partition that was created in the Partitioning form.

   **Select From Layout** lets you select a partition directly from the layout window.

**Style** lets you select and load one of the available styles. The style you select determines the controls that are made available on the Placement Planning form.

The rest of this section describes the options available with the Assisted CMOS style, which lets you define rows with full flexibility for device or cell-level placement styles.

**Style File** lets you load a user-defined style if none of the predefined styles matches your requirements.

   **Browse** opens the Open File form.

   **Load** loads the file specified in the **Style File** field.

**Load Row Template For** activates the buttons and fields for the template file. You can then choose whether to load the template file for the **Entire Design** or the **Selected Partition** only.

**Template File** is the template file created using the **Design – Save To Template** command.

   **Browse** opens the Open File form.
Load loads the specified template file.

Regions lets you specify a region, which is a boundary within which to apply the parameters specified in the form. After the rows are created, the regions no longer exist, but you can still manipulate the rows. You can draw rows outside the region.

Region Shape determines the shape of the region.

- **Rectangle** creates a rectangular region.
  - **Origin X** sets the X coordinate of the lower left corner of the lowest numbered row (region) being edited.
  - **Origin Y** sets the Y coordinate of the lower left corner of the row (region).
  - **Width** lets you enter the width of the row you want (larger than the height if you want horizontal rows).
  - **Height** lets you enter the height of the row you want (larger than the width if you want vertical rows).

- **Polygon** creates a polygonal region.
  - **Points** field appears after selecting the Polygon option. Type in the coordinates.

Draw lets you use the mouse to draw the shape you want in the layout window. When creating a Rectangle, the shape you draw sets the placement region and the X, Y, Width, and Height fields are filled in with the appropriate values. When creating a Polygon, the coordinates of the individual vertices are updated in the Points text field.

Update updates the region.

Delete removes the drawn region.

Properties lets you set properties for the pins.

- **Allow For Pins** secures a space from the boundary edge equal to the size of the pin plus the spacing rule. This prevents rows from overlapping pins.

- **Allow Rows Beyond Region** automatically expands the row beyond the region if needed. Specify the direction of expansion to be either horizontal or vertical.
  - **Horizontally** draw stretch rows in a horizontal direction.
  - **Vertically** draw stretch rows in a vertical direction.

- **Allow Rows Beyond Partition** automatically expands the row beyond the partition if needed.
Rows lets you create rows.

- **Row Name** specifies a name for your rows. If you do not specify a row name, the default row name is assigned, starting with Row1, Row2, and so on.

- **Direction** is the default direction for rows, either Horizontal or Vertical.

**Utilization**

- **Utilization Inside Rows** specifies how much of a row is used by the components and how much is left for routing. Select 100% to specify that all of the row space is available for use by components or specify a percentage value in the Specify % field.

- **% Area Covered By Rows** specifies how much of a region is occupied by rows and how much is left for routing channels. Provide a row Spacing or specify a percentage value in the Specify % field.

**Number Of Rows** lets you Specify the number of rows to be created or leave it to the placer to create the Maximum number possible in the space available.

**Create Pin Row** creates a pin row centered in the design along the y-axis. During the placement step, all pins that are not constrained to the cell boundary are placed in this center pin row and optimized to minimize wire length. Use this option if you are developing standard cell libraries, where placing the pins on the boundary is not desired.

**Spacing Between N-Row and P-Row** controls the distance between the N-P pair.

- **Minimum N-P Component Spacing** specifies the minimum distance between the N-P pair. The default value is the largest of all the minimum spacing values for the current cellview layers.

- **Top and Bottom Edge Offset** specifies the offset from top or bottom of the region to the nearest component.

**Rails** lets you specify information for power and ground pins.

**Power**

- **Layer** specifies the layer on which the rails are drawn.

- **Width** specifies the width of the rails.

- **Net** specifies the net to which the rails belong.

- **Rail to Device Spacing** specifies the minimum distance between the power rail and PMOS and NMOS devices. If you do not specify a value, the software uses the largest minimum spacing value specified for PMOS/NMOS device layers to power rail layers.
Ground

- **Layer** specifies the layer on which the rails are drawn.
- **Width** specifies the width of the rails.
- **Net** specifies the net to which the rails belong.
- **Rail to Device Spacing** specifies the minimum distance between the ground rail and PMOS and NMOS devices. If you do not specify a value, the software uses the largest minimum spacing value specified for PMOS/NMOS device layers to ground rail layers.

**Pattern** specifies how N and P devices alternate between rows or how cells alternate. The patterns that you can use are **N-P-P-N**, **P-N-N-P**, **N-P**, or **P-N**.

**Position** controls the relative position of rails with regard to rows. Specify either **Inside**, **Outside**, or **Center**.

Components

**PMOS**

- **Width Threshold** is the maximum width allowed after MOS device folding is performed. The default value comes from the CDF data for the device. If the threshold value is 0, the value of the **Fold Threshold** option set in the **Edit Component Types** form is used.

- **Diffusion Spacing** sets the minimum spacing required between adjacent MOS chains. This value affects the estimation of the number of rows required.

**NMOS**

- **Width Threshold** is the maximum width allowed after MOS device folding is performed. The default value comes from the CDF data for the device.

- **Diffusion Spacing** sets the minimum spacing required between adjacent MOS chains. This value affects the estimation of the number of rows required.

**Align Components** controls the relative alignment of devices in adjacent rows.

- **Inside** aligns the P and N devices toward each other.
- **Outside** aligns the devices away from each other.
- **Center** aligns the devices with the row center line.

**Allow Chaining** enables components to be chained.

- **Stack Partition Parameter** sets the device chaining partition parameters.
Max Chain Size sets the maximum number of devices in chain.

Layout Generation controls whether components are regenerated if necessary or only missing components are added.

- Regenerate All regenerates all components.
- Generate Only Missing Components generates only missing components.
- Preserve Constrained Objects is used with only the Regenerate All option. The constrained objects are not deleted.
- Adjust Boundary lets the placer estimate and change the boundary size when the layout is updated.

Estimates creates the rows but does not generate any layout components.

Calculate Estimates gives you an estimate of the PMOS and NMOS devices within a row.

Update Layout updates the current layout with the assumption that you do not want to change any of the parameters.

Placement Planning (Assisted Standard-Cell)

Partition lets you select a partition that was created in the Partitioning form.

- Select From Layout lets you select a partition directly from the layout window.

Style lets you select and load one of the available styles. The style you select determines the controls that are made available on the Placement Planning form.

The rest of this section describes the options available with the Assisted Standard-Cell style, which lets you define rows with full flexibility for device or cell-level placement styles.

Style File lets you load a user-defined style if none of the predefined styles matches your requirements.

- Browse opens the Open File form.
- Load loads the file specified in the Style File field.

Load Row Template For activates the buttons and fields for the template file. You can then choose whether to load the template file for the Entire Design or the Selected Partition only.
Template File is the template file created using the Design – Save To Template command.

Browse opens the Open File form.

Load loads the specified template file.

Regions lets you specify a region, which is a boundary within which to apply the parameters specified in the form. After the rows are created, the regions no longer exist, but you can still manipulate the rows. You can draw rows outside the region.

Region Shape determines the shape of the region.

- **Rectangle** creates a rectangular region.
  - **Origin X** sets the X coordinate of the lower left corner of the lowest numbered row (region) being edited.
  - **Origin Y** sets the Y coordinate of the lower left corner of the row (region).
  - **Width** lets you enter the width of the row you want (larger than the height if you want horizontal rows).
  - **Height** lets you enter the height of the row you want (larger than the width if you want vertical rows).

- **Polygon** creates a polygonal region.
  - **Points** field appears after selecting the Polygon option. Type in the coordinates.

Draw lets you use the mouse to draw the shape you want in the layout window. When creating a Rectangle, the shape you draw sets the placement region and the X, Y, Width, and Height fields are filled in with the appropriate values. When creating a Polygon, the coordinates of the individual vertices are updated in the Points text field.

Update updates the region.

Delete removes the drawn region.

Properties lets you set properties for the pins.

- **Allow For Pins** secures a space from the boundary edge equal to the size of the pin plus the spacing rule. This prevents rows from overlapping pins.

- **Allow Rows Beyond Region** automatically expands the row beyond the region if needed. Specify the direction of expansion to be either horizontal or vertical.
  - **Horizontally** draw stretch rows in a horizontal direction.
Vertically draw stretch rows in a vertical direction.

Allow Rows Beyond Partition automatically expands the row beyond the partition if needed.

Rows lets you create rows.

Row Name specifies a name for your rows. If you do not specify a row name, the default row name is assigned, starting with Row1, Row2, and so on.

Direction is the default direction for rows, either Horizontal or Vertical.

Utilization

Utilization Inside Rows specifies how much of a row is used by the components and how much is left for routing. Select 100% to specify that all of the row space is available for use by components or specify a percentage value in the Specify % field.

% Area Covered By Rows specifies how much of a region is occupied by rows and how much is left for routing channels. Provide a row Spacing or specify a percentage value in the Specify % field.

Number Of Rows lets you Specify the number of rows to be created or leave it to the placer to create the Maximum number possible in the space available.

Standard Cell Grid is a user-enforced grid. The value should not contradict the standard cell widths. The grid can be a fraction of the greatest common divisor of the standard cell widths. For example, if you have two cells of widths 2.5 and 4.5. The greatest common divisor is 0.5 so you can choose values of 0.01, 0.1, 0.5, and so on.

Use Filler Cells inserts filler cells that have been defined in the Edit Component Types form between the standard cells in the row. Type the names of the filler cells into the field, or click Choose Types and choose the names of the filler cells from the list that appears in the Choose Component Types form.

Use Substrate Contacts inserts standard cell substrate contacts that have been defined in the Edit Component Types form between the standard cells in the row. Type the names of the standard cell substrate contacts into the field or click Choose Types and choose the names of the standard cell substrate contacts from the list that appears in the Choose Component Types form.

Rails lets you specify information for power and ground pins.

Power

Layer specifies the layer on which the rails are drawn.
Width specifies the width of the rail.

Net specifies the net to which the rails belong.

Ground

Layer specifies the layer on which the rails are drawn.

Width specifies the width of the rail.

Net specifies the net to which the rails belong.

Pattern specifies how ground and power rails alternate between rows. The patterns that you can use are G-P, P-G, G-P-P-G, or P-G-G-P.

Rail To Rail Spacing specifies the spacing between power and ground rails for the same row.

Layout Generation controls whether components are regenerated if necessary or only missing components are added.

Regenerate All regenerates all components.

Generate Only Missing Components generates only missing components.

Preserve Constrained Objects is used with only the Regenerate All option. The constrained objects are not deleted.

Adjust Boundary lets the placer estimate and change the boundary size when the layout is updated.

Estimates creates the rows but does not generate any layout components.

Calculate Estimates gives you an estimate of the PMOS and NMOS devices within a row.

Update Layout updates the current layout with the assumption that you do not want to change any of the parameters.

Placement Planning (Assisted Mixed CMOS/Standard-Cell)

Partition lets you select a partition that was created in the Partitioning form.

Select From Layout lets you select a partition directly from the layout window.

Style lets you select and load one of the available styles. The style you select determines the controls that are made available on the Placement Planning form.
The rest of this section describes the options available with the Assisted Mixed CMOS/Standard-Cell style, which lets you define rows with full flexibility for device or cell-level placement styles.

**Style File** lets you load a user-defined style if none of the predefined styles matches your requirements.

- **Browse** opens the Open File form.
- **Load** loads the file specified in the *Style File* field.

**Load Row Template For** activates the buttons and fields for the template file. You can then choose whether to load the template file for the *Entire Design* or the *Selected Partition* only.

**Template File** is the template file created using the *Design – Save To Template* command.

- **Browse** opens the Open File form.
- **Load** loads the specified template file.

**Regions** lets you specify a region, which is a boundary within which to apply the parameters specified in the form. After the rows are created, the regions no longer exist, but you can still manipulate the rows. You can draw rows outside the region.

- **Region Shape** determines the shape of the region.
  - **Rectangle** creates a rectangular region.
    - **Origin X** sets the X coordinate of the lower left corner of the lowest numbered row (region) being edited.
    - **Origin Y** sets the Y coordinate of the lower left corner of the row (region).
    - **Width** lets you enter the width of the row you want (larger than the height if you want horizontal rows).
    - **Height** lets you enter the height of the row you want (larger than the width if you want vertical rows).
  - **Polygon** creates a polygonal region.
    - **Points** field appears after selecting the *Polygon* option. Type in the coordinates.

**Draw** lets you use the mouse to draw the shape you want in the layout window. When creating a *Rectangle*, the shape you draw sets the placement region and the $X$, $Y$,
Width, and Height fields are filled in with the appropriate values. When creating a Polygon, the coordinates of the individual vertices are updated in the Points text field.

Update updates the region.

Delete removes the drawn region.

Properties lets you set properties for the pins.

- Allow For Pins secures a space from the boundary edge equal to the size of the pin plus the spacing rule. This prevents rows from overlapping pins.

- Allow Rows Beyond Region automatically expands the row beyond the region if needed. Specify the direction of expansion to be either horizontal or vertical.
  - Horizontally draw stretch rows in a horizontal direction.
  - Vertically draw stretch rows in a vertical direction.

- Allow Rows Beyond Partition automatically expands the row beyond the partition if needed.

Rows lets you create rows.

- Row Name specifies a name for your rows. If you do not specify a row name, the default row name is assigned, starting with Row1, Row2, and so on.

- Direction is the default direction for rows, either Horizontal or Vertical.

Utilization

- Utilization Inside Rows specifies how much of a row is used by the components and how much is left for routing. Select 100\% to specify that all of the row space is available for use by components or specify a percentage value in the Specify % field.

- % Area Covered By Rows specifies how much of a region is occupied by rows and how much is left for routing channels. Provide a row Spacing or specify a percentage value in the Specify % field.

Number of Rows lets you Specify the number of rows to be created or leave it to the placer to create the Maximum number possible in the space available.

Standard Cell Grid is a user-enforced grid. The value should not contradict the standard cell widths. The grid can be a fraction of the greatest common divisor of the standard cell widths. For example if you have two cells of widths 2.5 and 4.5. The greatest common divisor is 0.5 so you can choose values of 0.01, 0.1, 0.5, and so on.
Use Filler Cells inserts filler cells that have been defined in the Edit Component Types form between the standard cells in the row. Type the names of the filler cells into the field, or click Choose Types and choose the names of the filler cells from the list that appears in the Choose Component Types form.

Use Substrate Contacts inserts standard cell substrate contacts that have been defined in the Edit Component Types form between the standard cells in the row. Type the names of the standard cell substrate contacts into the field or, click Choose Types and choose the names of the standard cell substrate contacts from the list that appears in the Choose Component Types form.

Rails lets you specify information for power and ground pins.

**Power**
- Layer specifies the layer on which the rails are drawn.
- Width specifies the width of the rail.
- Net specifies the net to which the rails belong.

**Ground**
- Layer specifies the layer on which the rails are drawn.
- Width specifies the width of the rail.
- Net specifies the net to which the rails belong.

Pattern specifies how ground and power rails alternate between rows. The patterns that you can use are G-P, P-G, G-P-G-P, or P-G-G-P.

**Rail To Rail Spacing** specifies the spacing between power and ground rails for the same row.

**Minimum N-P Component Spacing** specifies the minimum distance between the N-P pair. The default value is the largest of all the minimum spacing values for the current cellview layers.

Components

**PMOS**
- Width Threshold is the maximum width allowed after MOS device folding is performed. The default value comes from the CDF data for the device. If the threshold value is 0, the value of the Fold Threshold option set in the Edit Component Types form is used.
Diffusion Spacing sets the minimum spacing required between adjacent MOS chains. This value affects the estimation of the number of rows required.

NMOS

Width Threshold is the maximum width allowed after MOS device folding is performed. The default value comes from the CDF data for the device.

Diffusion Spacing sets the minimum spacing required between adjacent MOS chains. This value affects the estimation of the number of rows required.

Align Components controls the relative alignment of devices in adjacent rows.

- Inside aligns the P and N devices toward each other.
- Outside aligns the devices away from each other.
- Center aligns the devices with the row center line.

PMOS Offset To Row Edge specifies the minimum distance between a PMOS device and the edge of the row that contains it.

NMOS Offset To Row Edge specifies the minimum distance between an NMOS device and the edge of the row that contains it.

Allow Chaining enables components to be chained.

- Stack Partition Parameter sets the device chaining partition parameters.
- Max Chain Size sets the maximum number of devices in chain.

Layout Generation controls whether components are regenerated if necessary or only missing components are added.

- Regenerate All regenerates all components.
- Generate Only Missing Components generates only missing components.
- Preserve Constrained Objects is used with only the Regenerate All option. The constrained objects are not deleted.
- Adjust Boundary lets the placer estimate and change the boundary size when the layout is updated.

Estimates creates the rows but does not generate any layout components.

Calculate Estimates gives you an estimate of the PMOS and NMOS devices within a row.

Update Layout updates the current layout with the assumption that you do not want to change any of the parameters.
Placement Planning (Manual User-Defined)

**Partition** lets you select a partition that was created in the Partitioning form.

- **Select From Layout** lets you select a partition directly from the layout window.

**Style** lets you select and load one of the available styles. The style you select determines the controls that are made available on the Placement Planning form.

The rest of this section describes the options available with the *Manual User-Defined* style, which lets you define rows with full flexibility for device or cell-level placement styles.

**Style File** lets you load a user-defined style if none of the predefined styles matches your requirements.

- **Browse** opens the Open File form.
- **Load** loads the file specified in the *Style File* field.

**Load Row Template For** activates the buttons and fields for the template file. You can then choose whether to load the template file for the *Entire Design* or the *Selected Partition* only.

**Template File** is the template file created using the *Design – Save To Template* command.

- **Browse** opens the Open File form.
- **Load** loads the specified template file.

**General**

**Row Name** specifies a name for your rows. If you do not specify a row name, the default row name is assigned, starting with *Row1, Row2*, and so on.

**Direction** is the default direction for rows, either *Horizontal* or *Vertical*.

**Initialize First Row**

- **Draw** lets you use the mouse to draw the shape you want in the layout window. The shape you draw sets the placement region and the *X, Y, Width*, and *Height* fields are filled in with the appropriate values.

**Spacing Between Rows** specifies how much of the region is occupied by rows and how much is left for routing channels.

**Number Of Rows to Create** is the total number of rows in the list box.
Types

Create lets you change the options from Component Grid on down to create the component type.

Update lets you select the component type in the Types field and change any of the attributes.

Delete removes the component type from the Types field.

Component Types displays the component type that you selected in the Choose Component Types form.

Edit opens the Choose Component Types form. Select an existing component type and click OK.

Component Grid defines the component grid for the rows.

Reference Point For Type sets a reference point for a component type to be either Top, Center, Bottom, or Origin.

Reference Point For Row sets a reference point for a row to be either Top, Center, or Bottom.

Type “x” Offset From Row “y” is the distance between the reference point on the row and the reference point on the component.

Allowed Orientations are R0, MY, R90, MXR90, MX, R180, MYR90, R270, AS IS

Rails

Rails lets you create a user-defined rail using the Layer, Width, and Net options.

Add lets you add the user-defined rail to the Rails field.

Update Rail lets you update the Layer, Width, and Net Name for the selected rail.

Delete lets you delete the selected rail.

Layer is the layer of the rail.

Width is the width of the rail.

Net is the name of the rail.

Reference Point For Rail sets a reference point for the rail to be either Top, Center, or Bottom.

Reference Point For Row sets a reference point for a row to be either Top, Center, or Bottom.
**Rail “x” Offset From Row “y”** is the distance between the reference point on the row and the reference point on the rail.


**Number of Rows** is the number of rows generated after running Calculate Estimates.

**Search** lets you search for rows.

- **Choose From Layout** lets you select a row in the layout and have it highlighted in the list box.
- **Select All** selects all rows in the list box.
- **Deselect All** deselects all selected rows in the list box.

**Create New** creates new rows.

**Update Selected Rows** changes any of the properties of the selected row.

**Delete Selected** lets you delete any selected row.
Preparing Your Design for Routing

This chapter explains how the Virtuoso® XL layout editor (Virtuoso XL) processes connectivity information and presents several ways to prepare your design for routing. This chapter discusses the following topics:

- **Connectivity Extraction** on page 296
- **Hierarchical Extraction** on page 298
- **Pseudo-Parallel Connections** on page 302
- **Layer Selection** on page 312
- **Connecting Nets** on page 314
- **Checking Connectivity with Flight Lines** on page 319
- **Checking Connectivity with Markers** on page 320
- **Physical Vias** on page 322
- **Using the Virtuoso Compactor on a Routed Design** on page 326

For information about how to use the Virtuoso custom router to route your design, see the *Virtuoso Custom Placement and Routing Preparation Guide*.

For information about running the router, see the *Online Help* (available in PDF format in your router installation hierarchy).

For information on Virtuoso XL forms, see “Online Forms” on page 327.
Connectivity Extraction

Virtuoso XL accepts only connections

- On layers that are defined in the technology file as extractable layers. For more information, see “Layer Rules” on page 32.

- Between two touching shapes or instance pins on the same layer.

- Between two touching shapes or instance pins on different layers if the layers have been defined as equivalent layers or one of them is defined as a via layer. For more information, see “Layer Rules” on page 32.

- Between two layers by means of a contact instance having a pin on each layer and both pins on the same terminal.

- Between two layers by means of a via
**Note:** By default, Virtuoso XL uses symbolic vias defined in the technology file. You can also define physical vias using the Virtuoso layout editor if you need shapes not supported by the symbolic via. For more information, see “Physical Vias” on page 322.

There are two distinct types of extraction performed by Layout XL.

- **Top-level design extraction**, where the extractor extracts only the current top-level design. The extractor’s visibility into the design hierarchy is defined by the `extractStopLevel` environment variable.

  You do this by turning on the *Connectivity extractor* option in the *Layout XL Options* form.

- **Hierarchical extraction**, where the extractor extracts either the entire hierarchical design or selected lower-level cellviews from the current top level.

  You do this using the *Connectivity – Re-Extract Layout* command.

**Top-Level Design Extraction**

The extractor extracts the current top-level design and by default considers only

- Top-level objects; i.e., shapes, vias, and instances used to implement pins.

- Instance pins of top-level instances. These pins represent the interface of top-level instances to the lower levels in the design and as such are themselves considered to be at the top level.

**Extraction Stop Level**

The `extractStopLevel` environment variable increases the extractor’s visibility into the design hierarchy. For example,

<table>
<thead>
<tr>
<th><code>extractStopLevel</code></th>
<th>Means...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>extractStopLevel</code> 0 + objects at level 1 + instance pins at level 2</td>
</tr>
<tr>
<td>2</td>
<td><code>extractStopLevel</code> 1 + objects at level 2 + instance pins at level 3</td>
</tr>
</tbody>
</table>

The greater the `extractStopLevel`, the more accurate the extraction.

- Top-level nets that are incomplete with `extractStopLevel = 0` may become complete with `extractStopLevel > 0`. 
For example, assume that \( i0.p0 \) is an instance pin of top-level instance \( i0 \) and \( r1 \) is a top-level shape. \( i0.p0 \) and \( r1 \) are disjoint and on netA, therefore netA is incomplete.

Now increase the extractStopLevel. The extractor sees a lower-level shape \( i0.r2 \) inside \( r0 \). If top-level shape \( r1 \) is overlapping \( i0.r2 \), then netA becomes complete.

- There is better detection of shorts between top-level shapes and lower-level shapes. Note that, in this context, the connectivity of a lower-level shape is not the shape net itself but the top-level net (if it exists) found by the bottom-up series of terminals and instance terminals that starts at the shape.

- There is better detection of shorts between different instances.

\[ \text{Important} \]

Whatever the extractStopLevel, the extractor only ever extracts the current top level; i.e., it only ever changes the connectivity of objects or creates markers at the top level of the design. For information on hierarchical extraction, see “Hierarchical Extraction” on page 298.

The connectivity extractor recognizes connectivity only if all the shapes on a net have connectivity information. Otherwise, it reports incomplete nets even though the router reports that the design is 100% routed.

To verify that all connections are made in the layout as specified in the schematic, use an Assura® verification product, a Dracula® verification tool, or an Assura hierarchical physical verification product to run a layout versus schematic (LVS) check on the completed design.

**Hierarchical Extraction**

Hierarchical extraction in Virtuoso XL lets you extract lower-level cellviews from the current top level of the design.

You need to re-extract your design if it was changed after it was last extracted. This can happen if you modify a lower-level cellview in an application other than Layout XL (for example, in Layout L) or using a SKILL command which leaves the cellview’s extraction time stamp older than its modification time stamp.

As design is fully re-extracted only when you use the Connectivity – Re-Extract Layout command. This command is enabled when the Connectivity Extractor is switched on in the Layout XL Options form and honors the other extraction settings on that form.
Important

The *Update Components And Nets* command only partially re-extracts a design. It corrects the connectivity of the instances in the design and re-extracts only those instances and objects connecting to them. It does not perform a full re-extraction.

Re-Extracting a Top-Level Design

To re-extract a top-level design,

1. From the CIW, open a layout cellview and choose *Connectivity – Re-Extract Layout*.

   The *Virtuoso XL Connectivity Extraction* form is displayed.

   ![Virtuoso XL Connectivity Extraction Form](image)

   If there are lower level cellviews that need to be re-extracted, they are listed in the *Select Cellviews* pane.

2. Switch on *Save Extracted Cellviews* to automatically save the cellview after re-extraction.

3. (Optional) To increase or decrease the extractor’s visibility into the hierarchy of the design, change the value of the *Extract Connectivity to Level* option in the *Layout XL Options* form.

   For more information, see “*Extraction Stop Level*” on page 297.
4. Click OK or Apply.

   The top-level design is re-extracted.

   **Tip**

   You set the stop level for the extraction and the display options for weak-connect and must-connect marker violations in the *Layout XL Options* form.

**Re-Extracting Selected Hierarchical Cellviews**

To re-extract only selected hierarchical cellviews,

1. From the CIW, open a layout cellview and choose *Connectivity – Re-Extract Layout*.

   The *Virtuoso XL Connectivity Extraction* form is displayed.

   ![Virtuoso XL Connectivity Extraction Form]

   If there are lower level cellviews that need to be re-extracted, they are listed in the *Select Cellviews* pane.

2. Switch on *Save Extracted Cellviews* to automatically save the cellview after re-extraction.
3. (Optional) To increase or decrease the extractor’s visibility into the hierarchy of the design, change the value of the *Extract connectivity to level* option in the **Layout XL Options** form.

For more information, see “**Extraction Stop Level**” on page 297.

4. In the **Select Cellviews** pane, choose the cellviews you want to re-extract.

You can filter by library name, cell name, and view name or select all of the cellviews listed. Note that the filter mechanism has no wildcard capability. For example, to select a cell called ‘resistor’, you must type ‘r’, ‘re’, ‘res’, and so on. If you type ‘sis’, the cell is not selected.

5. Click **OK** or **Apply** to re-extract the selected lower level cellviews.

The selected cellviews are re-extracted all the way up to the top level of the design.

**Hierarchical Extraction of Parameterized Cells**

For hierarchical connectivity extraction to work for pcells, the connectivity must be defined on the shapes within the pcell.

You assign connectivity for a pcell shape by descending into the pcell and using the *Connectivity – Add Shape to Net* command. For more information see “**Adding and Deleting Shapes on Nets**” in the **Virtuoso Layout Editor User Guide**.

For more information on pcells, see the **Virtuoso Parameterized Cell Reference**.

⚠️ **Important**

For non-pcells, you assign connectivity interactively using Cadence® SKILL functions or using the *Connectivity – Assign Nets* command. For more information, see “**Adding Pins to a Layout**” on page 199.

**False Overlap and Short Markers**

To avoid false overlap and short markers

- Add the **lxBlockOverlapCheck** property to a shape, instance, or instance master.

The **lxBlockOverlapCheck** property tells the extractor not to check whether a nonoverlap layer of this shape, instance, or instance master is touching a nonoverlap layer at any cellview level.
Virtuoso XL does not trace connectivity through floating shapes that are at lower levels of hierarchy. Connectivity is traced only through the terminals of the lower level blocks. The primary reason for this is that a flat, LVS-type approach would significantly impair performance for large designs.

**Via Mosaic Support (CDB only)**

The CDB version of Virtuoso XL supports vias and contacts that are implemented as mosaics. The OpenAccess version does not.

![Caution]

* Mosaic instances are supported only for vias and contacts. Mosaic instances for components are not supported.

You can create, delete, move, and remaster mosaic vias and contacts. You can also change the numbers of rows and columns in the mosaic.

 Via mosaics are considered non-sticky. They are assigned to nets dynamically depending on the overlaps that are present after each move. If there is no overlap, there is no net assignment, even if you assign the net manually.

![Caution]

* The extractor ignores any `lxStickyNet` property set on a via mosaic.

Connectivity is propagated through mosaics. For example, shape A on net n1 on layer M1 connects to a M1_M2 via mosaic with no net. This via mosaic connects to shape B (with no net) on layer M2. When you extract the design, Virtuoso XL assigns both the via mosaic and shape B to net n1.

**Pseudo-Parallel Connections**

A pseudo-parallel connection represents a group of instance terminals on the same net, which need to be physically connected at the current hierarchy level. It is similar to a must-connect; the only difference is that it is implemented not using pins but instance terminals that belong to a specified net.

The main advantage of recognizing that a net can be made pseudo-parallel is that it saves space because there is less routing required to connect up the net.
What is a Pseudo-Parallel Connection

Consider a simple inverter comprising an NMOS and PMOS pair. If you set the \texttt{mfactor=2} on the inverter, then when it is flattened in the layout, the connectivity is something like in the picture below.

\begin{center}
\begin{tikzpicture}
\draw (-2,0) -- (2,0);
\draw (-1,-1) -- (-1,1);
\draw (1,-1) -- (1,1);
\draw (-1,0) -- (1,0);
\draw[dashed] (-1,-1) -- (1,1);
\end{tikzpicture}
\end{center}

The broken line represents an internal net connecting the source and drain of each PMOS and NMOS. This net has four instance terminals; however, the router does not need to actually make the connection from the P.1/N.1 pair to the P.2/N.2 pair in order for the circuit to function correctly.

Because it is mfactored, the nets connecting to all of the other terminals of P.1 and P.2 are exactly the same, as are the nets connecting to all the other terminals of N.1 and N.2. Therefore, the voltage on the node between P.1 and N.1 is the same as that on the node between P.2 and N.2, removing the need to connect them up with a wire.

To represent this situation in the database, the instance terminals are partitioned into sub-subnets. Each sub-subnet on a pseudoparallel net contains two instance terminals which must be connected together, but no connection is necessary between the instance terminals on different sub-subnets.

In the example above, there would be two such sub-subnets. The instance terminals from P.1/N.1 would be contained in one of them, representing the fact that they must be connected in order for the circuit to work. The instance terminals from P.2/N.2 would be in the other, again representing the fact that they must be connected in order for the circuit to work. But no connection is required between the two. The circuit would work equally well without the connecting wire, as shown below.

\begin{center}
\begin{tikzpicture}
\draw (-2,0) -- (2,0);
\draw (-1,-1) -- (-1,1);
\draw (1,-1) -- (1,1);
\draw (-1,0) -- (1,0);
\end{tikzpicture}
\end{center}
The model extends depending on the mfactor. In the example above, if the mfactor = x, then for the "internal" net connecting all of the sources and drains together there are a total of x sub-subnets each containing two instance terminals.

Virtuoso XL does not consider the mfactor in the source when determining whether a net is pseudo-parallel or not. Instead it considers the net, the instances attached to it, and all of the other connections to those instances.

In general, any time there is a pair of devices in series in the schematic, each with the same mfactor value, then the net implementing the series connection can be made pseudo-parallel in the layout.

**Examples of Pseudo-Parallel Connections**

Pseudo-parallel connections are useful in situations as shown below;

In this example, nodes A and B can be considered equipotential, i.e., there is no current flowing through them. You can save area in your design if this connection is skipped, even though the connectivity reference requires all instance terminals on the same net to be physically connected. With pseudo-parallel connections on A and B, interconnection is not required between the two instance terminals even though they are on the same net.
The picture below shows an explicit pseudo-parallel connection of two or more groups of parallel devices in series.
The picture below shows an implicit pseudo-parallel connection of two or more groups of mfactored devices in series.
Important

Both examples show above qualify as pseudo-parallel, but the second example is the recommended method.

Virtuoso XL automatically finds possible pseudo-parallel connections and treats them as such when you generate the layout using the Design – Gen from Source command.

If the layout exists already, you can create pseudo-parallel connections automatically by running the Connectivity – Update – Components and Nets command.

Alternatively, you can create pseudo-parallel connections manually by

- Choosing the Connectivity – Define Pins – Pseudo Parallel Connect command.
- Setting the setPPConn environmental variable.

Defining a Pseudo Parallel Connection

Using the Pseudo Parallel Connect Command

To define a pseudo-parallel connection for a set of instance terminals on a net,

1. From the layout window menu bar, choose Connectivity – Define Pins – Pseudo Parallel Connect.

   The layout window status bar prompts you to click to select a net.

2. Click on the net containing the instance terminals you want to connect.

   Flight lines appear between the pins on the specified net that can be pseudo-parallel connected.

   Note: If you were showing incomplete nets, those flight lines are removed automatically.

   The layout window prompts you to select an instance terminal.

3. Click on one or more pins of the selected net to select the instance terminals you want to connect in a pseudo-parallel connection.

   The pseudo-parallel connection is defined and the flight lines updated to show the connection.

4. When you are done, press Esc to cancel the command.
Instead of selecting nets by clicking in the layout window, you can use the Define Pseudo Parallel Connected Net form.

To open the form,

1. Press F3 while the *Pseudo Parallel Connect* command is active.
   - The Define Pseudo Parallel Connected Net form is displayed. The form lists all of the nets with no I/O pins.
2. In the form, type the name of a net in the *Net* field or select the net names from the list.

   *Tip*
   - You can work on more than one net at a time by choosing multiple nets in the form or using drag-by-area when the command line prompts you to select a net.
   - **Note:** You must define pseudo-parallel connectivity within a net. You cannot define pseudo-parallel connectivity between nets.
3. When you are done, press Esc to cancel the command.

   *Tip*
   - You can change between the *Must Connect*, *Strongly Connected*, *Weakly Connected*, and *Pseudo Parallel Connect* commands using the right mouse button.

**Using the setPPConn Environment Variable**

Setting the *setPPConn* environmental variable to t automatically identifies nodes that can qualify as pseudo-parallel connections and defines them as such.

To use the *setPPConn* environment variable,

1. Add the following line to the .cdsenv file.

   ```
   layoutXL setPPConn boolean t
   ```

**Resetting a Pseudo-Parallel Connection**

To reset a pseudo-parallel connection,

1. Delete one of the instances in the pseudo-parallel connection.
2. Set the *setPPConn* environment variable to nil.
3. Run Connectivity – Update – Components and Nets or rerun Design – Gen From Source to regenerate the deleted instance.

Generating a Layout without Chaining and Folding on Pseudo-Parallel Nets

Pseudo-parallel nets are shown as must-connects when the setPPConn environmental variable is set to t and the layout is generated without transistor chaining and folding.

To do this,

1. From the layout window menu bar, choose Design – Gen From Source.

   The Layout Generation Options form is displayed.

2. Turn off the Transistor Chaining and Transistor Folding options.

3. Set the rest of the options for your layout and click OK.
4. Choose Connectivity – Show Incomplete Nets and select the pseudo-parallel net.

**Caution**

Abutting instances with pseudo-parallel nets is order-dependent. If you manually abut the instances with the pseudo-parallel nets they will abut correctly. If you then abut instances that do not have pseudo-parallel nets they will not abut and the pseudo-parallel nets that were previously abutted will no longer be pseudo-parallel nets.
Generating a Layout with Chaining and Folding on Pseudo-Parallel Nets

When the `lxAllowPseudoParallelNets` environment variable is set to `t`, layout generation optimizes pseudo-parallel nets for implicit pseudo-parallel connections only.

**Note:** This also works with the `Create – Pick from Schematic` command.

To do this,

1. Add the following line to your `.cdsenv` file.
   ```
   layoutXL lxAllowPseudoParallelNets boolean t
   ```
2. From the layout window menu bar, choose `Design – Gen From Source`.
   The Layout Generation Options form is displayed.
3. Turn on the Transistor Chaining and Transistor Folding options.
4. Set the rest of the options for your layout and click OK.
5. To verify that the pseudo-parallel nets are complete, choose Connectivity – Show Incomplete Nets. Notice that the pseudo-parallel net connection is complete.
Using Pick From Schematic with Pseudo-Parallel Nets

You can set pseudo-parallel nets when using the Create – Pick from Schematic command if

- The Group As In Schematic option is turned on in the Pick From Schematic form.
- The nets are internal to the instances created in the scratch cellview.
- The setPPConn environment variable is set to t.
- All of the rules for pseudo-parallel nets are met.

For more information, see “Pseudo-Parallel Connections” on page 302.

Layer Selection

When working in the layout window, you can select the layer to work on directly in the Layer Selection Window or you can switch on the automatic tapping options to specify how Virtuoso XL chooses layers during interactive routing using wires, paths, and shapes.

Selecting a Layer in the Layer Selection Window

To select the layer you want to use to connect design components,

- In the Layer Selection Window, select the layer you want to use.
The layer you selected appears in the current display.

Note: The layout window Create – Path command automatically changes the current layer to the layer of the first pin you select to start the path.

Selecting a Layer Automatically when Creating a Rectangle, Polygon, or Path

When you use the Create – Rectangle, Create – Polygon, or Create – Path commands from overlapping shapes, the drawing layer changes automatically to be the same layer as the shape that you clicked on if Change Layer option on the respective form is switched.

- The Change Layer option on the Create Path form is controlled by the pathSwitchLayer environment variable.
- The Change Layer options on the Create Polygon and Create Rectangle forms are controlled by the shapeSwitchLayer environment variable.

Creating Paths Starting From Overlapping Layers

To change the drawing layer to the same layer as the shape that you first click on,

1. From the layout editor window, choose Create – Path and press F3 to show the Create Path form.
2. Switch on the Change Layer and Switch Layer Purpose options.
3. Move your cursor into the layout window and click on a shape.

   The LSW updates the current layer to be the same layer as the shape that you clicked on. The purpose of the layer is drawing, which is the first purpose listed in the Layer Purpose Order.

4. Change the Layer Purpose Order in the Create Path form to net pin drawing.

5. Click on a shape in the layout window.

   The current layer is the same layer as the shape that you clicked on but now has purpose net.

Creating Rectangles and Polygons Starting From Overlapping Layers

1. Set the shapeSwitchLayer environment variable to t in your .cdsenv file.

   layoutXL shapeSwitchLayer boolean t

   Note: You can also set the shapeSwitchLayer to t by turning on the Change Layer option in the Create Rectangle or Create Polygon forms.

2. Choose Create – Rectangle (or Create – Polygon) and move your cursor into the layout window and click on any shapes that overlap.

   The LSW updates the current layer to be the same layer as the shape that you clicked on.

Connecting Nets

To create paths that make the required connections between each device in the layout, you can use

- The layout window Create – Path command. The Create – Path command highlights the net you are wiring in both the schematic and the layout.

  For more information, see “Creating Paths” in the Virtuoso Layout Editor User Guide.

- Design shapes created by the layout editor: the Create – Rectangle, Create – Polygon, Create – Pin, and Create – Conics commands, which connect two or more
design elements. For more information, see “Connecting Nets with Design Shapes” on page 318.

Creating Paths

To create paths follow these steps.

1. From the layout window, choose Create – Path.
2. In the layout window, click where you want the first point of the path.
   Virtuoso XL highlights the corresponding net in the schematic and the layout.

   ➤ Important
   You cannot start a path where there is a short.

3. Continue clicking points to draw the path.

1st click ➤ 2nd click ➤ 3rd click ➤ Press Return

You draw the path centerline. ➤ Each click draws another segment. ➤ Press Return to close the path. ➤ The finished path appears.
Virtuoso XL creates the path using the layer of the shape where you click first. If you click where there is no shape, Virtuoso XL uses the layer selected in the LSW.

The default width of the path is the \texttt{minWidth} layer property defined in the technology file. For more information, see “Physical Rules” on page 34.

4. To change the width of the path, press \texttt{F3} to display the Create Path options form and type in the new value in the \textit{Width} field.

To erase the segments you draw, press the \texttt{Back Space} key.

If you try to start a path or a shape where there is already a path or a shape on more than one layer, a dialog box appears that shows you what layers are active and asks you to choose one.

The layers listed are determined by the \textit{Connectivity Extract to Level} option set in the \textit{Layout XL Options} form, as well as the \textit{Display Levels} set in the \textit{Display Options} form.

5. To complete the path, press \texttt{Return} on the last point.

\textbf{Note:} Do not double click to end a path. If you have set the layout window Design – Options – Display command to \textit{x-first} or \textit{y-first} snap modes, the double click might make a notch in the path, which triggers an error message and can cause errors in mask layout.

6. To change the current entry layer, choose a different layer in the LSW.

The current entry layer in the LSW is updated. The path you create uses the new layer.
Connecting Nets with Path Stitching

You can draw a path that switches from one layer to another, automatically placing an appropriate contact at the point where the layer changes. For more information, see "Path Stitching" in the *Virtuoso Layout Editor User Guide*.

To perform path stitching, follow these steps.

1. While the *Create – Path* command is active, press F3.
   
The Create Path form is displayed. For more information, see “Creating Paths” in the *Virtuoso Layout Editor User Guide*.

2. To drop a via and change from the current layer to a new layer, choose a different layer in the *Change to Layer* cyclic field.

3. Change any other options as required.
4. On the layout window, click where you want the via placed.

   Virtuoso XL places the via between the two layers. The next segment you draw appears on the new layer.

   You can then switch back and forth between layers, choosing the new layer from the Create Path form each time you change layers.

**Connecting Nets with Design Shapes**

To create design shapes (rectangles, polygons, or conic shapes—circles, ellipses, and donuts) that you can use to connect devices,

1. From the LSW, choose the layer to use.

2. From the layout window menu, choose the Create command and the design shape you want (Polygon, for example).

   The Create Polygon form is displayed. For more information, see “Creating Polygons” in the *Virtuoso Layout Editor User Guide*.

   ![Create Polygon Form](image)

3. Change the Snap Mode to the setting you want.

4. Click to place the first point.

5. Continue clicking points to draw the shape.

   Press the **Backspace** key to undo the previous segment you entered.
6. To complete the shape, press Return.

Note: The Virtuoso compactor does not compact nonorthogonal shapes or shapes other than paths, except for pins.

Checking Connectivity with Flight Lines

To see nets for which connectivity has not yet been made, you can use the Show Incomplete Nets command to turn on flight lines in the layout window. Flight lines show incomplete electrical connections between devices in each net, including
- Unrouted instance terminals with no pins
- Unrouted instance terminals with pins that are not extractable
- Instance terminals with dangling must-join instance pins.

The Command Interpreter Window (CIW) reports how many incomplete nets you have.

For more information on Show Incomplete Nets, see “Identifying Incomplete Nets” in the Virtuoso Layout Editor User Guide.

If you draw a path between two components that completes the connection, the flight lines disappear. If the path does not complete the connection, the flight lines remain.

To see flight lines in the layout window, from the layout window, choose Connectivity – Show Incomplete Nets.
Checking Connectivity with Markers

Markers are flashing boxes in the layout window that indicate electrical shorts or invalid overlaps. When the *Show Incomplete Nets* command is active (flight lines are displayed), the markers might be more difficult to see.

Finding Markers

To find markers, follow these steps.

1. From the layout window menu, choose *Verify – Markers – Find*.
   
   The Find Marker form is displayed. For more information, see “Finding Markers” in the *Virtuoso Layout Editor User Guide*.

2. Turn on *Zoom To Markers*.

3. Click *Apply*.

   Virtuoso XL zooms in on the error and warning markers one by one.

Explaining Markers

To find out what each marker means, follow these steps.
1. From the layout window, choose Verify – Markers – Explain.
   The layout window prompts you to point at a marker.

2. Click on the marker to explain.
   A dialog box appears identifying the location and explaining the short.

3. Click on another marker to explain, or press Escape to exit the command.

Deleting Single Markers

To delete (turn off) a single marker, follow these steps.

1. From the layout window, choose Verify – Markers – Delete.
   The layout window prompts you to point at a marker.

2. Click on the marker to delete.
   The marker disappears.

3. Click on another marker to delete or press Escape to exit the command.

Deleting All Markers

To delete all the markers, follow these steps.

1. From the layout window, choose Verify – Markers – Delete All.
The Delete All Markers form is displayed. Choose the type of markers to delete and the levels of hierarchy from which you want to delete them. For more information, see “Deleting All Markers” in the Virtuoso Layout Editor User Guide.

2. Indicate your choices on the form.

3. Click OK.

All the markers disappear.

Physical Vias

If you need a via that has more than five layers or one with special shapes not supported by symbolic vias, you can build a physical via using the layout editor.

Defining a Physical Via

To define a physical via,

1. From the CIW menu bar, choose File – New – Cellview.
The Create New File form is displayed.

2. Choose the library you want from the Library Name cyclic field and type the cell name you want to use in the Cell Name field.

3. Type layout in the View Name field, choose Virtuoso from the Tool cyclic field, and click OK.

A new layout window appears.

4. From the layout window menu bar, choose Create – Rectangle and draw one via shape as indicated in the picture below.

5. From the layout window menu bar, choose Create – Pin to create one pin on each of the two layers you want to connect. The names of the two pins must be identical.

6. Select the via in the layout window and from the menu bar, choose Design – Properties.
The Edit Cellview Properties form is displayed.

7. Click Property at the top of the form if it is not already selected.

![Edit Cellview Properties form](image)

For more information, see “About the Edit Properties Form” in the Virtuoso Layout Editor User Guide.

8. Click Add.

The Add Property form is displayed.

![Add Property form](image)

For more information, see “About the Add Property Form” in the Virtuoso Layout Editor User Guide.

9. Define the following properties:
10. To add each property, click Apply.

   The values appear in the Edit Cellview Properties form.

11. Click OK to accept the changes.

12. Choose Design – Save from the layout window menu bar.

   The via is saved.

**Using Via Mosaics**

The CDB version of Virtuoso XL supports vias and contacts that are implemented as mosaics. The OpenAccess version does not.

Mosaic instances for components are not supported on any database.

For more information, see “Via Mosaic Support (CDB only)” on page 302.
Using the Virtuoso Compactor on a Routed Design

If you want to compact a routed design with the Virtuoso compactor, remember that if you run the compactor using the layout window Tools – Compact command, the compactor recognizes symbolic connections only and it cannot compact all of the physical connections recognized by Virtuoso XL.

If you want the compactor to recognize physical connections, as well as symbolic connections, you must run the compactor when Virtuoso XL is running and use the layout window Compact – Compact command. You might have to stretch the layout window horizontally from its Virtuoso XL default size to be able to see the Compact menu on the far right.

To maintain connectivity after compaction if you run the compactor from the Tools menu, you must use connections made with the Create Path command, use paths with centerlines, and make sure the centerlines touch the centerlines of other paths to which you want to make connections or the connection regions of pins on electrically equivalent layers.

The best way to maintain logical connectivity while compacting a Virtuoso XL design is to set the syPresvOrigNet property on the design to Preserve Net before compacting.

For more information about the syPresvOrigNet property, see “Logical Connectivity” in the Virtuoso Compactor Reference Manual.
Online Forms

Virtuoso XL Connectivity Extraction

**Extract Connectivity to Level** specifies how much of the design hierarchy the extractor considers when extracting the design. For more information on this option, see `extractStopLevel`.

⚠️ **Important**

Whatever value you set for *Extract Connectivity to Level*, the extractor only ever extracts the top level; i.e., it only ever changes the connectivity of objects or creates markers at the top level of the design. To extract hierarchical cellviews, set the value of this option to a value greater than 0 and use the options in the *Hierarchical Cellviews* group box.

**Show Weak-Connect Violations** toggles the display of weak-connect violation markers in the current cellview. If you change the setting, you need to click OK to see the changes in the layout window.

**Show Must-Connect Violations** toggles the display of must-connect violation markers in the current cellview. If you change the setting, you need to click OK to see the changes in the layout window.

⚠️ **Important**

Weak-connect and must-connect markers are updated only if the cellview in question is highlighted in the *Select Cellviews* pane and extracted when the form is applied.

**Hierarchical Cellviews** lets you extract and save some or all of the hierarchical cellviews in your design. The options in this group are enabled only when *Extract Connectivity to Level* is set to a value other than 0.

- **Extract Hierarchical Cellviews** switches on the extraction of hierarchical cellviews
- **Save Extracted Cellviews** automatically saves the extracted cellviews.
- **Select Cellviews** lists the lower-level cellviews that need to be updated.
  - **Select** lets you filter the entries in the list using the library, cell, or view names. Type in the first characters of the names you want to select.
Note that the filter mechanism has no wildcard capability. For example, to select a cell called ‘resistor’, you must type ‘r’, ‘re’, ‘res’, and so on. If you type ‘sis’, the cell will not be selected.

**Number Selected** indicates the number of cellviews selected.

**Select All** selects all of the cellviews listed in the *Select Cellviews* pane.

**Deselect All** deselects all of the cellviews listed in the *Select Cellviews* pane.
Wire Editing

This chapter discusses the following:

- Introduction
- Requirements
- The Wire Editing Environment
- Enabling Wire Editing
- Routing Wires
- Preventing and Checking Design Rule Violations
- Routing Options and Styles
- Using Vias
- Editing Routed Connections
- Reporting
- Setting Constraints
- Troubleshooting
- Online Forms
Introduction

Wire editing within Virtuoso® XL Layout Editor (Virtuoso XL) environment, allows design rule correct interactive routing and editing features.

This section describes

- **Main Features**
- **Wire Editing Commands**
- **Virtuoso Router to Virtuoso XL Command Mapping**

Main Features

The following are the main wire editing features:

- Route design-rule-correct single and multiple wires based on connectivity
- Push wires and components while routing
- Input commands and constraints through a `.do` file
- Check existing routes for violations
- Reshape wire segments
- Copy existing routes to unrouted connections
- Generate reports about routing conditions or objects
- Eliminate notches and removes extra bends in selected wires
- Compact wires within a selected area

Wire Editing Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Create – Wire</em></td>
<td>Creates a single wire or multiples wires which are assigned to the net of the selected connectivity object(s). Wires are created DRC correct and can push routed wires and paths assigned to nets.</td>
</tr>
</tbody>
</table>
Virtuoso Router to Virtuoso XL Command Mapping

The interactive routing commands of the Virtuoso Router are mapped to the wire editing commands as follows:

<table>
<thead>
<tr>
<th>Virtuoso Router</th>
<th>Wire Editing Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit Route</td>
<td>Create – Wire</td>
</tr>
<tr>
<td>Move</td>
<td>Edit – Wire – Stretch / Edit – Wire – Split</td>
</tr>
</tbody>
</table>
Wire editing requires the following.

- **Rule Information**
- **Net Connectivity Information**
- **Routing Area Boundary**

### Rule Information

In addition to the Virtuoso XL requirements, you must define which layers and vias are to be used for routing. This information is defined in a rules file or in the technology file.

You can create a rules file interactively using the Rules Editor and either save the rule information to your technology file or to an ASCII file. An ASCII rules file takes precedence over rules specified in your technology file. To use the rules specified in an ASCII rules file, you can load the file using the *Routing Options* form, or you can also use environment variables to load the rules file.

For more information on using environment variables, see “Using Environment Variables to Load a Rules File” on page 340.

For more information about Virtuoso XL requirements, see Chapter 2, “Editing Your Technology File.”

For information about creating a rules file, see “Creating or Editing Rules” in the *Virtuoso Custom Placement and Routing Preparation Guide.*
For information about loading a rules file, see “Loading ASCII Rules Files” on page 338.

**Note:** If the rules file is not loaded or the rule information is not specified in your technology file, when you enable the wire editing a warning message is sent to the Command Interpreter Window:

```
icclayers are not defined
```

**Net Connectivity Information**

Connectivity source requirements are the same as those for the Virtuoso XL.

For more information, refer to “Connectivity and the Requirements” in the *Virtuoso Custom Placement and Routing Preparation Guide*.

**Routing Area Boundary**

The routing area boundary is used to confine the routing to a designated area. By default a boundary is generated by the *Gen From Source* command. You can also draw a boundary using the *Create – Rectangle* command. When creating a boundary you can specify a routing area boundary per layer, which allows areas of a cellview to be used only by that layer.

For information about defining a boundary layer for a specific routing layer, refer to “Defining Boundary Layers” in the *Virtuoso Custom Placement and Routing Preparation Guide*.

For more information about drawing a boundary, refer to “Defining the Design Boundary” on page 105.

**The Wire Editing Environment**

This section describes

- Valid Connectivity Objects
- Types of Routing Objects
- Net Assignment
- Preview Wires and Routing Aids
- Mouse Button Behavior
- Grid Controls
- Using Environment Variables
Valid Connectivity Objects

The following are valid connectivity objects used when routing wires.

- instTerm
- pin
- via instance having a pin on each layer and both pins on the same terminal
- existing wire
- path assigned to a net

Types of Routing Objects

path
A single layer object. Paths are constructed by defining coordinates, point by point, along a center line.

wire
A collection of paths and vias, or interconnecting layers, used to connect same net instTerms and pins of the cellview.

route
A collection of paths and vias, or interconnecting layers, used to connect same net instTerms, routes, and pins of the cellview. Depending on the topology, may also include multiple branches or T junctions.

Note: Virtuoso XL currently does not support route objects.
**net**

Logical implementation of a single signal or a group of signals that implements the connections to instance pins and pins. For example:

- \(a[0:56]\) - a net (vectored net).
- \(a[0]\) - a net (scalar net), and a signal.
- \(a[0]\) - a net member of \(a[0:56]\)

A net could be defined as \(A B C[0:7]\), and the signals defined as \(A, B, C[0]...C[7]\). Another net could be defined as \(C[4:6]\), with the signals defined as \(C[4], C[5],\) and \(C[6]\). Signal \(C[4]\) is part of both nets, one of which is a bundle that includes a vector and two scalar nets \((A B C[0:7])\), the other a vector net \((C[4:6])\).

Also see **signal**.

**net bundle**

A collection of nets.

**signal**

A database object that represents a scalar net. A signal maps to a scalar net and can have the same name as the scalar net. Signals and nets have different database IDs and they are accessed through different SKILL functions. From a signal, you can find all the nets that it is associated with. From a net, you can find all the member signals.

An analogy would be a phone network. The plastic bundle represents the net and the actual copper wires represents the signals.

Also see **net**.

**Net Assignment**

In the Virtuoso environment there are two types of nets, known and unknown nets.

- known nets: nets that are defined by pins and components (for example \(clkA\)).

  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. However, zero level geometry (not instTerms connections) can be assigned and reassigned as you edit.
If shape on a known net touches another shape on a known net, this will cause a short.

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

If a shape touches nothing, it is considered a floating net. When floating nets are assigned to a net, they are given the $lxStickyNet$ property. For more information, see “Routing Floating Wires” on page 344.

**Sticky Nets**

The $lxStickyNet$ property allows a shape to retain the assigned connectivity.

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, and the extractor is free to assign and reassign the shape to a net.

To assign a net to the power rail shape and not allow the extractor to re-assign to a net, you can assign a net name, such as vdd and the $lxStickyNet$ property is added automatically. The power rail will retain its connectivity.

As a result of this, if you create a connection between S-netA and the power rail shape you will create a short marker.

**Preview Wires and Routing Aids**

While you are routing, a preview wire of the active wire segment stretches from the cursor to the last point you digitized to show the intended wire. This preview wire is shown as a dashed outline to indicate the wire-to-wire clearance rule. Small arrows and alignment marks appear when the pointer aligns with a nearby wire, pin, or via that is on the net you are routing.

**Mouse Button Behavior**

- Clicking the middle button invokes the command-context-specific Layout pop-up menu for wire editing commands.

- Control-click enables the *Cycle Control Wire* when routing multiple wires.

- Shift-click enables the *Rotate Bus Cursor* when multiple wires.
Grid Controls

You can control the grid snapping of wires and vias while wire editing by,

- using a do file

or

- selecting either the manufacturing grid or the display grid from the layout menu.

Setting the Grid Using a .do File

To set grid snapping for wires and vias, follow these steps:

1. **Create a .do file** with the required snap values. For example

   ```
   # wire grid
   grid wire .1 metal2 (direction x) (offset 0)
   grid wire .1 metal2 (direction y) (offset 0)
   # via grid
   grid via .1 (direction x) (offset 0) via1
   grid via .1 (direction y) (offset 0) via1
   ```

2. **Load the .do file.**

   For more information about loading .do files, see “Loading .do Files” on page 394.

3. **Choose Create – Wire from the design window.**

   Using the above example, metal2 wires will snap to a .1 grid.

   **Note:** There currently is no automatic mapping of the Options – Display – Grid Control settings. A do file is required to set grid snapping.

Switch Between the Manufacture Grid and the Display Grid

When routing wires, you can press the middle button to open the Layout pop-up menu and select either Use manufacturing grid or Use display grid.

The display grid is defined under Options – Display and the manufacturing grid is defined in the technology file.
Using Environment Variables

For a list of environment variables available when wire editing, see “Environment Variables” on page 470.

Enabling Wire Editing

Wire editing is enabled once you have started Virtuoso XL.

For information about starting Virtuoso XL, see Chapter 6, “Generating Your Layout.”

Virtuoso XL Autorouting

You can route a design in batch mode using the Virtuoso XL Routing – Autorouting command. By setting the appropriate options using environment variables in a .cdsenv file, and by providing a do file with routing instructions, you can route the current design without explicitly invoking the router.

For more information, see “About the Autorouting Command” in the Virtuoso Custom Placement and Routing Preparation Guide.

Loading ASCII Rules Files

Rule information such as the layers and vias to be used when routing, minimum width rules, and the routing direction for layers can be specified in a rules file and saved to your technology file or an ASCII file. By default, the rules in your technology file are used. An ASCII rules file will take precedence over the rules specified in the technology file.

You can load a rules file through the Routing Options form or by specifying environment variables in a .cdsenv file. See “Using Environment Variables to Load a Rules File” on page 340.

To create a rule file, see “Creating or Editing Rules” in the Virtuoso Custom Placement and Routing Guide.

To load an ASCII rules file through the Routing Options form, follow these steps:

1. From the layout window, choose Options – Routing.
The Routing Options form is displayed.

2. Turn on Rules.

3. Choose the Rules file you want to use by doing one of the following:
   - In the Rules field, type in the name and path of the rules file.
   - Click on Browse.

   The Find File form is displayed.

   a. Navigate to the rules file.
   b. Choose the rules file.
   c. Click OK.

4. In the Routing Options form, click OK.

   The contents of the rules file are referenced when the next command is executed.
Note: Click Refresh in the Routing Options form to reload updated rule information in the following situations:

- Contents the rule file is changed.
- Name of the rules file is changed.
- A new rules file is being used.

Using Environment Variables to Load a Rules File

You can automate the loading of a rules files by using the following environment variables.

```
iccTranslator.ExportEditor rulesFile string "ruleFileName"
iccTranslator.ExportEditor useRulesFile boolean t
```

Routing Wires

Wire editing allows you to create design-rule-correct single or multiple wires between connectivity objects. While routing, you can insert vias to change the routing layer and push aside wires and paths assigned to nets, depending on the options you have set.

You can use the middle button Layout pop-up menu to access routing options. For details about setting up the routing environment and controlling the routing style, see “Routing Options and Styles” on page 349.

This section describes

- Routing a Single Wire
- Routing Multiple Wires
- Routing Floating Wires

Routing a Single Wire

To create a single wire, follow these steps:

1. From the layout window, choose Create – Wire.
When routing wires, you can use the middle button to open the Layout pop-up menu and change the width of a wire or Matching Wire Width and Pin Widths to the wire you are editing.

2. Do one of the following:
   - Create a floating wire.
   - Click or area select a connectivity object. See “Valid Connectivity Objects” on page 334.

Flight lines appear showing the connectivity of the selected net, and the status bar shows the name of the selected net.

The option Change Layer in the Create Wire form determines whether the entry layer will remain on the same layer or switch to the layer of the object you have selected. By default, the routing layer is changed to the same layer as the connectivity object. When starting a wire at a coordinate where there are overlapping layers or from an object with more than one layer, such as the source or drain of a MOS device, a dialog box appears showing you the selectable layers and prompts you to choose one. Turning the Change Layer option off disables the dialog box.

3. Digitize points where you want to route the wire.
Wires and vias are prevented from creating violations and will push aside other wires and vias if Push Routing is on. By default, Push Routing is on.

For information about changing layers, see “Changing Layers and Adding Vias” on page 369.

For information about violation checking, see “Preventing and Checking Design Rule Violations” on page 345.

4. Finish the wire by doing one of the following:

- From the Layout pop-up menu, choose Finish Routing.
  The connection is finished from the last digitized point. An area for the wire must lie within the routing search area or the automatic operation is terminated. Some connections may require layer changes and vias to be added in order to finish the connection.
- Continue digitizing points to complete one or more connections.
- Double click or press Return.
  The wire is terminated where the cursor is positioned.

**Note:** To erase the segments you draw, press the Backspace key. Each time you press Backspace, the last digitized segment is deleted.

**Routing Multiple Wires**

You can route arbitrary wire sets, buses, differential pairs, and bundles. All multiple wire routing is considered bus routing. Differential pairs and bundles retain their association after routing is completed. In general, only connections that use the same layers and vias can be routed together. The exception is tandem layer routing.

For details about setting up the routing environment and controlling the routing style for buses, refer to “Routing Options and Styles” on page 349.

When routing multiple wires, follow these steps:

1. From the layout window, choose Create – Wire.
   The layout window prompts you to enter a point or select an area to start the wire.
2. Select adjacent pins, vias, existing wires, or paths assigned to nets.
   You can Shift-click to add additional objects to your selection set.
Flight lines appear showing the connectivity of the selected nets, and the status banner shows the names of the selected nets.

**Note:** *Enable Bus Routing* must be turned on in the Routing Options form *Bus* tab. The default setting for *Enable Bus Routing* is on.

To select multiple wires on the same net, turn on *Allow Same Net Picking*. For more information, see “*Allow Same Net Picking*” on page 344.

3. Digitize points where you want to route the wire.

Wires starting from pins that are not at minimum spacing are gathered to the minimum separation at the first digitized point. To gather bus wires before the first digitized point, you can turn gathering bus wires on and also specify the spacing of wires.

When routing multiple wires, the pointer attaches to a control wire. *Cycling the control wire* can help you when you are changing the routing direction to avoid obstacles.

For more information about change layers, see “*Using Vias Patterns on Multiple Wires*” on page 370.

4. Finish the wires by doing one of the following:

- From the Layout pop-up menu, choose *Finish Routing*.
  
  The connections are finished from the last digitized point. Some connections may require layer changes and vias to be added in order to finish the connection. An area for the wire must lie within the routing search area or the automatic operation is terminated.

- From the Layout pop-up menu, choose *Route As Many As Possible*.
  
  If routing space is limited to fewer than all the wire segments of the set, you can choose to automatically drop the wires that cannot be routed, or have no new segments added. For more information, see “*Working with Insufficient Routing Space*” on page 356.

- From the Layout pop-up menu, choose *Route Only If All Succeed*.
  
  No new segments are added if there is insufficient routing space for all the wires of the set.

- Continue digitizing points to complete one or more connections.

- Double click or press *Return*.
  
  The wires will ended where the cursor is positioned.

**Note:** To erase the segments you draw, press the *Backspace* key. Each time you click
Backspace, the last digitized segment is deleted.

Allow Same Net Picking

The Allow Same Net Picking option is used to select multiple wires on the same net during area selection or by using Shift-click to add to a selected object. For example, you can route multiple floating wires which are assigned to the same net as show below.

1. In the Routing Options form, under the Bus tab, turn on Allow Same Net Picking.
2. Choose Create Wire.
3. Select the ends of the wires.

Routing Floating Wires

You can create floating wires, meaning the wire is not physically connected to a connectivity object.

To create a floating wire,

2. In the Routing Options menu, turn on Allow Floating Nets and click OK.
4. Optional – To assign the wire to a net, in the Create Wire form, type a name in the Net Name field.

When floating nets are assigned to a net, they are given the lxStickyNet property.

Note: When creating floating nets which you want to be sticky nets, do not use the prefix of sp_net, which is used internally by the software. Assigning a net name with the prefix of sp_net to sticky nets could result in the loss of data or loss of the assigned lxStickyNet property.
For information about net assignment and sticky nets, see “Net Assignment” on page 335.

**Preventing and Checking Design Rule Violations**

Wire editing supersedes DRD Editing. Wires created or edited are not checked for violations interactively in DRD Editing. Violations are displayed as post-edit markers at the end of the command, when post-edit DRD checking is on.

This section describes

- **Interactive Checking**
- **Same Net Checking**
- **Checking Regions**
- **Checking Route and Pin Violations**

**Interactive Checking**

Prevent clearance rule violations between routing objects such as wires, vias, and pins while routing by doing the following:

1. From the layout window, choose *Options – Routing*.
   
   The *Routing Options* form is displayed.

2. Turn *Interactive Checking* on.

   Conflicts and design rule violations such as clearance rules are not allowed. *Interactive Checking* also prevents routing outside of the boundary area.

   **Note:** When you are in an active command, you can turn interactive checking on by choosing *Turn checking on* from the *Layout* pop-up menu.
You can check for specific types of routing and pin violations through the Check Routing form. For more information, see “Checking Route and Pin Violations” on page 347.

**Same Net Checking**

By default, same net violations are not checked. Prevent same net violations such as gaps, notches, and self-intersecting nets by doing the following:

**Note:** The *Same Net Checking* option can slow the performance of wire editing. In best practice, this should be used after you have finished all routing and before running final verification on your design.

1. From the layout window, choose *Options – Routing*.
2. The *Routing Options* form is displayed.
3. Turn *Same Net Checking* on.
   - Same net gaps, notches, self intersecting nets, or loops are not allowed.
   - You can check for existing same-net violations through the Check Routing form. For more information, see “Checking Route and Pin Violations” on page 347.

**Checking Regions**

*Check Region* controls whether wire segments are checked at region boundaries when you are creating or editing wires. A region is a routing area where you want different clearances or width rules to apply. For more information about creating regions, see “Creating Regions” on page 347.

When *Check Region* is on, wire segments are halted at a region boundary until you click at the boundary. A pseudopin is inserted for each net, and then you can continue routing or moving the wire segments into or out of the region. If you are routing multiple wires that are not part of a bundle or differential pair, the wires fan out to accommodate different width or clearance rules.

When *Check Region* is off, region boundaries and rule differences are ignored when you route or move wire segments into or out of a region.

Control checking of regions by doing the following:

1. From the layout window, choose *Options – Routing*.
2. The *Routing Options* form is displayed.
3. Turn Check Region on.

**Creating Regions**

Define routing areas where you want different clearances or wire widths by creating a `.do` file with region and clearance rules. Region rules have the highest precedence in the rule hierarchy; therefore, the region clearance and width rules override all other clearance and width rules.

To define a region, follow these steps:

1. **Create a `.do` file** to define a region and assign rules to the region using the following syntax.

   Choose the shape (rectangle or polygon), specify its layer and location, and assign the clearance rules and width rules you want to apply within the region. You can also assign an optional region name. For example:

   ```
   define (region REG1 (rect metal1 10.5 2.75 55.05 5.5) (rule (width .8) (clearance 1.2 (type via_wire)))
   ```

   You can assign rules for the entire region, for a single class or net within the region, or between two classes within the region.

   For more information about using the `define` command to create regions, see *Define – Region – By Coordinates* in the *Virtuoso Chip Assembly Router User Reference*.

2. **Load the `.do` file.**

   For more information about loading `.do` files, see “*Loading do Files*” on page 394.

**Checking Route and Pin Violations**

Use the Check Routing form to control which rule violations and pin data problems are checked and graphically marked.

To check specific types of violations, doing the following:

1. From the layout window, choose *Verify – Check Routing*. 
The Check Routing form is displayed.

2. Turn on the options you want to check.

3. Click OK.

The checking process is initiated.

**Checking Violation Markers**

Violations will appear as markers, which can be checked using the Verify – Markers – Find command.

To check markers, see “Checking Connectivity with Markers” on page 320.
Routing Options and Styles

This section describes

- Matching Wire Width and Pin Widths
- Matching Wire Width and Pin Widths for Multiple Wires
- Gathering Bus Wires
- Spacing for Gathered Bus Wires
- Overriding Bus Spacing
- Rotating the Bus Cursor
- Cycling the Control Wire
- Working with Insufficient Routing Space
- Allowing Redundant Wiring
- Allowing Orthogonal Jogs
- Route To Cursor
- Using Alternate Views
- Connecting Multiple Component Pins
- Pushing Routes and Components
- Routing Shielded Nets
- Routing Tandem Layer Pairs
- Showing Length Rules Constraints

Matching Wire Width and Pin Widths

Matching the wire width and pin width can be set with using the following layout environment variables in the .cdsenv file or through menu items.

Setting the Wire Width to Match a Pin or Wire Using Environment Variables

matchPinWidth

Sets the width of the active wire segment to the width of the selected pin.
matchPinWidthValue

Sets the width of the active wire segment to the width of the narrow or wide edge of the selected pin and is active only when matchPinWidth is t.

matchTargetPinWidth

Sets the width of the active wire segment to the width of the narrow or wide edge of the target pin.

matchTargetPinWidthValue

Sets the width of the active wire segment to the width of the narrow or wide edge of the target pin and is active only when matchTargetPinWidth is t.

matchWidthSticky

Preserves the matchPinWidth, matchWireWidth, and matchPinWidthValue values.

The following will occur if matchPinWidth, matchWireWidth, matchPinWidthValue matchTargetPinWidth, matchTargetPinWidthValue values are set, and matchWidthSticky is t,

When the wire is finished the subsequent Create Wire command or session will not reset the environment variables to the default.

The environment variables values are pre-set for the first Create Wire session.

matchWireWidth

Sets the width of the active wire segment to the width of the wire segment you are editing.

Preserving the Match Width

When matchWidthSticky is t, matchPinWidth, matchWireWidth, matchPinWidthValue values are not reset when new wire editing session is started or when a new Create Wire command is performed.

Setting the Wire Width to Match a Pin or Wire Using Menu Items

To set the wire width to match a pin or an existing wire, follow these steps:

1. From the layout window, choose Create – Wire.
The Create Wire form is displayed.

**Note:** You can also use the middle button to open the Layout pop-up menu and choose any of the matching options once the Create – Wire command is active.

2. Click or area select the pin or existing wire to be routed.

3. Do one of the following:
   - To match the wire width to the width of the pin you are editing, turn on Match Pin Width and either choose Narrow, which matches the wire width to the shortest side of the pin, or Wide which matches the wire width to the longest side of the pin.
   - To match the wire width to the width of the wire you are editing, choose Match Wire Width.

The width change is not implemented if it introduces a clearance violation.
Matching Wire Width and Pin Widths for Multiple Wires

To match the wire width or specify a width for multiple wires, follow these steps

1. From the layout window, choose Create – Wire.
2. Click or area select the pins or wires to be routed.
3. Turn on Match Pin Width or Match Wire Width.

When multiple wires and pins are selected and Match Pin Width or Match Wire Width are on, AS IS is displayed in the Width field.

4. Do one of the following:
   - Type the width to be used for all wires in the Width field.
   - To match each of the wire widths to the width of the corresponding pins, turn on Match Pin Width and either choose Narrow, which matches the wire width to the shortest side of the pin, or Wide which matches the wire width to the longest side of the pin.
   - To match each of the wire widths to the width of the corresponding wire, choose Match Wire Width.
Gathering Bus Wires

Wires starting from pins that are not at minimum spacing are gathered to the minimum separation at the first digitized point. You can gather bus wires and change the spacing between wires before digitizing points by doing the following.

➤ From Layout pop-up menu, choose *Turn gather bus wires on*.

The bus cursor changes and the wires are gathered to the minimum spacing rule. You can change the orientation of the bus cursor by using the *Rotate Bus Cursor* command.

Spacing for Gathered Bus Wires

You can change the default wire-to-wire spacing for gathered bus wires. You might want to choose spacing based on the via pattern you use when you change layers and add vias. For more information, refer to “Adding Vias to Gathered Bus Wires” on page 371.

1. From the Layout pop-up menu, choose *Set up*. 
The **Routing Options** form is displayed.

2. Click on the **Bus** tab and choose from the following spacing options:

- **Wire-Wire**
- **Wire-Via**
- **Via-Via**
- **Specify Clearance**

If you want to change the spacing while you are routing multiple wires, choose **Setup** from the Layout pop-up menu, make your change, and regather the wires. You can also override bus spacing when you are preparing to change routing layers. For more information, see “**Overriding Bus Spacing**” on page 355.
Overriding Bus Spacing

You can override the current spacing for gathered bus wires and make the spacing change take place immediately on the layer you are routing, or you can have the change take place when you add a via pattern and change routing layers.

1. From the Layout pop-up menu, choose Set up.

   The Routing Options form is displayed.

2. Do one of the following:

   - If you want the new spacing to take effect immediately on the current routing layer, choose Use Override and type a new spacing value in the Override Value field.

   - If you want the new spacing to take effect on the next routing layer after you add a via pattern, choose Use Override On Next Via and type a new spacing value in the Override Value field.

3. Click OK.
Note: If you want to return to the default spacing value while routing the multiple wires, choose Ignore Override.

Rotating the Bus Cursor

When you gather wires, the bus cursor automatically orients perpendicular to the set of active wire segments.

To rotate the cursor, do the following:
➤ While routing, press Control-right-click.
  
  The cursor will toggle back and forth between horizontal and vertical each time you press Control-right-click.
  
  After you rotate the bus cursor, the cursor orientation stays fixed for the current routes. If you want to change the orientation, you must use the command again.

Cycling the Control Wire

As you are routing multiple wires, your cursor points to a wire segment called the control wire. The control wire serves as a pivot point and allows you to control the position of via patterns by changing which wire segment the cursor attaches to as you route the set of wires. By default, the center wire is the control wire and is visually identified by a white outline on the end of the control wire.

To cycle the control wire, do the following:
➤ While routing, press Shift-right-click.
  
  Repeatedly using this command cycles the control wire between the two outermost wires and the center wire of the set.

Working with Insufficient Routing Space

If there is insufficient routing space for all the wires of a multiple wire set, you can drop wires and finish routing a portion of the wires or have no new wire segments added.

To drop wires and route a portion of the wire set, do the following:
➤ From the middle button Layout pop-up menu, choose Route As Many As Possible.
  
  As many wires as possible are routed and the rest are dropped. The dropped wires are replaced by guides drawn from the last digitized locations to pins on their respective nets.
and you can continue routing a subset of wires. However, even with the option set on, no new segments are added if there is insufficient space to route at least one wire of the set.

If wires are dropped you can use the *Picking Up Dropped Wires* command to restart the set from a new location and reroute the dropped wires.

To have no new segments added, do the following:

➤ From the Layout pop-up menu, choose *Route Only If All Succeed*.

   No new segments are added if there is insufficient routing space for all the wires of the set.

**Picking Up Dropped Wires**

If you chose *Route As Many As Possible* from the Layout pop-up menu, wires that can cause violations are dropped as you route a multiple wire set. You can restart some or all of the dropped wires at the currently digitized location of the set. Only the wires that do not introduce violations are restarted, and guides are drawn from the new locations back to the locations from which these wires were dropped. You can then route the set of picked up wires from the new locations.

To route the dropped set of wires, follow these steps:

1. Click to digitize the locations of the undropped wires.

2. From the Layout pop-up menu, choose *Pick Up Dropped Wires*.

   All the wires of the set that do not create violations restart at locations consistent with the currently digitized locations.

3. Continue routing the set of wires.

4. After you complete the routing of the dropped wires, you must route each dropped wire from the location where it was dropped to the location where it was restarted.

**Allowing Redundant Wiring**

Redundant wiring requires two steps. The first step is creating wire. The second step is editing the wire and creating a loop or crossing over the wire by adding an additional wire. This is considered redundant wiring and by default is not allowed. You might need redundant wires on selected nets for increased current capacity or to create power rings. Redundant wiring is permitted on nets when *Allow Redundant Wiring On Enabled Nets* is turned on and the *allow_redundant_wiring* rule is attached to the net.
Note: Self intersecting wires are allowed by default. When you create a wire and cross over the wire in one step, this is considered a same-net violation and by default is not checked. For more information about same net violations, see “Same Net Checking” on page 346.

To allow redundant wiring, follow these steps:

1. Create a .do file with the following syntax:
   
   ```
   rule net netName (allow_redundant_wiring on)
   ```

   You can also apply the allow_redundant_wiring rule to a class of nets. For example:
   
   ```
   rule class className (allow_redundant_wiring on)
   ```

2. Load the .do file.
   
   For more information about loading .do files, see “Loading .do Files” on page 394.

3. From the Routing Options form, turn Allow Redundant Wiring On Enabled Nets on.

   Wire loops are allowed on nets with the attached allow redundant wiring rule.

Allowing Orthogonal Jogs

You can permit orthogonal jogs in existing wires when they are pushed with the Create Wire, Stretch Wire, and Pull commands by selecting Allow Orthogonal Jogs. The Push Routing option must be on to enable wire pushing. By default, Allow Orthogonal Jogs is turned on.

To permit orthogonal jogs, do the following:

➤ From the Routing Options form, turn Allow Orthogonal Jogs on.

To prevent orthogonal jogs, do the following:

➤ From the Routing Options form, turn Allow Orthogonal Jogs off.

Note: Also see Route To Cursor.

Figure 10-1 Orthogonal Jogs
Allow Orthogonal Jogs turned on

Allow Orthogonal Jogs turned off

Route To Cursor

*Route To Cursor* creates wire segments, on single or multiple layers, that follow the direction of the cursor from the previously digitized point.

To use the *Route to Cursor* command, do the following:

1. From the layout window, choose *Create – Wire*.
2. From the layout window, choose *Options – Routing*.
The Routing Options form is displayed.

3. Turn Route To Cursor on.

Note: To avoid unexpected jogs, press the Shift key when routing to cursor. Pressing and holding the Shift key constraints the mouse motion which help to keep the x or y coordinates unchanged.

Single Layer

To route multiple wire segments on a single layer, do the following:

1. Turn Single Layer on and click Apply.

2. Click or area select a connectivity object (pin, via, or an existing wire).
   The multi-segment wire is routed on the active layer and follows the direction of the cursor.

Multiple Layers

To route wires on multiple layers, do the following:

1. Turn Multiple Layers on and click Apply.
2. Click or area select a connectivity object (pin, via, or an existing wire).

   The wire follows the cursor and changes layers. Vias are added automatically as layers change.

   For example, when routing a `metal1` wire and crossing over another `metal1` wire with the cursor, wires will not be pushed, the layer is changed, and a via is automatically inserted.

**Follow Layer Direction**

To route wires that follow the layer directions set in the rules file, do the following:

1. From the Routing Options form, turn *Multiple Layers* on.
2. Turn *Follow Layer Direction* on and click *Apply*.
3. Click or area select a connectivity object (pin, via, or an existing wire).
The wires are routed on multiple layers, vias are added automatically as layers change, and the layer direction set in the rules file is strictly enforced.

For example, when the routing direction in the rules file is as follows:

```
metal1 vertical
metal2 horizontal
```

When routing from a metal1 pin in the horizontal direction, the layer direction is strictly followed; a via is inserted and the wire is routed on metal2. The layer direction continues to be followed as the wire is routed and vias are automatically inserted as the layer direction changes.

**Note:** The *Route To Cursor* setting is not available when you are routing a bus, bundle, or differential pair.

**Using Alternate Views**

You can use the *Alternate Views* option in the Routing Options form to specify views other than the layout view, such as an abstract view of your data. If you have abstract views, choosing to use them rather than the layout views can make run time faster because there is much less data to be processed by the software.
Connecting Multiple Component Pins

*Multiple Component Connection* lets you connect multiple component pins (terminals of components) or top level component pins (instance pins) to the same net.

Use the Create Wire command to connect multiple same net component pins that are in alignment by clicking on the two outermost pins. The resulting wire contains connections to each pin and separate wire segments between each adjacent pair of pins.

**Note:** If the pins are not in alignment, a single wire is created with connections only to the pins that are clicked on. Shape pins (top-level shapes that are also pins) are considered io_ports and will not be connected using the *Multiple Component Connection* option.

For more information, see “Pins” in the *Virtuoso Custom Placement and Routing Preparation Guide*.

Pushing Routes and Components

**Push Routing**

*Push Routing* controls whether wires and vias are pushed aside while routing to maintain current clearance rules. By default *Push Routing* is on.

The *Create – Wire*, *Stretch Wire*, and *Split Wire* commands can be used to push wires.

To control pushing of wires, do the following:

1. From the layout window, choose *Options – Routing*.
   - The *Routing Options* form is displayed.
2. Turn *Push Routing* off or on.

**Note:** When *Push Routing* is on, use *Allowing Orthogonal Jogs* to control whether
orthogonal jogs are allowed in the nets you are pushing.

Pushing Components

*Push Component* controls whether components are pushed aside while routing if there is insufficient space to route the wire.

Only the *Stretch Wire* command can be used to push components.

To control pushing of components, do the following:

1. From the layout window, choose *Options – Routing*.
   
The *Routing Options* form is displayed.

2. Turn *Push Component* off or on.

Components are pushed aside only if there is insufficient routing space. If there is sufficient routing area for the wires you are routing, the wires are routed around the component, rather than push the component.

Routing Shielded Nets

The *Auto Shield* option automatically routes shield wires around nets that have a shield rule attached to the net.

To route shielded nets, follow these steps:

1. Create a .do file using the *circuit* and *rule* commands to assign shielding rules to the net.

   Choose the type of shield (parallel, tandem or coax), specify the net to be used as a shield, and assign the overhang, width and gap rules you want to apply. For example:

   ```
circuit net NET5 (shield on (type parallel) (use_net GND))
   rule net NET5 (tandem_shield_overhang 1.5)
   rule net NET5 (shield_width 1)
   rule net NET5 (shield_gap 1.5)
   ```

   For more information about shielding rules, refer to *shield_descriptor* in the *Virtuoso Chip Assembly Router User Reference*.

2. Load the .do file.

   For more information about loading .do files, see “Loading .do Files” on page 394.

3. From the Routing Options form, turn *Auto Shield* on.
4. From the layout window, choose *Create – Wire*.

5. Click or area select a connectivity object (pin, via, or an existing wire) of the net you want to shield.

6. Digitize points where you want to route the wire.

7. Finish the wire by doing one of the following:
   - From the Layout pop-up menu, choose *Finish Routing*.
     The connection is finished from the last digitized point. An area for the wire must lie within the routing search area or the automatic operation is terminated. Some connections may require layer changes and vias to be added in order to finish the connection.
   - Double click or press *Return*.
     The selected net is automatically routed with a shield.

     The shield command requires sufficient clearance around the wires to be shielded. Wherever there is insufficient space, the command leaves a gap in the shield wire.

8. Tie down the shielding net to the source.

     The shielding net is left floating and must to be manually connected to the source.

**Routing Tandem Layer Pairs**

To route tandem layer pairs, follow these steps,

1. From the layout window, choose *Create – Wire*.
   
   The layout window prompts you to enter a point or select an area to start the wire.

2. From the *Layout* pop-up menu, choose *Set up*.

3. Click the *Bus* tab.

4. Turn on *Enable Tandem Pair*.

5. Click or area select the adjacent connectivity objects of the layer pairs you want to route.
   
   The *Choose which layer to use* form is displayed.

6. Choose a layer and click *OK*.

   Wires are routed on both layers.

7. Digitize points where you want to route the wire.
The wires are gathered and overlaid at the first digitized point.

**Showing Length Rules Constraints**

To display wire lengths relative to length rules as you are routing wires,

1. **Create a .do file** using the `circuit` command to assign length rules to the net.
   
   For example:
   
   ```plaintext
circuit net NET5 (length 6)
circuit net CK (length 14 5)
```
   
   Minimum and maximum rules can be set.

2. **Load the .do file.**
   
   For more information about loading .do files, see “Loading .do Files” on page 394.

3. From the Routing Options form, **Show Timing/Length Rule Constraints** field, turn on `Meter`, `Octagons`, or both `Meter` and `Octagons`.

4. From the layout window, choose **Create – Wire.**

5. Click or area select a connectivity object (pin, via, or an existing wire) of the net which is assigned the length rule.

6. Digitize points where you want to route the wire.

**Note:** Rectangles are not included in length rule calculations, only wires are considered when displaying length rules. The wire length is calculated from pin-edge to pin-edge.

**Using the Meter Display**
The meter indicator displays the status of the wire length relative to minimum and maximum length limits for the wire. The meter indicator consists of a horizontal line with two vertical tick-marks labeled MIN and MAX. These marks indicate the minimum and maximum length limits. As you create a wire with assigned length rules, an X appears on the horizontal line at a position relative to the MIN and MAX marks. The X indicates the current wire length relative to minimum and maximum length rules for the net.

If the current wire length is between the minimum and maximum length limits, there is no length rule violation. If the X is to the left of the MIN mark or to the right of the MAX mark, the current wire length violates the minimum or maximum length rule. If you set only a minimum or only a maximum length rule, the meter graphic displays a single MIN or a single MAX limit line. Also, you might not see both limit lines, if the current wire position is too far from one of the limits. The line for the rule limit you are closest to is always displayed.

A signed number appears above the MIN or MAX limit mark and indicates the distance to the closest length limit. The displayed number switches from one limit line to the other as you route the current wire across the midpoint distance. Negative numbers mean the current length is below a limit, and positive numbers mean that the current length is above a limit. The number is displayed in green if length rules are not violated and in red if any length rule is violated.
Using the Octagons Display

You can use the octagon rule display to compare wire length to length or timing rules while you are routing a connection. The octagon bounds the area that you can route within from the current pointer position without violating timing or length rules. Two octagon types are available, dependent on what type of timing or length rules are set for the wire you are routing.

If a maximum timing or length rule applies to the current wire, a maximum octagon encloses the routing area. This octagon is drawn assuming orthogonal routing and allows only one "U turn." If a maximum timing or length rule is not set for the wire you are routing, or the wire exceeds the maximum length rule, this maximum octagon is not displayed.

If a minimum timing or length rule applies to the current wire, a minimum octagon appears when the wire length is below the minimum rule. This octagon shows the area to route within in order to comply with the minimum length rule. When the minimum octagon appears, you must cross the minimum octagon boundary and add enough wire length to satisfy the minimum length or timing rule.
The minimum octagon is calculated by assuming orthogonal routing and by allowing only one "U turn." After the wire you are routing exceeds the minimum length or if no minimum length rule applies, this octagon is not displayed.

If you are editing a net that has a net timing or length rule, the Manhattan lengths of all guide wires of the net contribute to the current net length.

**Using Vias**

You can change layers and add vias while routing individual wires or you can change via patterns while routing multiple wires.

This section describes

- Changing Layers and Adding Vias
- Using Vias Patterns on Multiple Wires
- Adding Vias to Gathered Bus Wires
- Legal Via Sites
- Rotating Vias
- Minimum Number of Cuts
- Pseudo Vias
- Stacking Vias

**Changing Layers and Adding Vias**

To change layers and add a via while routing, do one of the following:

- From the Layout pop-up menu, choose *Add Via*.

  The *Add Via* form is displayed and all available vias and routing layers that can be reached from the current routing layer are displayed. Choose the applicable layer you want to switch to or choose a via type on the layer.

  Inaccessible layers are grayed out.

  **Note:** You can control whether you can add a via to the current layer without changing layers when using the *Add Via* form with the `allowViaOnCurrentLayer` environment
Vias are added to the last digitized point.

- Press the space bar.

If only one layer and one via are available for selection, vias are placed and the layer is changed to the only available layer. If the current routing layer has more than one layer available for selection, the Add Via form is opened for you to choose a layer or via type.

If you try to add vias to locations that violate design clearance rules, you can use **Via Assistance** to display Legal Via Sites.

Vias can be added to the current layer without changing to another layer. For information about how to disable the ability to add a via without changing layers, see **allowViaOnCurrentLayer** on page 610.

**Using Vias Patterns on Multiple Wires**

The procedure for adding a set of vias when routing multiple wires is the same as the procedure for adding individual vias, but the result is that multiple vias are added simultaneously in a predefined pattern. You can choose from several predefined via patterns: *Perpendicular, Diagonal 1, Diagonal 2, Stagger, Out Taper, and In Taper*.

To add vias and choose via patterns while routing multiple wires, follow these steps:
1. From the Layout pop-up menu, choose Via Pattern.

   The Via Pattern Pop-up appears.

2. Click the icon of the via pattern you want to use.

   Diagonal 1 is the default via pattern. The display indicates how the pattern would be placed for the current routing direction.

   If the selected via pattern cannot be placed, single circles are displayed indicating the via site are not legal. Use the Fit Via Pattern option to search for nearby available via sites for those vias that have clearance conflicts. For more information, refer to Legal Via Sites.

Adding Vias to Gathered Bus Wires

The following are guidelines for adding vias to gathered buses.

When the bus wire spacing is set to the following:

- wire-via
  Use any via pattern except the perpendicular pattern to keep the wires from spreading.

- via-via
  Use all patterns without spreading the wires.

- wire-wire
  All via patterns are likely to spread the wires (if the wire-wire clearance rules specify the smallest spacing possible).

  After the vias are added, the wires will gather back to minimum wire spacing.
**Note:** For non-gathered bus wires, if the required via spacing differs from the minimum wire spacing, the wires are spread to accommodate the via spacing and the new via pitch is used as the new wire spacing.

### Legal Via Sites

A legal via site is a location where you can add a via during routing and not violate design clearance rules.

When the add via operation fails, you can use *Via Assistance* to display legal via sites. When vias are placed using *Via Assistance*, via pattern resembles the original pattern but with some via positions modified. If *Snap* or *Display* are turned on, legal via sites are displayed as concentric circles. Single circles are via sites that are not reachable from the current digitized location.

- If *Via Assistance* is off, no via sites are displayed when you try to add a via to an illegal site.
- If *Snap* is on and the digitized point is within one via radius of a legal site, the tool routes to the legal via site and adds a via. If it is not within one via radius, legal via sites are displayed and no via is added.
- If *Display* is on and the last digitized point is a legal site, a via is added. If it is not a legal via site, nearby legal via sites are displayed and no via is added.

### Rotating Vias

Use the *Rotate* button on the *Add Via* form to rotate vias 90 degrees as you place them. If the last digitized point is on or within one via radius of a legal site, a rotated via (or via pattern for multiple wires) is added automatically. Otherwise, the legal via sites in the area are displayed (when *Via Assistance* is on), and you can click one of them to add the rotated via or via pattern.
Note: If you are routing multiple wires, all the rectangular vias of the via pattern are rotated.

Minimum Number of Cuts

The minimum number of cuts rule specifies the number of cuts a via must have when it is on a wide wire or pin whose width is greater than \( \text{minWidth} \).

The \text{minNumCut} rule is stored in the technology file as \text{tableSpacing}, one-dimension table indexed by width inside the \text{physicalRules} section. The rule is specified only for layers with material type \text{li} or \text{cut}.

For more information, see the \textit{Technology File and Display Resource File ASCII Syntax Reference Manual} or “Technology File Class physicalRule” in the \textit{Virtuoso Custom Placement and Routing Guide}.

Pseudo Vias

Pseudo vias are used in the case of two layers that are self-conducting. They are defined in your rules file and used internally by Virtuoso XL but never appear in the DFII database.

For information about defining pseudo vias in a rules file, see “Local Interconnects (Pseudo Vias)” in the \textit{Virtuoso Custom Placement and Routing Preparation Guide}.

Using Default Vias

The environment variable \texttt{layout useDefaultVia} is supported when using \texttt{Route to Cursor} or \texttt{Finish Routing}. When \texttt{useDefaultVia} is \texttt{t}, you can specify which of multiple vias is the default by adding the device property in the technology file, for example:

\begin{verbatim}
tfcDefineDeviceProp(
    ; (viewName    deviceName    propName    propValue)
    ( symbolic M1M2 t )
    ( symbolic M1M2TYPE2 defaultVia t )
    ( symbolic M2M3TYPE2 defaultVia t )
) ;tfcDefineDeviceProp
\end{verbatim}

The \texttt{viaNameTYPE2} vias are automatically picked. If \texttt{useDefaultVia} is set to \texttt{nil}, then a popup will appear when multiple vias are available, to allow you to choose which one to use.
Stacking Vias

To stack vias, do the following:

1. **Create a .do file** using the `rule` command to allow via stacking.
   
   For example:
   ```plaintext
   rule IC (stack_via any_overlap (stack_cut))
   ```
   
   For more information about length rules, refer to the *Virtuoso Chip Assembly Router User Reference*.

2. Load the .do file.
   
   For more information about loading .do files, see “Loading .do Files” on page 394.

3. From the layout window, choose *Create – Wire*.

4. From the Layout pop-up menu, choose *Add Via*.
   
   The **Add Via** form is displayed and all available vias and routing layers that can be reached from the current routing layer are displayed.

5. From the Add Via form select the correct via on an adjacent layer.
   
   **Note:** You are allowed to choose a layer that is not adjacent to the current layer, however, this does not allow you to choose a via. In most cases your design rules will require you to place vias with extended overlaps when stacking vias. To stack specific vias you must stack each via individually.

6. Once the first via is placed, again from the Layout pop-up menu, choose *Add Via*.

7. From the Add Via form select the correct via on an adjacent layer.
   
   You can continue to place and stack vias. Via stacking adheres to the stack via depth rule, `prMaxStackVias`.

---

**Editing Routed Connections**

- Stretching Wires and Vias
- Splitting and Stretching Wires
- Copying Routes
- Removing Notches and Extra Wire Bends
- Compacting Wires Using Pull
Stretching Wires and Vias

When using the Stretch Wire command, you can stretch a wire segment while maintaining the minimum spacing and width rules, and push aside existing wires.

To stretch a wire segment follow, these steps:

1. Choose Edit → Wire → Stretch.
   
The layout window prompts you to select a figure to be stretched and the design window shows the active command as Stretch Wire.

2. Click or area select the wire you want to stretch.

3. Move the cursor and click to place the wire.

The following options affect the Stretch Wire command.

- Interactive Checking
  When Interactive Checking is on, wires are prevented from creating violations.

- Push Routing
  When Push Routing is on, the stretched wire will push aside other wires and vias.

- Push Components
  When Push Component is on, the stretched wire will push aside components if there is insufficient space to stretch the objects.

- Allow Orthogonal Jogs
  When Allow Orthogonal Jogs and Push Routing are on, orthogonal jogs are created in wires that you are pushing.

**Note:** Press the Escape key to terminate the Stretch Wire command.

Splitting and Stretching Wires

The Split Wire command reshapes wires by cutting and stretching segments while maintaining spacing and width rules. The steps include cutting selected segments, using the right mouse button to cycle to the segments you want to stretch, and stretching the segments to the final position.

To split and stretch wires, follow these steps:

1. Choose Edit → Wire → Split.
2. Select the wires you want to split.

The design window shows the active command as *Split Wire*.

The Split form is displayed.

3. Digitize the points of the split line.

The split line in the example above creates three separate sets of stretchable wire segments.

4. Press Return after the last digitized point of the split line.
5. Choose the segments you want to stretch by clicking the right button and cycling through each set of wire segments.

6. Stretch the wire segments to the new location and click to place the segments.

The following options affect the split command.

- **Interactive Checking**

  When *Interactive Checking* is on, the stretched wire segments are prevented from creating violations.
Push Routing

When Push Routing is on, the stretched wire segments will push aside other wires and vias.

Allow Orthogonal Jogs

When Allow Orthogonal Jogs and Push Routing are on, orthogonal jogs are created in wires that you are pushing.

Copying Routes

You can copy existing wires and vias (complete or incomplete connections, but not floating nets) to unrouted connections that have similar lengths and topologies. You can copy single wires to one or more target pins or a set of wires, and you can mirror the copied wires along the x axis and y axis.

To copy a single route or multiple routes, follow these steps:


   The layout window prompts you to select the route to be copied.

2. Select the route you want to copy.

   The layout window prompts you to select the next route segment or point to the destination location.
   - Click once to select one segment of a route.
   - Shift-click to add additional route segments to your selection.
   - Double click to select the entire route.

   If rule checking is turned on, the copy is successful only if it does not cause clearance rule violations.

   **Note:** If the copied wire has a t-junction, only the wire segments up to the t-junction can be selected for copying. You cannot pick the entire wire. T-junctions might introduce ambiguous routing topologies that can cause the command to fail.

3. From the Layout pop-up menu, choose Mirror – X, – Y or – XY to mirror the copied route.

4. Click on a target pin to place the copied route.

   When you copy a single wire to a target pin with different width or clearance rules, the tool applies these rules to the copied wire.
Removing Notches and Extra Wire Bends

The *Critic Wire* command eliminates notches and removes extra bends in routed wires by attempting to adjust the existing wires rather than reroute the wire. You can use the *Critic Wire* command on a single wire or a group of wires.

To remove notches and extra bends from wires, do the following:

1. Choose *Edit – Wire – Critic*.
2. Select a wire or group of wires.
The following figures show before and after using the *Critic Wire* command on all wires.

Before using the Critic Wire command

![Before using the Critic Wire command](image1)

After using the Critic Wire command

![After using the Critic Wire command](image2)
Compacting Wires Using Pull

The *Pull* command compacts wires and vias within a rectangular area. The compaction area is defined by digitizing a starting point and ending point, which creates a pull bar, and then digitizing a third point, which completes the definition of the compaction area. The pull bar is used to pull the routes in a vertical or horizontal direction.

To compact wires and vias within an area, do the following:

2. Set a pulling bar by doing one of the following:
   - Digitize a starting point, move the pointer vertically, and click to define the end points of a vertical pulling bar.
   - Digitize a starting point, move the pointer horizontally, and click to define the end points of a horizontal pulling bar.

   A vertical or horizontal line represents the pulling bar.

3. Sweep the pointer away from the pulling bar in either orthogonal direction.

   As you sweep the pointer, the wires and vias are pulled toward the pulling bar.

4. Click when the pointer defines the compaction area you want to use.

The following options will affect the how the wires are compacted:

- **Interactive Checking**

  When *Interactive Checking* is on, the compaction is limited by clearance and spacing rules. Pulling stops for a particular wire or via if a violation would occur.

  When *Interactive Checking* is off, rules are not observed, and the pulling tends to pile up the wires at or near the pulling bar.

- **Allow Orthogonal Jogs**

  When *Allow Orthogonal Jogs* and *Push Routing* are on, orthogonal jogs are created for any wire segment that passes through the compaction area. Wire segments within the area are pulled, while wire segments outside are not.

  When *Allow Orthogonal Jogs* is off, the entire wire segment is pulled.
Reporting

This section describes

- Displaying Routing Status Reports
- Displaying Network Reports
- Displaying Component Reports
- Displaying Net Reports
- Creating Rules Reports
- Search Reports
- Saving Reports

You can display reports, which give you detailed information about routing status, network information, a specified component, a specified net, and current design rules. You can also save reports in text files.

To display reports, follow these steps:

The Reports form is displayed.

Displaying Routing Status Reports

Routing reports contain a routing summary and a table of design statistics by layer. The routing summary includes information about the completion rate, unconnects, wiring, and the conflicts and rule violations encountered.

To display a Route Status report, do the following:

1. From the Report form, click the General tab.
2. Turn Route Status on.
3. Click OK.
The Routing Status Report window appears.

For descriptions of the information contained in the report, see “Route Status Report Window” on page 406.

**Displaying Network Reports**

Network reports contain design statistics, which include the number of pins, vias, t-junctions, conflicts, and length information for each routing layer.

To create a Network Report, do the following:

1. From the Report form, click the *General* tab.
2. Turn *Network on*.
3. Choose *Length By Layer* or *Total Lengths*. 
Length By Layer displays the report with the nets sorted by the length of the layers. Total Length lets you display the report with the nets sorted by Name, Ratio, Length, or Extra.

4. Click OK.

The Network Report window appears.

The report identifies the nets in the current design and how the netlist was sorted.

For descriptions of the information contained in this report, refer to “Network Report Window” on page 407.

**Displaying Component Reports**

Component report contain information about a single component in the design.

To create a Component report, do the following:

1. From the Report form, click the Component tab.
The tab lists components in the current design.

2. Choose the component from the list for which you want to create a report.

3. Click OK.
The Instance Report window appears.

![Instance Report Window]

For descriptions of the information contained in this report, refer to “Instance Report Window” on page 408.

**Displaying Net Reports**

Net reports contain information about a single net in the design.

To create a Net report, do the following:

1. From the Report menu, click the *Net* tab.
The tab lists nets in the current design.

2. Choose a net from the list for which you want to create a report.
3. Click OK.
The Net Report window appears.

For descriptions of the information contained in this report, refer to “Net Report Window” on page 409.

**Creating Rules Reports**

Rules reports contain information about current constraints and technology process rules for the current design. You can specify which rules to include by precedence level in the rules hierarchy.
To create a Rules report, do the following:

1. From the Report form, click the *Rules* tab.
   
The form updates.

2. Choose *All* or *Specify*.
   
   You can list all the design rules in the current design or you can specify which rules to include by precedence level in the rules hierarchy.

3. Click *OK*. 
The Rules Report window appears.

For descriptions of the information contained in this report, refer to “Rules Report Window” on page 411.

**Search Reports**

To search a report, do the following:

1. From the report window, choose *File – Search.*
The Search form is displayed.

2. In the Search for field, type the specific words, strings, or regular expressions.
3. Click OK.

Saving Reports

To save the report in a text file, do the following:

1. From the report window, choose File – Save As.

   The Save As form is displayed.

2. In the File Name field, type a name for the text file.
   
   **Note:** You can include the path to the file if you want to specify a directory other than the current working directory.

3. Click OK.

Setting Constraints

This section describes
Using the Virtuoso Constraint Manager

Use the Virtuoso Constraint Manager to set net-based constraints which, once they are saved to the cellview, are followed when routing.

The following are net-based constraints you can apply using the constraint manager.

- **Net or Class**
  - Control the order in which nets or classes are routed.
  - Control the width of nets or classes.

- **Net To Net / Shielding**
  - Set the minimum distance between two nets on different layers.
  - Create tandem shielding.
  - Create parallel shielding.
  - Create differential pairs.

- **Class To Class**
  - Set the minimum distance between nets on different layers.

- **Class To Net**
  - Set the minimum distance between nets on different layers.

**Note:** If you create or modify a constraint while in a wire editing command, you will need to restart the command to make the constraint active.

For more information, see the *Virtuoso Constraint Manager User Guide*.

Using .do Files

A .do file is an external ASCII file you can create using any text editor. In the Virtuoso XL environment, .do files are used to apply constraints and definitions. In contrast, .do files used in the Virtuoso Custom Router and Cadence® Chip Assembly Router environment can be used as a script to drive a batch job.
You can load a .do file or change a .do file at any time during a session. The .do file is not cumulative; once you load a new .do file or change the data within the file, the initial data is lost. However, you can use the do command to load additional .do files. For example, the following line in a .do file will cumulatively load rules from both files:

```
do routing1.do routing2.do
```

### .do File Commands

The following commands can be included in a .do file. For more information, see Appendix D, “.do File Commands”.

- `circuit`
- `define/forget`
- `do`
- `rule`
- `set`

### Loading .do Files

To load an ASCII .do file, follow these steps:

1. From the layout window, choose `Options – Routing`.
   The Routing Options form is displayed.

2. Turn on `Do File`.

3. Choose the .do file by doing one of the following:
   - In the Do File field, type the name and path of the .do file.
   - Click on Browse.
The Find File form is displayed.

![Find File Form]

**a.** Navigate to the `.do` file.  

**b.** Choose the `.do` file.  

**c.** Click *OK*.

4. In the Routing Options form, click *OK*.  

The contents of the `.do` file are referenced when the next command is executed.  

**Note:** Click *Refresh* in the Routing Options form to reload updated `.do` file information in the following situations:

- Contents of the `.do` file is changed.  
- Name of the `.do` file is changed.  
- New `.do` file is being used.  

**Note:** Verify constraints and rules set in the current design by using the *Verify – Report Routing* command. For more information, refer to “Creating Rules Reports” on page 389.
Online Forms

Add Via

Displays vias and routing layers that have been defined in the rules file or technology file. You can either choose the layer you want to change to or choose a via type on the layer.

The Add Via form consists of a matrix of layer and via buttons in rows and columns. Each row in the matrix represents a routing layer and contains a button for the layer and buttons for the vias that can reach that layer from the current layer. The first column contains the buttons for the layers. The other columns contain buttons for each available via type for the given layer. The buttons are color coded by layer color, and the layer name appears on each layer button.

Buttons for layers that can be reached from the current active layer are enabled, while buttons for inaccessible layers are grayed out.

Rotate Via rotates vias 90 degrees as you place them.

Check Routing

Rules

Conflict checks for shorts and clearance violations.

Crosstalk checks for crosstalk rule violations. To control crosstalk, limit the lengths of parallel wire segments for a given wire-to-wire gap. The parallel_segment crosstalk rule limits the distance that wire segments on the same layer can be parallel, and the tandem_segment crosstalk rule limits the distance that wire segments on adjacent layers can be parallel.

Limit Way checks for limit_way rule violations. The limit_way rule defines the maximum wrong-way distance permitted in a connection. The wrong way direction is vertical on horizontal routing layers and horizontal on vertical routing layers.

Maximum Process Wire Width checks for maximum process width rule violations.

Minimum Mask Edge Length checks for the minimum dimension for any edge of a shape at the mask level.

Miter checks for unmitered corners.

Polygon Wire checks for clearance violations from polygon wire shapes to pins, fixed wires, protected wires, and fixed or protected vias.
Re-Entrant Path checks for wire loops that have been created while routing or editing.

Segment checks for maximum segment rule violations. The max_segment rule controls the total routed length on a layer.

Use Layer checks for use_layer rule violations. The use_layer rule assigns one or more routing layers to a net, class, group, group set, or fromto.

Wire Extension checks for wire extension violations. Wire extension rules define the amount of routing wire material needed beyond the wire end at a via.

Corner - Corner checks for a larger corner-to-corner clearance, rather than an edge-to-edge clearance between shapes. Corner-to-corner clearance is the minimum corner-to-corner clearance between two shapes and can vary from the edge-to-edge clearance to a maximum value of $1.414 \times$ (edge-to-edge clearance value) at 45 degrees.

Length checks for length_rule violations.

Maximum Stacked Via Depth checks for stack_via_depth violations. stack_via_depth limits how many vias can be stacked at one location.

Minimum Process Wire Width checks for minimum process width rule violations.

Maximum Total Via checks for the max_total_vias rule, which controls the maximum number of vias permitted on a net.

Net Order checks routed wiring for violations of the net ordering rules. The net ordering choices are, Starburst, Daisy, Mid-Driven Daisy, Balanced Daisy.

Protected checks for clearance violations between protected wires or vias and other protected or fixed objects.

Same Net checks for same-net violations such as notches, gaps, and loops.

Stub checks for maximum stub length rule violations. Stub length is the distance between a pin or via and a tjunction. Stub length is measured from the edge of a via or pin to the center of the tjunction.

Use Via checks for use_via rule violations. You can control which via is used to route a group, net, or class of nets by setting a use_via rule.

Pin Data

Pin Spacing checks the edge-to-edge spacing between pins.

Off Manufacturing Grid Pin checks for pins that are located off the manufacturing grid.

Off Wire Grid Pin checks for pins that are not centered on a wire grid.
Pin Larger Than Maximum Process Width checks for pins that are larger than the maximum process width rule. Both dimensions of rectangular pins are checked.

Pin Smaller Than Minimum Process Width checks for pins that are smaller than the minimum process width rule. Both dimensions of rectangular pins are checked. Non-rectangular pins are not checked.

Inaccessible Pin checks for pins that are blocked by a keepout, boundary, or other obstructions.

Create Wire

Width lets you enter a value for the width of the current wire.

Note: Setting a width value or applying a width change to a wire does not alter the width rule for the wire.

Match Pin Width sets the net width to the width of the pin you are editing.

  Narrow sets the width of the active segment to match the shortest side of the pin to which the wire is attached.

  Wide sets the width of the active wire segment to match the longest side of the pin to which the wire is attached.

Match Wire Width sets the net width to the width of the net you are editing.

Change Layer specifies whether the entry layer will remain on the same layer or switch to the layer of the object you have selected.

Snap Mode limits how the cursor snaps when you create the wire.

  anyAngle creates wires at any angle.

  diagonal creates wires parallel to the x or y axis or at a 45-degree angle to the axes.

  orthogonal creates wires parallel to the x or y axis.

  L90XFirst creates orthogonal two-segment wires and creates the first line in the X direction.

  L90YFirst creates orthogonal two-segment wires and creates the first line in the Y direction.

Net Name assigns a net name to floating wires.
Find File

Directory lists all directories in the current directory.

File lists all files in the current directory.

Note: The text field displays the current directory. Choosing a file and clicking OK updates the file information in the Routing Options form.

Layout

The Layout pop-up menu is context specific and shows only options for the active command.

![Layout Menu]

Finish Routing finishes the routing connections from the last digitized points. If the routing area is obstructed, the command is terminated.

Route As many As Possible while bus routing, the Finish Routing command will try to complete as many connections as the available routing area permits.

Route Only If All Succeed while bus routing, the Finish Routing command must complete all wires successfully or none are routed.

Turn gather bus wires on/off changes the cursor to a single bus cursor and wires are spaced by the amount specified in the Spacing for Gather Bus Wires settings.

Add Via opens the Add Via form, and all available vias and routing layers that can be reached from the current routing layer are displayed for selection. Buttons for inaccessible layers are grayed out.

Via Pattern opens the Via Pattern Pop-up. When routing multiple wires, multiple vias are added according to a via pattern that you choose.
Pick up dropped wires restarts bus routing at a new location after some wires have been dropped. Guides appear from the new starting points and back to the dropped wires. The dropped wires can then be routed from the new location. After routing the wires from the new location, you can route the dropped wires individually from the location where it was dropped to the location where the wires were restarted.

Set up opens the Routing Options form, which sets the routing controls for two categories; General and Bus.

Width

  Match Pin Width sets the net width to the width of the pin you are editing.

    Narrow sets the width of the active segment to match the shortest side of the pin to which the wire is attached.

    Wide sets the width of the active wire segment to match the longest side of the pin to which the wire is attached.

  Match Wire Width sets the net width to the width of the net you are editing.

    Specify opens the Create Wire form, which lets you specify a new width for the wire.

Turn checking off/on prevents routing rule violations between routing objects, such as wires, vias, and pins.

Use display grid allows switching between the technology file manufacturing grid and the grid defined under Options – Display.

Reports

General

  Route Status creates a report that includes information about unconnects, conflicts, rule violations, number of pins, number of components, vias, and length information for each routing layer.

  Network creates a report about all nets in the design. You can sort the net information by the length of the layers or select the Total Lengths Boolean to further sort them.

    Length By Layer displays the report in a new window with the nets sorted by the length of the layers.

    Total Lengths - Sort By

      Name sorts the information about the nets alphabetically according to the net name.
Ratio sorts the information from the highest to the lowest ratio of the actual routed length divided by the manhattan length.

Length sorts the information from the highest to lowest length rule.

Extra sorts the information from the highest to the lowest difference between the actual routed length and the manhattan distance.

Component displays a list of components placed in the current design. The report includes component placement information, nets, and pins associated with the component and component pin positions.

Net displays a list of nets for the current design. The report includes rules applied to the specified net, connections, coordinates, components, component pins, layers, and vias.

Rules displays a report that contains information about routing rules applied to the current design. You can specify to view All rules associated with the current design or Specify which rules are displayed in the report.

Routing Options

General Tab

Interactive Checking prevents conflict and clearance violations between routing objects, such as wires, vias, and pins. Also prevents routing outside of the boundary area.

Via Assistance controls whether visual and interactive assistance is available to search for legal via locations when you are routing a connection and want to add a via.

Snap either snaps the wire you are routing to the closest via site or displays nearby via sites, depending on the distance of the closest legal via site to the last digitized point on the wire.

- If the last digitized point is within one via radius of a legal via site, the wire is automatically routed to the closest via site and a via is added.
- If the last digitized point is not close to a legal via site, nearby via sites are displayed as concentric circles but no via is added.

Display adds a via if the last digitized point is on a via site. If the last digitized point is not at a legal via site, nearby via sites are displayed as concentric circles and no via is added.

With both the Snap and Display options, when via sites are displayed you can move the pointer to a via site and click to add a via.
Allow Orthogonal Jogs, when Allow Orthogonal Jogs and Push Routing are on, permits orthogonal jogs in wires that are pushed using the Create Wire, Stretch Wire, Split Wire, and Pull commands.

Alternate Views, lets you specify abstract or other views rather than layout views of cells. Type a list of views separated by spaces. The translator uses the first view found in the list for each particular cell.

Show Timing/Length Rule Constraints displays length rule indicators as you are editing a wire with a length rule.

- **Meter** displays the current wire length relative to the minimum or maximum length rules of the wire.

- **Octagons** displays an octagon showing the minimum or maximum length rule of the wire you are editing.

Route To Cursor when on, wires are created by following the direction of the cursor from the previously digitized point. This option is not available when you are routing a bus, bundle, or differential pair.

- **Single Layer** multiple wire segments are routed on the current active layer.

- **Multiple Layers** wires are routed using multiple layers. Vias are added automatically as layers change.

  - **Follow Layer Direction** controls whether the layer direction set in the rules file is strictly followed.

Snap to Pin Origin controls whether routing connections snap to pin origins when you route them. If Snap to Pin Origin is on, a wire snaps to the origin of pins when you click anywhere inside the pin. If Snap to Pin Origin is off, a connection to a pin is considered complete if it is within the boundary of the pin shape.

Push Routing controls whether wire segments and vias can push aside other wires and vias in order to comply with current width and clearance rules.

Same Net Checking prevents same net rule violations such as gaps and notches which occur when the exposed facing edges of objects on the same net are closer than the minimum clearance rules permit. Also prevents loops or self intersecting nets.

Check Region check wire segments at region boundaries during routing operations.

- When Check Region is on (the default), wire segments are blocked from extending across a region boundary until you click at the boundary. If you are routing the wires, the router inserts a pseudopin for each net. Then you can continue routing or moving the wire segments into or out of the region. If you are routing multiple wires and the applicable
wire width or clearance rules are different inside the region, the wires fan out if necessary to accommodate these rules.

- When Check Region is off, region boundaries and region rules are ignored during routing and move operations.

**Auto Adjust Length** attempts to meet minimum length rules when finishing a connection with Finish Route and highlights the connection if any rule violations occur.

**Multiple Component Connection** when on, the Create Wire command connects same net instance pins and top level instance pins that are in alignment. The resulting wire contains connections to each pin and separate wire segments between each adjacent pair of pins. When off or the pins are not in alignment, a single wire is created with connections only to the pins that are clicked on.

**Push Component** controls whether wire segments or vias can push aside components in order to comply with current clearance rules.

**Auto Shield** when on, nets that have a shielding constraint attached to them are automatically shielded. Shields are generated when the Create Wire operation is completed.

**Allow Redundant Wiring On Enabled Nets** allows wire loops and extra vias on selected nets for increased current capacity. Redundant wiring is permitted on selected nets when Allow Redundant Wiring On Enabled Nets is turned on and the allow_redundant_wiring on rule is attached to the net.

**Rules** lets you load a technology process rules file.

**Do File** lets you load a specified .do file. The .do file is used for setting constraints within your design and also for setting definitions, such as defining bundles and classes.

**Refresh** reloads updated rules and .do files. Use this when you manually edit an ACSII rules file or .do file and want to reload the new file.

**Browse** opens the Find File form, which lets you navigate to the rules file or .do file you want to load.

**Bus Tab**

**Enable Bus Routing** controls whether you can simultaneously route multiple wires.

- **Enable Tandem Pair** controls whether the router lets you simultaneously route wires on different layers.

- **Fit Via Pattern** controls whether the router automatically adjusts the locations of the vias to fit in the available space when you place a via pattern.
Allow Same Net Picking is used to select multiple wires on the same net during area selection.

Spacing For Gathering Bus Wires controls the minimum wire clearance during a multiple wire gathering operation. The options are Wire-Wire, Wire-Via, Via-Via, and Specify Clearance.

Override Spacing For Gathering Bus Wires change the current bus spacing when you are routing multiple wires.

- Override Value lets you set a new bus spacing value.
- Use Override the override value will take effect immediately on the current routing layer.
- Use Override On Next Via the override value will take effect on the next routing layer after you add a via pattern.
- Ignore Override returns to the default spacing value while routing the multiple wires.

Save As

File Name lets you type the name of the file to which you want to save the report.

Search

Search for lets you type a word, text string, or regular expression for which you want to search.

When Found

- select selects words, strings, or expressions specified in the Search for text field.
- deselect deselects words, strings, or expressions specified in the Search for text field.
- scroll to next match scrolls to the next occurrence of the word, string, or expression specified in the Search for text field.

Match Options

- whole word controls whether the search is restricted to the entire word, string, or expression or a partial word, string, or expression.
- exact case controls whether the search is case sensitive.
Scan the Whole File, when off, selects or deselects, depending on the requested operation, the first occurrence of the word, string, or expression. When on, scans the entire file and selects or deselects all occurrences of the word, string, or expression.

Split

Snap mode controls the direction in which you can draw the split lines.

anyAngle creates split lines at any angle.
diagonal creates split lines parallel to the x or y axis or at a 45-degree angle to the axes.
optogonal creates split lines parallel to the x or y axis.
L90XFirst creates orthogonal two-segment split lines and creates the first line in the X direction.
L90YFirst creates orthogonal two-segment split lines and creates the first line in the Y direction.

Via Pattern Pop-up

Sets the via pattern when multiple vias are added to bus wires. The via patterns are Perpendicular, Diagonal 1, Diagonal 2, Stagger, Out Taper and In Taper. The default is Diagonal 1.
Online Reports

Route Status Report Window

<table>
<thead>
<tr>
<th>Report Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#VirtuosoXL Wire Editor Version</td>
<td>The software version number and the version date and time.</td>
</tr>
<tr>
<td>#Host</td>
<td>The CPU host ID.</td>
</tr>
<tr>
<td>#ROUTING STATUS &lt;&lt;&lt; design &gt;&gt;&gt;</td>
<td>The name of the design file.</td>
</tr>
<tr>
<td>Report Time:</td>
<td>The time the report was generated.</td>
</tr>
<tr>
<td>Nets = Connections =</td>
<td>The number of nets and the number of two-pin connections in the entire design.</td>
</tr>
<tr>
<td>Current Wire = Reroute wires =</td>
<td>The current or last wire routed and the total number of wires to be routed. This information is relevant only for a particular routing pass.</td>
</tr>
<tr>
<td>Completion = Unconnections =</td>
<td>The completion percentage is: (1 - ((unconnections + wires in conflicts) / connections)) x 100. Unconnections are two-pin connections that are not routed.</td>
</tr>
</tbody>
</table>

#WIRING STATISTICS

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC Area =</td>
</tr>
<tr>
<td>Components =</td>
</tr>
<tr>
<td>Signal Layers = Power Layers =</td>
</tr>
<tr>
<td>Wire Junctions = , at vias = Total Vias =</td>
</tr>
</tbody>
</table>
Manhattan length = Horizontal = Vertical = Total manhattan length for the design is displayed and is further broken down into the horizontal and vertical components of the total manhattan length. This information is valuable for determining the necessary layer directions.

Routed length = Horizontal = Vertical = The total routed length is listed and broken down into its horizontal and vertical components.

Ratio Actual / Manhattan = The ratio of the actual wire length to the calculated manhattan length.

Unconnected length = Horizontal = Vertical = The total calculated horizontal and vertical manhattan lengths of unconnects is listed, and the total unconnected length broken down into vertical and horizontal components.

**Report Field**

**Description**

Layer
Routing layer name.

Direct
Layer biasing direction.

Pin
Number of component pins on the layer.

Vias
Number of vias on the layer.

TJs
Number of t-junctions at wires on the layer.

Conflicts
Number of conflicts on the layer.

Length
Total routed length on the layer.

Horizontal
Horizontal routed length on the layer.

Vertical
Vertical routed length on the layer.

**Network Report Window**

**Report Field**

**Description**

Name
The net names in the network file.

Pins
The number of components pins in the net.

Vias
The number of vias used to route the net.
The number of two-point connections in the net.

The number of t-junctions used in the net.

The calculated manhattan length required to route the net based on the number of two-point connections.

The actual length of routed connections.

The actual routed length divided by the Manhattan length.

The difference between the actual routed length and the manhattan distance.

**Instance Report Window**

<table>
<thead>
<tr>
<th>Report Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>The reference designator (component ID) of the component for which the report has been generated.</td>
</tr>
<tr>
<td>Component Mirroring</td>
<td>The x or y axis about which a component is mirrored.</td>
</tr>
<tr>
<td>Component Rotation</td>
<td>The degrees of rotation (counterclockwise) from the working origin.</td>
</tr>
<tr>
<td>Location (X,Y)</td>
<td>The location of the origin of the component.</td>
</tr>
<tr>
<td>Part Image</td>
<td>The library image name that defines the component.</td>
</tr>
<tr>
<td>Placement Grid</td>
<td>Manufacturing grid resolution.</td>
</tr>
<tr>
<td><strong>Component Pins</strong></td>
<td></td>
</tr>
<tr>
<td>Pin</td>
<td>The physical pin name (pin ID) of each pin.</td>
</tr>
<tr>
<td>X</td>
<td>The current X coordinates of each pin.</td>
</tr>
<tr>
<td>Y</td>
<td>The current Y coordinates of each pin.</td>
</tr>
<tr>
<td>Z</td>
<td>The layer numbers on which each pin has a contact point.</td>
</tr>
<tr>
<td>Rotate</td>
<td>The rotation of each pin or via.</td>
</tr>
<tr>
<td>Wires</td>
<td>The number of two-pin connections in which each pin appears.</td>
</tr>
</tbody>
</table>
Net Report Window

<table>
<thead>
<tr>
<th>Report Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net name</td>
<td>Reports whether the net is fixed. The report includes the classes the net is assigned to, the number of pins, vias, wires, and t-junctions, and data on manhattan versus actual (routed) lengths.</td>
</tr>
<tr>
<td>Rules In Effect</td>
<td>IC, class, and net width and clearance rules; net ordering; t-junction rule status; timing rules; bend, crossing, and via limits; the net's routing priority; and crosstalk and noise rules.</td>
</tr>
<tr>
<td>Layer Rules In Effect</td>
<td>The layer rules that override the net rules currently in effect for this net. They are listed by layer. The wire width and all object-to-object clearance rules are listed for each signal layer.</td>
</tr>
<tr>
<td>Fromto Rules In Effect</td>
<td>Fromtos in the following form and includes any corresponding fromto rules in effect: From component_id pin_id To component_id pin_id</td>
</tr>
</tbody>
</table>

Network Description

<table>
<thead>
<tr>
<th>Pin</th>
<th>Pin names identified by reference designator and pin number.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>The object type the pins are attached to.</td>
</tr>
<tr>
<td>X</td>
<td>The X coordinate location of each pin.</td>
</tr>
<tr>
<td>Y</td>
<td>The Y coordinate location of each pin.</td>
</tr>
<tr>
<td>Layer</td>
<td>Each layer on which the pins can be connected (the layers on which a pad has been defined for the pin).</td>
</tr>
</tbody>
</table>

Connections Description

<table>
<thead>
<tr>
<th>From</th>
<th>The starting point of a connection identified by component reference designator and pin number.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>The object type the from starting point of the connection is attached to.</td>
</tr>
<tr>
<td><strong>To</strong></td>
<td>The ending point of a connection identified by component reference designator and pin number.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Lists the object type the <em>to</em> ending point of the connection is attached to.</td>
</tr>
<tr>
<td><strong>Manhattan Length</strong></td>
<td>The minimum wire length needed to route a connection orthogonally (sum of X and Y distances).</td>
</tr>
<tr>
<td><strong>Actual Length</strong></td>
<td>The actual routed length of a connection.</td>
</tr>
<tr>
<td><strong>Belongs to Group(s)</strong></td>
<td>The group or groups each connection belongs to.</td>
</tr>
<tr>
<td><strong>Routing</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>From</strong></td>
<td>The starting point of each wire segment identified by component reference designator and pin number.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>The object type the <em>from</em> starting point of the wire segment wire is connected to.</td>
</tr>
<tr>
<td><strong>X</strong></td>
<td>The X coordinate location of each <em>from</em> wire segment starting point.</td>
</tr>
<tr>
<td><strong>Y</strong></td>
<td>The Y coordinate location of each <em>from</em> wire segment starting point.</td>
</tr>
<tr>
<td><strong>To</strong></td>
<td>The ending point of each wire segment of the net identified by component reference designator and pin number.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>The object type the <em>to</em> ending point of the wire segment is connected to.</td>
</tr>
<tr>
<td><strong>X</strong></td>
<td>The X coordinate location of each <em>to</em> wire segment ending point.</td>
</tr>
<tr>
<td><strong>Y</strong></td>
<td>The Y coordinate location of each <em>to</em> wire segment ending point.</td>
</tr>
<tr>
<td><strong>Attribute</strong></td>
<td>The attribute or attributes assigned to each wire segment.</td>
</tr>
<tr>
<td><strong>Layer</strong></td>
<td>The layer name each wire segment is routed on.</td>
</tr>
<tr>
<td><strong>Width</strong></td>
<td>The wire width of each wire segment.</td>
</tr>
</tbody>
</table>
Rules Report Window

<table>
<thead>
<tr>
<th>Report Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC Grid Values</td>
<td>The via, wire, and manufacturing grid by edge values.</td>
</tr>
<tr>
<td>Inter Layer Rules</td>
<td>Clearances between objects on different layers.</td>
</tr>
<tr>
<td>IC Rules</td>
<td>Global rules for all connections in the design. This is the lowest precedence level.</td>
</tr>
<tr>
<td>Layer Structure</td>
<td>Routing and reference layers and the rules that have been applied to them as defined in the translation rules file.</td>
</tr>
<tr>
<td>Layer Rules</td>
<td>Rules that apply to all wires routed on a particular layer.</td>
</tr>
<tr>
<td>Class Rules</td>
<td>Rules that apply to a class of nets.</td>
</tr>
<tr>
<td>Group_Set Rules</td>
<td>Rules that apply to a set of groups.</td>
</tr>
<tr>
<td>Net Rules</td>
<td>Rules that apply to nets.</td>
</tr>
<tr>
<td>Group Rules</td>
<td>Rules that apply to a group of fromtos.</td>
</tr>
<tr>
<td>FromTo Rules</td>
<td>Rules that apply to pin-to-pin connections.</td>
</tr>
<tr>
<td>Class_Class Rules</td>
<td>Rules that apply between wires, pins, and vias of one or more net classes.</td>
</tr>
<tr>
<td>Via_Image Rules</td>
<td>Clearance rules that apply to via padstacks.</td>
</tr>
<tr>
<td>Region Rules</td>
<td>Clearance or width rules that apply to any connection routed within an area of the design.</td>
</tr>
</tbody>
</table>

Setting Environment Variables

You can set the variables to the value you commonly use for a single session or you can set them permanently in setup files such as the .cdsenv file and the .cdsinit file.

To set environment variables for a single session, do the following:

- Type the environment setting in the CIW, or include the environment setting in any Cadence SKILL file you load.

  ```
  envSetVal("layout" "viaAssistance" 'cyclic none )
  ```

To set environment variables permanently, do any of the following:
Include the environment variables in the `.cdsenv` file in your home directory. For example:

```bash
layout matchWireWidth boolean t
```

Include the environment setting in your `.cdsinit` file.

```bash
envSetVal("layout" "$fitViaPattern" 'boolean t)
```

To determine the current value of any Virtuoso XL environment variable, type in the CIW. For example:

```bash
envGetVal("layout" "$fitViaPattern" 'boolean)
```

## Troubleshooting

This section provides information about what to do if you are unable to route from a connectivity object (pin, via, or an existing wire) during the Virtuoso® XL wire editing process.

- Make sure the routing layers are defined in the rules file or the technology file. If the routing layers are not defined, the following warning message is sent to the Command Interpreter Window:
  
  icclayers are not defined

  For more information, see “Rule Information” on page 332.

- Make sure object has connectivity. If there is no connectivity assigned to the object the following warning message is sent to the Command Interpreter Window:

  No wires are attached to that terminal

- Check if there are rule violations which are preventing you from creating or extending a wire. A dashed outline surrounds the wire to indicate the wire-to-wire clearance rule.

- Check if Routing Options – Route To Cursor – Follow Layer Direction is set. When Follow Layer Direction is turned on, the routing layer direction, which is set in the rules file, is strictly followed. If the layer direction of is set, for example to horizontal on `metal1`, routing from a pin in the vertical direction on `metal1` will not be allowed.

- Check if the instance is placed off grid. If the instance is place off grid, you will not be allowed to route from the pins within the instance.

- Make sure that all devices to be routed are inside a prBoundary. For more information, see “Routing Area Boundary” on page 333.

- Check if the routing angle is set to `diagonal` or `orthogonal` in the Create – Wire form. In some cases wires will not route correctly in `anyAngle` mode.
Checking Design Data

This chapter explains how to use the Virtuoso® XL Layout Editor to check your design as you work.

This chapter discusses the following topics.

- Finding Design Elements on page 414
- Checking Shorts and Opens on page 420
- Comparing Design Elements (Check Against Source) on page 422

For information on Virtuoso XL forms, see “Online Forms” on page 423.
Finding Design Elements

Probing lets you find a component, net, or pin in the layout (or schematic window) and highlight the corresponding design element in the schematic (or layout) window.

To probe a design element,

1. From the layout window, choose *Connectivity – XL Probe*.
   
   The layout window prompts you to point to a design object.

2. Click on a design object in the layout or schematic.
   
   Virtuoso XL highlights the design object you selected in both the layout and schematic windows.

   If you have several versions of a layout cellview open, selecting a design element in the schematic highlights the corresponding element in all of the open layout cellviews.

3. Press \( \text{F}3 \) to change the probe options.
The Probe Options form is displayed. For more information, see Online Forms on page 423.

4. In the Object Filter section, turn on Pins, Nets, or Devices to choose the design elements to probe.

When you click on a design element, the Probe Options form displays information about the selected device in the layout and schematic.

For example, if you click on the vss pin in the layout, vss is highlighted in both the schematic and the layout using the hilite color displayed in the Probe Options form Display Layer box and the Probe Options form displays information about vss.

pin: (lay)vss -> (sch)vss

If you want this information to be displayed in the Command Interpreter Window (CIW), switch on Send Messages to CIW.
If you want to create a net class, so that you can probe nets in groups relevant to your design, select Tools – Constraint Manager. Change the Select Entity to Support Entities. The form updates to display Classes.

You can also set the Show cyclic field to have the list box below display the names of all the pins, nets, net classes, or devices in the design. You can click on the names in the list box (instead of clicking on the objects in the design windows) to probe them.

To zoom in on the selected pins, nets, net classes, or devices in the list box, click the Zoom button.

**Note:** You can change the hilite drawing# layer colors from the Display Resources Editor.

5. To add additional elements to the Probe Information list, or to deselect items, hold down the Control key when you click on subsequent elements in the schematic, layout, or Probe form. You can select a range of names in the Probe Options form list by clicking on the top name and then clicking on the bottom name with Shift-click.

If Virtuoso XL cannot find the design element in the schematic that corresponds to the one in the layout, you see a question mark (?) at the end of the listing in the form. The device might not exist in the schematic, or the schematic window might be iconified or closed.

```
dev:(lay)Q12->(sch)?
```

If you click on a place where there is more than one design element, Virtuoso XL highlights design elements in the following order:

1. Pins
2. Nets
3. Devices

If you click on a place where there is more than one of the same kind of design element, a message window opens asking which one you want.
6. In the message window, click on the design element to probe and click OK.

   Virtuoso XL highlights the component, net, or pin you choose and displays the information in the Virtuoso XL Probe Options form (if open).

**Note:** If you are cross-probing components bound by one-to-many/many-to-many/many-to-one mapping, probing one component bound by such mapping highlights all the components in the mapping group. If you probe an external net of a mapping group, the corresponding external net in the other window is highlighted. Probing an internal net of a mapping group highlights the entire mapping group.

**Note:** If you open a schematic and two copies of the same layout, the *Probe* command applies to both layouts. If you have a schematic and two different layouts open, the *Probe* command applies to only the layout from which you selected the command.

**Probing Hierarchical Designs**

To probe a hierarchical design, follow these steps.

1. Open the layout and schematic (the cell view pair, pair #1) in Virtuoso XL.

2. Select the device to probe.

   In the diagram below, the device to probe is an inverter (CV1). It is represented in the layout as pair of transistors (a flat representation).

   ![Diagram](image)

3. Choose *Design – Hierarchy – Edit in Place.*
The schematic of the inverter opens (CV3 in the diagram).

If you select one of the transistors in this schematic to probe, the corresponding transistor in the layout in the level above is highlighted (CV2 in the diagram).

4. In the inverter schematic (CV3), select one of the two NMOS elements.

5. In the schematic window Tools menu, choose Design Synthesis – Layout XL to open the layout of the inverter (CV4 in the diagram) in Virtuoso XL.

This creates another cellview pair (pair #2).
6. To descend into the schematic of the inverter, select one of the NMOS elements and from the schematic window choose **Design – Hierarchy – Edit in Place**.

The schematic of the NMOS opens (CV5 in the diagram).

If you probe the transistor in this schematic, the corresponding transistor in the layout in the level above is highlighted (CV4 in the diagram).

7. In the schematic window **Tools** menu, choose **Design Synthesis – Layout XL** to open the layout of the transistor (CV6 in the diagram) in Virtuoso XL.

This creates another cellview pair (pair # 3). You can probe from NMOS: Schematic back to Top: Layout, Inv: Layout, and NMOS: Layout.
Removing Probes

To remove an individual probe from the layout,
➤ Hold down the Control key and click on the probe to delete.

To remove all probes from windows,
➤ Click on an empty space in the layout window.

Exiting the Probe Command

To exit the Probe command,
➤ Click Cancel in the Virtuoso XL Probe Options form or press Escape.

Showing the Options Form

To set the Probe Options form to appear each time you use the Probe command, follow these steps.
1. From the CIW, choose Utilities – User Preferences.
   The User Preferences form is displayed.
2. Click Options Displayed When Commands Start.
3. Click OK.
   The options form opens automatically the next time you use the Probe command.

Checking Shorts and Opens

You can use the Connectivity – Check – Shorts and Opens command to check the number of incomplete nets, short markers, and illegal overlap markers in your design.

Tip
The command relies on extractor markers. If you have deleted these markers, you must re-extract your design before checking for shorts and opens. For more information, see “Re-Extracting a Top-Level Design” on page 299.

To check the number of shorts, opens, and illegal overlap markers
1. Make sure that the connectivity extractor is turned on in the Layout XL Options form.

2. From the layout window menu bar, choose Connectivity – Check – Shorts and Opens.

A text window opens reporting the number of incomplete nets, shorts, and illegal overlaps currently in the layout window. This command also reports shorts and opens in nets where pins are weakly connected and must connected.

![Virtuoso XL Info]

**Important**

*Connectivity – Check – Shorts and Opens* marks in the layout window shorts found at lower levels of hierarchical designs but does not report these shorts in the Virtuoso XL Info window. Open circuits at lower levels of hierarchical designs are neither marked in the layout window nor reported in the Virtuoso XL Info window.

For more information, see Chapter 13, “Using Connectivity,” in the Virtuoso Layout Editor User Guide.
Comparing Design Elements (Check Against Source)

You can check whether

- All the devices present in the schematic are present in the layout
- All the devices present in the layout are present in the schematic
- All the properties listed for devices in the schematic are associated with equivalent properties on devices in the layout

To compare design elements in the schematic and layout,

➤ From the layout window, choose Connectivity – Check – Against Source.

Note: Check - Against Source does not report missing or extra I/O pins and instances which have a property specified in the propsUsedToIgnoreObjs environment variable.

If there are devices in the schematic that are missing in the layout, a Virtuoso XL Info window opens listing the name of the components. The components are also highlighted in the schematic.

Check – Against Source also reports shapes that are on redundant or unmatched nets which no longer exist in the schematic but it does not put markers on these shapes. You can check the shapes and delete them manually as required.
If a device that is not in the schematic is present in the layout, it appears with a blinking marker.

If a device parameter in the layout is different from the corresponding parameter in the schematic, a text window appears and lists the device names and properties (unless the device is ignored).

To save the information in an ASCII file or close the Virtuoso XL Info window,

➤ Use the commands on the *File* menu.

**Using CDF Callbacks with Check Against Source**

*Check Against Source* uses the schematic list of parameters, but the layout value of the width (for folding), $sParam$ (for $sfactor$), and $mfactor=1$ and $sfactor=1$ to verify the list of parameters. The total width/$sParam$ is compared before applying the callback, all other parameters are compared with the corresponding master after applying the callback.

**Online Forms**

**Probe Options**

*Object Filter* specifies the objects highlighted by the probe command.

- **Pins** specifies that the probe command highlights pins.
- **Nets** specifies that the probe command highlights nets.
- **Devices** specifies that the probe command highlights devices.

*Display Layer* shows which of the *hilite drawing #* layers is to be used in the next probe drawn.
Cycle, when turned on, changes the hilite layer to the next color when a new object is probed. When Cycle is not turned on, the hilite layer does not change to the next color each time a new object is probed.

Send Messages to CIW, when turned on, sends the information about the objects probed to the CIW (as well as displaying it in the box at the bottom of the form, when the form is open).

Show displays in the list box the names of the nets, pins, net classes, or devices in the design (depending on which category you select in the cyclic field). When you click on one or more items in the list box, the items are probed (highlighted) in the schematic and in the layout and the name, window, and equivalency information about each item is shown in the display box. When you remove a probe, that information disappears from the display box.

Zoom to the bounding box of the current probing objects based on the probe type.
Updating Design Data

This chapter explains how to use the Virtuoso® XL Layout Editor to make design changes and update your design as you progress.

It contains information on the following areas.

- Updating Components and Nets (Engineering Change Order Mode) on page 426
- Updating Layout Parameters on page 432
- Updating Schematic Parameters on page 434
- Updating Device Correspondence on page 436
- Creating Device Correspondence on page 444
- Updating the Connectivity Reference on page 449
- Changing the Device (Instance) View on page 450

For information on Virtuoso XL forms, see “Online Forms” on page 451.
Updating Components and Nets (Engineering Change Order Mode)

Virtuoso XL can automatically update the layout to take account of any devices, pins, or connectivity changes you have added to the schematic. This operation is sometimes known as Engineering Change Order (ECO) mode.

While *Gen From Source* deletes any existing components in the layout view and regenerates everything from scratch, *Update Components And Nets* updates only the components that have changed; adding new devices and pins, deleting (or marking) old devices and pins, and updating masters and connectivity to match that of the schematic.

**Tip**

Activate *Connectivity – Show Incomplete Nets* before you update components and nets. This lets you verify that the connectivity you want is made.

To update the components and nets in a layout with changes made to a schematic,

1. In the schematic window select *Check – Current Cellview*.

   The schematic editor extracts the connectivity of the design and reports the results in a Schematic Check information window.

2. From the layout window, choose *Connectivity – Update – Components and Nets*.

   Virtuoso XL checks the components and nets in the schematic against the ones in the layout.

   If the schematic has changed, you see a dialog box indicating that schematic extraction is needed to continue. Virtuoso XL issues a message to the CIW indicating the cellviews that are out of date.

   - If you click *OK* in the dialog box, Virtuoso XL performs a hierarchical extraction on the schematic and all of its reference libraries. However, the extracted cellviews are not automatically saved.

   - If you want a different extraction behavior, click *Cancel* in the dialog box and select the *Check – Hierarchy* command in the schematic window. Click *OK* to run the *Check Hierarchy* command.

     If there are no differences, a message appears in the Command Interpreter Window (CIW) saying that the check has been completed successfully.

     If there are differences, they are reported in the CIW.
If there are devices or pins in the layout that are not in the schematic, they are highlighted with markers in the layout. If you want the software to delete unmatched instances and pins, set the deleteUnmatchedInsts and deleteUnmatchedPins environment variables. The advantage of deleting redundant pins is that redundant nets and terminals are also deleted from the layout view at the same time.

For more information, see “deleteUnmatchedInsts” on page 487 and “deleteUnmatchedPins” on page 488.

If there are devices or pins in the schematic that are missing from the layout, the Layout Generation Options form opens. For more information about using this form, see “Layout Generation Options” on page 163.

If the I/O Pins option is turned on, the pins shown in the I/O Pins section of the form are those that exist in the schematic but are missing from the layout. Any pins generated are automatically snapped to the placement grid.

Note: The Layout Generation Options form always reflects the default settings for the design (that is, the state of the schematic) not the settings from the previous time the form was used.

3. In the Layout Generation section at the top of the form, check that the options you want are switched on.
Transistor chaining and transistor folding

Turn on the Transistor Chaining option to recreate the abutment into chains of MOS transistors or fingers of folded MOS transistors that were unabutted since you created them using the Design – Gen From Source command.

Turn on the Transistor Folding option to recreate and fold MOS transistors that were unfolded or all fingers of which were deleted since you created them with the Design – Gen From Source command.

If you switch on these options, the Update Components and Nets command chains and folds only new devices. New chains are not attached to existing chains. Devices that were chained prior to running the command are not moved from their existing positions.

If the Transistor Chaining and Transistor Folding options are switched off, and you have deleted all the fingers of a folded transistor, Update Components and Nets generates a single device. If you have deleted some of the fingers of a folded device, it creates markers only if fingers in a numbered sequence are missing; otherwise it does nothing about missing fingers of folded transistors.

To preserve existing many-to-many mapping of devices between the connectivity source and the layout, turn on Preserve Mappings.

Note: This function does not report missing devices or shapes within a mapped group.

4. For each pin you want to add, click Add a Pin.

An empty pin row opens. Enter the Net Name, Pin Type, Layer/Master, Width, Height, and specify the number of these pins that you want.

5. Set the Pin Label Shape for all pins.

a. Set Label to generate labels for each pin on the text dg layer-purpose pair.

Note: Virtuoso XL does not generate labels for symbolic pins.

b. Set Text Display to generate text for each pin on the text dg layer-purpose pair.

To view the text, set the Pin Names option on in the Display Options form.

c. Set None to prevent label generation for pins.

6. To set the style of the pin labels, click the Display Pin Name Option button.

The Display Pin Name Option form is displayed.

7. Choose the options you want from the Set Pin Label Text Style form and click OK.
8. If you want Virtuoso XL to draw a boundary, follow the steps below.

   a. Choose the layer you want to use for the boundary.

   b. In the Shape field, choose Rectangle or Polygon.
      - Specify the dimensions of a rectangular boundary using the Utilization %, Aspect Ratio (W/H), Boundary Width, and Boundary Height options.
        Use the Left and Bottom fields to specify the origin of the boundary. Type the x coordinate in the Left field and the y coordinate in the Bottom field.
      - Specify the dimensions of a polygonal boundary by entering the X and Y coordinates of each angle of the polygon in the Points type-in field in the following format. For example, for a rectangle: (0 0) (0 10) (10 8) (8 0) (0 0).

When you move the cursor in the layout window, the X and Y coordinates of the location of the cursor are displayed in the Status Line at the top of the layout window.
9. In the *Template File* section, to load a template file, (an ASCII file containing the information entered in this form in a previous session and saved to a file) follow these steps.

   **Load Template File for Layout Generation**

   ![Image of Load Template File for Layout Generation]

   **a.** Select the *Load Template File for Layout Generation* option.
   
   **b.** Click the *Browse* button and the Open File form is displayed.
   
   **c.** Click on the directories in the left list box to descend into your file hierarchy until you reach the name of the template file to use.
   
   **d.** Click on the name of the template file in the right list box to enter it in the *Name* field.
   
   The left and right arrows at the right side of the form let you go up and down in the file hierarchy.
   
   **e.** Click *OK* in the Open File form.
   
   The values in the template file are loaded into the Layout Generation Options form and are used during the design session.
   
   You can set an environment variable called `templateFileName` to always load the name of a template you specify as the value of the variable.

10. To save the information you enter in the Layout Generation Options form from the cellview to a template file (an ASCII file) to use in a later session, follow these steps.

   **a.** Select *Design – Save To Template*.
   
   The Template File form is displayed.
   
   **b.** Turn on the *Boundary* and *I/O Pins* options
   
   **c.** In the *Template file name* field type in a template file name.
   
   **d.** Click *OK*.
   
   The data you selected is saved in the template file.

11. Click *OK*. 
Virtuoso XL puts any missing pins and components below the design boundary and updates their connectivity.

If the schematic master has changed, Layout XL updates the layout master accordingly, as long as the `updateReplacesMasters` is set to `t`.

If there is a schematic instance with a master which does not exist on the layout side, Virtuoso XL descends into the schematic hierarchy to try to generate the instances contained inside.

**Note:** Permuted pins are not preserved by the *Update Components and Nets* command. If any component in the design needs to be updated, the command resets the permutation status of all of the cells to that originally stored in the schematic.

If a device that has been folded into fingers is missing in the layout, Virtuoso XL recreates the device.

If one or more fingers of a device that has been folded into fingers is missing in the layout, the CIW displays a warning that the fingers corresponding to the schematic instance are missing, but it does not recreate the fingers.

If a parameter regulating the width of the fingers of a folded device has changed in the schematic, it is not changed in the layout by this command. Use the *Connectivity – Update – Layout Parameters* command to implement parameter changes in the schematic that affect physical implementations in the layout.

**Note:** When Update Components and Nets is not able to find a valid view as specified in the `stopList` and the layout was not placed as a result, the following error message appears.

```
\w *WARNING* Could not find master cellView 'test,nmos_2v,layout compacted symbolic'.
\w *WARNING* Could not find master cellView 'test,nmos_2v,layout compacted symbolic'.
\w *WARNING* There were errors during Update Components and Nets.
\w *WARNING* Run Check Against Source for details.
```

*Connectivity – Update – Components and Nets* preserves many-to-one and many-to-many mapping between the connectivity source and the layout cellview.

**Note:** You can use the *Edit – Undo* command to undo any changes that you made with the *Connectivity – Update – Components and Nets* command.
Updating Layout Parameters

To update the parameters of instances in the layout to match those in the schematic, follow these steps.

1. From the layout window, choose Connectivity – Update – Layout Parameters.

   If you preselected instances, Virtuoso XL checks parameters of those instances in the schematic against their parameters in the layout and updates the layout parameters when it finds differences.

   If you did not preselect instances, the CIW and the layout window prompt you to select devices whose parameters you want to update.

   If there is no schematic referenced by the layout, the Define Connectivity Reference form is displayed. If there is currently a connectivity reference for the layout, that information is displayed. You can change the connectivity reference in this form.

   **Note:** Connectivity-Update – Layout Parameters will update folded devices if the Update Parameter option is turned on in the XL Options form. Check the size of folded devices manually to avoid overriding the specified size.

2. Click OK on the Define Connectivity Reference form.

3. Select a device in the layout or schematic window.

   You can use Shift-click to select more than one instance. Virtuoso XL highlights the instance in the layout and the schematic window.

4. Move the cursor to the layout window and press Return.

   Virtuoso XL checks parameters of those instances in the schematic against their parameters in the layout and updates the layout parameters when it finds differences.

5. To see a list of the layout devices, move the cursor to the layout window and press F3.
The Update Layout Device List form is displayed. For more information about this form, see Change Inst View Device List Form / Update Layout Device List Form / Update Schematic Device List Form on page 452.

6. Click a device name in the Update Layout Device List form to select it.

   You can use Control-click to select more than one device and to deselect selected devices.

7. Click Apply.

   Virtuoso XL checks parameters of those instances in the schematic against their parameters in the layout and updates the layout parameters when it finds differences (unless you have set the lvsIgnore or ignore property on a device).

   Information about layout instances that contain properties that are not on their schematic counterparts is displayed in the Virtuoso XL Info window.
When Virtuoso XL checks or updates any instance, a Virtuoso XL Info window reports the updates that were made.

![Virtuoso XL Info window](image)

If updating parameters results in a change in the layout, Virtuoso XL updates the connectivity and indicates any shorts with markers.

If you have CDF callbacks **Update Layout Parameter** applies the callback first then applies all differing parameters on the source instance.

**Note:** **Update Components and Nets** does not update split transistors if the sizes of the resulting transistors are different because such transistors may be the result of manual splitting. If the layout figure selected for update is a transistor that has been folded, Virtuoso XL displays a message box that says that it is not updating the parameters.

8. Press **Escape** to exit the command.

**Note:** If you are using inherited connections in your design to assign more than one value to a global net, remember that the `netSet` properties on schematic instances, which specify the new value of a global signal, are not copied over with the instance from the schematic to the layout when you use the **Gen From Source**, **Pick from Schematic**, or **Update Layout Parameters** commands.

For more information, see the *Inherited Connections Flow Guide*.

### Updating Schematic Parameters

To update the parameters of devices in the schematic to match those in the layout, follow these steps.

**Note:** You must have the schematic window open in edit mode to use this command.

1. From the layout window, choose **Connectivity – Update – Schematic Parameters**.
If you preselected instances, Virtuoso XL checks parameters of the selected instances in the schematic against the parameters of the corresponding instances in the layout and updates the schematic parameters when it finds differences. A Virtuoso XL Info window reports the change.

If you did not preselect instances, the CIW and layout window prompt you to select instances whose parameters you want to update.

2. Select a device in either the layout or schematic window.

You can use Shift-click to select more than one device. Virtuoso XL highlights the device in the window.

3. Move the cursor to the layout window and press Return.

Virtuoso XL checks parameters of those instances in the schematic against their parameters in the layout and updates the schematic parameters when it finds differences (unless you have set the lvsIgnore or ignore property on a device).

4. To see a list of the layout devices, move the cursor to the layout window and press F3.

The Update Schematic Device List form is displayed. For more information about this form, see Change Inst View Device List Form / Update Layout Device List Form / Update Schematic Device List Form on page 452.

5. Select a device name in the Update Schematic Device List form.

You can use Control-click to select more than one device.

6. Click Apply.
Virtuoso XL checks parameters of those instances in the schematic against their parameters in the layout and updates the schematic parameters when it finds differences.

If you have CDF callbacks *Update Layout Parameter* applies the callback first then applies all differing parameters on the source instance.

When Virtuoso XL checks or updates any instance, a Virtuoso XL Info window reports the updates that were made.

![Virtuoso XL Info](image)

7. Press *Escape* to exit the command.

### Updating Device Correspondence

You can use the *Connectivity – Update – Device Correspondence* command to maintain connectivity in a layout when you want to

- Apply the connectivity from a symbol in the default schematic to a device in the layout
- Apply the connectivity from a group of symbols in the default schematic to a device or group of devices in the layout (*many-to-many* and *many-to-one* mapping)
- **Modifying Many-to-Many or Many-to-One Mapping Between Components**
- **Deleting Many-to-Many or Many-to-One Mapping Between Components**
- Apply the connectivity from a single device in the schematic to multiple devices in the layout.
- Apply connectivity to devices in a layout from a relevant schematic if you did not use Virtuoso XL to generate your layout
Note: Use the Connectivity – Update – Device Correspondence command with designs that are not LVS correct and have been created using Virtuoso XL. Use the Create – Device Correspondence command for LVS correct designs that have not been created using Virtuoso XL.

It is helpful to have the Show Incomplete Nets command active when you update device correspondences so that you can see that the connectivity you want is made.

You cannot preselect devices to use with the Update Device Correspondence command.

Important

The Connectivity – Update – Device Correspondence command does nothing to ensure that parameters are in sync, nor does add any missing parameters to the layout master.

Updating Connectivity from Schematic to Layout

To update the connectivity of a device or devices in the layout from a schematic,

1. From the layout window, choose Connectivity – Update – Device Correspondence. The schematic window prompts you to click on one or more symbols.

2. In a schematic window, click on the symbols whose connectivity you want to update in the layout.

Note: If you click on more than one symbol, the connectivity from those symbols is applied as a group to the device or devices you select in the layout. For example, as shown in the diagram, you might select N1, N2, R1, and R2 in the schematic to correspond to the device X in the layout.
3. With the cursor in the schematic window, press Return.
   The layout window prompts you to select one or more corresponding objects in the layout window.

4. In the layout window, click on the object or objects to associate with the symbol.
   
   **Important**
   Device correspondence will not list mismatched instances that have the same name as an existing instance. Rename the instance so that no name space overlap exists.

5. With the cursor in the layout window, press Return.
   
   **Important**
   The correspondence is defined only when you press Return.
   A message such as this one appears in the CIW saying that the connection has been made:
   
   `Virtuoso XL: Instance 'I1' in cellView 'overview amp1 layout' is renamed 'R21'.`
   
   If the *Show Incomplete Nets* command is active, flight lines show that the connection has been made. If the connection cannot be made, a message box identifies the problem.

6. To cancel the command, press Escape.
   
   **Note:** If you have set the *lvsIgnore* or *ignore* property on a device and then try to make that device correspond to another device, Virtuoso XL opens a dialog box asking you to verify the removal of the *ignore* or *lvsIgnore* property.
   
   If you click *OK*, Virtuoso XL removes the property and sets the two devices to correspond. If you click *Cancel*, Virtuoso XL does not set the devices to correspond.

### One-to-Many Assignment with Update Device Correspondence

You can also map a single device in the schematic to multiple devices in the layout using the *Update – Device Correspondence* command. You might want to do this if you need to map a device in the schematic to a flattened layout instance.

**Note:** You can map One-to-Many devices only between cellview pairs; that is, between the schematic and layout that correspond to each other and are at the same hierarchical level. For example, you cannot map an instance in the one to multiple instances accessed through a descend edit in the other.
To map an instance in the schematic to multiple instances in the layout, follow these steps.

1. From the layout Connectivity menu, choose Update – Device Correspondence.

   The schematic window and the CIW prompt you to choose one or more instances from the schematic.

2. Select a component from the schematic and press Return in the schematic window.

   The layout window and the CIW prompt you to choose one or more instances from the layout window.

   Preselected components are considered part of the selection unless you specifically deselect them.

   For example, in the diagram below, you can select the resistor A from the schematic to be mapped to the resistors R21, R22, and R6 in the layout.

3. Select the instances in the layout window and press Return.

   **Important**

   The correspondence is defined only when you press Return.

   Virtuoso XL checks your selection for connectivity. If the number of external nets in the layout selection is less than the number of external nets in the schematic selection, the mapping is not set.

   If the number of external nets in the layout selection is the same as or greater than the number of external nets in the schematic selection, a dialog box shows the nets that do not match and asks if you want to use Assign Nets to assign the nets of the layout components to nets corresponding to those in the schematic.

   a. Choose Yes.
The Assign Nets function highlights all unmatched nets from the schematic and layout.

The CIW and layout window prompt you to select a pin within an instance to add to a net.

You can press F3 to see a list of the nets in the design.

b. Click on a pin in one of the instances in the layout that you want to correspond to the instance in the schematic.

   The CIW and layout window prompt you to select a net to add the pin to.

c. Click on a net in the schematic or a path in the layout.

   Assign Nets connects the pin to the net.

d. When you have finished assigning nets, click Cancel on the Assign Nets Form or press Escape to cancel the Assign Nets command.

   The Assign Nets command is not canceled and the prompt to select a pin does not disappear until all the unassigned pins are connected to nets.

The Device Correspondence command checks to see that the components are all in one group.

   If they are all in the same group, Virtuoso XL leaves them in the group and the CIW displays a message that mapping has been successful.

   If only some of them are in a group, Virtuoso XL puts the rest in the group and the CIW displays a message that mapping has been successful.

   If none of them is in a group, Virtuoso XL creates a new group and puts them all in. The CIW displays a message that mapping has been successful.

   If they are in different groups, the CIW displays a message that mapping cannot be set because they are in different groups.

   If Virtuoso XL needs to create a group, the group is called 1xMTM. A property called 1xMTM is created and attached to the group in the layout. The property value is the name of all the selected components from the schematic. All the components selected from the layout are the members of the group.

   Note: More than one group in a layout cellview can be named 1xMTM. But a component can belong to only one such group.

   You can repeat this process to assign all the elements in the “many” group to nets.

4. Press Escape to cancel the Device Correspondence command.
Using Many-to-Many or Many-to-One Mapping

You can map multiple devices in the schematic to one or more devices (including shapes) in the layout by using the *Update Components and Nets* command. You might want to do this if you need to map a symbol or symbols in the schematic to a flattened layout instance.

**Note:** You can map many-to-many devices only between cellview pairs; that is, between the schematic and layout that correspond to each other and are on the same hierarchical level. For example, you cannot map an instance in the one to multiple instances accessed through a descend edit in the other.

To map multiple symbols in the schematic to one or more instances in the layout, follow these steps.

1. From the layout *Connectivity* menu, choose *Update – Device Correspondence*.

   The schematic window and the CIW prompt you to choose one or more instances from the schematic.

2. Select the components to be mapped from the schematic and press Return in the schematic window.

   The layout window and the CIW prompt you to choose one or more instances from the layout window.

   For example, in the diagram below, you could select the resistors R1 and R2, NMOS transistors N1 and N2, and devices X and Y from the schematic to be mapped to the devices L and M in the layout (many-to-many) or the block MISC in the layout (many-to-one).

   The number of external nets in the selected group in the schematic (in the example, nets A, B, and C) must be the same as the number of external nets in the selected group in the layout.
The software lets you ignore global nets, such as \textit{Vdd} and \textit{Gnd}, in the schematic because global nets are not indicated on many schematics. But if a schematic has global nets, they must be shown in the layout.

3. Select the instances in the layout window and press \textbf{Return}.

\textbf{Important}

The correspondence is defined only when you press \textbf{Return}.

If you are assigning many-to-many, the \textit{Device Correspondence} command checks to see that the layout components are all in one group.

- If they are all in the same group, Virtuoso XL leaves them in the group and the CIW displays a message that mapping has been successful.
- If only some of them are in a group, Virtuoso XL puts the rest in the group and the CIW displays a message that mapping has been successful.
- If none of them is in a group, Virtuoso XL creates a new group and puts them all in. The CIW displays a message that mapping has been successful.
- If they are in different groups, the CIW displays a message that mapping cannot be set because they are in different groups.

4. If Virtuoso XL needs to create a group, the group is called \textit{1xMTM}. A property called \textit{1xMTM} is created and attached to the group in the layout. The property value is the name...
of all the selected components from the schematic. All the components selected from the layout are the members of the group.

*Note:* More than one group in a layout cellview can be named lxMTM. But a component can belong to only one such group.

After the components are checked, the *Device Correspondence* command creates the new group.

### Modifying Many-to-Many or Many-to-One Mapping between Components

To modify a mapping between components, follow these steps.

1. From the layout window *Connectivity* menu, choose *Update – Device Correspondence*.
   
The layout window and the CIW prompt you to select one or more instances from the schematic window.

2. In the schematic, select one of the components in the group that you want to modify.
   
   All the components in the group are highlighted.

3. To delete some schematic components from or add others to the group, use Control-click to select those components to add and Shift-click to deselect the components to be deleted.

4. Press Return in the schematic window.

   The layout window and the CIW prompt you to choose one or more instances from the layout window.

5. To delete some layout components from or add others to the group, use Control-click to select those components to add and Shift-click to deselect the components to be deleted.

6. Press Return in the layout window.

   The new mapping is checked and, if found acceptable, created.

### Deleting Many-to-Many or Many-to-One Mapping between Components

To delete a mapping between components, use the *Update – Device Correspondence* command to set a one-to-one mapping between the components.
To delete a many-to-many (or many-to-one) mapping between components, follow these steps.

1. From the layout window Connectivity menu, choose Update – Device Correspondence.
   
The layout window and the CIW prompt you to select one or more instances from the schematic window.

2. In the schematic, select one of the components in the group whose mapping you want to delete.
   
   All the components in the group are highlighted.

3. Use Shift-click to deselect all the components except one.

4. Press Return in the schematic window.
   
   The layout window and the CIW prompt you to choose one or more instances from the layout window.

5. Use Shift-click to deselect the components to be deleted except the one you want to map to the schematic component.

6. Press Return in the layout window.
   
   The many-to-many (or many-to-one) mappings are deleted.

Creating Device Correspondence

For layout designs that have not been created using Virtuoso XL, or to bind an existing layout to another schematic, use the Create – Device Correspondence command to set correspondence between a schematic and layout.

The Create – Device Correspondence command

- Takes an LVS-correct layout design and migrates it into Virtuoso XL.
- Updates the layout instances’ and nets’ naming convention to the Virtuoso XL naming conventions so that legacy layouts can be reused in Virtuoso XL.
- Enables connectivity-driven layout editing.

Needed Mode

Needed mode supports the following;
Enforce master match.

Check connectivity once match is enforced.

mfactor and sfactor.

Prerequisites for the *Needed* mode are as follows;

- All selected instances must have the same master.
  - pins/instance in schematic must be mapped to a pin/instance in the layout.

To create device correspondence using the *Needed* mode;

1. Select *Create – Device Correspondence*.
2. The Create Device Correspondence form is displayed.
1. Leave the Show option to Instances. If you want to display the pins change the Show option to Pins.

2. By default the mode is set the Needed.

3. Change the option to Show Instance Master Name and the library, cell, and view names are displayed in the list boxes.

   Use the text field to highlight a particular device or set of devices. For example p* will highlight the devices starting with “P”.

4. Click on a device in the left list box. Only one instance or pin can be selected.

   The left list box lists the schematic instances and pins.

   The Message field prompts you to select layout objects.

5. Click on a corresponding layout instances or pin in the right list box. Multiple instances can be selected. If multiple instances are selected, they must all be in series or parallel. Multiple selection is only permitted if the selected schematic symbol has a sfactor or a hierarchical mfactor.

   The Message field prompts you to press map or select more layouts.

   **Note:** If you selected a NMOS instance in the schematic list box, and a PMOS instance in the layout list box then a Virtuoso XL dialog box appears. The dialog box indicates that you cannot map a NMOS device to a PMOS device and that you must make another selection.

   The Message field at the bottom of the XL Create Device Correspondence form tells you that you must adjust connectivity or layout selection.

6. To create device correspondence select the Map option. Once the instances are mapped the instance names and the names of the nets attached to the layout instances are changed according the Virtuoso XL naming convention.

   The list box updates showing the instances/pins that are mapped.

7. To unmap mapped objects, select the mapped objects and select Unmap.

   The objects are unmapped and placed in the schematic and layout list boxes.

8. Click Apply.

   The matches are removed from the map list box.

   The mapped instances are changed to the Virtuoso XL naming convention only when you click Apply or OK. Until then the information is merely visual in the form. You can cross-select, but no names are changed.
Computer Aided Mode

_Comp_**uter Aided** mode supports the following;

- Same check as _**N**eeded_ mode with additional LVS mappings
- Requires that all pins are mapped first then instances.
- Needs information from the user. The more information that you add to the form the better match you will have.
- Correspondence is set from the pins.

To create device correspondence using the _**C**omputer Aided_ mode,

1. Select _Create – Device Correspondence_.
2. Change the mode to _Computer Aided_.
   - The _Sort By_ option and _Show Instance Master Name_ is greyed out because these options only apply to instances.
   - The Message field asks you to create correspondence for all pins.
3. Change the _**S**how_ option to _Pins_.
   - Whenever there are unmatched pins in both the connectivity source and in the layout the _Pins_ option is displayed in the _Show cyclic_ field.
4. Select pins in the schematic and the corresponding pins in the layout list box.
5. Select _Map_ to map the pins.
6. Once all of the pins are mapped change the _Show_ option to _Instances_.
7. Turn on the _Show Instance Master Name_ option to view the library, cell, and view names.
8. A partial LVS algorithm will find possible mappings and will list them in the lower-left listbox. If the mappings are correct, the mappings can be mapped using the “>” button.
   - Mappings can also be performed manually as in _**N**eeded_ mode.
   - For every map/unmap operation, device correspondence incrementally searches for new proposed LVS matches. The more mappings are realized, the incremental LVS algorithm will recognize the connectivity and will propose more matches.

**Note:** If a large number of instances are unmapped or the number of different masters is small, the LVS algorithm might return old mappings. In this case, manual mapping from the top boxes will provide the algorithm with new data and the algorithm will start...
converging towards finding correct mappings. You are responsible for validating the possible mappings proposed by the algorithm.

**Updating the Connectivity Reference**

In Virtuoso XL, the default schematic is the schematic with the same cellview name as the layout.

You can change this setup so that you can update your layout using a schematic different from the default schematic by following these steps.

1. Choose *Connectivity – Update – Source*.

   The **Define Connectivity Reference** form is displayed.

2. Choose *Schematic* as the connectivity source type.

3. Do one of the following:

   - Type the name of the library, cell, and view of the device to update in *Library*, *Cell*, and *View* fields.
   - Click *Browse* to see the cells available in the Library Browser, and click on a cell name in the Library Browser.
   - Click *Sel by Cursor* and then click on an open schematic.

4. Click *Update layout cellviews hierarchically* to update the library setting for any lower-level cellviews in a hierarchical cellview, except...
Where the lower-level cellview is a leaf (i.e., there is no instance in the cellview)

The current setting points to a non-existent cellview.

Lower-level cellviews updated in this way are not saved until you confirm that you want to do so at the end of the session

5. Click OK.

Virtuoso XL sets the connectivity reference to the schematic you selected.

Changing the Device (Instance) View

To change the layout view of a device or selected set of devices in a hierarchical design, follow these steps.

1. From the layout window, choose Connectivity – Change Instance View.

The Change Instance View form is displayed.

2. Click on the device whose layout view you want to change (or type the name in the Dev Name field).

To select multiple devices, hold down Shift. You can move from one selected device to the next using Next and Previous. The Switch to View Name cyclic field changes to show the device name and available view names for the device displayed in the cyclic field. A dialog box appears and tells you when you reach the first or last device.

3. From the Switch to View Name cyclic field, select the view name you want.

4. To see all the devices in your layout and the view names available for them, click List Devices.
The Change Inst View Device List form is displayed, showing all devices associated with the layout you have open. For more information on this form, see Change Inst View Device List Form / Update Layout Device List Form / Update Schematic Device List Form on page 452.

5. Double click on a device name.
   The device name appears in the Dev Name field in the Change Instance View form.

6. In the Change Instance View form, click Apply.
   Virtuoso XL applies the new view name to the device.

Online Forms

- Change Instance View on page 451
- Change Inst View Device List Form / Update Layout Device List Form / Update Schematic Device List Form on page 452
- Creating Device Correspondence on page 444

Change Instance View

Dev Name specifies the name of a device whose layout view you want to change.
common selects multiple devices whose layout view you want to change to a common layout view. When common is selected, clicking on Apply associates the cell view displayed in the cyclic field to all the devices chosen (the number shown at the bottom of the form).

Switch to View Name changes to include the name of the device you select (Switch cell “res” to View Name) and lets you choose from the cyclic field the cell view to which you want to attach the device. The choices shown in the cyclic field are set by the stopList variable. If no cell view is associated with the device you choose, this field is not shown.

n of n fields displays the order and number of devices you have chosen to attach to a given cell view. You can enter data in these fields.

List Devices shows the devices present in your layout and the view names associated with each.

Change Inst View Device List Form / Update Layout Device List Form / Update Schematic Device List Form

The Device List form shows all the devices in the design. Several Virtuoso XL commands use the Device List form.

To select a device or devices on which you want to perform an action, click on the device you want to select, or hold down Control and click to select more than one device.

Search finds and selects the names of devices you specify.

You can type in the field to the right of the Search button:

- A device name
  The Device List form highlights the device and scrolls the list to make the device name visible.

- A part of a device name ("IN," for example)
  The Device List form highlights all the devices with “IN” as an element in the name (IN3, IN6, IN7, IN12, for example) and scrolls the list to make the first example visible.

- A regular SKILL expression using part of a device name (“*mos,” for example)
  The Device List form highlights all devices with the string “*mos”, (nmos, pmos, for example) and scrolls the list to make the first example visible.
Create Device Correspondence

From the Create-Device Correspondence command.

Options allows you to control the display of pins and instances.

- **Show** displays either pins or instances in the schematic and layout list boxes.
- **Sort By** sorts the instances by Instance Name or Master Name.
- **Show Instance Master Name** displays the library, cell, and view name for either the Instance Name or Master Name.

Needed allows you to traverse the schematic down to the symbols that have possible layout masters to be instantiated.

Computer Aided recognizes net names and inspects instances connected to these nets. After the pin correspondence is established, the instances that connect to the outside nets (which have a pin) can be scanned for possible matching groups. The matched objects are presented in the Device Correspondence form for validation. Upon validation, a new set of nets is recognized.

Next allows you to search for objects by using the filters provided at the top of each list box. Use the Next button to go to the next search result.

Right List Box lists the connectivity source.

Left List Box list the layout to be mapped to the connectivity source.

Map verifies the consistency of the selected set.

Unmap is a reverse action of the Map button. The mappings that you create can be unmapped without creating Virtuoso XL names.

Next allows you to search for a mapped objects by using the filters provided at the top of each list box. Use the Next button to go to the next search result.

Bottom List Box shows objects that have been mapped.

Bottom Left List Box (Computer Aided Mode only) shows the proposed matches to be validated.

Message displays what action you need to take next.
Troubleshooting

This chapter tells you what to do if unexpected results occur during the Virtuoso XL design process. It covers the following topics.

Problems with the Interface

- Invalid Markers from Previous Software Versions on page 455
- Options Form Does Not Appear on page 456
- Virtuoso XL Performance Is Slow on page 457

Problems with Editing

- Components Move Slowly on page 457
- Extra Probes Appear on page 457
- Layout Generation Options Form Does Not Keep Values from the Last Entry on page 458
- Parameters Not Updated on page 458
- Schematic Not Editable on page 458
- Warning to Update Your Design Appears at Startup on page 458

Problems with Connectivity

- Connections Not Made on page 459
- Incomplete Nets Command Does Not Recognize Connected Pins and Nets on page 459
- Markers for Nonexistent Overlaps and Shorts Appear on page 460
- Path Ends Not Accepted on page 460
- Placement and Routing Do Not Run on page 461
Problems with Cloning

- Structure Already Exists in the Layout on page 462
- Connectivity Structure is Different on page 463
- Master Cells are Different on page 463
- More Source Components are Selected than Target Components on page 464
- Properties are Different on page 465

Problems with Pick From Schematic

- Hierarchical Designs on page 467
- Global Terminals and Implicit Inherited Connection Terminals on page 467

Problems with Parameter Evaluation

- Netlisting Mode on page 467

Problems with the Interface

Invalid Markers from Previous Software Versions

If you are opening layouts you developed using previous versions of Virtuoso software, you can clean up the layout window and get rid of invalid markers.

1. From the layout window, choose Connectivity – Update – Components and Nets.
   Virtuoso XL resynchronizes the layout connectivity with the schematic connectivity.
2. From the layout window, choose Verify – Markers – Delete All.
   The markers disappear.
3. Choose Design – Save.
4. From the layout window, choose Window – Close.
The layout window closes.

5. From the schematic window, choose Window – Close.

6. The schematic window closes.

7. From the Command Interpreter Window (CIW), choose File – Open.
   The Open File form is displayed.

8. From the Library Name cyclic field, choose the library name of your design.

9. In the Cell Name field, type or select from the Cell Names list the cell name of your design.

10. From the View Name cyclic field, choose the view name of your design.

11. Click OK.
   The layout window appears, displaying the updated layout configuration.

12. In the layout window, choose Tools – Layout XL.
   The Layer Selection Window (LSW) and the schematic view appear.

Note: If you still have problems, make sure that lxExtractLayers is defined.

Options Form Does Not Appear

When you press F3 to see an options form listing possible choices for a command and the command does not have an options form, the message box shown below appears:

![Options Form Does Not Appear](image)
If you want the message box to appear every time you press F3 for a command that has no options form, click Yes.

If you do not want the message box to appear every time you press F3 for a command that has no options form, click No.

**Virtuoso XL Performance Is Slow**

The following considerations can help you optimize Virtuoso XL performance speed.

- Virtuoso XL is targeted for use in designs with fewer than 10,000 devices. The performance of the online extractor degrades substantially beyond that limit.
- Incomplete net information is updated after every edit. If you have many incomplete nets, Virtuoso XL might take longer to redisplay your design. Consider turning off *Show Incomplete Nets* when working on interconnect layers. Use the highlighting features of the path command or probing to determine connections or display only nets you are working on or that are relevant to the current task.
- If you need extra pins in the layout for feedthrough nets or substrate connections, consider adding those after you have completed the main portion of the design. Adding extra pins increases the number of nets the extractor manages and degrades performance.
- Define `lxExtractLayers` only on layers to be used as interconnect during wiring with Virtuoso XL. Using more layers than necessary causes the extractor to check all shapes on those layers, which slows performance. If you do not have interconnect to wells, do not make the well layers `lxExtractLayers`.

**Problems with Editing**

**Components Move Slowly**

If you find that the *Move* command works too slowly, especially when you have many iterated instances, turn off the layout window *Connectivity – Show Incomplete Nets* command or display only nets that are relevant to the current task.

**Extra Probes Appear**

To delete unwanted probes in the schematic or layout windows, while the Probe command is active,
➤ Click on an empty space in the design.

or at any time

➤ Press Control-Shift-L.

**Layout Generation Options Form Does Not Keep Values from the Last Entry**

The Layout Generation Options form displays default values each time it opens, it does not carry over values from the previous entry. If you want to use values from a previous entry, you must save these values to a template and then use the *Load* option on the form to open the template and update the fields in the form with the information in the template.

**Parameters Not Updated**

If the parameters are not updated the way you expect, check for the presence of the `lxParamsToIgnore` property on components of your design.

**Schematic Not Editable**

If you cannot make changes to the schematic,

- Check that the schematic is editable (the Schematic window *Design* menu shows the *Make Read Only* command, not the *Make Editable* command).

- Check with your system administrator that you have a license for the Virtuoso schematic editor (product 302).

**Warning to Update Your Design Appears at Startup**

When you open a design in Virtuoso XL and an Info window similar to this appears, it means that the schematic has been changed after the last time the layout was changed, so the layout is not the latest version of the design.

You can update the layout from the changed schematic by using the layout window *Connectivity – Check – Against Source* command and then using either the
Connectivity – Update – Components and Nets or Connectivity – Update – Device Correspondence commands as needed.

Problems with Connectivity

Connections Not Made

If the software does not accept an electrical connection you make, make sure that the following information is set in the technology file:

- Connectivity information in the lxRules class
  - lxExtractLayers (leConducting in version 4.3.4 and earlier) defines conducting layers
  - lxNoOverlapLayers (leOverlap in version 4.3.4 and earlier) defines layers that must not overlap

- Layer information in the layerRules class
  - equivalentLayers (leEquivalent in version 4.3.4 and earlier) defines layers that are electrically equivalent
  - viaLayers (leEquivalent in version 4.3.4 and earlier) defines layers used for vias and the layers they connect

Also check that the layer properties were set when you made the connection.

Incomplete Nets Command Does Not Recognize Connected Pins and Nets

If a pin is connected to a net but has the lxBlockExtractCheck property on it, the incomplete nets command does not recognize that the pin is connected.
If a layer you use for pcells is not listed in the technology file in the \texttt{lxExtractLayers} list, the incomplete nets command does not recognize incomplete nets noted on that layer. Furthermore, the probe code does not highlight any layer that is not listed in the technology file in the \texttt{lxExtractLayers} list.

**Markers for Nonexistent Overlaps and Shorts Appear**

If there are markers for nonexistent overlaps and shorts in the layout, make sure that the appropriate layer was active when the path was created.

If this does not fix the problem, make sure that the connectivity information (\texttt{lxExtractLayers} and \texttt{lxNoOverlapLayers}) and layer information (\texttt{equivalentLayers} and \texttt{viaLayers}) are set correctly in the technology file.

**Path Ends Not Accepted**

If the software rejects the final segment of a path or wire connection and displays the following message.

\begin{quote}
First or last segment of created path has length less than or equal to half the path width
\end{quote}

➤ Press Return to end a path instead of double clicking the mouse (do not double click to end a path).

or

➤ Set the layout window Design – Options – Display command to a snap mode other than \texttt{x-first} or \texttt{y-first}.

When you use the \texttt{x-first} or \texttt{y-first} snap modes, the double click often makes a notch in the path, which triggers the error message and can cause errors in mask layout.
Placement and Routing Do Not Run

Virtuoso XL does not let you place or route the elements of a design unless you have defined which layers of your layout design are conducting layers.

If you try to place or route your design before defining the conducting layers, you get an error message like this.

<table>
<thead>
<tr>
<th>Virtuoso XL</th>
</tr>
</thead>
</table>
| There are no ‘lxExtractLayers’ defined in library ‘cmos_test’.
To get Virtuoso XL connectivity, you need to update your technology file. |

Define the layers you want to be conducting layers as `lxExtractLayers` in the `lxRules` class of the technology file.

Virtuoso XL Does Not Recognize Physical Viases

If you have an existing design in which Virtuoso XL does not recognize physical vias, you need to add a property with name = “function”, value = “via” to the master cellview of the via or to the component description format (CDF) for via layout.

Moving Software Executables To a New Location

If you move software executables to a different location other than specified by manufacturing, the Virtuoso custom placer may take longer to start because it is searching for the executable.

Problems with Cloning

If a target cannot be cloned, no outline of the layout structure appears and a dialog box is displayed with the following message.

Cloning could not be completed.
Check for:
Target partially implemented
Connectivity Structure mismatch
Mismatch of parameters or masters

Structure Already Exists in the Layout

The structure or part of the structure that you selected as the target already exists in the layout. For example, here R1, R2, and Q1 in the layout were the source components and R4 (which is already implemented), R5, and Q2 were the target components.
**Connectivity Structure is Different**

The connectivity structure of the source and the target are not the same; for example, if you choose R1, R2, and Q1 as source components and R6, R7, and Q2 as target components.

**Master Cells are Different**

The master cells of the source and the target instances are not the same.
More Source Components are Selected than Target Components

More source components are selected than target components. For example, if you selected R1, R2, and Q1 as source components and selected only R4 and Q2 for target components.
Properties are Different

The properties of the source and target instances are not the same. For example, you could not use R1 and R2 in this diagram to clone R4 and R5 because their resistance value properties are different.

The target instance can have properties that are additional to the properties of the source instance, but the target instance must have all the same properties with the same values as the source instance has.
In the example below, you can use R1, R2, and Q1 as source components to clone R4, R5, and Q2 because the targets have the same properties and values as the sources, even though one of the targets has additional properties.

You can assign the property `lxParamsToIgnore` to source instances with properties that are different from the target instance properties, so that the source instances can be cloned successfully as target instances.

**Problems with Pick From Schematic**

If the `Create – Pick From Schematic` command fails, check that

- The instance is not already placed in the layout.
  
  If it is already placed, you cannot use `Create – Pick From Schematic` to place it again.

- All of the required masters and reference views are present, including
  
  - The default layout master for the specified schematic component.
  - Any layout master specified by the `useCell` attribute for the schematic instance or master.
  - The required layout view as derived from the `stopList`. 
Hierarchical Designs

If you push into a hierarchical schematic view beyond the layout stop point, you cannot use the *Create – Pick From Schematic* command to generate lower-level instances inside that top-level cell.

For example, assume there is a layout view derived from the specified *stopList* corresponding to a *cell1* instantiated in the schematic. If you descend into an instance of *cell1* and try to generate the lower-level instances, the command fails.

Global Terminals and Implicit Inherited Connection Terminals

*Create – Pick From Schematic* does not generate global terminals or implicit inherited connection terminals (i.e., wires with *netSet* properties).

Problems with Parameter Evaluation

Netlisting Mode

To ensure that CDF parameters are always evaluated correctly, make sure that the *CDS_Netlisting_Mode* shell environment variable is set to *Analog*.

To set *CDS_Netlisting_Mode*,

➤ Type the following commands in the CIW.

```bash
setShellEnvVar("CDS_Netlisting_Mode=Analog")
cdsSetNetlistMode()
```

To check which mode is currently set,

➤ Type the following command in the CIW.

```bash
cdsGetNetlistMode()
```

For more information on *CDS_Netlising_Mode*, see “Customizing the Simulation Environment” in the *Open Simulation System Reference*. 
Evaluating CDF Callbacks by Default

To ensure that SKILL callbacks defined on CDF parameters (for example, lxCombination) are evaluated by default, you must set the lxEvalCDFCallbacks environment variable to t.

lxEvalCDFCallbacks affects any CDF parameter with a callback that is executed whenever the value of the parameter changes.

For more information, see “Defining Parameters” in the Component Description Format User Guide.
Environment Variables

This appendix provides information on the names, descriptions, and graphical user interface equivalents for the Virtuoso® XL Layout Editor (Virtuoso XL) and wire editor environment variables.

Important

Only the Virtuoso XL environment variables documented in this chapter are supported for public use. All other Virtuoso XL environment variables, regardless of their name or prefix, and undocumented aspects of the environment variables described below, are private and are subject to change at any time.

- Virtuoso XL Layout Editor on page 471
- Wire Editor on page 607

For more information on setting environment variables, see “Setting Environment Variables” on page 62.

Inherited Environment Variable Settings

Many of the environment variables honored by Virtuoso XL are set in Virtuoso Layout Editor. Information on these environment variables is not duplicated in this section.

For more information on Virtuoso Layout Editor environment variables, see Appendix B, “Environment Variables,” in the Virtuoso Layout Editor User Guide.
Virtuoso XL Layout Editor

abutPerpSnapOn

layout abutPerpSnapOn 'boolean { t | nil }

Description

For information on this environment variable, see abutPerpSnapOn in the Virtuoso Layout Suite L User Guide.
allowRotation

`layoutXL.placement allowRotation 'boolean { t | nil }`

**Description**

Specifies whether the Virtuoso custom placer can rotate components as part of its optimization. If this variable is set to `nil`, the placer can move components but not rotate them.

The default is `t`.

**GUI Equivalent**

Command:  *Place – Placer*

Field:  *Allow Rotation*
autoAbutment

layoutXL autoAbutment 'boolean { t | nil }

Description

Abuts correctly prepared transistors automatically so that they can share pins.

The default is t.

When autoAbutment is set to t, devices that have not been abutted but are overlapping are abutted automatically.

Note: If the devices are already abutted, the automatic abutment feature does not re-abut them when you start Virtuoso XL.

GUI Equivalent

Command: Options – Layout XL
Field: Auto Abutment
**autoArrange**

`layoutXL autoArrange 'boolean { t | nil }

**Description**

Specifies whether Virtuoso XL automatically rearranges the windows on your desktop on startup.

The default is `t`.

**GUI Equivalent**

Command: `Options – Layout XL`

Field: `Auto Arrange Windows`
autoExtractEnabled

layoutXL autoExtractEnabled 'boolean { t | nil }

Description

For information on this environment variable, see autoExtractEnabled in the Virtuoso Layout Suite L User Guide.
autoExtractSaveEnabled

layoutXL autoExtractSaveEnabled 'boolean { t | nil }

Description

For information on this environment variable, see autoExtractSaveEnabled in the Virtuoso Layout Suite L User Guide.
autoSpace

layoutXL autoSpace 'boolean { t | nil }

Description

Switches on automatic spacing, which allows components with the properties vxlInstSpacingDir and vxlInstSpacingRule to be spaced automatically according to the values specified in the properties.

The default is t.

GUI Equivalent

Command: Options – Layout XL
Field: Auto Space
checkTimeStamps

layoutXL checkTimeStamps 'boolean { t | nil }

Description

Specifies whether Virtuoso XL checks the schematic time stamp and warns you if the schematic has changed since Virtuoso XL generated the corresponding layout.

The default is t.

GUI Equivalent

None.
checkOldIgnoredParamsProps

layoutXL checkOldIgnoredParamsProps 'boolean { t | nil }

Description

Specifies that Virtuoso XL uses the old lxIgnoredParams and lxIgnoreParamsForCAS properties when checking for parameters and properties to be ignored.

By default, Virtuoso XL uses the new lxParamsToIgnore and lxParamsToIgnoreForCheck properties.

⚠️ Caution

Enabling this environment variable can adversely affect the performance of Virtuoso XL.

GUI Equivalent

None.
ciwWindow

`layoutXL ciwWindow string pair_of_coordinates`

**Description**

Specifies the position of the command interpreter window (CIW) window on the screen.

The default is `((0 0) (0 0))`.

**GUI Equivalent**

None.
cloningDoExactMatch

layoutXL cloningDoExactMatch 'boolean { t | nil }

Description

Controls whether the instance parameter should be checked when finding a match between connectivity sources and connectivity targets.

GUI Equivalent

<table>
<thead>
<tr>
<th>Command</th>
<th>Create – Clone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Exact Match</td>
</tr>
</tbody>
</table>
**colorDraglines**

`layoutXL colorDraglines 'boolean { t | nil }

**Description**

Displays each dragline in a different color. The color cycle is defined in the Show Incomplete Nets form.

This environment variable is honored by the Create – Pick From Schematic, Create – Clone, Edit – Move, and Edit – Stretch commands.

**Tip**

For more information on displaying draglines, see also hideDraglinesForGlobalNets and showDraglinesForDistantConns.

**GUI Equivalent**

Command: Options – Layout XL
Field: Display – Draglines – Use Colors
compTypeRefLibs

layoutXL compTypeRefLibs 'string string

Description

Specifies the names of the reference libraries to which component type definitions are applied.

The default is none.

GUI Equivalent

Command: Design – Component Types
Field: Ref Libs
constraintAssistedMode

`layoutXL constraintAssistedMode 'boolean { t | nil }

Description

Specifies whether the Move and Stretch commands use the algorithms from the constraint-driven placer when placing components in a layout. When set to t, fixed constraints prevent locked objects from being moved and stretched.

The default is nil.

GUI Equivalent

Command: Options – Layout XL
Field: Constraint Assisted Move and Stretch
**createBoundaryLabel**

layoutXL createBoundaryLabel 'boolean { t | nil }

**Description**

Specifies whether the *Gen From Source* command automatically creates a label containing the library, cell, and view name of the design when creating a boundary from the Layout Generation Options form.

The default is `nil`.

**GUI Equivalent**

None.
crossSelect

layoutXL crossSelect 'boolean { t | nil }

Description

Specifies whether cross-selection is operational. Cross-selection means that when you select a component in the layout, the corresponding component is selected in the schematic and vice versa.

The default is t.

GUI Equivalent

Command: Options – Layout XL
Field: Cross Selection
deleteUnmatchedInsts

*layoutXL deleteUnmatchedInsts 'boolean \{ t | nil \}*

**Description**

Controls whether or not Virtuoso XL deletes from the layout instances that no longer exist in the schematic.

The default is `nil`, meaning that unmatched instances are not deleted when updating components and nets. Instead unmatched instances are indicated with a marker in the layout view.

**GUI Equivalent**

None.
**deleteUnmatchedPins**

`layoutXL deleteUnmatchedPins 'boolean { t | nil }

**Description**

Controls whether or not Virtuoso XL deletes from the layout pins that no longer exist in the schematic. The advantage of deleting redundant pins is that redundant nets and terminals are deleted from the layout view at the same time.

The default is `nil`, meaning that unmatched pins are not deleted when updating components and nets. Instead unmatched pins are indicated with a marker in the layout view.

**GUI Equivalent**

None.
**Environment Variables**

**drdUseNetName**

`layoutXL drdUseNetName 'boolean { t | nil }`

**Description**

Controls whether an object that is being moved retains the net name of the pin or instance terminal to which it was connected prior to the move. This in turn lets the design-rule-driven (DRD) editing functionality use the name to flag short violations during the move operation.

For example, when using DRD editing, if you move an object on net A towards an object on net B, the system displays a message warning you about the short violation. If the move leads to the object becoming disconnected from an I/O pin or instance terminal, the `drdUseNetName` environment variable controls the behavior as follows.

- When set to `t`, the object retains the net name of the pin or instance terminal to which it was connected prior to the move and DRD editing uses this name to flag short violations.
- When set to `nil`, the object does not retain the net name, Virtuoso XL is unaware of any connectivity associated with the object, and therefore no short message is displayed.

The original net assignment remains intact in both cases. The default value is `nil`.

**Additional Information**

- The `lxStickyNet` property overrides any setting of `drdUseNetName`. For information on `lxStickyNet`, see “Sticky Nets” on page 336.

- For information on design-rule-driven editing, see “Design-Rule-Driven Editing” in the Virtuoso Layout Editor Turbo User Guide.

- For information on how `drdUseNetName` affects DRD editing, see “Editing Commands in Virtuoso XL Layout Editor” in the Virtuoso Layout Editor Turbo User Guide.

**Important**

`drdUseNetName` has no effect when you stretch an object because assigned connectivity is always maintained during the stretch operation. However, if all the points of the object are selected prior to stretching, the stretch is in fact a move and the behavior is as described above.

**GUI Equivalent**

None.
extractEnabled

layout extractEnabled 'boolean { t | nil }

Description

For information on this environment variable, see extractEnabled in the Virtuoso Layout Suite L User Guide.
**extractShowMustConnectMarkers**

```plaintext
layout extractShowMustConnectMarkers 'boolean \{ t | nil \}
```

**Description**

For information on this environment variable, see `extractShowMustConnectMarkers` in the *Virtuoso Layout Suite L User Guide*. 
extractShowUnimplementedInstTerms

layout extractShowUnimplementedInstTerms 'boolean { t | nil }

Description

For information on this environment variable, see extractShowUnimplementedInstTerms in the Virtuoso Layout Suite L User Guide.
extractShowWeakMarkers

layout extractShowWeakMarkers 'boolean { t | nil }

Description

For information on this environment variable, see extractShowWeakMarkers in the Virtuoso Layout Suite L User Guide.
extractStopLevel

layout extractStopLevel 'int { 0 | 1 | 2 | ... | 32 }

Description

For information on this environment variable, see extractStopLevel in the Virtuoso Layout Suite L User Guide.
flightLineEnable

layoutXL flightLineEnable 'boolean { t | nil }

Description

Controls whether draglines are displayed during the Move, Stretch, Pick From Schematic, and Create Clone commands. The draglines connect the pins of the object you are moving to pins of the nearest objects.

The default is t.

For information on how to change the display of the draglines, see colorDraglines, hideDraglinesForGlobalNets, and showDraglinesForDistantConns.

GUI Equivalents

Command: Edit – Move
Edit – Stretch

Field: Display Draglines

Command: Create – Pick From Schematic
Create – Clone

Field: Draglines
**globalPlacement**

`layoutXL.placement globalPlacement 'boolean { t | nil }`

**Description**

Switches on the global placement function, which places components without regard to any initial placement. The placer optimizes global objectives such as total wire length and overlap violations between components might be ignored.

You would typically use this option the first time you place a design or to discard previous unsatisfactory placement results.

The default is `t`.

**GUI Equivalent**

| Command: | Place – Placer |
| Field:   | Global Placement |
**hideDraglinesForGlobalNets**

`layoutXL hideDraglinesForGlobalNets 'boolean { t | nil }

**Description**

Hides draglines for global nets and power and ground nets specified using the `lxGroundNetNames` and `lxSupplyNetNames` environment variables.

This environment variable is honored by the *Create – Pick From Schematic, Create – Clone, Edit – Move, and Edit – Stretch* commands.

**Tip**

For more information on displaying draglines, see also `colorDraglines` and `showDraglinesForDistantConns`.

**GUI Equivalent**

Command: *Options – Layout XL*

Field: *Display – Draglines – Hide Global Nets*
ignoreNames

layoutXL ignoreNames ’string “list_of_properties”

Description

Tip

Cadence recommends that you use the propsUsedToIgnoreObjs and propsUsedToIgnoreObjsForCheck environment variables instead of ignoreNames. However, ignoreNames, if defined, takes precedence.

Defines a list of properties that cause pins and instances to be ignored during Virtuoso XL updates. If any of the properties in the ignoreNames list is set to true for a particular instance then that instance is ignored.

The software searches for the properties in the order in which the names are listed in the environment variable. In the example below, the vxlIgnore property is used because it is first in the list.

    envSetVal(“layoutXL” “ignoreNames” ’string “vxlIgnore lvsIgnore ignore nlAction”)

You can also use the Layout XL Options form to set the ignoreNames environment variable to define other properties to use as ignore properties. For more information, see “Setting Environment Variables” on page 62.

Note: A schematic device that has the string property nlAction with a value of ignore behaves the same as one with an lvsIgnore or ignore property.

GUI Equivalent

Command: Options – Layout XL
Field: Ignore Names
IgnoredParams

layoutXL IgnoredParams ’string { lxIgnoredParams | dleIgnoredParams | lxStopList | dleStopList | lxUseCell | dleUseCell | maskLayout | ViewName | posi | instancesLastChanged | instNamePrefix | pin# | lxTimeStamp | dleSchExtractPath | xPlacementStatus }

Description

Tip

Cadence recommends that you use the paramsToIgnore environment variable instead of IgnoredParams. However, IgnoredParams, if defined, takes precedence.

Lists the parameters that are not reported as mismatches during Connectivity – Check – Against Source and are not updated by the Connectivity – Update – Components and Nets command.

The valid values are all the default values plus any other parameters you need to add.

Note: Do not remove the default values for IgnoredParams. The database-specific values are used by Virtuoso XL to traverse the schematic.

If there are devices in your schematic with properties that you do not want to be checked when you use the Connectivity – Check – Against Source command or transferred to the layout when you use the Connectivity – Update – Layout Parameters command, specify those device parameters to be excluded using the lxIgnoredParams property.

You can also set this property on layout components that are the same except for their parameters, so that the differing parameters are ignored by the cloning functionality.

For information on the properties mentioned above, see “Properties” on page 736.

GUI Equivalent

Command: Options – Layout XL
Field: Show Parameters to Ignore
IgnoreParamsForCAS

`layoutXL IgnoreParamsForCAS 'string string`

**Description**

Specifies a list of parameters and properties to be ignored when using the *Connectivity – Check – Against Source* command. Instances with any of the listed parameters or properties are not included in the check.

⚠️ **Important**

Cadence recommends that you use the `paramsToIgnoreForCheck` environment variable instead of `IgnoreParamsForCAS`. However, `IgnoreParamsForCAS`, if defined, takes precedence.

**GUI Equivalent**

- **Command:** *Check – Against Source*
- **Field:** None
incNetCycleHilite

layoutXL incNetCycleHilite 'boolean { t | nil }

Description

Controls whether the Show Incomplete Nets command cycles through the y0 through y9 drawing layers when displaying incomplete nets. When set to nil, the layer specified in the incNetHiliteLayer environment variable is used to highlight all incomplete nets.

The default is t.

GUI Equivalent

Command: Connectivity – Show Incomplete Nets
Field: Cycle
**incNetHiliteLayer**

```plaintext
layoutXL incNetHiliteLayer 'string { y1 | ... | y9 }
```

**Description**

Specifies which layer the *Show Incomplete Nets* command uses to display incomplete nets. The environment variable is effective only if the `incNetCycleHilite` environment variable is set to `nil`.

The default is `y0` drawing.

**GUI Equivalent:**

- **Command**: *Connectivity – Show Incomplete Nets*
- **Form Field**: *Cycle*
infoWindow

layoutXL infoWindow 'boolean { t | nil }

Description

Controls where messages issued by Virtuoso XL are displayed. When set to t, Virtuoso XL messages are displayed in a separate Info window. When set to nil, messages are sent to the CIW.

The default is t.

GUI Equivalent

None.
initAspectRatio

`layoutXL initAspectRatio 'float floating_point_number`

**Description**

Specifies the width-to-height aspect ratio of the design to be automatically generated or reinitialized. For example, a value of 1 specifies a square boundary, 0.5 specifies a boundary twice as high as it is wide, and 2 specifies a boundary twice as wide as it is high.

The default is 1.0.

**GUI Equivalent**

Command:  *Design – Gen From Source*

Field:  *Aspect Ratio (W/H)*
**initAspectRatioOption**

`layoutXL initAspectRatioOption 'string { Aspect Ratio W/H | Boundary Width | Boundary Height }`

**Description**

Specifies which of the boundary options is set when you open the Layout Generation Options form.

The default is Aspect Ratio W/H.

**GUI Equivalent**

None.
initBoundaryLayer

layoutXL initBoundaryLayer 'string string

Description

Defines the layer Virtuoso XL uses to draw the design boundary in the Virtuoso layout window.

The default is prBoundary drawing.

Note: If the layer purpose pair prBoundary boundary is not defined in the technology file, then prBoundary drawing is used.

GUI Equivalent

None.
**initCreateBoundary**

`layoutXL initCreateBoundary 'boolean { t | nil }`

**Description**

Controls whether Virtuoso XL creates a design boundary in the Virtuoso layout window.

The default is `nil`.

**GUI Equivalent**

Command:  
*Design – Gen From Source*

Field:  
*Boundary*
initCreateInstances

layoutXL initCreateInstances 'boolean { t | nil }

Description

Specifies that Virtuoso XL is to generate instances during the Design – Gen From Source command. The default is t.

GUI Equivalent

Command: Design – Gen From Source
Field: Instances
initCreateMTM

`layoutXL initCreateMTM 'boolean { t | nil }

Description

Specifies that Virtuoso XL is to preserve existing many-to-many mappings during Design – Gen From Source. The default is t.

GUI Equivalent

Command: Design – Gen From Source
Field: Preserve Mappings
initCreatePins

layoutXL initCreatePins ‘boolean { t | nil }

Description

Specifies that Virtuoso XL is to create pins during the Design – Gen From Source command. The default is t.

GUI Equivalent

Command: Design – Gen From Source
Field: Create
**initDoFolding**

layoutXL initDoFolding `boolean { t | nil }

**Description**

Specifies that Virtuoso XL is to fold prepared transistors into separate fingers during Design – Gen From Source. The default is nil.

**GUI Equivalent**

Command:  *Design – Gen From Source*
Field:  *Transistor Folding*
**initDoStacking**

`layoutXL initDoStacking 'boolean { t | nil }

**Description**

Specifies that Virtuoso XL is to abut MOS transistors in a specific order during Design – Gen From Source. The default is nil.

**GUI Equivalent**

Command: Design – Gen From Source
Field: Transistor Chaining
initEstimateArea

`layoutXL initEstimateArea 'boolean { t | nil }`

**Description**

Determines whether Virtuoso XL estimates the size of the design boundary based on the **Boundary Area Estimation** options in the Layout Generation Options form.

The default is `t`.

**GUI Equivalent**

*Command:*  *Design – Gen From Source*

*Field:*  *Boundary Area Estimation* groupbox
initGlobalNetPins

layoutXL initGlobalNetPins 'boolean { t | nil }

Description

Determines whether Virtuoso XL Gen from Source command creates pins for global nets. The default is t.

GUI Equivalent

Command: Design – Gen From Source
Field: I/O Pins
initIOLabelType

layoutXL initIOLabelType 'string ( Label | Text Display | None )

Description

Specifies the type of label that Virtuoso XL generates for I/O pins. When set to None, no pin label is created.

This setting is honored by the Gen From Source and Pick From Schematic commands.

The default is Label.

Important

Virtuoso XL does not create labels for symbolic pins.

Note: For initIOLabelType to take effect, you must set the Layout L environment variable createPinLabel to t. For information on createPinLabel, see Appendix B, “Environment Variables,” in the Virtuoso Layout Editor User Guide.

GUI Equivalent

Command: Design – Gen From Source
Field: Pin Label Shape
initIOPinLayer

layoutXL initIOPinLayer 'string “layer_purpose_pair”'

Description

 Specifies which of the layers defined in the technology file is used as the pin layer when you choose the Gen from Source or Update – Components and Nets commands.

The default value is the current entry layer and purpose from the LSW.

Arguments

layer_purpose_pair

Specifies the layer and purpose on which pins are created. The string must be enclosed in quotation marks; for example, “metal1 drawing”

GUI Equivalent

Command:  Design – Gen From Source
Field:     Layer
initIOPinName

`layoutXL initIOPinName 'string string`

**Description**

Specifies the name of the pins generated when you use the *Gen from Source* or *Update Components and Nets* command.

**GUI Equivalent**

- **Command:** *Design – Gen From Source*
- **Field:** *Master*
initPinHeight

`layoutXL initPinHeight 'float floating_point_number`

Description

Specifies the pin height used when you choose the Gen from Source or Update Components and Nets command.

GUI Equivalent

Command: Design – Gen From Source
Field: Height
initPinMultiplicity

layoutXL initPinMultiplicity 'int integer

Description

Specifies the number of pins for each type to be placed.

GUI Equivalent

Command: Design – Gen From Source
Field: Num
initPinWidth

layoutXL initPinWidth 'float floating_point_number

Description

Specifies the pin width used when you choose the Gen from Source or Update Components and Nets command.

GUI Equivalent

Command: Design – Gen From Source
Field: Width
initPrBoundaryH

layoutXL initPrBoundaryH 'float floating_point_number

Description

Specifies the height of the design boundary used when you choose the Gen from Source or Update Components and Nets command. The default is 10.0 or the last value.

GUI Equivalent

Command:    Design – Gen From Source
Field:       Boundary Height
**initPrBoundaryW**

```
layoutXL initPrBoundaryW 'float floating_point_number
```

**Description**

Specifies the width of the design boundary used when you choose the *Gen from Source* or *Update Components and Nets* command. The default is 10.0 or the last value.

**GUI Equivalent**

Command: *Design – Gen From Source*

Field: *Boundary Width*
initSymbolicPins

layoutXL initSymbolicPins ’boolean { t | nil }

Description

Determines whether or not Virtuoso XL creates symbolic pins. The default is nil.

GUI Equivalent

Command: Design – Gen From Source
Field: Pin Type
initUtilization

layoutXL initUtilization 'float floating_point_number

Description

Specifies the percentage of area within the design boundary that is filled when the *Boundary Area Estimation* options are used in the Layout Generation Options form. The utilization value is applied and the boundary calculated only after any chaining and folding is complete.

The default is 25.0.

GUI Equivalent:

Command  
*Design – Gen From Source*

Form Field  
*Utilization (%)*
layoutWindow

layoutXL layoutWindow 'string pair_of_coordinates

Description

Determines the position of the layout window when Virtuoso XL is initiated. The default is (0 0) (0 0).

GUI Equivalent

None.
IswWindow

layoutXL lswWindow 'string pair_of_coordinates

Description

Determines the position of the layer selection window (LSW) when Virtuoso XL is initiated. The default is ((0 0) (0 0)).

GUI Equivalent

None.
**lxAllowPseudoParallelNets**

`layoutXL lxAllowPseudoParallelNets 'boolean { t | nil }`

**Description**

Enables the detection of pseudo-parallel nets when chaining is turned on. A pseudo-parallel net connects nodes that are always the same voltage, so the current cannot pass through the net. The default is `nil`.

**GUI Equivalent**

None.

**Additional Information**

To see the significance of pseudo-parallel nets, consider a symmetric series of N-transistors, A and B, tied in a series pull-down chain to ground, as shown in the following figure.

Both A and B are folded into two legs, with A1, net_PP, and B1 connected in series to ground, and likewise A2 and B2 connected in series to ground, with both pairs in parallel. The nodes between A1 and B1 and between A2 and B2 are electrically equivalent. In reality no current passes through that connection and it need not physically be made.

The Assura™® verification tool understands pseudo-parallel connections and does not report incomplete nets in such situations.
lxDeltaWidth

layoutXL lxDeltaWidth 'float floating_point_number

Description

Determines the effective width of folded transistors. The effective total width is defined as
\[ W_{\text{eff}} = M \times (W + lxDeltaWidth) \]
where \( M \) is the number of fingers and \( W \) is the identical width of each transistor.

GUI Equivalent

None.
**lxEvalCDFCallbacks**

`layoutXL lxEvalCDFCallbacks 'boolean { t | nil }`

**Description**

Causes any SKILL callbacks defined on CDF parameters to be evaluated by default. The callbacks are evaluated in the order in which they are listed in the CDF.

⚠️ **Important**

`lxEvalCDFCallbacks` affects any CDF parameter with a callback that is executed whenever the value of the parameter changes.

For more information, see “Defining Parameters” in the *Component Description Format User Guide*.

The default is `nil`.

**GUI Equivalent**

None.
lxFingeringNames

layoutXL lxFingeringNames 'string string

Description

Specifies property names other than fingers or finger to split the number of fingers. The default is finger and fingers.

GUI Equivalent

Command: Options – Layout XL
Field: pCell fingering Names
lxGenerationOrientation

layoutXL lxGenerationOrientation 'string string

Description

Lets you control the orientation of Virtuoso XL layout devices generated from the schematic. The default is preserved, where there is no change in device orientation.

GUI Equivalent

None.
lxGenerationTopLevelOnly

layoutXL lxGenerationTopLevelOnly 'boolean { t | nil }

Description

Generates layout instances only from top-level schematic instances during the Design – Gen From Source and the Create – Pick From Schematic commands. Hierarchical schematic instances are not generated in the layout.

The default is nil.

GUI Equivalent

None.
lxGetSignifDigits

layoutXL lxGetSignifDigits 'cyclic { 0 | 3 | 4 ... 30 }

Description

Controls the number of significant digits used by Virtuoso XL when calculating the value of layout CDF parameters.

When set, lxGetSignifDigits overrides the setting of aelGetSignifDigits for calculations in the layout cellview. This lets you specify a different precision to be used in layout cellview calculations; for example, those involving parameterized cells.

Valid values are 0 and any integer between 3 and 30. The default is 0, which means that the environment variable is switched off. The higher the value you set, the more precise are the values used in the calculations.

GUI Equivalent

None.
lxGroundNetNames

`layoutXL lxGroundNetNames 'string string`

**Description**

To achieve optimized chaining results in Virtuoso XL device level (i.e. flat) schematics, you can set the environment variable `lxGroundNetNames` to specify the ground net name.

The default values are `gnd, gnd!, gnd:, vss, vss!, and vss:`.

**Note:** The `lxGroundNetNames` environment variable can also be used by the Virtuoso placer to filter out ground nets. The placer would not consider wire length for ground nets during placement.

**GUI Equivalent**

None.
lxInitResetSource

layoutXL lxInitResetSource 'boolean { t | nil }

Description

If set to t, prompts you to reset the source before starting Virtuoso XL from the layout. The default is nil.

GUI Equivalent

None.
lxLocalAbutment

layoutXL lxLocalAbutment ’boolean { t | nil }

Description

Controls device abutment in layout editing mode and unabutment during property edits.

The default is nil, which means that when you open a layout view in Virtuoso XL, the tool automatically abuts any devices which overlap but have not been abutted previously. If the devices are already abutted, they are not re-abutted.

If lxLocalAbutment is set to t,

- Devices are not abutted automatically when you open a cellview in Virtuoso XL. The software automatically abuts only objects that are subsequently edited interactively.

- Devices are not unabutted and re-abutted during when you change a property on the cellview.

GUI Equivalent

None.
Environment Variables

lxSchExtractTopLevelOnly

layoutXL lxSchExtractTopLevelOnly 'boolean { t | nil }

Description

When the lxSchExtractTopLevelOnly environmental variable is set to t, then only the top-level of the schematic is extracted. Virtuoso XL extracts the schematic only when needed when the following commands are invoked: Update Components and Nets, Gen From Source, Check Against Source, Update Device Correspondence, and Cloning. The lower level schematics are not extracted, even if they need to be extracted. The default is nil.

GUI Equivalent

None.
IxStackMinimalFolding

layoutXL IxStackMinimalFolding 'boolean { t | nil }

Description

Controls folding to yield the minimum number of legs for each folded device. If set to t, the folding will yield the minimal number of fingers, whether it is even or odd. The default is nil.

GUI Equivalent

Command: Options – Layout XL
Field: Generate Minimal Folding
**lxStackPartitionParameters**

`layoutXL lxStackPartitionParameters 'string "(integer integer)"`

**Description**

Sets the device chaining partition parameters.

The first integer controls how far up the hierarchy Virtuoso XL looks for opportunities to abut devices. A value of 1 permits abutment of devices only within the same gate. A value of 2 permits abutment of any devices in the same master. Higher values permit abutment with devices at progressively higher levels of the design hierarchy.

The second value sets the maximum number of devices permitted in a chain.

The default is "(1 100)".

**GUI Equivalent**

None.
IxSupplyNetNames

layoutXL IxSupplyNetNames ’string string

Description

To achieve optimized chaining results in Virtuoso XL device level (i.e. flat) schematics, you can set the environment variable IxSupplyNetNames to specify the power net name.

Default values are vcc, vcc!, vcc:, vdd, vdd!, and vdd:.

Note: The IxSupplyNetNames environment variable can also be used by the Virtuoso placer to filter out power nets. The placer would not consider wire length for supply nets during placement.

GUI Equivalent

None.
**lxToggleLocalAbutment**

```
hiSetBindKey("Layout" "<Key>" "lxToggleLocalAbutment()"")
```

**Description**

A user-defined bindkey that toggles the `lxLocalAbutment` environment variable.

For more information, see “lxLocalAbutment” on page 536.

**GUI Equivalent**

None.
lxUseLibList

layoutXL lxUseLibList 'string list_of_libraries

Description

Specifies a list of libraries for Virtuoso XL to search for a layout view that corresponds to a particular schematic or symbol view.

When generating or updating layout components from a schematic or symbol view, Virtuoso XL uses the layout cellview specified by the lxUseCell property for the schematic or symbol in question.

If lxUseCell is not specified, Virtuoso XL searches for a layout cellview of the same name stored in the same library as the schematic or symbol cellview. If there is no such cellview, the software searches any libraries specified by lxUseLibList.

The software searches the libraries in the sequence in which they are listed and uses the first layout cellview it finds with a name that matches the name of the schematic or symbol. If it is unable to find a corresponding layout cellview, the software issues a warning message.

The following commands honor the lxUseLibList environment variable.

- Design – Gen From Source
- Create – Clone
- Create – Pick From Schematic
- Edit – Transistor Chaining
- Edit – Transistor Folding
- Connectivity – Check – Against Source
- Connectivity – Update – Components and Nets
- Connectivity – Update – Device or Schematic Parameters

GUI Equivalent

None.
**lxWidthTolerance**

`layoutXL lxWidthTolerance 'string pair_of_coordinates`

**Description**

Sets the allowed device width variation for folding. Of the two positive numbers provided as the value, the first in the list is the absolute value of the negative tolerance; the second is the positive tolerance. The default is `((0 0) (0 0))`.

**GUI Equivalent**

None.
maintainConnections

layoutXL maintainConnections 'boolean { t | nil }

Description

Enables the automatic *Maintain Connections* option when Virtuoso XL is initiated. The default is nil.

GUI Equivalent

Command:   Options – Layout Editor
Field:     *Maintain Connections*
mfactorNames

layoutXL mfactorNames 'string "list_of_names"

Description

Lists the names of properties used in the schematic to specify the multiplication factor (mfactor) for transistors.

The default is “m M”.

Note: The mfactor property value can be expressed as an integer or an expression.

Arguments

list_of_names

A list of mfactor property names each separated by a space. The list must be enclosed in quotation marks; for example, “m M”

GUI Equivalent

Command: Options – Layout XL
Field: mfactor Names
mfactorSplit

`layoutXL mfactorSplit 'boolean { t | nil }`

**Description**

Controls whether Virtuoso XL places schematic devices with the `mfactor` property as multiple devices in the layout.

The default is `t`.

**Note:** You can override this environment variable for a given instance by setting the boolean property `lxMfactorSplit`.

**GUI Equivalent**

Command: *Options – Layout XL*

Field: *Generate Multiple Instances*
mfactorSplitParamNames

layoutXL mfactorSplitParamNames 'string "list_of_parameters"

Description

Lists the names of the device parameters to be calculated or checked during Design – Gen From Source, Connectivity – Check – Against Source, Connectivity – Update – Layout Parameters.

Virtuoso XL checks each instance for the listed parameter names and changes or updates the value of the matching parameters such that

\[
    \text{number of mfactored instances} \times \text{mfactorSplit value} = \text{source parameter value}
\]

The default value is “w”.

Important

Virtuoso XL currently checks only for width parameters.

Arguments

list_of_parameters

A list of parameter names each separated by a space. The list must be enclosed in quotation marks; for example, “w”

GUI Equivalent

Command: Options – Layout XL
Field: mfactor Split Param Names
moveAsGroup

layoutXL moveAsGroup 'boolean { t | nil }

Description

Determines whether or not to move groups of components bound by placement constraints as a unit. The default is t.

GUI Equivalent

Command:   Edit – Move
Field:     together
openWindow

layoutXL.placement openWindow 'boolean { t | nil }

Description

Opens a window. The default is nil.

GUI Equivalent

Command: Place – Placer
Field: Save As
optimizePlacement

`layoutXL.placement optimizePlacement 'boolean { t | nil }

Description

Switches on the *Optimize Placement* option on the Auto Placer form which determines whether detailed placement passes are run. The default is `t`.

GUI Equivalent

Command: *Place – Placer*
Field: *Optimize Placement*
paramsToIgnore

layoutXL paramsToIgnore "string "list_of_parameters"

Description

Important

paramsToIgnore has been introduced to replace the IgnoredParams environment variable. In the current release, if IgnoredParams is defined, it supersedes paramsToIgnore.

Lists the parameters that are ignored by the following Virtuoso XL update and check commands.

- Design – Gen From Source
- Create – Pick From Schematic
- Create – Clone
- Connectivity – Check – Against Source
- Connectivity – Update – Components and Nets
- Connectivity – Update – Layout Parameters
- Connectivity – Update – Schematic Parameters

The following parameters are always ignored.

dleIgnoredParams dleSchExtractPath dleStopList dleUseCell
instancesLastChanged instNamePrefix
lxIgnoredParams lxIgnoreParamsForCAS lxIgnoreParamsForCheck
lxMFactorNum lxParamsToIgnore lxParamsToIgnoreForCheck
lxPlacementStatus lxStopList lxTimeSpan lxUseCell
maskLayoutViewName pin# posi

Arguments

list_of_parameters

A list of parameter names each separated by a space. The list must be enclosed in quotation marks.
GUI Equivalent

Command:   Options – Layout XL
Field:     Generation – Show Parameters to Ignore
**paramsToIgnoreForCheck**

```
layoutXL paramsToIgnoreForCheck 'string "list_of_parameters"
```

**Description**

⚠️ **Important**

`paramsToIgnoreForCheck` has been introduced to replace the `IgnoreParamsForCAS` environment variable. In the current release, if `IgnoreParamsForCAS` is defined, it supersedes `paramsToIgnoreForCheck`.

Lists the parameters to be ignored during the *Connectivity – Check – Against Source* command. Mismatches for any of the listed parameters are not reported by the check.

The list automatically includes all of the parameters specified by the `paramsToIgnore` environment variable.

**Arguments**

- `list_of_parameters`

  A list of parameter names each separated by a space. The list must be enclosed in quotation marks.

**GUI Equivalent**

- **Command:** *Options – Layout XL*
- **Field:** *Verification – Show Parameters to Ignore*
paramTolerance

layoutXL paramTolerance 'float floating_point_number

Description

Specifies the relative tolerance allowed when comparing values between the layout and the schematic.

This tolerance is applied by any function that deals with parameters, including

- Connectivity – Check – Against Source
- Connectivity – Update – Layout Parameters
- Connectivity – Update – Schematic Parameters

The default is 1E-6.

GUI Equivalent

Command: Options – Layout XL
Field: Tolerance
**pathLayerFilter**

`layoutXL pathLayerFilter 'string "list_of_layers"`

**Description**

Specifies the layers to be ignored by the Choose which object to use dialog box when using the Change Layer option on the Create Path form; for example,

```
layoutXL pathLayerFilter string "met1 met2 act"
```

This means that objects on met1, met2, or act are not listed in the Choose which object to use dialog. If the `pathLayerFilter` excludes all layers, then the path command reverts to the LSW for layer selection.

**Arguments**

`list_of_layers`

A list of layer names each separated by a space. The list must be enclosed in quotation marks; for example,

```
"met1 met2 act"
```

**GUI Equivalent**

Command: `Create – Path`

Field: `Change Layer`
pathProbe

layoutXL pathProbe 'boolean { t | nil }

Description

Determines whether the Path command automatically highlights the net attached to the shape under the starting point for the path. It also changes the current layer to the layer of the shape under the starting point for the path.

The default is t.

GUI Equivalent

Command: Create – Path
Field: Probe Nets
pathPurposeList

layoutXL pathPurposeList 'string "list_of_purposes"

Description

Specifies the layer purpose order used when setting the entry layer in the layer selection window (LSW).

The default is “drawing net pin”.

For example, if you point to a pin with its shape on the metal1 pin layer purpose, and the layer is in the technology file, the entry layer changes to the metal1 drawing layer.

Arguments

list_of_purposes

A list of layer purpose names each separated by a space. The list must be enclosed in quotation marks; for example, “drawing net pin”

GUI Equivalent

Command: Create – Path, Create – Rectangle, and Create – Polygon
Field: Layer Purpose Order
**pathSwitchLayer**

```lisp
layoutXL pathSwitchLayer 'boolean { t | nil }
```

**Description**

Controls whether the entry layer remains the same as it was prior to starting the *Create – Path* command or switches to the layer clicked at the first path point.

The default is `t`.

**GUI Equivalent**

- **Command:** *Create – Path*
- **Field:** *Change Layer*
pathSwitchPurpose

`layoutXL pathSwitchPurpose 'boolean { t | nil }

Description

Controls whether to switch purpose during the `Create – Path`, `Create – Rectangle`, and `Create – Polygon` commands.

When switched on, Virtuoso XL parses the `pathPurposeList` and searches the technology file and switches to the first layer purpose that matches one of the layer purposes specified in the list.

The default is `t`.

GUI Equivalent

Command: `Create – Path`, `Create – Rectangle`, and `Create – Polygon`  
Field: `Switch Layer Purpose`
pinTextSamePurpose

layoutXL pinTextSamePurpose 'boolean { t | nil }

Description

Creates the pin name label on the same layer purpose as the pin it corresponds to. When set to nil, the pin name is created on the purpose defined by pinTextPurpose.

The default is nil.

For more information on pinTextPurpose, see the Virtuoso Layout Editor User Guide.

GUI Equivalent

None.
preserveTerminalContacts

`layoutXL preserveTerminalContacts 'boolean { t | nil }`

**Description**

Instructs the abutment utility to maintain terminal contacts when abutting devices.

The default is `nil`.

**Tip**

To set this environment variable for a single instance, create a boolean property called `preserveTerminalContacts` with a value of `t` on the instance in question. This lets you preserve contacts on selected instances while dropping contacts for all other instances.

**GUI Equivalent**

None.
probeCycleHilite

layoutXL probeCycleHilite 'boolean { t | nil }

Description

Specifies whether the *Probe* command cycles through the hilite drawing0 – hilite drawing9 entry layers for showing probes. The default is t.

GUI Equivalent

Command: *Connectivity – XL Probe*
Field: *Cycle*
probeDevice

layoutXL probeDevice 'boolean { t | nil }

**Description**

Determines whether or not the *Probe* command highlights devices on the schematic and layout. The default is *t*.

**GUI Equivalent**

- Command: *Connectivity – XL Probe*
- Field: *Devices*
**probeHiliteLayer**

```plaintext
layoutXL probeHiliteLayer 'string { hilite | drw to hilite | dr9 layers }
```

**Description**

Determines whether the **hilite** entry layer or another layer is used to display probes. The default is **hilite drawing**.

**GUI Equivalent**

- **Command:** *Connectivity – XL Probe*
- **Field:** *Display Layer*
**probeInfoInCIW**

```plaintext
layoutXL probeInfoInCIW "boolean { t | nil }
```

**Description**

Determines whether or not the output of the *Probe* command appears in the CIW as well as in the Probe form. The default is nil.

**GUI Equivalent**

- **Command:**  *Connectivity – XL Probe*
- **Field:**  *Send Messages to CIW*
**probeNet**

```plaintext
layoutXL probeNet 'boolean { t | nil }
```

**Description**

Determines whether or not nets can be highlighted in the schematic and layout by the *Probe* command. The default is *t*.

**GUI Equivalent**

- **Command:** Connectivity – XL Probe
- **Field:** Nets
probePin

layoutXL probePin 'boolean { t | nil }

Description

Determines whether or not pins can be highlighted on the schematic and layout by the *Probe* command. The default is `t`.

GUI Equivalent

Command: *Connectivity – XL Probe*
Field: *Pins*
propsUsedToIgnoreObjs

`layoutXL propsUsedToIgnoreObjs 'string "list_of_properties"`

**Description**

**Important**

`propsUsedToIgnoreObjs` is introduced to replace the `ignoreNames` environment variable. In the current release, if `ignoreNames` is defined, it supersedes `propsUsedToIgnoreObjs`.

Lists the properties that cause pins and instances to be ignored during the following Virtuoso XL commands. Any object with one of the listed properties set to `t` is ignored during these operations.

- **Design – Gen From Source**
- **Create – Pick From Schematic**
- **Create – Clone**
- **Connectivity – Check – Against Source**
- **Connectivity – Update – Components and Nets**
- **Connectivity – Update – Layout Parameters**
- **Connectivity – Update – Schematic Parameters**

By default, the list the contains the following properties.

```plaintext
ignore lvsIgnore lxRemoveDevice nlAction
```

**Arguments**

`list_of_properties`

A list of property names each separated by a space. The list must be enclosed in quotation marks.

**GUI Equivalent**

Command:  
`Options – Layout XL`
Field: Generation – Show Properties Used to Ignore Objects
propsUsedToIgnoreObjsForCheck

```
layoutXL propsUsedToIgnoreObjsForCheck 'string "list_of_properties"
```

**Description**

> **Important**

propsUsedToIgnoreObjsForCheck is introduced to replace the ignoreNames environment variable. In the current release, if ignoreNames is defined, it supersedes propsUsedToIgnoreObjsForCheck.

Lists the properties used to ignore objects during the Connectivity – Check – Against Source command. Any object with one of the listed properties set to `t` is ignored during this operation.

The list automatically includes all of the parameters specified by the propsUsedToIgnoreObjs environment variable.

**Arguments**

`list_of_properties`

A list of property names each separated by a space. The list must be enclosed in quotation marks.

**GUI Equivalent**

Command: `Options – Layout XL`

Field: `Verification – Show Properties Used to Ignore Objects`
rowGroundLayer

layoutXL.placement rowGroundLayer 'string string

Description

Specifies the layer used for ground rails during row generation.

GUI Equivalent

Command: Place – Placement Planning
Field: Layer
rowGroundName

`layoutXL.placement rowGroundName 'string string`

**Description**

Specifies the net name used for ground rails during row generation.

**GUI Equivalent**

Command: *Place – Placement Planning*

Field: *Net Name*
rowGroundWidth

layoutXL.placement rowGroundWidth 'float floating_point_number

Description

Specifies the width used for the ground rails during row generation. The default is 1.0.

GUI Equivalent

Command:  Place – Placement Planning
Field:     Width
Virtuoso XL Layout Editor User Guide
Environment Variables

**rowPowerLayer**

```
layoutXL.placement rowPowerLayer 'string string
```

**Description**

Specifies the layer used for power rails during row generation.

**GUI Equivalent**

- **Command:** *Place – Placement Planning*
- **Field:** *Layer*
rowPowerName

layoutXL.placement rowPowerName 'string string

Description

Specifies the net name used for power rails during row generation.

GUI Equivalent

Command:  Place – Placement Planning
Field:     Net Name
rowPowerWidth

layoutXL.placement rowPowerWidth 'float floating_point_number

Description

Specifies the width used for the power rails during row generation set in the Placement Style form. The default is 1.0.

GUI Equivalent

Command:  Place – Placement Planning
Field:     Width
rowSupplyPosition

layoutXL.placement rowSupplyPosition 'cyclic { Inside | Outside | Center }

Description

Specifies the default for the supply rail positions. The default value is Outside.

GUI Equivalent

Command: Place – Placement Planning
Field: Style (Assisted CMOS) – Rails – Position
rowSupplySpacing

layoutXL.placement rowSupplySpacing ’string string

Description

Determines the required spacing between edges of the power and ground rails in a standard cell row.

GUI Equivalent

Command: Place – Placement Planning
Field: Style (Assisted Standard Cell or Assisted Mixed CMOS/Standard Cell) – Rails – Rail Spacing
rowMOSSupplyPattern

layoutXL.placement rowMOSSupplyPattern 'cyclic { N-P-P-N | P-N-N-P | N-P | P-N }

Description

Specifies the default pattern for CMOS rows set in the Placement Planning form. The default value is N–P–P–N.

GUI Equivalent

Command: Place – Placement Planning
Field: Style (Assisted CMOS) – Rails – Rail Pattern
rowSTDAllowFlip

*layoutXL.placement rowSTDAllowFlip ’boolean { t | nil }*

**Description**

Specifies whether standard cells can be flipped around the vertical axis. The default value is `t`.

**GUI Equivalent**

None.
rowSTDSupplyPattern

layoutXL.placement rowSTDSupplyPattern 'cyclic { G-P-P-G | P-G-G-P | G-P | P-G }

Description

Specifies the default pattern for standard cell rows set in the Placement Planning form. The default value is G–P–P–G.

GUI Equivalent

Command:   Place – Placement Planning
Field:     Style (Assisted Standard cell or Assisted Mixed CMOS/Standard Cell) – Rails – Rail Pattern
rulesFile

layoutXL.placement rulesFile ’string string

Description

Specifies the name of the rules file. If you do not set this environment variable, the translator uses the rules in the technology file. If the rules are not present in the technology file a warning message appears indicating that you need to provide a rules file.

GUI Equivalent

Command: Place – Placer
Field: Rules File
**runTime**

layoutXL.placement runTime 'cyclic { quick | moderate | optimized }

**Description**

Controls the level of effort the Virtuoso custom placer uses to optimize placement. The default value is moderate.

**GUI Equivalent**

Command: Place – Placer
Field: Runtime
saveAs

`layoutXL.placement saveAs 'boolean { t | nil }

**Description**

Saves the results of the placement to be saved to a cellview that is different from the current layout. The default is nil.

**GUI Equivalent**

Command: **Place – Placer**
Field: **Save As**
saveAsCellName

layoutXL.placement saveAsCellName 'string string

Description

Specifies the cell name of the saveAs cellview. The default is the source cell name.

GUI Equivalent

Command: Place – Placer
Field: Cell
saveAsLibName

`layoutXL.placement saveAsLibName 'string string`

**Description**

Specifies the library name of the saveAs cellview. The default is the source library name.

**GUI Equivalent**

- Command: *Place – Placer*
- Field: *Library*
saveAsViewName

layoutXL.placement saveAsViewName 'string string

Description

Specifies the view name of the saveAs cellview. The default is the source view name (for example, layout.plc).

GUI Equivalent

Command: Place – Placer
Field: View
schematicWindow

layoutXL schematicWindow 'string pair_of_coordinates

Description

Determines the position of the Virtuoso Schematic window on the monitor screen when Virtuoso XL is initiated. The default is ((0 0) (0 0)).

GUI Equivalent

None.
setPPConn

layoutXL setPPConn 'boolean { t | nil}

Description

Recognizes pseudo-parallel nets and defines them as pseudo-parallel. If set to \texttt{nil}, you must explicitly define pseudo-parallel nets through the \textit{Connectivity – Define Pins – Pseudo Parallel Connect} command. The default is \texttt{t}.

GUI Equivalent

None.
sfactorNames

layoutXL sfactorNames 'string "list_of_names"

Description

Lists the names of properties used in the schematic to specify the number of series-connected (sfactor) devices to be generated in the layout.

Virtuoso XL checks each instance for a property matching one of the names on the list and uses the value of that property to generate the appropriate number of series-connected devices.

The default is “s S”.

Note: The sfactor property value can be expressed as an integer or an expression.

Arguments

list_of_names

A list of sfactor property names each separated by a space. The list must be enclosed in quotation marks; for example, “s S”

GUI Equivalent

Command: Options – Layout XL
Field: sfactor Names
**sfactorParam**

layoutXL sfactorParam 'string "list_of_parameters"

**Description**

Lists the names of the device parameters to be split among generated series-connected devices during *Design – Gen From Source, Connectivity – Check – Against Source, Connectivity – Update – Layout Parameters*.

Virtuoso XL checks each instance for the listed parameter names and changes or updates the values of the matching parameters.

The default list includes names for resistor, capacitor, and inductor parameters, “r R c C l L”.

**Arguments**

$list_of_parameters$

A list of parameter names each separated by a space. The list must be enclosed in quotation marks; for example, “r R c C l L”

**GUI Equivalent**

Command: *Options – Layout XL*

Field: *sfactor Split Param Names*
shapeSwitchLayer

layoutXL shapeSwitchLayer 'boolean { t | nil }

Description

Controls whether the entry layer switches to the layer clicked at the first shape point or remains the same as it was prior to starting the Create – Rectangle or Create – Polygon commands.

- When set to t, Virtuoso XL automatically switches to the layer clicked at the first shape point. If the first shape point is on overlapping layers, the system displays a window asking you to choose the layer on which you want to draw the shape.
- When set to nil, the entry layer remains the same as it was prior to starting the command.

The default is nil.

GUI Equivalent

Command: Create – Polygon and Create – Rectangle
Field: Change Layer
showDraglinesForDistantConns

layoutXL showDraglinesForDistantConns 'boolean { t | nil }

Description

Displays draglines for all of the connections to the instance that is being manipulated. When switched off, draglines are shown only for the connections closest to the current instance.

This environment variable is honored by the *Create – Pick From Schematic, Create – Clone, Edit – Move, and Edit – Stretch* commands.

Tip

For more information on displaying draglines, see also *colorDraglines* and *hideDraglinesForGlobalNets*.

GUI Equivalent

Command: *Options – Layout XL*

Field: *Display – Draglines – Show Distant Connections*
**showIncNetEnable**

`layoutXL showIncNetEnable 'boolean { t | nil }`

**Description**

Specifies that the *Show Incomplete Nets* command is active when Virtuoso XL is initiated. The default is nil.

**GUI Equivalent**

- **Command:** *Options – Layout XL*
- **Field:** *Show All Incomplete Nets*
**stopList**

`layoutXL stopList 'string string`

**Description**

Determines the view of a design that permits no further descent in a hierarchical design.

**GUI Equivalent**

- **Command:** *Options – Layout XL*
- **Field:** *Show Layout Instance Views*
templateFileName

layoutXL templateFileName ’string string

Description

Specifies a default template file for the Gen from Source command. If its value is the empty string ("", the default), the default filename is layout_cell_name.lxt. If it is anything other than the empty string, its value is used as the default filename.

GUI Equivalent

Command:  Design – Gen From Source
Field:  Load Template File for Layout Generation
traverseMixedHierarchies

layoutXL traverseMixedHierarchies 'boolean { t | nil }

Description

Specifies whether the design hierarchy is a Cadence data base (CDBA), netlist driven layout (NDL), or mixed CDBA and NDL hierarchy. If set to t, Virtuoso XL switches between the CDBA and NDL hierarchies. The default is nil.

GUI Equivalent

None.
**updateReplacesMasters**

```text
layoutXL updateReplacesMasters 'boolean { t | nil }
```

**Description**

Updates an existing instance that uses an incorrect master to use the correct master during the *Connectivity – Update – Device Correspondence* and *Connectivity – Update – Components and Nets* commands.

When set to **t**,

- With **updateWithMarkers** also set to **t**, the system puts a marker on the incorrect instance in the layout window and renames it `name_old`. It then creates a new instance with the correct master and places it below the design boundary.

- With **updateWithMarkers** set to **nil**, the system updates the instance to use the correct master.

When set to **nil**, the behavior differs between the two commands.

- In *Update Device Correspondence*, the binding to the incorrect master is accepted with a warning message in the log file.

- In *Update Components and Nets*, Virtuoso XL does not update the instance master in the layout.

The default is **t**.

**GUI Equivalent**

Command: *Options – Layout XL*
Field: *Update Layout Instances*
updateWithMarkers

layoutXL updateWithMarkers 'boolean { t | nil }

Description

Important

This environment variable is enabled only when updateReplacesMasters is set to t.

Puts a marker on an instance with an incorrect master and renames it name_old during the Connectivity – Update – Components and Nets command. It then creates a new instance with the correct master and places it below the design boundary.

When set to nil, Layout-XL the system updates the instance to use the correct master.

The default is t.

GUI Equivalent

None.
vcpCellBoundaryLPPs

```bash
layoutXL.placement vcpCellBoundaryLPPs 'string "(layer purpose)...)"
```

**Description**

Specifies the layer-purpose pairs (LPPs) to be used by the Virtuoso custom placer to derive the cell boundary. Use this environment variable to derive a cell boundary from shapes on an LPP that is not one of the defaults.

You can specify more than one LPP. The placer processes them in the following order.

1. The list of layer-purpose pairs from `vcpCellBoundaryLPPs`.
2. The `(prboundary drawing)` layer.
3. The `(prboundary boundary)` layer.
4. The `(instance drawing)` layer.
5. If the layers in the iccRules file have layers with functions: `metal`, `cut`, `*_diffusion` and `polysilicon`, the bounding box is computed based on those layers.

If there are no shapes on any of the LPPs listed above, the boundary is derived such that it encloses all of the objects in the cell.

**GUI Equivalent**

None.
vcpConductorDepth

layoutXL.placement vcpConductorDepth ’int integer

Description

Specifies the conductor depth for the Design Framework II (DFII) to Virtuoso custom placer translation.

The default value is 2 and is recommended for device level placement. For block and standard cell placements, Cadence recommends that you set the vcpConductorDepth to 0.

GUI Equivalent

None.
vcpGlobalPathScript

layoutXL.placement vcpGlobalPathScript 'string string

Description

Provides the placer with a script that you define for LBS that returns the network path for the current working directory. If the string is empty, then the placer will add the current path (/net/hostname/currentPath.)

GUI Equivalent

None.
vcpKeepoutDepth

layoutXL.placement vcpKeepoutDepth 'int integer

Description

Specifies the keepout depth for the DFII to Virtuoso custom placer translation.

The default value is 2 and is recommended for device level placement. For block and standard cell placements, Cadence recommends that you set the vcpKeepoutDepth to 0.

GUI Equivalent

None.
vcpVerboseLevel

layoutXL.placement vcpVerboseLevel 'int { 0 | 1 | 2 | 3 }

Description

Specifies that the verbose level for the placer is between zero and three. Zero specifies that there is no status except for the final placement statistics. Three specifies full messaging.

GUI Equivalent

None.
vcpWriteToCIW

`layoutXL vcpWriteToCIW 'boolean { t | nil }

**Description**

Specifies if the placer messages are written to the command interpreter window (CIW) or to a special window.

**GUI Equivalent**

None.
**viewList**

```
layoutXL viewList 'string list_of_views
```

**Description**

Specifies the design views used to descend through a hierarchical design to find stopping views (see *stopList* on page 595). The default is schematic, netlist, symbol, layout, compacted, symbolic.

**GUI Equivalent**

- **Command:** Options – Layout XL
- **Field:** Traverse Hierarchy View
Wire Editor

allowJogs

layout allowJogs 'boolean { t | nil }

Description

Allows orthogonal jogs in wires. If set to t, orthogonal jogs in wires that are pushed are permitted. The default is t.

GUI Equivalent

Command: Options – Routing
Field: Allow Orthogonal Jogs
allowRedundantWiring

layout allowRedundantWiring 'boolean { t | nil }

**Description**

Eliminates wire loops and extra vias on enabled nets. If set to `true`, wire loops and extra vias are allowed. The default is `t`.

**GUI Equivalent**

Command: `Options – Routing`
Field: `Allow Redundant Wiring On Enabled Nets`
allowSameNetPicking

layout allowSameNetPicking 'boolean { t | nil }

Description

Determines whether to select multiple wires on the same net when routing bus wires. The default is nil.

GUI Equivalent

Command: Options – Routing
Field: Allow Same Net Picking
**allowViaOnCurrentLayer**

`layout allowViaOnCurrentLayer 'boolean { t | nil }`

**Description**

Controls whether or not you can stay on the same layer while adding a via during wire creation. When `t`, you can add a via with or without changing the current layer during wire creation. When `nil`, vias which allow you to remain on the current layer are grayed out on the *Add Via* form, forcing an automatic change of layers.

The default is `t`.

**GUI Equivalent**

None.
alternateViews

iccTranslator.ExportEditor alternateViews "string list_of_views"

Description

When lets you specify a list of alternate views separated by spaces.

GUI Equivalent

Command: Options – Routing
Field: Alternate Views
autoAdjustLength

layout autoAdjustLength 'boolean { t | nil }

Description

Allows the length of a route to be adjusted automatically based on terminal to terminal, timing constraints, and minimum and maximum length rules. The default is nil.

GUI Equivalent

Command: Options – Routing
Field: Auto Adjust Length
autoShield

layout autoShield 'boolean { t | nil }

Description

Controls whether shields are automatically routed on nets with shielding rules. The default is nil.

GUI Equivalent

Command: Options – Routing
Field: Auto Shield
**busOverride**

layout busOverride 'cyclic
{ ignore | useOverrideValue | useOverrideValueOnNextVia }

**Description**

Allows the busWireSpacing value to be overridden. The default value is ignore.

**GUI Equivalent**

Command: Options – Routing
Field: Override Spacing For Gathering Bus Wires
busOverrideValue

layout busOverrideValue 'float floating_point_number

Description

Specifies the override minimum spacing between wires when busOverride is on. The default is 1.0.

GUI Equivalent

Command: Options – Routing
Field: Override Value
busWireSpacing

layout busWireSpacing "float floating_point_number"

Description

Specifies the override minimum spacing between wires, when routing multiple wires. The default is 0.0.

GUI Equivalent

Command: Options – Routing
Field: Spacing for Gathering Bus Wires
**busWireSpacingType**

```
layout busWireSpacingType 'cyclic { wirewire | wirevia | viavia | specify }
```

**Description**

When routing multiple wires, specifies the minimum spacing between wires. The default value is `wirewire`.

**GUI Equivalent**

Command: *Options – Routing*
Field: *Spacing for Gathering Bus Wires*
checkConflict

layout checkConflict 'boolean { t | nil }

Description

Controls whether conflicts are considered during route passes and whether they are displayed. The default is t.

GUI Equivalent

Command: Verify – Check Routing
Field: Conflict
checkCornerCorner

layout checkCornerCorner 'boolean { t | nil }

Description

Controls whether a larger corner-to-corner clearance, rather than the edge-to-edge clearance, is checked between shapes. The default is nil.

GUI Equivalent

Command: Verify – Check Routing
Field: Corner-Corner
checkCrosstalk

layout checkCrosstalk 'boolean { t | nil }

Description

Controls whether crosstalk rules are checked. The default is t.

GUI Equivalent

Command: Verify – Check Routing
Field: Crosstalk
checkLength
layout checkLength 'boolean { t | nil }

Description
Controls whether length rules are checked. The default is t.

GUI Equivalent
Command: Verify – Check Routing
Field: Length
checkLimitWay

layout checkLimitWay 'boolean { t | nil }

Description

Controls whether limit_way rules are checked. The default is nil.

GUI Equivalent

Command: Verify – Check Routing
Field: Limit Way
checkMaxProcessWireWidth

layout checkMaxProcessWireWidth 'boolean { t | nil }

Description

Controls checking for maximum process width rule violations. The default is nil.

GUI Equivalent

Command: Verify – Check Routing
Field: Maximum Process Wire Width
checkMaxStackViaDepth

layout checkMaxStackViaDepth 'boolean { t | nil }

Description

Controls checking for stack_via_depth violations. stack_via_depth limits how many vias can be stacked at one location.

GUI Equivalent

Command: Verify – Check Routing
Field: Maximum Stacked Via Depth
checkMaxTotalVia

layout checkMaxTotalVia ‘boolean { t | nil }

**Description**

Controls whether the maximum via rules for nets, classes, groups, and fromtos are checked. The default is nil.

**GUI Equivalent**

Command: Verify – Check Routing
Field: Maximum Total Via
checkMinMaskEdgeLength

layout checkMinMaskEdgeLength ‘boolean { t | nil }"

Description

Controls whether the min_mask_edge_length rule is checked. The default is nil.

GUI Equivalent

Command: Verify – Check Routing
Field: Minimum Mask Edge Link
checkMinProcessWireWidth

layout checkMinProcessWireWidth 'boolean { t | nil }

Description

Controls whether a minimum process width rules are checked. The default is t.

GUI Equivalent

Command: Verify – Check Routing
Field: Minimum Process Wire Width
checkMiter

layout checkMiter ‘boolean { t | nil }

Description

Controls whether unmitered corners are checked. The default is nil.

GUI Equivalent

Command: Verify – Check Routing
Field: Miter
checkNetOrder

layout checkNetOrder 'boolean { t | nil }

Description

Controls whether wiring is checked against net order rules. The default is nil.

GUI Equivalent

Command: Verify – Check Routing
Field: Net Order
**checkOffManGridPin**

`layout checkOffManGridPin ‘boolean { t | nil }`

**Description**

Controls whether pins that are located off the manufacturing grid are checked. The default is nil.

**GUI Equivalent**

- **Command:** Verify – Check Routing
- **Field:** Off Manufacturing Grid Pin
checkOffWireGridPin

layout checkOffWireGridPin 'boolean { t | nil }

Description

Controls whether pins that are not centered on a wire grid are checked. The default is nil.

GUI Equivalent

Command: Verify – Check Routing
Field: Off Wire Grid Pin
checkPinSpacing

layout checkPinSpacing ‘boolean { t | nil }

Description

Controls whether edge-to-edge spacing between pins is checked. The default is t.

GUI Equivalent

Command: Verify – Check Routing
Field: Pin Spacing
checkPolygonWire

`layout checkPolygonWire 'boolean { t | nil }`

**Description**

Controls whether clearance from polygon wire shapes to pins, fixed wires, protected wires, and fixed or protected vias are checked. The default is `t`.

**GUI Equivalent**

- **Command:** *Verify – Check Routing*
- **Field:** *Polygon Wire*
checkProtected

layout checkProtected 'boolean { t | nil }

Description

Controls the checking of clearance violations between protected wires or vias and other protected or fixed objects. The default is nil.

GUI Equivalent

Command:  Verify – Check Routing
Field:  Protected
checkReentrantPath

layout checkReentrantPath 'boolean { t | nil }

Description

Controls whether wire loops created by interactive routing or editing are checked. The default is t.

GUI Equivalent

Command: Verify – Check Routing
Field: Re-Entrant Path
checkRegion

layout checkRegion 'boolean { t | nil }

Description

Controls whether wire segments are halted at a region boundary until you click at the boundary. The default is t.

GUI Equivalent

Command: Options – Routing
Field: Check Region
checkSameNet

layout checkSameNet 'boolean { t | nil }

Description

Controls whether same-net notches are checked. The default is nil.

GUI Equivalent

Command: Verify – Check Routing
Field: Same Net
checkSegment

layout checkSegment 'boolean { t | nil }

**Description**

Controls whether maximum segment rules are checked. max_segment rules control the total routed length on a layer. The default is nil.

**GUI Equivalent**

Command: Verify – Check Routing
Field: Segment
checkStub

layout checkStub ‘boolean { t | nil }

Description

Controls whether maximum stub length rules are checked. The default is nil.

GUI Equivalent

Command: Verify – Check Routing
Field: Stub
checkUseLayers

layout checkUseLayers 'boolean { t | nil }

Description

Controls whether use layer rules are checked. The default is nil.

GUI Equivalent

Command: Verify – Check Routing
Field: Use Layer
checkUseVias

`layout checkUseVias 'boolean { t | nil }

**Description**

Controls whether use via rules are checked. The default is `nil`.

**GUI Equivalent**

- **Command:** *Verify – Check Routing*
- **Field:** *Use Via*
checkWireExtension

layout checkWireExtension 'boolean { t | nil }

Description

Controls whether wire_extension rules are checked. Wire extension rules control overhang at vias on routing layers. The default is nil.

GUI Equivalent

Command: Verify – Check Routing
Field: Wire Extension
doFile

layout doFile 'string string

Description

Specifies the .do file to be used when useDoFile is set to true.

GUI Equivalent

Command: Options – Routing
Field: Do File
enableBusRouting

layout enableBusRouting 'boolean { t | nil }

Description

Enables the routing of multiple wires (bundles, differential pairs, buses, and arbitrary sets). The default is t.

GUI Equivalent

Command: Options – Routing
Field: Enable Bus Routing
enableTandemPair

layout enableTandemPair 'boolean { t | nil }

Description

Enables the wire editor to route pairs on tandem layers. The default is nil.

GUI Equivalent

Command: Options – Routing
Field: Enable Tandem Pair
**fitViaPattern**

`layout fitViaPattern 'boolean { t | nil }`

**Description**

Allows the wire editor to automatically adjust the individual via positions of a via pattern when clearance conflicts occur. The default is `nil`.

**GUI Equivalent**

Command: *Options – Routing*

Field: *Fit Via Pattern*
gatherBusWires

layout gatherBusWires 'boolean { t | nil }

**Description**

Allows bus wires to be gathered to the minimum spacing value or to specified bus spacing value. The default is nil.

**GUI Equivalent**

Command: Layout pop-up menu
Field: Turn gather bus wires on/off
inaccessiblePin

layout inaccessiblePin 'boolean { t | nil }

Description

Controls whether pins that are inaccessible because of a keepout or because the pin is too close to a boundary are checked. The default is t.

GUI Equivalent

Command: Verify – Check Routing
Field: Inaccessible Pins
**interactiveChecking**

`layout interactiveChecking 'boolean { t | nil }

**Description**

Enables interactive checking, which prevents design rule violations between routing objects. The default is `t`.

**GUI Equivalents**

- **Command:** Options – Routing
  - Field: Interactive Checking
- **Command:** Layout pop-up menu
  - Field: Turn checking off/on
**matchPinWidth**

```
layout matchPinWidth 'boolean { t | nil }
```

**Description**

Sets the width of the active wire segment to the width of the selected pin. The default is nil.

**GUI Equivalent**

- **Command:** *Create – Wire*
- **Field:** *Match Pin Width*
**matchPinWidthValue**

`layout matchPinWidthValue 'cyclic { narrow | wide }`

**Description**

Sets the width of the active wire segment to the width of the narrow or wide edge of the selected pin. The default value is narrow.

**GUI Equivalent**

Command:  *Create – Wire*
Field:  *Match Pin Width*
matchTargetPinWidth

layout matchTargetPinWidth 'boolean { t | nil }

Description

Sets the width of the active wire segment to the width of the narrow or wide edge of the target pin. The default is nil.

GUI Equivalent

Command: Create – Wire
Field: Match Target Pin Width
**matchTargetPinWidthValue**

```
layout matchTargetPinWidthValue 'cyclic { narrow | wide }
```

**Description**

Sets the width of the active wire segment to the width of the narrow or wide edge of the target pin and is active only when `matchTargetPinWidth` is `t`.

**GUI Equivalent**

- **Command:**  *Create – Wire*
- **Field:**  *Match Target Pin Width*
**matchWidthSticky**

```plaintext
layout matchWidthSticky 'boolean { t | nil }
```

**Description**

Preserves the `matchPinWidth`, `matchWireWidth`, and `matchPinWidthValue` values.

The following will occur if `matchPinWidth`, `matchWireWidth`, `matchPinWidthValue`, `matchTargetPinWidth`, `matchTargetPinWidthValue` values are set, and `matchWidthSticky` is `t`,

When the wire is finished the subsequent *Create Wire* command will not reset the environment variables to the default.

The subsequent *Create Wire* session will not reset the environment variables to the default.

The environment variables values are pre-set for the first *Create Wire* session.

**GUI Equivalent**

Command: *Options – Routing*

Field: *‘Sticky’ Match Width Settings*
**matchWireWidth**

```haskell
layout matchWireWidth : boolean { t | nil }
```

**Description**

Sets the width of the active wire segment to the width of the wire segment you are editing. The default is `nil`.

**GUI Equivalent**

- **Command:** *Create – Wire*
- **Field:** *Match Wire Width*
multiplePinsConnection

layout multiplePinsConnection 'boolean { t | nil }

Description

Allows the wire editor to connect multiple pins in a straight line on a single wire. The default is t.

GUI Equivalent

Command: Options – Routing
Field: Multiple Component Connection
**pinLargerMaxProcessWidth**

layout pinLargerMaxProcessWidth 'boolean { t | nil }

**Description**

Controls checking for pins that are larger than the maximum process width rule. The default is nil.

**GUI Equivalent**

Command: *Verify – Check Routing*

Field: *Pin Larger Than Maximum Process Width*
pinSmallerMinProcessWidth

layout pinSmallerMinProcessWidth 'boolean { t | nil }

Description

Controls whether pins that are smaller than the minimum process width rule are checked. The default is t.

GUI Equivalent

Command: Verify – Check Routing
Field: Pin Smaller Than Minimum Process Width
pushComponent

layout pushComponent 'boolean { t | nil }

Description

Allows wire segments and vias to push aside components. The default is nil.

GUI Equivalent

Command: Options – Routing
Field: Push Component
pushRouting

layout pushRouting 'boolean { t | nil }

Description

Allows wire segments and vias to push aside other wires and vias while maintaining clearance rules. The default is t.

GUI Equivalent

Command: Options – Routing
Field: Push Routing
routeAsManyAsPossible

```
layout routeAsManyAsPossible 'boolean { t | nil }
```

**Description**

When set to `true`, as many wires as possible are routed and the rest are dropped. When set to `nil`, wires are routed only if all can be routed successfully. The default is `nil`.

**GUI Equivalent**

**Command:** `Layout` pop-up menu

**Field:**
- `Route As Many As Possible` and `Route Only If All Succeed`
routeToCursor

`layout routeToCursor 'boolean { t | nil }

**Description**

Allows a wire to follow the direction of the cursor to complete an optimal connection. The default is `nil`.

**GUI Equivalent**

- **Command:** *Options – Routing*
- **Field:** *Route To Cursor*
routeToCursorStyle

layout routeToCursorStyle 'cyclic
   { singleLayer | multipleLayers | followLayerDirection }

Description

When routeToCursor is set to true, controls the style of routing. singleLayer routes a wire only on the current active layer. multipleLayers changes the layers and adds vias as needed. followLayerDirection strictly follows the direction of the layer set in the rules file. The default value is singleLayer.

GUI Equivalent

Command: Options – Routing
Field: Route To Cursor
sameNetChecking

layout sameNetChecking 'boolean { t | nil }

Description

Control rule checking for gap and notch violations between objects on the same net. The default is nil.

GUI Equivalent

Command: Options – Routing
Field: Same Net Checking
showTimingMeter

layout showTimingMeter 'boolean { t | nil }

Description

Displays a meter showing the length of the wire being routed relative to the minimum or maximum length rules set for the wire. The default is nil.

GUI Equivalent

Command: Options – Routing
Field: Show Timing/Length Rule Constraints – Meter
showTimingOctagon

layout showTimingOctagon 'boolean { t | nil }

Description

Shows an octagon display with the length of the wire being routed relative to the minimum or maximum length rules set for the wire. The default is nil.

GUI Equivalent

Command:   Options – Routing
Field:     Show Timing/Length Rule Constraints – Octagons
**snapToPinOrigin**

`layout snapToPinOrigin 'boolean { t | nil }

**Description**

Allows a wire to snap to the origin of pins when you click anywhere inside the pin. The default is nil.

**GUI Equivalent**

Command:  *Options – Routing*
Field:  *Snap to Pin Origin*
**useDisplayGrid**

`layout useDisplayGrid `boolean { t | nil }

**Description**

Toggles between the manufacturing grid or display grid. The display grid is defined under *Options – Display* and the manufacturing grid is defined in the technology file. If `useDisplayGrid` is `t`, then display grid is used, if `useDisplayGrid` is `nil` then manufacturing grid is used. The default value is `nil`.

**GUI Equivalent**

Command:  *Layout* pop-up menu
Field:  *Use manufacturing grid or Use display grid*
useDoFile

layout useDoFile 'boolean { t | nil }

Description

If set to true, a .do file, specifying constraints and entered into the Route Options form’s General tab is used. The default is nil.

GUI Equivalent

Command: Options – Routing
Field: Do File
useRulesFile

`iccTranslator useRulesFile 'boolean { t | nil }`

**Description**

If set to `true`, a rules file is entered into the Route Options form (General tab) and it takes precedence over the rules specified in the technology file. The default is `nil`.

**GUI Equivalent**

Command: `Options – Routing`

Field: `Rules`
**viaAssistance**

`layout viaAssistance 'cyclic { snap | display | none }`

**Description**

Provides assistance with locating legal via sites. The default value is `snap`.

**GUI Equivalent**

Command: *Options – Routing*

Field: *Via Assistance*
**viaPattern**

```plaintext
layout viaPattern 'cyclic
  { perp | diag_1 | diag_2 | stagger | out_taper | in_taper }
```

**Description**

Specifies the via pattern to be used when placing vias on multiple wires. The default value is `diag_1`.

**GUI Equivalent**

- **Command:** *Layout*
- **Field:** *Via Pattern*
# Command Quick Reference

Listed below are all the commands you can use in the Virtuoso® XL Layout Editor (Virtuoso XL) with a brief summary of what each command does and a link to more complete documentation of the command.

<table>
<thead>
<tr>
<th>Menu Name</th>
<th>Command Name</th>
<th>Command Function</th>
<th>Documented</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIW: File – Import</td>
<td>XL Netlist</td>
<td>Imports a CDL netlist to use as a connectivity source for Virtuoso XL</td>
<td>Chapter 6, “Importing a Netlist for a Connectivity Reference”</td>
</tr>
<tr>
<td>Design</td>
<td>Gen From Source</td>
<td>Creates and places layout cells below the boundary in the layout window</td>
<td>Chapter 6, “Generating a Layout with Components Not Placed (Gen From Source)”</td>
</tr>
<tr>
<td>Design</td>
<td>Template – Load From File</td>
<td>Loads pin, boundary, and placement information you saved to a file to generate a layout</td>
<td>Chapter 6, “Loading Template Files”</td>
</tr>
<tr>
<td>Design</td>
<td>Template – Save To File</td>
<td>Saves pin, boundary, and placement information to generate layout to a file for later use</td>
<td>Chapter 6, “Creating Template Files”</td>
</tr>
<tr>
<td>Create</td>
<td>Multipart Path</td>
<td>Creates relative object design (ROD) multi-part paths</td>
<td>Virtuoso Layout Editor User Guide, “Creating and Editing Multipart Paths”</td>
</tr>
<tr>
<td>Menu Name</td>
<td>Command Name</td>
<td>Command Function</td>
<td>Documented</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Create</td>
<td>Pick From Schematic</td>
<td>Creates and places interactively selected schematic elements in the layout</td>
<td>Chapter 6, “Moving Components from the Schematic into the Layout (Pick from Schematic)”</td>
</tr>
<tr>
<td>Create</td>
<td>Clone</td>
<td>Copies parts of schematic or layout including connectivity info to use as other parts</td>
<td>Chapter 6, “Cloning”</td>
</tr>
<tr>
<td>Edit</td>
<td>Other – Swap Components</td>
<td>Exchanges the position of two components</td>
<td>Chapter 7, “Swapping Components”</td>
</tr>
<tr>
<td>Edit</td>
<td>Other – Lock Selected</td>
<td>Creates temporary fixed constraints on layout objects</td>
<td>Chapter 7, “Using Device Locking”</td>
</tr>
<tr>
<td>Edit</td>
<td>Other – Unlock Selected</td>
<td>Removes temporary fixed constraints from layout objects</td>
<td>Chapter 7, “Using Device Locking”</td>
</tr>
<tr>
<td>Edit</td>
<td>Component Types</td>
<td>Controls how components are placed in rows, and defines their chaining and folding parameters</td>
<td>Chapter 7, “Component Types”</td>
</tr>
<tr>
<td>Edit</td>
<td>Transistor Chaining</td>
<td>Abuts prepared MOS transistors into chains</td>
<td>Chapter 5, “Device Abutment”</td>
</tr>
<tr>
<td>Edit</td>
<td>Transistor Folding</td>
<td>Splits prepared MOS transistors or chains into fingers (smaller devices connected in parallel).</td>
<td>Chapter 7, “Using Transistor Folding”</td>
</tr>
<tr>
<td>Edit</td>
<td>Place From Schematic</td>
<td>Places components from the schematic inside the layout boundary in the same relative position as in the schematic</td>
<td>Chapter 6, “Placing Components in a Layout in the Same Relative Position as in the Schematic (Place As In Schematic)”</td>
</tr>
<tr>
<td>Menu Name</td>
<td>Command Name</td>
<td>Command Function</td>
<td>Documented</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Assign Nets</td>
<td>Assigns instance pins with or without a connectivity source or schematic to nets in the layout</td>
<td>Chapter 7, “Assigning Pins to a Net”</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Show Incomplete Nets</td>
<td>Shows flight lines connecting nets that have not yet been routed (wired).</td>
<td>Chapter 7, “Identifying Incomplete Nets”</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Hide Incomplete Nets</td>
<td>Removes flight lines connecting nets that have not yet been routed (wired).</td>
<td>Chapter 7, “Identifying Incomplete Nets”</td>
</tr>
<tr>
<td>Connectivity</td>
<td>XL Probe</td>
<td>Highlights the design element in the layout or schematic when you click on the corresponding one in the other window</td>
<td>Chapter 11, “Finding Design Elements”</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Permute Pins</td>
<td>Switches the position of pins set up for permutation within an instance.</td>
<td>Chapter 7, “Permuting Component Pins”</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Check – Shorts and Opens</td>
<td>Reports the number of incomplete nets, shorts, and illegal overlaps currently in the layout window.</td>
<td>Chapter 11, “Checking Shorts and Opens”</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Check – Against Source</td>
<td>Reports the devices and parameters in the connectivity source that are missing in the schematic</td>
<td>Chapter 11, “Comparing Design Elements (Check Against Source)”</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Update – Components and Nets</td>
<td>Updates the layout to add devices, pins, or connectivity changes added to the schematic</td>
<td>Chapter 12, “Updating Components and Nets (Engineering Change Order Mode)”</td>
</tr>
<tr>
<td>Menu Name</td>
<td>Command Name</td>
<td>Command Function</td>
<td>Documented</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Update – Layout Parameters</td>
<td>Updates parameters of instances in the layout to match those in the schematic</td>
<td>Chapter 12, “Updating Layout Parameters”</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Update – Schematic Parameters</td>
<td>Updates parameters of devices in the schematic to match those in the layout</td>
<td>Chapter 12, “Updating Schematic Parameters”</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Update – Device Correspondence</td>
<td>Applies connectivity from schematic symbol(s) to layout device(s); allows implementation of one or many to many or one</td>
<td>Chapter 12, “Updating Device Correspondence”</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Update – Source</td>
<td>Allows the connectivity source for update to a layout to be a schematic, a netlist, or no source</td>
<td>Chapter 12, “Updating the Connectivity Reference”</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Change Instance View</td>
<td>Changes the layout view of a device or devices in a hierarchical design</td>
<td>Chapter 12, “Changing the Device (Instance) View”</td>
</tr>
<tr>
<td>Options</td>
<td>Layout XL</td>
<td>Sets values of Virtuoso XL environment variables</td>
<td>Chapter 4, “Setting Environment Variables”</td>
</tr>
</tbody>
</table>
Using Spice and CDL for Netlist-Driven Layout Generation

Introduction

The VXL Spice reader reads two styles of Spice, those being generic SPICE3 (Hspice-like) and CDL. Within these styles, only those constructs needed to support the netlist input requirements of VXL are read.

Specifying Spice Designs

A Spice design is specified to VXL via a single Spice netlist file. The cell defined at the file level (i.e., outside of any .SUBCKT block) is designated as the top level cell and is assigned the name *TOP-CELL-(last part of file name). For example, for the Spice file /mycds/mydir/des_a.sp, the top level cell is assigned the name *TOP-CELL-des_a. All other cells specified in the design must be defined using the .SUBCKT or .MODEL statements. Some additional cells may be created via an automatic mechanism as needed by the Spice reader.

The Spice netlist file can include other Spice netlists by using the .INCLUDE statement. All cells defined within included files must be defined using the .SUBCKT or .MODEL statements. Further, all cell names in the design must be unique. The spice style (SPICE3 or CDL) must be specified to VXL.

Cell Creation Rules

Cells associated with a SUBCKT instance must be created using a .SUBCKT statement.

Cells associated with a mosfet, jfet, mesfet, BJT (bipolar), or diode instance can be created one of 3 ways

1. By a .SUBCKT statement in which case the specified ports are used.
2. By a .MODEL statement.
a. Type nmos or pmos for a mosfet, in which case the ports D,G,S,B are used

b. Type njf or pjf for a jfet, in which case the ports D,G,S are used

c. Type nmf or pmf for a mesfet, in which case the ports D,G,S are used

d. Type npn or pnp for a BJT device, in which case the ports C,B,E are used

e. Type d for a diode, in which case the ports PLUS,MINUS are used

3. Automatically, by being an unresolved reference of a mosfet, jfet, mesfet, BJT, or diode instance in which case the ports D,G,S,B; D,G,S; D,G,S; C,B,D; or PLUS,MINUS respectively are used.

Cells associated with a capacitor or resistor can be created in one of four ways

1. By a .SUBCKT statement in which case the specified ports are used.

2. By a .MODEL statement.

   a. Type c for a capacitor, in which case the ports PLUS,MINUS are used

   b. Type r for a resistor, in which case the ports PLUS,MINUS are used

3. Automatically, by being an unresolved reference of a capacitor or resistor instance, in which case the ports PLUS,MINUS are used.

4. Automatically, for cases where no model is specified, in which case the cell “-RES-” is created for a resistor and “-CAP-” is created for a capacitor. In both cases, the ports PLUS,MINUS are used.

Inductor references do not allow specification of a model. However, a default cell -IND- is created for all inductors. The ports PLUS,MINUS are used.

Voltage Source references do not allow specification of a model. However, a default cell named -VOLT- is created for all voltage sources. The ports PLUS, MINUS are used.

**Note:** The implications of the above are that references to mosfet, jfet, mesfet, BJT, capacitor, resistor, and diode models may be assumed but references to subckt cells are never assumed.
Character Considerations

Case

In general case sensitivity is recognized, however the following rules and exceptions will apply

1. All keywords are treated as case insensitive.
2. The *Scale parameter on the .OPTION line is treated as case insensitive.
3. On fet references, if implied width and length values are specified, uppercase W and L are the assigned parameters.

The * Character

Spice

The * at the beginning of a column denotes the beginning of a comment which ends at the end of line. The * PORT statement is treated as a valid statement to determine port direction.

CDL

The * at the beginning of a column denotes the beginning of a comment which ends at the end of line. If this occurs before a command, then the command should be read by Dracula but not by the circuit simulator. For parsing CDL, the keywords .GLOBAL, .INCLUDE, .OPTION, and .PARAM are processed even if a * precedes them.

The $ Character

Spice

Under most circumstances the $ denotes the beginning of a comment which ends at the end of line. The exceptions are when the $ is part (but not the start) of a name or when it is part of a quoted string. In these cases $ is treated as any other character.
CDL

The $ before a non-command field denotes that the following field should be read by Dracula but not circuit simulator. The $ before a command indicates that the line should be treated as a comment. For parsing CDL, any $ before a name field is ignored.

Characters in Names

Spice

A name may start with an alpha character or the following 3 characters % _ / . The allowable characters following the first character are alphanumeric, or the following seventeen characters _ / + - [] < > ; : # $! *% / . However, the following three characters are stripped from the name < />.

CDL

A name may start with an alpha character or the following four characters % _ / $. The allowable characters following the first character are alphanumeric, or the following twenty one characters _ / + - [] < > ; : # $! *% / , ( ) `. However, the following fourteen characters are stripped from the name [ ] < > ; # $! / , ( ) `.

Pin, Terminal, and Net Names

Pins, terminals, and nets are given additional naming latitude. In addition to the names allowed by the general naming requirements under the Spice and CDL sections, pin, terminal, and net names are allowed to be a non-negative integer or a pair of non-negative integers separated by a colon. Examples of such allowable names are 0, 17, and 4:2. Further, net 0 is always treated as a global ground net.

Spice Statements

Conventions for Format Descriptions

{ } surround optional elements in a description of a syntax
/ denotes OR
... denotes a list
**numval** represents a value which must be numeric (4, -2.2, 3.75U, -3.56e-7).

**val** represents a value which may be numeric, a (variable) parameter name or a literal string enclosed in single quotes (4, -2.2, 3.75U, -3.56e-7, ppp, ‘any lit str’)

## File Level Statements

### First Line of File

The first line of a Spice or CDL file must be a comment.

**Format** (Spice or CDL) : *|$ {text}

* or $ is the comment character

**text** is an optional text string

**Ex** : * this is a Spice or CDL file

### .GLOBAL

The **Global** statement is used for specifying global nets.

**Format** (Spice or CDL) : .GLOBAL name{:P|G} ...

name is the net name

P or G are optional and represents Power or Ground

**Ex** : .GLOBAL vdd:P vss:g GNETA

**Note**: CDL allows a * before the . of the **GLOBAL** statement.

### .INCLUDE

The **Include** statement is used for including additional spice files from within a spice file.

The file included is treated similar to as if it were specified directly via command. There is no scoping done. However, all cells in an included file must be defined by .subckt or .model statements. If the specified file name supplied is not fully qualified, the run directory is used as the base directory for the file.

**Format** (Spice or CDL) : .INCLUDE ‘filename’

filename is the file to be included

**Ex** : .INCLUDE ‘/cdsabc/mydir/sfile’
Note: CDL will allow a * before the . of the .INCLUDE statement.

.INC

The .inc keyword is an alias for the .include keyword. See the .include keyword for details.

.OPTION

The .option statement is used for specifying option parameters.

This statement is reserved for specific Spice options, all but one of which are not used by VXL Spice processing. These options are read and stored. The one option processed for Spice handling is the scale. The scale factor acts as a multiplier for values such as length, width, area, etc. See the Parameter Resolution section for more details on the use of scale.

Format (Spice or CDL) : .OPTION param{=val} ...

param{=val} is the parameter set to an optional, in the parameter list

Ex : .OPTION scale=1.1

Note: CDL will allow a * before the . of the .OPTION statement.

.OPTIONS

The .options keyword is an alias for the .option keyword. See the .option keyword for details.

.MODEL

The .model statement is used for defining various types of model cells.

Format (Spice or CDL) : .MODEL name type {param=val}...

name is the model name

type is the model type

param=val is an optional parameter set to a value, in the parameter list

The allowable model types are:

- nmos and pmos for mosfet models, which are created using the ports D,G,S,B
- njf and pjf for jfet models, which are created using ports D,G,S
- *nmf* and *pmf* for MESFET models, which are created using ports *D,G,S*
- *nnp* and *pnp* for BJT (bipolar) models, which are created using the ports *C,B,E*
- *d* for diode models, which are created using the ports *PLUS,MINUS*
- *c* for capacitor models, which are created using the ports *PLUS,MINUS*
- *r* for resistor models, which are created using the ports *PLUS,MINUS*

Ex: .MODEL TN nmos  l = 1U w = 4U

**.SUBCKT**

The `.subckt` statement is used for defining subckt cells.

**Format** (Spice): `.SUBCKT` name pin … {param=val}…

  - `name` is the cell name
  - `pin` is a pin in the pin list
  - `param=val` is an optional parameter set to a value, in the parameter list

Ex: .SUBCKT AND O A B L = 3U AS = 1.5P

**Format** (CDL): `.SUBCKT` name {opin} … / {ipin} … {param=val}…

  - `name` is the cell name
  - `opin` is a pin in the output pin list
  - `/` is the separator between the output pin list and the input pin list
  - `ipin` is a pin in the input pin list
  - `param=val` is an optional parameter set to a value, in the parameter list

Ex: .SUBCKT AND O / A B L = 3U AS = 1.5P

**Note:** For the CDL format, there must be at least one pin in the opin or the ipin list. For Spice, port types are specified using the `*PORT` statement.

**.MACRO**

The `.macro` keyword is an alias for the `.subckt` keyword. See the `.subckt` keyword for details.

**.ENDS**

The `.ends` statement is used for ending a `.subcktcell` definition.
Format (Spice or CDL) : .ENDS

.EOM

The .eom keyword is an alias for the .ends keyword. See the .ends keyword for details.

Statements Allowed at File Level or within a Subckt Cell or a Top Level Cell

.PARAM

The .param statement is used for specifying lists of parameters.

Format (Spice or CDL) : .PARAM param=val …

  param=val is a parameter set to a value, in the parameter list

Ex : .PARAM w=5U p = wid str= ‘abc’

Note: CDL will allow a * before the . of the .PARAM statement.

Statements Allowed within a Subckt Cell or a Top Level Cell

* PORT

The * PORT statement is used for specifying pins as input or output pins.

Note: The *PORT statement is restricted to Spice only.

Format (Spice) : * PORT type pin …

  type is the pin type and can take the values input, output, bidirect
  tristate, power, ground, supply, open_drain, or unknown
  pin is a pin in the pin list

Ex : * PORT input i1 i2

Note: For CDL, port types are specified on the .SUBCKT.

Subckt Reference

Format (Spice) : Xabc term … master {param=val}…

  Xabc is the instance name
  term is a terminal in the terminal list
  master is the cell type (aka : master cell)
param=val is a parameter set to a value, in the parameter list

Ex: xanda a b o AND w=10U qqq=20

**Format** (CDL): Xabc term ... {/}master {param=val}...

- Xabc is the instance name
- term is a terminal in the terminal list
- / is the optional separator between the term list and the cell type
- master is the cell type (aka: master cell)
- param=val is a parameter set to a value, in the parameter list

Ex: xanda a b o / AND w=10U qqq=20

**Note:** Subckt references must always begin with x.

**Mosfet Reference**

**Format** (Spice or CDL): Mabc drain gate src bulk master {val1 {val2}}{param=val}...

- Mabc is the instance name
- drain gate src bulk is the order of the terminals
- master is the cell type (ie: model name) (aka: master cell)
- val1 is the value of the W parameter
- val2 is the value of the L parameter
- param=val is a parameter set to a value, in the parameter list

Ex 1: m1 vdd a q vdd TP W = 9e-7 L=1U as = 2e-12
Ex 2: m1 vdd a q vdd TP 9e-7 1U as = 2e-12
Ex 3: m1 vdd a q vdd TP wpa L=1pa

**Note:** Mosfet references must always begin with m.

**Jfet Reference**

**Format** (Spice or CDL): Jabc drain gate src master {area} {OFF} {param=val}...

- Jabc is the instance name
- drain gate src is the order of the terminals
- master is the cell type (ie: model name) (aka: master cell)
- area is the optional area factor, it will be assigned to the property
  “AREA”
- OFF is the optional initial condition flag for dc analysis
param=val is a parameter set to a value, in the parameter list

Ex :  J1 vdd a q 9e-7 OFF pp = 5

Note:  JFET references must always begin with j.

Mesfet Reference

Format  (Spice or CDL) :  Zabc drain gate src master (area) {OFF} {param=val}...

  Zabc  is the instance name
  drain gate src  is the order of the terminals
  master  is the cell type (ie : model name) (aka : master cell)
  area is the optional area factor, it will be assigned to the property “AREA”
  OFF is the optional initial condition flag for dc analysis
  param=val is a parameter set to a value, in the parameter list

Ex :  Z1 vdd a q zp 5e-7 OFF pa = 2.2

Note:  Mesfet references must always begin with z.

BJT (Bipolar) Reference

Format  (Spice or CDL) :  Qabc collector base emitter {substrate} master (area) {OFF} {param=val}...

  Qabc  is the instance name
  collector base emitter substrate is the order of the terminals with substrate being optional
  master is the cell type (ie : model name) (aka : master cell)
  area is the optional area factor, it will be assigned to the property "AREA"
  OFF is the optional initial condition flag for dc analysis
  param=val is a parameter set to a value, in the parameter list

Ex 1 :  Q1 vdd a q nv 9e-7 OFF pp = 5
Ex 2 :  Qa2 n1 n2 n3 n4 pl

Note:  BJT references must always begin with q.

Note:  The substrate node must be specified. If it is not, a syntax error will occur.
Diode Reference

**Format** (Spice or CDL) : Dabc term1 term2 master {area} {OFF} {param=val}...

Dabc is the instance name
drain gate src is the order of the terminals
master is the cell type (ie : model name) (aka : master cell)
area is the optional area factor, it will be assigned to the property
“AREA”
OFF is the optional initial condition flag for dc analysis
param=val is a parameter set to a value, in the parameter list

Ex : D1 vdd a 9e-7 OFF pp = 5

**Note**: Diode references must always begin with d.

Capacitor Reference

**Format** (Spice or CDL) : Cabc term1 term2 {numval}{model} {param=val}...

Cabc is the capacitor name
term1 term2 are the terminals
numval is the optional numerical capacitance value
model is the optional model name (aka : master cell)
param=val is a parameter set to a value, in the parameter list

Ex 1 : c0 gnd q 1e-14
Ex 2 : c1 gnd q capmod

**Note**: Either numval or model must be specified. It is allowable to specify both.

**Note**: Capacitor references must always begin with c.

Resistor Reference

**Format** (Spice or CDL) : Rabc term1 term2 {numval}{model} {param=val}...

Rabc is the resistor name
term1 term2 are the terminals
numval is the optional numerical resistance value
model is the optional model name (aka : master cell)
param=val is a parameter set to a value, in the parameter list

Ex 1 : r0 gnd q 1e-6
Ex 2 :  r1 gnd q resmod

**Note:** Either *numval* or *model* must be specified. It is allowable to specify both.

**Note:** Resistor references must always begin with *r*.

**Inductor Reference**

**Format** (Spice or CDL) :  Labc term1 term2 numval {param=val}...

- *Labc* is the inductor name
- *term1* term2 are the terminals
- *numval* is the numerical inductor value
- *param=val* is a parameter set to a value, in the parameter list

Ex 1 :  l1 gnd q 1e-6

**Note:** Inductor references must always begin with *l*.

**Voltage Source Reference**

**Format** (Spice or CDL) :  Vabc term1 term2 <<DC> numval> <AC ...>

- *Vabc* is the voltage source name
- *term1* term2 are the terminals
- *numval* is the optional voltage source value, which may be preceded by the optional DC flag.
- *AC ...* is the optional AC construct, which will be ignored

Ex 1 : v1 t1 t2   (for this example *numval* is given the value 0)
Ex 2 : V2 tt ss 1.0 AC p1 p2
Ex 3 : V3 pl mn DC 2.2

**Note:** Voltage source references must always begin with *v*.

**Spice Design Example**

**File: datafan.sp**

* spice design - datafan

```
.global vdd:p gnd:g
.option scale=1.1
```
.param ln=3u lp=4u wn=5u wp=6u

.include '/net/mycds/spice_files/latch.sp'
.include '/net/mycds/spice_files/models.sp'

.subckt MIV Q A
* port input A
* port output Q
m1 vdd A Q vdd TP W=7e-06 L=8e-06
m2 Q A gnd gnd TN 7e-06 8e-06
.ends

.subckt MND2 Q A B
* port input A
* port input B
* port output Q
m1 vdd A Q vdd TP W=wp L=lp
m2 vdd B Q vdd TP wn ln
m3 Q A temp gnd TN W=wpar I=1par
m4 temp B gnd gnd TN wpar lpar
.ends

.subckt MNR2 Q A B
.param ln=7u lp=8u
* port input A B
* port output Q
m1 vdd A temp vdd TP W=wp L=lp
m2 temp B Q vdd TP wn ln
m3 Q A gnd gnd TN W=9u L=9u
m4 Q B gnd gnd TN 9u 9u
.ends
* top level

* port input I CLK E
* port output L1 L1N
* port output L2 L2N
* port output L3 L3N

Xinv1 D1 I MIV
Xnand1 Q1 D1 E MND2
Xinv2 Q2 Q1 MIV
Xinv3 DATA Q2 MIV
Xlatch1 L1 L1N DATA E CLK LATCH ln1=7.5u lp1=lp + wn1=wn wp1=wp wn2=8u wp2=wp
Xlatch2 L2 L2N DATA E CLK LATCH
Xlatch3 L3 L3N DATA E CLK LATCH

File: latch.sp

* latch

.GLOBAL vdd:P gnd:G
.OPTION scale=1.2
.PARAM ln=3.5U lp=4.5U wn=5.5U wp=6.5U

.SUBCKT LATCH Q QN D E CLK ln1=2.5U lp1=lp wn1=3.5U wp1=wp
.PARAM ln2=1.5U lp2=lp wn2=wn wp2=wp
* PORT input CLK
* PORT input D
* PORT input E
* PORT output Q
* PORT output QN
c1 gnd vdd 1.81369e-14

m9 Q QN gnd gnd TN W=1.5U L=2.5U

m17 vdd QN Q vdd TP W=1.5U L=2.5U

m18 n9 Q vdd vdd TP W=wp2 L=lp2

m21 vdd D n4 vdd TP wp1 lp1

m22 vdd E n7 vdd TP W L

m23 vdd CLK n7 vdd TP

m24 n8 n7 vdd vdd TP

.EENDS
File: models.sp

$ models
.param lpar=1U wpar= 2U
.model TN nmos L=lpar W=wpar
.model TP pmos L=lpar W=wpar

CDL Design Example

File: datafan.cdl

* cd1 design - datafan

*.global vdd:p gnd:g
*.option scale=1.1
*.param ln=3u lp=4u wn=5u wp=6u

*.include '/net/mycds/cdl_files/latch.cdl'
.include '/net/mycds/cdl_files/models.cdl'

.subckt MIV Q / A
m1 vdd A Q vdd TP W=7e-06 L=8e-06
m2 Q A gnd gnd TN 7e-06 8e-06
.ends

.subckt MND2 Q / A B
m1 vdd A Q vdd TP W=wp L=lp
m2 vdd B Q vdd TP wn ln
m3 Q A temp gnd TN W=wpar L=lpar
m4 temp B gnd gnd TN wpar lpar
.ends

.subckt MNR2 Q / A B
.param ln=7u lp=8u
m1 vdd A temp vdd TP W=wp L=lp
m2 temp B Q vdd TP wn ln
m3 Q A gnd gnd TN W=9u L=9u
m4 Q B gnd gnd TN 9u 9u
.ends

* top level

Xinv1 D1 I / MIV
Xnand1 Q1 D1 E / MND2
Xinv2 Q2 Q1 / MIV
Xinv3 DATA Q2 / MIV
XLatch1 L1 L1N DATA E CLK / LATCH ln1=7.5u lp1=lp + wn1=wn wp1=wp wn2=8u wp2=wp
XLatch2 L2 L2N DATA E CLK / LATCH
XLatch3 L3 L3N DATA E CLK / LATCH

File: latch.cdl

* latch
.GLOBAL vdd:P gnd:G
.OPTION scale=1.2
.PARAM ln=3.5U lp=4.5U wn=5.5U wp=6.5U

.SUBCKT LATCH Q QN / D E CLK ln1=2.5u lp1=lp wn1=3.5u wp1=wp
*.PARAM ln2=1.5U lp2=lp wn2=wn wp2=wp
c1 gnd vdd 1.81369e-14

c2 gnd Q 1.37038e-14

c3 gnd QN 1.25655e-14

c4 gnd n8 1.2719e-14

c5 gnd n7 1.47611e-14

c6 gnd D 1.14232e-14

c7 gnd E 1.07987e-14

c8 gnd CLK 8.80052e-15

m9 Q QN gnd gnd TN W=1.5U L=2.5U

m10 gnd Q n10 gnd TN W=wn2 L=ln2

m11 n10 n7 QN gnd TN wn2 ln2

m12 n5 n8 QN gnd TN W=wn1 L=ln1

m13 gnd D n5 gnd TN wn1 ln1

m14 n1 CLK n7 gnd TN W=W L=L

m15 gnd E n1 gnd TN

m16 gnd n7 n8 gnd TN

m17 vdd QN Q vdd TP W=1.5U L=2.5U

m18 n9 Q vdd vdd TP W=wp2 L=lp2

m19 QN n8 n9 vdd TP wp2 lp2

m20 n4 n7 QN vdd TP W=wp1 L=lp1

m21 vdd D n4 vdd TP wp1 lp1

m22 vdd E n7 vdd TP W L

m23 vdd CLK n7 vdd TP

m24 n8 n7 vdd vdd TP

.ENDS
Parameter Resolution

Spice and CDL allow the specification of parameters within the netlist. The VXL spice reader has the capability, within some limitations, of reading and resolving these parameters. This section will describe and illustrate, using examples, the VXL spice reader's parameter handling capabilities and limitations.

Specifying Parameters

Parameters can be specified in Spice and CDL netlists on the .PARAM, .OPTION, .SUBCKT, .MODEL, Subckt reference, Fet reference, Capacitor reference, and Resistor reference statements. As described in the Spice Statements section, the possible formats for parameter specification are:

param , param = numval , param = val

where;

- numval represents a value which must be numeric (4, -2.2, 3.75U, -3.56e-7)
- val represents a value which may be numeric, a (parameter) name or a literal enclosed in single quotes (4, -2.2, 3.75U, -3.56e-7, ppp, ‘any lit str’)

Not all formats are allowed on every statement in which parameters can be specified. Refer to the individual statement formats to see which formats for parameter specification are allowed.

Caution

Restriction: The VXL Spice Reader does not allow a parameter to be set to an arithmetic expression. Ex: W = 2 * WPAR, is not allowed.
Parameter Levels

Every parameter specified in a netlist can be associated with a particular level of the netlist. The level of a parameter is determined by where it is defined within the netlist. There are three possible levels, those being, from highest to lowest, file, cell, and instance.

Parameters defined on the .OPTION statement are file level parameters. Parameters defined on the .PARAM statement outside of subckts are also file level. Parameters defined on the .PARAM statement within a subckt are cell level parameters. Parameters defined on the .SUBCKT or the .MODEL statement are also cell level. Parameters defined on Subckt reference, Fet reference, Capacitor reference, or Resistor reference statements are instance level parameters.

Parameter levels are an important consideration for parameter resolution.

Resolving Parameters

The VXL Spice Reader contains a resolver that resolves parameters to values. Resolution is based on a set of rules that take into account the value type of the parameter, the level of the parameter, and possibly the scale factor. Scaling is discussed in the Scaling Parameters section.

Literal string parameters are fully resolved as specified. Numerical parameters are also resolved as specified, except that they may need to be scaled by a scale factor. Variable parameters are resolved by various rules that are described below.

Rule 1: The first basic rule for variable parameter resolution is to look, at the next parameter level up, for the parameter of the variable name specified by the parameter being resolved.

For example, when attempting to resolve the parameter $w$ which was specified as $w = \text{wpar}$, the resolver will look for the parameter $\text{wpar}$ at the next parameter level up. So, for example, if $\text{wpar} = 6\text{U}$ at the next higher level, then $w$ will resolve to the value $6\text{U}$.

Restriction: The implication of this rule is that the resolver will not look at the same level to resolve a parameter.

For example, if in the specification of a particular instance, the parameters $w = \text{wpar}$ and $\text{wpar} = 4\text{U}$ are specified, the resolver will not resolve $w$ to the value $4\text{U}$.

File level variable parameters are considered resolved to their variable (string) value, since there is nowhere else to look to try to resolve them. For example, at the file level, the parameter $w = \text{wpar}$, $w$ will resolve to the string $\text{wpar}$. 
Cell level variable parameters can be resolved only by using the file level parameters of the file in which the cell resides. If the variable parameter value cannot be found as a parameter at the file level, then the cell level parameter is considered resolved. For example, at the cell level, are the parameters $W_1 = wp_1$, $W_2 = wp_2$, and $W_3 = wp_3$ and, at the file level, are the parameters $wp_1 = 4U$ and $wp_2 = wstr$. $W_1$ will resolve to $4U$, $W_2$ will resolve to the string $wstr$, and $W_3$ will resolve to the string $wp_3$.

The situation for instance level variable parameters is more complex than that of parameters at the file or cell level. This is because Spice and CDL allow parameters to be passed into an instance from the instance above it in the instance tree. Further, instance level parameters can be inherited from the master cell of the instance.

If a parameter exists on the master cell of an instance, but that parameter does exist on the instance itself, then the instance inherits that parameter from the master cell. For example, an instance does not contain the parameter $w$ but the master cell for that instances specifies, $w = 4U$, then the instance inherits the $w$ parameter and the value $4U$ is assigned to it.

Parameter passing plays a large role in variable parameter resolution for instances. The resolver follows certain rules for parameter passing.

Rule 2: Only parameters specified on the cell definition line (.SUBCKT or .MODEL line) of the cell in which an instance resides, can be passed into an instance.

For example, for instance $X_{i1}$ of the cell named $excel$ defined below,

```
.SUBCKT excel I O wp1 = 5U
.PARAM wp2 = 6U
X_{i1} I A inv W1 = wp1 W2 = wp2
```

A value for $wp_1$ may be passed into $X_{i1}$ but a value for $wp_2$ cannot. This is because $wp_1$ is specified on the .SUBCKT line but $wp_2$ is not.

Rule 3: The order of priority for variable parameter searching during resolution is

1. Parameters passed via instance.
2. Cell level parameters.
3. File level parameters.
4. Parameters from the master of the instance.
Sample Rules

The following Spice netlist example is used to illustrate various cases of parameter resolution. Results for all possible instance tree combinations down to the fet level can be found in the Complete ibuf Example Results section.

File: IBUF.SP

.GLOBAL vdd gnd
.option scale=1.0
.param PPP=600U W=400U L=400.5U P1=401U P2=402U P3=403U P4=404U P5=405U
.include '/net/mycds/spice_files/IBUF_I.SP'
.subckt IBUFTWO I O W=200U L=200.5U P1=201U P2=202U P4=204U
.param P3=203U PP3=213U
*port input I
*port output O
Xinv1 I A MIV W=100U L=L P1=101U P2=102U P4=104U
Xinv2 A B MIV W=P1 WW=105U
Xinv3 B C MIV P5=PPP
Xinv4 C O MIV P1=W P4=WW
.ends

* top level
*port input II
*port output OO
Xibuf1 II AA IBUFTWO W=300U P1=P3
Xibuf2 AA BB IBUFTWO W=P1 L=P4
Xibuf3 BB OO IBUFTWO L=PP3

File: IBUF_I.SP

*
.GLOBAL vdd:P gnd:G

.option scale = 1.0

.param P4=20U P5=25U PP1=21U PP2=22U PP3=23U

.model TN nmos level=2 L=1.0U W=2.0U WW=3U PP4=34U

.model TP pmos W=2.5U L=1.5U level=2 WW=4U

.subckt MIV A Q L=4.0U W=5.0U P1=6.0U P2=7.0U P3=8U P4=9U P5=PP1

.param WW=30U LL=31U PP3=32U

* port input A
* port output Q

m1 vdd A B vdd TP W=9e-07 L=1e-06

m2 B A gnd gnd TN 3.0U 2.0U

m3 vdd B C vdd TP W L

m4 C B gnd gnd TN W=P1 L=P2

m5 vdd C D vdd TP WW L=P3

m6 D C gnd gnd TN W=WW L=P4

m7 vdd D E vdd TP L=P5 W=PP2

m8 E D gnd gnd TN

m9 vdd E Q vdd TP W=WW L=LL

m10 Q E gnd gnd TN W=PP3 L=PP4

.ends

Ex 1 : InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m1@MIV

Parameter = W

Value = 9e-7

Reason : W is set to the value 9e-7. Note that for all instance trees ending m1@MIV, will take on the value 9e-7.
Ex 2 : InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m2@MIV

Parameter = L
Value = 2.0U
Reason : Fet m2, employs implied parameters. The first corresponds to W and the second to L. Here L set to 2.0U. For all instance trees ending at m2@MIV, L will take on the value 2.0U

Ex 3 : InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m4@MIV

Parameter = W
Value = 101U
Reason : Fet m4, specifies W = P1 Since P1 is specified on the .SUBCKT line of MIV, it can be passed into m4. Xinv1, the instance above m4 specifies P1 = 101U.

Ex 4 : InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m3@MIV

Parameter = W
Value = 100U
Reason : Fet m3, employs implied parameters. The first corresponds to W, resulting in W = W. Since W is specified on the .SUBCKT line of MIV, it can be passed into m3. Xinv1 specifies W = 100U.

Ex 5 : InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m5@MIV

Parameter = L
Value = 8U
Reason : Fet m5, specifies L = P3. Since P3 is specified on the .SUBCKT line of MIV, it can be passed into m5. However, Xinv1 doesn’t specify P3. So the value for P3 is 8U, being obtained from the cell level specification in MIV.
Ex 6 : InstTree = Xibuf1@IBUF / Xinv2@IBUFTWO / m6@MIV
   Parameter = W
Value = 30U
Reason : Fet m5 specifies W = WW. Since WW is not specified on the .SUBCKT line of MIV, it cannot be passed into m6. So the value for WW is 30U, being obtained from the cell level specification in MIV.

Ex 7 : InstTree = Xibuf1@IBUF / Xinv4@IBUFTWO / m4@MIV
   Parameter = W
Value = 300U
Reason : Fet m4 specifies W = P1. Since P1 is on the .SUBCKT line of MIV, it can be passed in from Xinv4. Xinv4 specifies P1 = W. Since W is on the .SUBCKT line of IBUFTWO, it can be passed into xinv4. Xibuf1 specifies W = 300U.

Ex 8 : InstTree = Xibuf2@IBUF / Xinv3@IBUFTWO / m7@MIV
   Parameter = L
Value = 600U
Reason : Fet m7 specifies L = P5. P5 can be passed into m7. Xinv3 specifies P5 = PPP. PPP cannot be passed into Xinv3. Further, PPP is not at the cell level for Xinv3 because it was not specified on the .SUBCKT IBUF2 nor in the .PARAM statement inside IBUFTWO. However, PPP can be found at the file level for Xinv3, being specified on the .PARAM statement of the file in which Xinv3 resides. Thus PPP = 600U.

Ex 9 : InstTree = Xibuf1@IBUF / Xinv4@IBUFTWO / m10@MIV
   Parameter = L
Value = 34U
Reason: Fet m10 specifies L = PP4. PP4 cannot be passed into m10. PP4 is not at the cell level for m10. PP4 is also not at the file level for m10. It is, however, at the master level, being specified as PP4 = 34U, on the parameter list for model TN.

Ex 10: InstTree = Xibuf1@IBUF / Xinv4@IBUFTWO / m8@MIV
    Parameter = W
    Value = 2.0U
Reason: Fet m10 does not specify. However, its master, TN, specifies W = 2.0U. Fet m10 inherits this value.

Ex 11: InstTree = Xibuf2@IBUF / Xinv4@IBUFTWO / m6@MIV
    Parameter = L
    Value = 30U
Reason: Fet m6 specifies L = P4. P4 can be passed into m6. Xinv4 specifies P4 = WW. WW cannot be passed into Xinv4. Further WW does not exist at the cell level nor the file level for Xinv4. However, WW = 30U is specified in MIV, which is the master of Xinv4.

Parameter Scaling

There are specific, named, fet level parameters that are scaled based on the scaling factor that may be specified on the .OPTION statement. The scaling factor used is the one that was specified for the file in which the fet resides. The parameters L, W, PS, and PD are scaled, with the rule being that the parameter value is multiplied by the scale factor. Also scaled are AS and AD, with the rule being that the parameter value is multiplied by the scale factor squared.

Note: Character case is not considered when determining whether or not to scale a parameter. For example, the parameters l, as, and ad, at the fet level, are scaled.

Ex: Scaling

Consider the following partial netlist.
File: s1.sp

* 
.option scale = 2.0
.param P1=1U
.include '/net/mycds/spice_files/s2.sp'
.subckt IV A Q P2= 2U
m1 vdd A B vdd TP W=P1 L=P2 AS=5U

File: s2.sp

* 
.option scale = 3.0
.subckt IB I O

Xiv1 I A IV P2=4U

For the instance tree Xiv1@IB / m1@IV, to take scaling into account:

- the parameter \(W\) resolves to 2U because \(W = P1 \times \text{(scale from s1.sp)} = 1U \times 2.0 = 2U\).

- the parameter \(L\) resolves to 8U because \(L = P2 \times \text{(scale from s1.sp)} = 4U \times 2.0 = 8U\).
  
  Note that \(P2\) is passed into \(m1\) from \(Xiv1\).

- the parameter \(AS\) resolves to 20U because \(AS = 5U \times \text{(scale from s1.sp)}^2 = 5U \times 4 = 20U\).
Complete ibuf Example Results

InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m1@MIV
  Parameter = W
  Value = 9.000000e-07
  Parameter = L
  Value = 1.000000e-06

InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m2@MIV
  Parameter = W
  Value = 3.000000e-06
  Parameter = L
  Value = 2.000000e-06

InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m3@MIV
  Parameter = W
  Value = 1.000000e-04
  Parameter = L
  Value = 2.005000e-04

InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m4@MIV
  Parameter = W
  Value = 1.010000e-04
  Parameter = L
  Value = 1.020000e-04

InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m5@MIV
  Parameter = W
  Value = 3.000000e-05
  Parameter = L
  Value = 8.000000e-06

InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m6@MIV
Parameter = W  
Value = 3.000000e-05

Parameter = L  
Value = 1.040000e-04

InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m7@MIV
  Parameter = W  
  Value = 2.200000e-05
  Parameter = L  
  Value = 2.100000e-05

InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m8@MIV
  Parameter = W  
  Value = 2.000000e-06
  Parameter = L  
  Value = 1.000000e-06

InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m9@MIV
  Parameter = W  
  Value = 3.000000e-05
  Parameter = L  
  Value = 3.100000e-05

InstTree = Xibuf1@IBUF / Xinv1@IBUFTWO / m10@MIV
  Parameter = W  
  Value = 3.200000e-05
  Parameter = L  
  Value = 3.400000e-05

InstTree = Xibuf1@IBUF / Xinv2@IBUFTWO / m1@MIV
  Parameter = W  
  Value = 9.000000e-07
Parameter  =  L
Value  =  1.000000e-06

InstTree  =  Xibuf1@IBUF / Xinv2@IBUFTWO / m2@MIV
Parameter  =  W
Value  =  3.000000e-06
Parameter  =  L
Value  =  2.000000e-06

InstTree  =  Xibuf1@IBUF / Xinv2@IBUFTWO / m3@MIV
Parameter  =  W
Value  =  4.030000e-04
Parameter  =  L
Value  =  4.000000e-06

InstTree  =  Xibuf1@IBUF / Xinv2@IBUFTWO / m4@MIV
Parameter  =  W
Value  =  6.000000e-06
Parameter  =  L
Value  =  7.000000e-06

InstTree  =  Xibuf1@IBUF / Xinv2@IBUFTWO / m5@MIV
Parameter  =  W
Value  =  3.000000e-05
Parameter  =  L
Value  =  8.000000e-06

InstTree  =  Xibuf1@IBUF / Xinv2@IBUFTWO / m6@MIV
Parameter  =  W
Value  =  3.000000e-05
Parameter  =  L
Value  =  9.000000e-06
InstTree = Xibuf1@IBUF / Xinv2@IBUFTWO / m7@MIV
  Parameter = W
  Value = 2.200000e-05
  Parameter = L
  Value = 2.100000e-05

InstTree = Xibuf1@IBUF / Xinv2@IBUFTWO / m8@MIV
  Parameter = W
  Value = 2.000000e-06
  Parameter = L
  Value = 1.000000e-06

InstTree = Xibuf1@IBUF / Xinv2@IBUFTWO / m9@MIV
  Parameter = W
  Value = 3.000000e-05
  Parameter = L
  Value = 3.100000e-05

InstTree = Xibuf1@IBUF / Xinv2@IBUFTWO / m10@MIV
  Parameter = W
  Value = 3.200000e-05
  Parameter = L
  Value = 3.400000e-05

InstTree = Xibuf1@IBUF / Xinv3@IBUFTWO / m1@MIV
  Parameter = W
  Value = 9.000000e-07
  Parameter = L
  Value = 1.000000e-06

InstTree = Xibuf1@IBUF / Xinv3@IBUFTWO / m2@MIV
  Parameter = W
Value  =  3.000000e-06
Parameter  =  L
Value  =  2.000000e-06

InstTree  =  Xibuf1@IBUF / Xinv3@IBUFTWO / m3@MIV
Parameter  =  W
Value  =  5.000000e-06
Parameter  =  L
Value  =  4.000000e-06

InstTree  =  Xibuf1@IBUF / Xinv3@IBUFTWO / m4@MIV
Parameter  =  W
Value  =  6.000000e-06
Parameter  =  L
Value  =  7.000000e-06

InstTree  =  Xibuf1@IBUF / Xinv3@IBUFTWO / m5@MIV
Parameter  =  W
Value  =  3.000000e-05
Parameter  =  L
Value  =  8.000000e-06

InstTree  =  Xibuf1@IBUF / Xinv3@IBUFTWO / m6@MIV
Parameter  =  W
Value  =  3.000000e-05
Parameter  =  L
Value  =  9.000000e-06

InstTree  =  Xibuf1@IBUF / Xinv3@IBUFTWO / m7@MIV
Parameter  =  W
Value  =  2.200000e-05
Parameter  =  L
Value  =  6.000000e-04
InstTree  = Xibuf1@IBUF / Xinv3@IBUFTWO / m8@MIV
  Parameter =  W
  Value    =  2.000000e-06
  Parameter =  L
  Value    =  1.000000e-06

InstTree  = Xibuf1@IBUF / Xinv3@IBUFTWO / m9@MIV
  Parameter =  W
  Value    =  3.000000e-05
  Parameter =  L
  Value    =  3.100000e-05

InstTree  = Xibuf1@IBUF / Xinv3@IBUFTWO / m10@MIV
  Parameter =  W
  Value    =  3.200000e-05
  Parameter =  L
  Value    =  3.400000e-05

InstTree  = Xibuf1@IBUF / Xinv4@IBUFTWO / m1@MIV
  Parameter =  W
  Value    =  9.000000e-07
  Parameter =  L
  Value    =  1.000000e-06

InstTree  = Xibuf1@IBUF / Xinv4@IBUFTWO / m2@MIV
  Parameter =  W
  Value    =  3.000000e-06
  Parameter =  L
  Value    =  2.000000e-06

InstTree  = Xibuf1@IBUF / Xinv4@IBUFTWO / m3@MIV
  Parameter =  W
Value = 5.000000e-06
Parameter = L
Value = 4.000000e-06

InstTree = Xibuf1@IBUF / Xinv4@IBUFTWO / m4@MIV
  Parameter = W
  Value = 3.000000e-04
  Parameter = L
  Value = 7.000000e-06

InstTree = Xibuf1@IBUF / Xinv4@IBUFTWO / m5@MIV
  Parameter = W
  Value = 3.000000e-05
  Parameter = L
  Value = 8.000000e-06

InstTree = Xibuf1@IBUF / Xinv4@IBUFTWO / m6@MIV
  Parameter = W
  Value = 3.000000e-05
  Parameter = L
  Value = 3.000000e-05

InstTree = Xibuf1@IBUF / Xinv4@IBUFTWO / m7@MIV
  Parameter = W
  Value = 2.200000e-05
  Parameter = L
  Value = 2.100000e-05

InstTree = Xibuf1@IBUF / Xinv4@IBUFTWO / m8@MIV
  Parameter = W
  Value = 2.000000e-06
  Parameter = L
  Value = 1.000000e-06
InstTree = Xibuf1@IBUF / Xinv4@IBUFTWO / m9@MIV
  Parameter = W
  Value = 3.000000e-05
  Parameter = L
  Value = 3.100000e-05

InstTree = Xibuf1@IBUF / Xinv4@IBUFTWO / m10@MIV
  Parameter = W
  Value = 3.200000e-05
  Parameter = L
  Value = 3.400000e-05

InstTree = Xibuf2@IBUF / Xinv1@IBUFTWO / m10@MIV
  Parameter = W
  Value = 9.000000e-07
  Parameter = L
  Value = 1.000000e-06

InstTree = Xibuf2@IBUF / Xinv1@IBUFTWO / m2@MIV
  Parameter = W
  Value = 3.000000e-06
  Parameter = L
  Value = 2.000000e-06

InstTree = Xibuf2@IBUF / Xinv1@IBUFTWO / m3@MIV
  Parameter = W
  Value = 1.000000e-04
  Parameter = L
  Value = 4.040000e-04

InstTree = Xibuf2@IBUF / Xinv1@IBUFTWO / m4@MIV
  Parameter = W
Value = 1.010000e-04
Parameter = L

Value = 1.020000e-04

InstTree = Xibuf2@IBUF / Xinv1@IBUFTWO / m5@MIV
Parameter = W
Value = 3.000000e-05

Parameter = L
Value = 8.000000e-06

InstTree = Xibuf2@IBUF / Xinv1@IBUFTWO / m6@MIV
Parameter = W
Value = 3.000000e-05

Parameter = L
Value = 1.040000e-04

InstTree = Xibuf2@IBUF / Xinv1@IBUFTWO / m7@MIV
Parameter = W
Value = 2.200000e-05

Parameter = L
Value = 2.100000e-05

InstTree = Xibuf2@IBUF / Xinv1@IBUFTWO / m8@MIV
Parameter = W
Value = 2.000000e-06

Parameter = L
Value = 1.000000e-06

InstTree = Xibuf2@IBUF / Xinv1@IBUFTWO / m9@MIV
Parameter = W
Value = 3.000000e-05

Parameter = L
Value = 3.100000e-05

InstTree = Xibuf2@IBUF / Xinv1@IBUFTWO / m10@MIV
  Parameter = W
  Value = 3.200000e-05
  Parameter = L
  Value = 3.400000e-05

InstTree = Xibuf2@IBUF / Xinv2@IBUFTWO / m1@MIV
  Parameter = W
  Value = 9.000000e-07
  Parameter = L
  Value = 1.000000e-06

InstTree = Xibuf2@IBUF / Xinv2@IBUFTWO / m2@MIV
  Parameter = W
  Value = 3.000000e-06
  Parameter = L
  Value = 2.000000e-06

InstTree = Xibuf2@IBUF / Xinv2@IBUFTWO / m3@MIV
  Parameter = W
  Value = 2.010000e-04
  Parameter = L
  Value = 4.000000e-06

InstTree = Xibuf2@IBUF / Xinv2@IBUFTWO / m4@MIV
  Parameter = W
  Value = 6.000000e-06
  Parameter = L
  Value = 7.000000e-06

InstTree = Xibuf2@IBUF / Xinv2@IBUFTWO / m5@MIV
Parameter = W
Value = 3.000000e-05

Parameter = L
Value = 8.000000e-06

InstTree = Xibuf2@IBUF / Xinv2@IBUFTWO / m6@MIV
Parameter = W
Value = 3.000000e-05
Parameter = L
Value = 9.000000e-06

InstTree = Xibuf2@IBUF / Xinv2@IBUFTWO / m7@MIV
Parameter = W
Value = 2.200000e-05
Parameter = L
Value = 2.100000e-05

InstTree = Xibuf2@IBUF / Xinv2@IBUFTWO / m8@MIV
Parameter = W
Value = 2.000000e-06
Parameter = L
Value = 1.000000e-06

InstTree = Xibuf2@IBUF / Xinv2@IBUFTWO / m9@MIV
Parameter = W
Value = 3.000000e-05
Parameter = L
Value = 3.100000e-05

InstTree = Xibuf2@IBUF / Xinv2@IBUFTWO / m10@MIV
Parameter = W
Value = 3.200000e-05
Parameter = L
Value = 3.400000e-05

InstTree = Xibuf2@IBUF / Xinv3@IBUFTWO / m1@MIV
Parameter = W
   Value = 9.000000e-07
Parameter = L
   Value = 1.000000e-06

InstTree = Xibuf2@IBUF / Xinv3@IBUFTWO / m2@MIV
Parameter = W
   Value = 3.000000e-06
Parameter = L
   Value = 2.000000e-06

InstTree = Xibuf2@IBUF / Xinv3@IBUFTWO / m3@MIV
Parameter = W
   Value = 5.000000e-06
Parameter = L
   Value = 4.000000e-06

InstTree = Xibuf2@IBUF / Xinv3@IBUFTWO / m4@MIV
Parameter = W
   Value = 6.000000e-06
Parameter = L
   Value = 7.000000e-06

InstTree = Xibuf2@IBUF / Xinv3@IBUFTWO / m5@MIV
Parameter = W
   Value = 3.000000e-05
Parameter = L
   Value = 8.000000e-06
InstTree = Xibuf2@IBUF / Xinv3@IBUFTWO / m6@MIV
    Parameter = W
    Value = 3.000000e-05
    Parameter = L
    Value = 9.000000e-06

InstTree = Xibuf2@IBUF / Xinv3@IBUFTWO / m7@MIV
    Parameter = W
    Value = 2.200000e-05
    Parameter = L
    Value = 6.000000e-04

InstTree = Xibuf2@IBUF / Xinv3@IBUFTWO / m8@MIV
    Parameter = W
    Value = 2.000000e-06
    Parameter = L
    Value = 1.000000e-06

InstTree = Xibuf2@IBUF / Xinv3@IBUFTWO / m9@MIV
    Parameter = W
    Value = 3.000000e-05
    Parameter = L
    Value = 3.100000e-05

InstTree = Xibuf2@IBUF / Xinv3@IBUFTWO / m10@MIV
    Parameter = W
    Value = 3.200000e-05
    Parameter = L
    Value = 3.400000e-05

InstTree = Xibuf2@IBUF / Xinv4@IBUFTWO / m1@MIV
    Parameter = W
Value = 9.000000e-07
Parameter = L
Value = 1.000000e-06

InstTree = Xibuf2@IBUF / Xinv4@IBUFTWO / m2@MIV
Parameter = W
Value = 3.000000e-06
Parameter = L
Value = 2.000000e-06

InstTree = Xibuf2@IBUF / Xinv4@IBUFTWO / m3@MIV
Parameter = W
Value = 5.000000e-06
Parameter = L
Value = 4.000000e-06

InstTree = Xibuf2@IBUF / Xinv4@IBUFTWO / m4@MIV
Parameter = W
Value = 4.010000e-04
Parameter = L
Value = 7.000000e-06

InstTree = Xibuf2@IBUF / Xinv4@IBUFTWO / m5@MIV
Parameter = W
Value = 3.000000e-05
Parameter = L
Value = 8.000000e-06

InstTree = Xibuf2@IBUF / Xinv4@IBUFTWO / m6@MIV
Parameter = W
Value = 3.000000e-05
Parameter = L
Value = 3.000000e-05
InstTree  = Xibuf2@IBUF / Xinv4@IBUFTWO / m7@MIV
    Parameter  =  W
        Value  =  2.200000e-05
    Parameter  =  L
        Value  =  2.100000e-05

InstTree  = Xibuf2@IBUF / Xinv4@IBUFTWO / m8@MIV
    Parameter  =  W
        Value  =  2.000000e-06
    Parameter  =  L
        Value  =  1.000000e-06

InstTree  = Xibuf2@IBUF / Xinv4@IBUFTWO / m9@MIV
    Parameter  =  W
        Value  =  3.000000e-05
    Parameter  =  L
        Value  =  3.100000e-05

InstTree  = Xibuf2@IBUF / Xinv4@IBUFTWO / m10@MIV
    Parameter  =  W
        Value  =  3.200000e-05
    Parameter  =  L
        Value  =  3.400000e-05

InstTree  = Xibuf3@IBUF / Xinv1@IBUFTWO / m1@MIV
    Parameter  =  W
        Value  =  9.000000e-07
    Parameter  =  L
        Value  =  1.000000e-06

InstTree  = Xibuf3@IBUF / Xinv1@IBUFTWO / m2@MIV
    Parameter  =  W
Value = 3.000000e-06
Parameter = L
Value = 2.000000e-06

InstTree = Xibuf3@IBUF / Xinv1@IBUFTWO / m3@MIV
Parameter = W
Value = 1.000000e-04
Parameter = L
Value = 2.130000e-04

InstTree = Xibuf3@IBUF / Xinv1@IBUFTWO / m4@MIV
Parameter = W
Value = 1.010000e-04
Parameter = L
Value = 1.020000e-04

InstTree = Xibuf3@IBUF / Xinv1@IBUFTWO / m5@MIV
Parameter = W
Value = 3.000000e-05
Parameter = L
Value = 8.000000e-06

InstTree = Xibuf3@IBUF / Xinv1@IBUFTWO / m6@MIV
Parameter = W
Value = 3.000000e-05
Parameter = L
Value = 1.040000e-04

InstTree = Xibuf3@IBUF / Xinv1@IBUFTWO / m7@MIV
Parameter = W
Value = 2.200000e-05
Parameter = L
Value = 2.100000e-05
InstTree = Xibuf3@IBUF / Xinv1@IBUFTWO / m8@MIV
  Parameter = W
  Value = 2.000000e-06
  Parameter = L
  Value = 1.000000e-06

InstTree = Xibuf3@IBUF / Xinv1@IBUFTWO / m9@MIV
  Parameter = W
  Value = 3.000000e-05
  Parameter = L
  Value = 3.100000e-05

InstTree = Xibuf3@IBUF / Xinv1@IBUFTWO / m10@MIV
  Parameter = W
  Value = 3.200000e-05
  Parameter = L
  Value = 3.400000e-05

InstTree = Xibuf3@IBUF / Xinv2@IBUFTWO / m1@MIV
  Parameter = W
  Value = 9.000000e-07
  Parameter = L
  Value = 1.000000e-06

InstTree = Xibuf3@IBUF / Xinv2@IBUFTWO / m2@MIV
  Parameter = W
  Value = 3.000000e-06
  Parameter = L
  Value = 2.000000e-06

InstTree = Xibuf3@IBUF / Xinv2@IBUFTWO / m3@MIV
  Parameter = W
Value = 2.010000e-04
Parameter = L
Value = 4.000000e-06

InstTree = Xibuf3@IBUF / Xinv2@IBUFTWO / m4@MIV
Parameter = W
Value = 6.000000e-06
Parameter = L
Value = 7.000000e-06

InstTree = Xibuf3@IBUF / Xinv2@IBUFTWO / m5@MIV
Parameter = W
Value = 3.000000e-05
Parameter = L
Value = 8.000000e-06

InstTree = Xibuf3@IBUF / Xinv2@IBUFTWO / m6@MIV
Parameter = W
Value = 3.000000e-05
Parameter = L
Value = 9.000000e-06

InstTree = Xibuf3@IBUF / Xinv2@IBUFTWO / m7@MIV
Parameter = W
Value = 2.200000e-05
Parameter = L
Value = 2.100000e-05

InstTree = Xibuf3@IBUF / Xinv2@IBUFTWO / m8@MIV
Parameter = W
Value = 2.000000e-06
Parameter = L
Value = 1.000000e-06
InstTree = Xibuf3@IBUF / Xinv2@IBUFTWO / m9@MIV
  Parameter = W
  Value = 3.000000e-05
  Parameter = L
  Value = 3.100000e-05

InstTree = Xibuf3@IBUF / Xinv2@IBUFTWO / m10@MIV
  Parameter = W
  Value = 3.200000e-05
  Parameter = L
  Value = 3.400000e-05

InstTree = Xibuf3@IBUF / Xinv3@IBUFTWO / m1@MIV
  Parameter = W
  Value = 9.000000e-07
  Parameter = L
  Value = 1.000000e-06

InstTree = Xibuf3@IBUF / Xinv3@IBUFTWO / m2@MIV
  Parameter = W
  Value = 3.000000e-06
  Parameter = L
  Value = 2.000000e-06

InstTree = Xibuf3@IBUF / Xinv3@IBUFTWO / m3@MIV
  Parameter = W
  Value = 5.000000e-06
  Parameter = L
  Value = 4.000000e-06

InstTree = Xibuf3@IBUF / Xinv3@IBUFTWO / m4@MIV
  Parameter = W
Value = 6.000000e-06
Parameter = L
Value = 7.000000e-06

InstTree = Xibuf3@IBUF / Xinv3@IBUFTWO / m5@MIV
Parameter = W
Value = 3.000000e-05
Parameter = L
Value = 8.000000e-06

InstTree = Xibuf3@IBUF / Xinv3@IBUFTWO / m6@MIV
Parameter = W
Value = 3.000000e-05
Parameter = L
Value = 9.000000e-06

InstTree = Xibuf3@IBUF / Xinv3@IBUFTWO / m7@MIV
Parameter = W
Value = 2.200000e-05
Parameter = L
Value = 6.000000e-04

InstTree = Xibuf3@IBUF / Xinv3@IBUFTWO / m8@MIV
Parameter = W
Value = 2.000000e-06
Parameter = L
Value = 1.000000e-06

InstTree = Xibuf3@IBUF / Xinv3@IBUFTWO / m9@MIV
Parameter = W
Value = 3.000000e-05
Parameter = L
Value = 3.100000e-05
InstTree  = Xibuf3@IBUF / Xinv3@IBUFTWO / m10@MIV
  Parameter  =  W
  Value  =  3.200000e-05
  Parameter  =  L
  Value  =  3.400000e-05

InstTree  = Xibuf3@IBUF / Xinv4@IBUFTWO / m1@MIV
  Parameter  =  W
  Value  =  9.000000e-07
  Parameter  =  L
  Value  =  1.000000e-06

InstTree  = Xibuf3@IBUF / Xinv4@IBUFTWO / m2@MIV
  Parameter  =  W
  Value  =  3.000000e-06
  Parameter  =  L
  Value  =  2.000000e-06

InstTree  = Xibuf3@IBUF / Xinv4@IBUFTWO / m3@MIV
  Parameter  =  W
  Value  =  5.000000e-06
  Parameter  =  L
  Value  =  4.000000e-06

InstTree  = Xibuf3@IBUF / Xinv4@IBUFTWO / m4@MIV
  Parameter  =  W
  Value  =  2.000000e-04
  Parameter  =  L
  Value  =  7.000000e-06

InstTree  = Xibuf3@IBUF / Xinv4@IBUFTWO / m5@MIV
  Parameter  =  W
Value  =  3.000000e-05
Parameter  =  L
Value  =  8.000000e-06

InstTree  =  Xibuf3@IBUF / Xinv4@IBUFTWO / m6@MIV
  Parameter  =  W
  Value  =  3.000000e-05
  Parameter  =  L
  Value  =  3.000000e-05

InstTree  =  Xibuf3@IBUF / Xinv4@IBUFTWO / m7@MIV
  Parameter  =  W
  Value  =  2.200000e-05
  Parameter  =  L
  Value  =  2.100000e-05

InstTree  =  Xibuf3@IBUF / Xinv4@IBUFTWO / m8@MIV
  Parameter  =  W
  Value  =  2.000000e-06
  Parameter  =  L
  Value  =  1.000000e-06

InstTree  =  Xibuf3@IBUF / Xinv4@IBUFTWO / m9@MIV
  Parameter  =  W
  Value  =  3.000000e-05
  Parameter  =  L
  Value  =  3.100000e-05

InstTree  =  Xibuf3@IBUF / Xinv4@IBUFTWO / m10@MIV
  Parameter  =  W
  Value  =  3.200000e-05
  Parameter  =  L
  Value  =  3.400000e-05
.do File Commands

When Virtuoso XL wire editing is enabled, you can use the commands below in a .do file and execute the file through Virtuoso XL.

- circuit
- define/forget
- do
- limit
- rule
- set

For information about loading .do files, see “Using .do Files” on page 393.

Rule Hierarchy

Based on the available commands, use an object keyword with each command to determine at which level of rule precedence the routing rules are applied. The following table describes the routing rule hierarchy from highest to lowest.

**Rule Hierarchy**

region class_class
region net
region class
region
via_image
class_class layer
class_class
Rule Hierarchy

- group layer
- group
- net layer
- net
- group_set layer
- group_set
- class layer
- class
- layer
- ic

For example,

```plaintext
rule net (inter_layer_clear 3 (layer_pair poly diffusion))
```

would overwrite the following rule,

```plaintext
rule ic (inter_layer_clear 3 (layer_pair poly diffusion))
```

For more information about rules precedence, see “Understanding the Routing Rule Hierarchy” in the *Virtuoso Chip Assembly Router User Reference*.
Commands

circuit

Use the `circuit` command to assign length, delay, capacitance, resistance, shielding rules, routing priorities, vias, and routing layers to the following:

```
circuit net

Example
```
circuit net J1 (length 4.4 3.9 (type ratio))
```

Syntax

For information, see `circuit` in the *Virtuoso Chip Assembly Router User Reference*.
**define/forget**

Use the define command to create or define the following types of objects or conditions:

define/forget group
define/forget group_set
define/forget keepout
define/forget region

**Syntax**

For information, see the following in the *Virtuoso Chip Assembly Router User Reference*.

**group**

**group_set**

**keepout**

**Example**

define (keepout myKeepout (rect signal 0 0 5 5))
do

Use the do command to read and execute other .do files. Any additional .do files you include in a .do file, support the circuit, cost/tax, define/forget, limit, rule, and set commands only.

For more information, see “Using .do Files” on page 393.
limit

Use the limit command to set the number of vias allowed on a route.

limit vias

Syntax

For more information, see limit in the Virtuoso Chip Assembly Router User Reference.

Example

limit via 3
rule

Use the rule command to set design rules or routing constraints.

rule ic
rule class
rule class_class
rule group
rule group_set
rule layer
rule net

Syntax

For more information, see rule in the Virtuoso Chip Assembly Router User Reference.

Example

rule layer M1 (width 2.5) (clearance 1.5) (add_diag_clear 0.05)
rule net wr2 (shield_gap 1.0)
rule net wr2 (shield_width 3.0)
set

The `set` command controls the following routing options.

```plaintext
set extend_wire_for_via
set trim_extensions
```

Syntax

For more information, see `set` in the *Virtuoso Chip Assembly Router User Reference*. 
Properties

This appendix provides property names, descriptions, and graphical user interface (GUI) equivalents for the Virtuoso® XL Layout Editor (Virtuoso XL).

Important

Only the properties documented in this chapter are supported for public use. All other Virtuoso XL properties, regardless of their name or prefix, and undocumented aspects of the properties described below, are private and are subject to change at any time.

You can set some of the properties described in this chapter in the graphical user interface. This section tells you how to access the appropriate form.

Accessing the Symbol Master Add Property Form

To add a user property to a symbol master,

1. From the Virtuoso Symbol Editing window, choose Edit – Properties – Cellview.

   The Edit Cellview Properties form appears.

2. In the User Property section, click Add.

   The Add Property form appears.

Accessing the Add CDF Parameter Form

To add a CDF parameter to a layout or schematic master,

1. From the CIW, choose Tools – CDF – Edit.

   The Edit Component CDF form is displayed.

2. Specify the library and cell names and click Apply.

   The form is populated with the options you can set.
3. In the Component Parameters section, click Add.  
   The Add CDF Parameter form is displayed.

**Accessing the Layout Instance Add Property Form**

To add a user property to an instance in the layout,

1. In the layout window design area, select the instance to which you want to add a property.

2. From the menu bar, choose Edit – Properties.
   The Edit Instance Properties form is displayed.

3. Choose the Property radio button and click Add.
   The Add Property form is displayed.
abutAccessDir

Defines the directions the pins are able to abut.

Property Name  abutAccessDir
Value  left, right, bottom, top
Value Type  list

Additional Information

The abutAccessDir property ensures that the proper edges of devices are abutting; for example, a left side to a right side, or a bottom to a top. Cells with rail pins need to add abutAccessDir properties for both sides because there is one pin for both right and left sides of the cell left right. MOS cells have just one pin per side, so they need only one direction left.

Example

dbReplaceProp(objId "abutAccessDir" "list" list("left"))
**abutClass**

Lets two different cells abut, even if their master cells are not the same.

**Property Name**  abutClass

**Value**  user defined name

**Value Type**  string

**Additional Information**

The `abutClass` property is placed on the pin. Each pin targeted for abutment can belong to only one class, but pins on different edges of a cell can belong to different classes. Therefore, a cell can have multiple abutment classes but only one class per pin. The property can also be added to the cellview to implement a global class for all the pins. Automatic abutment checks for the `abutClass` property first on the pin and then on the master.

**Example**

```plaintext
dbReplaceProp(leftObj~>dbId "abutClass" "string" d_layer)
```
abutCondInclusion

Defines the name of a boolean property that adds or removes contact shapes over the diffusion pin.

Property Name       abutCondInclusion
Value               name of pcell parameter
Value Type          string

Additional Information

The abutCondInclusion parameter adds or removes the source or drain contact. You can conditionally include or exclude the contacts and connecting metal tab. You must conditionally exclude these features if you are creating a graphical pcell. The pcell parameter must remove all features that are required for metal hookup and leave only the diffusion material to be stretched. For CMOS device abutment, you add this property to both the drain and the source sides of the device.

Example

dbReplaceProp(rightObj~>dbId "abutCondInclusion" "string" "rightCnt")
abutFunction

Specifies a user defined callback to process abutment and unabutment.

Property Name       abutFunction

Value                name of SKILL callback

Value Type           string

Additional Information

The abutFunction is a callback that you create to process abutment and unabutment. The value of this property is a string consisting of the name of a user-defined Cadence® SKILL function that is executed before abutment takes place. Automatic abutment passes this function eight arguments in a list made up of the following information in this order:

1. The database identifier of the cell being abutted
2. The database identifier of the cell being abutted to
3. The database identifier of the pin figure of the cell being abutted
4. The database identifier of the pin figure of the cell being abutted to
5. The abutment access direction of pin being abutted (integer):
   - 1 - top
   - 2 - bottom
   - 4 - left
   - 8 - right
6. An integer with a valid value of 1 or 2 that indicates connection condition:
   - 1 - pins are connected to the same net and do not connect to any other pin
   - 2 - pins are connected to the same net and the net connects to other pins
7. An integer specifying automatic abutment events:
   - 1 - abutFunction computes and returns abutment offset (in the direction of abutment)
   - 2 - abutFunction adjusts pcell parameters for abutment. This is called before the offset event.
abutFunction adjusts pcell parameters for unabutment

abutFunction computes and return spacing offset for non-abutment.

8. A database identifier that is the abutment group pointer available to events 2 and 3. It is used to store information so that unabutment can return pins to their original state. It is a generally-accepted method to use the group attribute of a cell to hold original parameter values prior to modification by abutment.

abutFunction returns different values for different automatic abutment events:

- Event 1 returns a floating-point number, which is the offset in user units needed to abut the cells in the direction of abutment.
  
  If it returns two floating-point numbers, the second number is the distance in user units that the abutting cell is moved in the direction perpendicular to the abutting edge.

- Event 2 returns a value of true to allow abutment and nil to block abutment. When event 2 returns nil, it must not change any parameters in any way, including setting them to new values and then resetting them to the original values.

- If the event is 3, it can return anything; the value returned is not used.

Example

Below is an example of the abutFunction property:

``` procedure( abutFunction(iA iB pA pB pASide connection event @optional (group nil)) prog((result)
    case(event
      (1 ; Compute offset
        result = getAbutmentOffset(iA iB pA pB pASide connection)
      )
      (2 ; Adjust pcell parameters
        result = setAbutmentParams(iA iB pA pB pASide connection
                                 group)
      )
      (3 ; Adjust pcell parameters back to default
        resetAbutmentParams(group iA iB)
        result = t
      )
      (4 ; To get the spacing offset when pins are on different nets
        if(instA~>w > 3.0 && instB~>w > 3.0
          then
            result = 3
          else
            result = 1
          then
```
Note: If you are using pcells, you can use the default abutment functions.

If each pin being abutted has a different abutFunction defined, automatic abutment calls only the function defined by the pin in the cell being moved.

Example

dbReplaceProp(obj "abutFunction" "string" "mosAbutFunc")
**abutGateNet**

Defines the name of the net the gate shapes are on.

- **Property Name**: abutGateNet
- **Value**: name of pcell parameter
- **Value Type**: string

**Additional Information**

This parameter identifies the gate pins so the abutment software knows where the edge of the gate is. In the example, the value G is the name of the gate net inside the pcell.

**Example**

```plaintext
dbReplaceProp(leftObj~>dbId "abutGateNet" "string" "G")
```
**abutMosStretchMat**

Governs how a material is stretched towards or away from the gate when the instance is altered for abutment.

*Property Name*  
**abutMosStretchMat**

*Value*  
abutFlush, abutGateSpaceSeries, abutGateSpaceParallel, abutEncloseGate, abutEncloseContact, abutSmallerSeries, abutSmallerParallel, abutEncloseDogBoneContact, abutMinContactWidth

*Value Type*  
list

**Additional Information**

The picture below illustrates the MOS diffusion extension rules for the different *abutMosStretchMat* property values.

The *abutMosStretchMat* diffusion extension rules are described below.

**A = abutFlush (Not shown).** Applied to the pin which does not move when abutting two pins of equal width.

**B = abutGateSpaceSeries.** Applied to the moving pin when abutting two pins of equal width with no contact required between the gates. The *abutFlush* rule is applied to the non-moving pin.
C = abutGateSpaceParallel. Applied to the moving pin when abutting pins of equal width where a contact is required between the gates and the pin widths are greater than or equal to the abutMinContactWidth rule. The abutFlush rule is applied to the non-moving pin.

D = abutEncloseGate. Applied to the larger of two abutting pins when a contact is not required between the gates. The abutSmallerSeries rule is applied to the smaller of the two pins being abutted.

E = abutEncloseContact. Applied to the larger of two different-width abutting pins when a contact is required between the gates but only if the width of the larger pin is greater than or equal to the abutMinContactWidth rule. The abutSmallerParallel rule is applied to the smaller of the two pins being abutted.

F = abutSmallerSeries. Applied to the smaller of two abutting pins when a contact is not required between the gates.

G = abutSmallerParallel. Applied to the smaller of two abutting pins when a contact is required between the gates.

H = abutEncloseDogBoneContact. Applied when contacts are required and the width of the device is less than the abutMinContactWidth rule. It is applied to the larger of two different-width pins or to the moving pin when the two pins are of equal width.

I = abutMinContactWidth. Determines whether to use the abutEncloseContact rule or the abutEncloseDogBoneContact rule when a contact is required between gates.

Examples

```
list(t_stretchParamName
   list("abutFlush" f_value)
   list("abutGateSpaceSeries" f_value)
   list("abutGateSpaceParallel" f_value)
   list("abutEncloseGate" f_value)
   list("abutEncloseContact" f_value)
   list("abutSmallerSeries" f_value)
   list("abutSmallerParallel" f_value)
   list("abutEncloseDogBoneContact" f_value)
   list("abutMinContactWidth" f_value)
)
```

The example below illustrates the properties being applied to pins on the left and right edges of a MOS pcell.

- The pin rectangles were created by the rodCreateRect command and are referenced by two variables called leftObj and rightObj pointing to their rod object IDs.
The pcell parameters that control the stretching of the left and right pin extensions past the gate are called leftSt and rightSt.

All abutment extension rule values in the abutMosStretchMat property list must be numeric.

The pcell parameters that control whether or not the left or right pins have contacts over them are leftCnt and rightCnt.

The contact parameters must be of type Boolean.

;; Definition of variables used in the abutRuleList calculations for the abutMosStretchMat property.

;; cw = contact width
;; doc = diffusion overlap of contact
;; pps = poly to poly space
;; pcs = poly to contact space
;; pds = poly to diffusion space

abutRuleList = list(
    list("abutFlush" 0.0)
    list("abutGateSpaceSeries" pps)
    list("abutGateSpaceParallel" max((cw + (pcs * 2)) pps))
    list("abutEncloseGate" doc)
    list("abutEncloseContact" doc + cw + pcs)
    list("abutSmallerSeries" max((pps - doc) pds))
    list("abutSmallerParallel" max((pcs - doc) (pps - pcs - cw - doc) pds))
    list("abutEncloseDogBoneContact" cw + doc + max(pds+doc pcs))
    list("abutMinContactWidth" cw + (2 * doc))
)

;; Definition of variables used for abutClass and autoSpacer property values.
;; d_layer = diffusion layer (used for abutClass property value)
;; dds = diffusion to diffusion space (used by autoSpacer)

;; Add the abutment and spacer properties to the left diffusion pin.

leftObj->dbId->pin->name = "leftContact"
dbReplaceProp(leftObj->dbId "abutCondInclusion" "string" "leftCnt")
dbReplaceProp(leftObj->dbId "abutMosStretchMat" "list"
    cons("leftSt" abutRuleList))
dbReplaceProp(leftObj->dbId "abutClass" "string" d_layer)
dbReplaceProp(leftObj->dbId "abutGateNet" "string" "G")
dbReplaceProp(leftObj->dbId "abutAccessDir" "list" list("left"))
dbReplaceProp(leftObj->dbId "vxlInstSpacingDir" "list" list("left"))
dbReplaceProp(leftObj->dbId "vxlInstSpacingRule" "float" dds)

;; Add the abutment and spacer properties to the right diffusion pin.

rightObj->dbId->pin->name = "rightContact"
dbReplaceProp(rightObj->dbId "abutCondInclusion" "string" "rightCnt")
dbReplaceProp(rightObj->dbId "abutMosStretchMat" "list"
    cons("rightSt" abutRuleList))
dbReplaceProp(rightObj->dbId "abutClass" "string" d_layer)
dbReplaceProp(rightObj->dbId "abutGateNet" "string" "G")
dbReplaceProp(rightObj->dbId "abutAccessDir" "list" list("right"))
dbReplaceProp(rightObj->dbId "vxlInstSpacingDir" "list" list("right"))
dbReplaceProp(rightObj->dbId "vxlInstSpacingRule" "float" dds)
abutStretchMat

Stretches the material toward or away from the gate when the instance is altered for abutment.

Property Name: abutStretchMat
Value: drainStretch, abutMinExt, abutRule1Ext, abutRule2Ext, abutContactExt
Value Type: list

Additional Information

The default value of the stretch parameter is the distance from the edge of the material to the edge of the gate. Numeric values are in user units.

- The first element in the list (drainStretch) is the name of the stretchable material parameter. You name this parameter yourself to identify the material to be stretched.

- The second element in the list (abutMinExt) is the rule for minimum diffusion overhang from the edge of the poly gate. The argument to this element (0.5) is the rule value.

- The third element in the list (abutRule1Ext) is the polysilicon-layer-to-polysilicon-layer rule. This rule is used when the net connecting the two instances does not share the net with any other pin.

- The fourth element in the list (abutRule2Ext) is the polysilicon-layer-to-diffusion-layer rule. The rule is used when the transistor width is different for the two instances being abutted.

Caution

The placer does not take the abutRule2Ext into consideration, resulting in illegal overlap markers.

- The fifth element in the list (abutContactExt) is the diffusion extension value used when one of the contacts needs to be added during the abutment process. With the contacts off by default, the abutment program needs to know how close to move the selected cell to abut properly.
Example

abutStretchMat list ("drainStretch" illist
  list ("abutMinExt" 0.5)
  list ("abutRule1Ext" 0.75)
  list ("abutRule2Ext" 1.0)
  list ("abutContactExt" 3.0))
**abutOffset**

Specifies a distance in user units to offset the reference edge to the outside direction of the cell. A negative number causes the pins to overlap.

**Note:** The reference edge is the outside edge of the pin that triggered the abutment on each cell in the direction defined by the abutment direction.

**Property Name**  
abutOffset

**Value**  
floating point number

**Value Type**  
float

**Additional Information**

If the `abutFunction compute offset event (event 1)` returns `nil`, automatic abutment looks for an `abutOffset` property on the pin and applies the value of this property as the offset. If there is no `abutOffset` property, automatic abutment uses the outside edge of the pin with no offset.
If the pin of the device being abutted overlaps the other pin on the bottom or top, the pins are aligned by the bottom or top edges respectively. This also applies from side to side when abutment is vertical.

Interactive move places one cell approximately overlapping another

Cells are first aligned and then the offset is applied. In this case, abutOffset = 0 or is not set

Example

abutOffset 5
ignore

Ignores the corresponding layout device so that it does not appear in the layout.

Form Fields

Table A-1 In the Symbol Master Add Property Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>ignore</td>
</tr>
<tr>
<td>Type</td>
<td>boolean</td>
</tr>
<tr>
<td>Value</td>
<td>t, nil</td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table A-2 In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>boolean</td>
</tr>
<tr>
<td>name</td>
<td>ignore</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>t, nil</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Additional Information

If you do not want the corresponding layout device for a symbol in the schematic to appear in the layout, you can exclude these symbols by setting the ignore property for the symbol. You can also set the lvsIgnore property for layout devices if you do not want Virtuoso XL to check them. The ignore property can be set in either the Edit Properties form or the Edit Component CDF form.
**lxAutoAbut**

**Important**

`lxAutoAbut` is only a CDF property.

If either instance of an abutment pair has the `lxAutoAbut` property set to `nil`, the pair are prevented from abutting.

**Form Fields**

**Table A-3  In the Add CDF Parameter Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Add After Parameter</code></td>
<td><code>As First Parameter</code></td>
</tr>
<tr>
<td><code>paramType</code></td>
<td><code>boolean</code></td>
</tr>
<tr>
<td><code>name</code></td>
<td><code>ignore</code></td>
</tr>
<tr>
<td><code>prompt</code></td>
<td><code>N/A</code></td>
</tr>
<tr>
<td><code>defValue</code></td>
<td><code>t, nil</code></td>
</tr>
<tr>
<td><code>use</code></td>
<td><code>N/A</code></td>
</tr>
<tr>
<td><code>display</code></td>
<td><code>N/A</code></td>
</tr>
</tbody>
</table>
**lxAutoSpace**

**Note:** `lxAutoSpace` is only a CDF property.

If any instance of the abutment pair has the CDF property `lxAutoSpace` set to nil, then spacing will not happen for the pair.

**Form Fields**

**Table A-4 In the Add CDF Parameter Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>boolean</td>
</tr>
<tr>
<td>name</td>
<td>ignore</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td><code>t, nil</code></td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>
**IxBlockExtractCheck**

`IxBlockExtractCheck boolean { t | nil }

Prevents the extractor from checking the connectivity between a shape or instance and other shapes or instances it touches.

You can attach the `IxBlockExtractCheck` property on a shape or instance master. Use this property to prevent extraction of instances of masters that are not compatible with Layout XL or which would adversely affect the performance of the extractor.

---

**Layout instance Add Property form**

<table>
<thead>
<tr>
<th>Name</th>
<th><code>IxAutoSpace</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td><code>Boolean</code></td>
</tr>
<tr>
<td>Value</td>
<td>`t</td>
</tr>
</tbody>
</table>

---

**Example**

![Diagram](image)

In the illustration above, the layer that crosses net1 and net2 has the property `IxBlockExtractCheck`, so there is no connectivity checking between this shape and all others it touches. If the shape does not already have connectivity, the software assigns a net name to it.

If you assign connectivity to the shape manually or using Cadence® SKILL code, it will keep the connectivity, but the shape will always be considered incomplete because the software will not check it when checking the connectivity.

For hierarchical connectivity to work for pcells, the connectivity must be defined on the shapes within the pcell. You can assign connectivity interactively using the `Connectivity – Assign Nets` command or using SKILL functions.
**lxBlockOverlapCheck**

Specifies that the extractor not check whether a nonoverlap layer of this shape or instance is touching a nonoverlap layer at the current cellview level. Defined on a shape, instance, or instance master.

**Form Fields**

**Table A-5 In the layout Add Property Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Name</em></td>
<td>lxBlockExtractCheck</td>
</tr>
<tr>
<td><em>Type</em></td>
<td>boolean</td>
</tr>
<tr>
<td><em>Value</em></td>
<td><em>t, nil</em></td>
</tr>
<tr>
<td><em>Choices</em></td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Additional Information**

In the illustration below, the *lxBlockOverlapCheck* property can be defined on the diffusion layer so that no short marker is created.

![Diagram](image)

A: extra layer by mistake

For example, if you define the *lxBlockOverlapCheck* property on a shape in cellview “A”, you will not see any markers when overlapping the shape with an instance. However, if cellview “A” is placed inside cellview “B”, and a *noOverlap* layer is drawn over the shape in cellview “A”, then a marker will appear. This happens even though the shape in cellview “A” has the *lxBlockOverlapCheck* property defined.
lxCombination

lxCombination string t_expression

Builds a complex set of devices comprising series (sfactor) and parallel (mfactor) connections.

Use lxCombination to define a set of resistors to help achieve maximum density by putting the resistors into spaces otherwise not utilized. This facilitates the creation of a close approximation of the resistance value as defined in the schematic, without violating the manufacturing grid.

Define lxCombination as a CDF parameter or an instance property, using a valid string value.

Symbol master Add Property form

<table>
<thead>
<tr>
<th>Name</th>
<th>lxCombination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value</td>
<td>t_expression</td>
</tr>
</tbody>
</table>

Add CDF Parameter form

<table>
<thead>
<tr>
<th>Add After Parameter</th>
<th>As First Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>paramType</td>
<td>string</td>
</tr>
<tr>
<td>name</td>
<td>lxCombination</td>
</tr>
<tr>
<td>defValue</td>
<td>t_expression</td>
</tr>
</tbody>
</table>

Notes

- You can also use lxCombination to define the topology for a network of capacitors. However, in this case, the property describes only the topology of the network; the values entered do not equate to the total capacitance of the network.
- To ensure that lxCombination expressions are evaluated correctly, you must set the lxEvalCDFCallbacks environment variable to t before generating your resistors.
- If the lxCombination property is defined on the schematic symbol, the r parameter is ignored by the Connectivity – Generate – All From Source and Connectivity –
Update – Components And Nets commands. The layout instances will have the correct parameters based on the lxCombination factor setting.

Naming Convention

Devices defined using the lxCombination property in the schematic are named instName.msinteger in the layout. For example, if you have a schematic instance named R0 with

lxCombination = 2*10K + 1*100K

the layout instances generated are named as follows.

<table>
<thead>
<tr>
<th>Schematic</th>
<th>Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R0.ms1</td>
</tr>
<tr>
<td></td>
<td>R0.ms2</td>
</tr>
<tr>
<td></td>
<td>R0.ms3</td>
</tr>
</tbody>
</table>

Example

The following lxCombination expression divides a single resistor in the schematic into a complex set of devices in the layout

(2k*2)/3+1500/4

where

2k, 2, 3, 1500, and 4 are device parameter values

* and + are series connections

/ is a parallel connection

Note: The difference between the series connection symbols is that + can be followed by an expression, whereas * can be followed only by an integer. The / parallel connection can also be followed only by an integer.
In the schematic cellview, a resistor with the name A and a resistance of 3k is connected to nets net1 and net2.

A resistance property is defined on this instance with a value of 3k. During layout generation, Layout XL reads the lxCombination property and ignores the 3k value. A total of 10 resistors are generated in a combination of series and parallel connections, based on the property value. In this example, there are two 2k resistors connected in a series, with each series connected in parallel three times. Also connected in series to the 2k resistors are four parallel connected 1.5k resistors.
**lxIgnoredParams**

Specify device parameters to be excluded when using the *Connectivity – Check – Against Source* command or transferred to the layout when using the *Connectivity – Update – Layout Parameters* command.

### Tip

Cadence recommends that you use `lxParamsToIgnore` instead of `lxIgnoredParams`. However, you can continue to use `lxIgnoredParams` by setting the `checkOldIgnoredParamsProps` environment variable to t.

### Form Fields

Table A-6 In the Symbol Master Add Property Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td><code>lxIgnoredParams</code></td>
</tr>
<tr>
<td>Type</td>
<td><code>string</code></td>
</tr>
<tr>
<td>Value</td>
<td><code>parameter name</code></td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table A-7 In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td><code>string</code></td>
</tr>
<tr>
<td>name</td>
<td><code>lxIgnoredParams</code></td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td><code>parameter name</code></td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>
lxIgnoreParamsForCAS

Specifies device parameters and properties to be ignored by the Connectivity – Check – Against Source command.

Tip

Cadence recommends that you use lxParamsToIgnoreForCheck instead of lxIgnoreParamsForCAS. However, you can continue to use lxIgnoreParamsForCAS by setting set the checkOldIgnoredParamsProps environment variable to t.

Form Fields

Table A-8 In the Symbol Master Add Property Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>lxIgnoreParamsForCAS</td>
</tr>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value</td>
<td>parameter/property name</td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table A-9 In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>string</td>
</tr>
<tr>
<td>name</td>
<td>lxIgnoreParamsForCAS</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>parameter/property name</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>
**lxMfactorSplit**

Controls whether Virtuoso XL places a schematic device with the `mfactor` property as multiple devices in the layout.

Use this property to override the global value set using the `mfactorSplit` environment variable.

**Form Fields**

**Table A-10 In the Symbol Master Add Property Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td><code>lxMfactorSplit</code></td>
</tr>
<tr>
<td>Type</td>
<td>boolean</td>
</tr>
<tr>
<td>Value</td>
<td><code>t, nil</code></td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Table A-11 In the Add CDF Parameter Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Add After Parameter</code></td>
<td><code>As First Parameter</code></td>
</tr>
<tr>
<td><code>paramType</code></td>
<td>boolean</td>
</tr>
<tr>
<td><code>name</code></td>
<td><code>lxMfactorSplit</code></td>
</tr>
<tr>
<td><code>prompt</code></td>
<td>N/A</td>
</tr>
<tr>
<td><code>defValue</code></td>
<td><code>t, nil</code></td>
</tr>
<tr>
<td><code>use</code></td>
<td>N/A</td>
</tr>
<tr>
<td><code>display</code></td>
<td>N/A</td>
</tr>
</tbody>
</table>
**lxNetNamePrefix**

Specify the `lxNetNamePrefix` property to add unique prefixes to the internal nets.

**Form Fields**

**Table A-12 In the Symbol Master Add Property Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td><code>lxNetNamePrefix</code></td>
</tr>
<tr>
<td>Type</td>
<td><code>string</code></td>
</tr>
<tr>
<td>Value</td>
<td><code>unique prefix</code></td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Table A-13 In the Add CDF Parameter Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td><code>As First Parameter</code></td>
</tr>
<tr>
<td>paramType</td>
<td><code>string</code></td>
</tr>
<tr>
<td>name</td>
<td><code>lxNetNamePrefix</code></td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td><code>unique prefix</code></td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Additional Information**

The names of the internal nets are generated in the layout using Virtuoso XL conventions. You can use the `lxNetNamePrefix` property to add unique prefixes to the internal nets (valid values: string).

**Note:** `lxNetNamePrefix` can only be used with devices which have the `lxCombination` property or `sfactor` parameter.
For example, generating layout devices from the R device in the schematic adds |R|net.ms1, |R|net.ms2, and |R|net.ms3 names to the internal nets.

Using the lxNetNamePrefix property, you can choose to add the prefix X to the internal net names. The names change to |R|XnetMS1, |R|XnetMS2, and |R|XnetMS3.
**lxParamsToIgnore**

Specifies device parameters to be ignored when using the following Virtuoso XL update and check commands.

- *Design – Gen From Source*
- *Create – Pick From Schematic*
- *Create – Clone*
- *Connectivity – Check – Against Source*
- *Connectivity – Update – Components and Nets*
- *Connectivity – Update – Layout Parameters*
- *Connectivity – Update – Schematic Parameters*

\[Important\]

Cadence recommends that you use `lxParamsToIgnore` instead of the `lxIgnoredParams` property.

**Form Fields**

**Table A-14 In the Symbol Master Add Property Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td><code>lxParamsToIgnore</code></td>
</tr>
<tr>
<td>Type</td>
<td><code>string</code></td>
</tr>
<tr>
<td>Value</td>
<td><code>parameter/property name</code></td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Table A-15 In the Add CDF Parameter Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Add After Parameter</code></td>
<td>As First Parameter</td>
</tr>
<tr>
<td><code>paramType</code></td>
<td><code>string</code></td>
</tr>
</tbody>
</table>
### Table A-15  In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>lxParamsToIgnore</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>parameter/property name</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>
lxParamsToIgnoreForCheck

Specifies device parameters and properties to be ignored by the Connectivity – Check – Against Source command.

⚠️ Important

Cadence recommends that you use `lxParamsToIgnoreForCheck` instead of the `lxIgnoreParamsForCAS` property.

Form Fields

### Table A-16 In the Symbol Master Add Property Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td><code>lxParamsToIgnoreForCheck</code></td>
</tr>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value</td>
<td><code>parameter/property name</code></td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Table A-17 In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>string</td>
</tr>
<tr>
<td>name</td>
<td><code>lxParamsToIgnoreForCheck</code></td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td><code>parameter/property name</code></td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>
lxRemoveDevice

Causes parasitic devices to be ignored.

Use lxRemoveDevice to merge nets connected to the terminals of a single instance at a time. Its use is not restricted to only two terminals; it also works for more than two terminals.

You can also use the property to merge nets connected to the terminals of an iterated instance. To do so, add the property to the instance being iterated, not to the hierarchical block containing the iterated instances.

Form Fields

Table A-18  In the Symbol Master Add Property Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>lxRemoveDevice</td>
</tr>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value</td>
<td>“(short(list_of_terminals)...[function({arguments}])”</td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table A-19  In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>string</td>
</tr>
<tr>
<td>name</td>
<td>lxRemoveDevice</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>“(short(list_of_terminals)...[function({arguments}])”</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Additional Information

⚠️ **Caution**

*If the Diva LVS `removeDevice` rule and the `lxRemoveDevice` property are set on the same instance or master, running Diva LVS causes an error.*

**Rules for lxRemoveDevice Merging**

The rules below are used to decide which net survives after the device is shorted.

1. Global nets or nets connected to I/O pins always survive.
2. If there are no global or external nets, the net at a higher level of hierarchy survives; for example, `net1` survives over `/I1/net2`.
3. If the nets are at the same level of hierarchy, the net with the shortest name survives; for example, `netA` survives over `netBCD`.
4. If the nets are at the same level of hierarchy and their names are of the same length, alphabetical order is used; for example, `netM` survives over `netN`.

**Rules for lxRemoveDevice Handling**

1. A single function applies for all the `short()` lists.
2. If the function is missing, the terminals are shorted as specified in the short rule.
3. If the device is not removed, the terminals are not shorted.
4. A warning is issued when the device is not ignored in the netlisting.

**Example**

For a resistor with terminal names PLUS and MINUS, a simple `lxRemoveDevice` property would be as follows.

- `(short(PLUS MINUS))`
  
  This property shorts the terminals.

- `(short(PLUS MINUS) funcR(r))`
  
  Here, the resistor is shorted only if the user-defined SKILL function `funcR` returns non-nil.
Important

The arguments to the function must be instance or cell parameter names. Virtuoso XL replaces the names with the evaluated values of the properties when calling the function.

Note: The initial and final parentheses must be included.

The function must already be defined. The recommended place to define this function is in the .cdsinit or libInit.il file. A sample funcR for the above case would be

```il
procedure( funcR(r)
    if(r<100 then t
    else nil)
)
```
lxRounding

Defines how the value of a specified parameter on a cell master or schematic instance is rounded when it is evaluated by Virtuoso XL.

You can use lxRounding with the lxDeviceWidth parameter to prevent folded devices from becoming off-grid.

You can also use it to set a tolerance when Virtuoso XL compares parameter values during the following commands.

- **Design – Gen From Source**
- **Connectivity – Check – Against Source**
- **Connectivity – Update – Layout Parameters**
- **Connectivity – Update – Schematic Parameters**

The definition comprises three parts; 

\[(\text{paramName float keyword})\].

- **paramName** is the name of the parameter to be rounded. Valid names are
  - Those specified by the mfactorSplitParamNames environment variable
  - Those specified by the sfactorParam environment variable
  - The width parameter name specified in the component type to which the cellview is assigned.

- **float** is a number. If the specified parameter is to be rounded, the value returned is a nonzero integer multiple of the float value.

  **Note:** Virtuoso XL never rounds to zero. If the closest rounded value is zero, no rounding takes place.

- **keyword** specifies the rounding method used. Allowed values are
  - floor rounds to the nonzero integer multiple at or below the current value
  - ceil rounds to the nonzero integer multiple at or above the current value
  - round rounds to the nonzero integer multiple at or closest to the current value

For example, \(\text{lxRounding} = (w \ 0.05 \ \text{round})\) rounds the value of \(w\) to the closest multiple of 0.05 nanometers.
Form Fields

Table A-20  In the Symbol Master Add Property Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>lxRounding</td>
</tr>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value</td>
<td>(paramName float keyword)</td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table A-21  In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>string</td>
</tr>
<tr>
<td>name</td>
<td>lxRounding</td>
</tr>
<tr>
<td>prompt</td>
<td>lxRounding</td>
</tr>
<tr>
<td>defValue</td>
<td>(paramName float keyword)</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>
lxSeriesTerms

Understands the connectivity of the devices with sfactor greater than 1 and instance terminals greater than 2.

Form Fields

Table A-22 In the Symbol Master Add Property Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>lxSeriesTerms</td>
</tr>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value</td>
<td>“PLUS MINUS”</td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table A-23 In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>string</td>
</tr>
<tr>
<td>name</td>
<td>lxSeriesTerms</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>“PLUS MINUS”</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Additional Information

The lxSeriesTerms property must be defined on the instance in the schematic in order for Virtuoso XL to understand the connectivity of the devices with sfactor >1 and instance terminals>2. The description below shows setting the lxSeriesTerms PLUS and MINUS on two instance terminals.

\[ \text{lxSeriesTerms} = \text{“PLUS MINUS”} \]

The Generate Multiple Instances option must be turned on in the Layout XL Options form.
**lxStickyNet**

Allows a shape to retain the assigned connectivity when it is modified in some way.

Virtuoso XL adds the `lxStickyNet` property automatically to a shape when a net is assigned to that shape using one of the commands listed below.

- Create – Rectangle
- Create – Polygon
- Create – Path
- Create – Multipart Path
- Create – Conics – Circle
- Create – Conics – Ellipse
- Create – Conics – Donut
- Connectivity – Add Shape to Net

The property remains in effect until you turn it off or remove it using the Edit – Properties command. You can also manually add the property using the Edit – Properties command.

**Form Fields**

**Table A-24 In the layout Add Property Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>lxStickyNet</td>
</tr>
<tr>
<td>Type</td>
<td>boolean</td>
</tr>
<tr>
<td>Value</td>
<td>t, nil</td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Additional Information**

In the Virtuoso environment there are two types of nets, known and unknown.

- known nets are nets that are defined by pins and components (for example `clkA`).

  If a shape that 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 cannot be changed. However, zero level geometry (not instTerm connections) can be assigned and reassigned as you edit.

If shape on a known net touches another shape on a known net, it causes a short.

- unknown nets: shapes that are not assigned to a net
  
  If a shape touches nothing, it is considered a floating net.

When floating nets are assigned to a net, they are given the lxStickyNet property, which allows the shape to retain the assigned connectivity when it is modified in some way.

In this example, the nets of the device's instance terminals are known (component net assignments, netA, netB, and vdd). The unknown net is the power rail. The power rail is not connected to the device, and the extractor is free to assign and reassign the shape to a net. To assign a net to the power rail shape and not allow the extractor to reassign it to a different net, assign a net name, such as vdd, and the software adds the lxStickyNet automatically. The power rail retains this connectivity when it is modified. For example, if you create a connection between S netA and the power rail shape, the system displays a short marker.
**lxStopList**

Lists the names of views at which Virtuoso XL stops traversing the design. The stop view is generally the view with the most detailed description or the one with no further layers of hierarchy.

**Form Fields**

**Table A-25 In the Symbol Master Add Property Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>lxStopList</td>
</tr>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value</td>
<td>layout compacted symbolic</td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Table A-26 In the Add CDF Parameter Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>string</td>
</tr>
<tr>
<td>name</td>
<td>lxStopList</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>layout compacted symbolic</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Additional Information**

For more information on hierarchy expansion using the `lxStopList` and `lxViewList` properties, see “How the Netlister Expands Hierarchy” in Chapter 3 of the *Virtuoso Analog Design Environment User Guide*. 
Note: In releases 4.3.4 and earlier, the `maskLayoutViewName` property specified which view to use in the layout. Virtuoso XL still recognizes this property if you have not defined the `lxStopList` property.

After generating a layout, you can change the view using the layout window `Connectivity – Change Instance View` command. The Change Instance View form lets you choose a view from the list defined in the `lxStopList` property.
lxUseCell

Specifies which of several layout cells associated with a device symbol to use in generating a layout.

Form Fields

Table A-27 In the Symbol Master Add Property Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>lxUseCell</td>
</tr>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value</td>
<td>t_libName t_cellName</td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table A-28 In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>string</td>
</tr>
<tr>
<td>name</td>
<td>lxUseCell</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>t_cellName</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Additional Information

If a symbol does not have an lxUseCell property or the property is not assigned a value, Virtuoso XL defaults to the layout cellview of the same name stored in the same library as the schematic or symbol cellview.

If there is no such cellview, the software searches any libraries specified by the lxUseLibList environment variable. For more information, see “lxUseLibList” on page 542.
If there is already an instance of the cell with the expected name in the layout, the software uses that instance.

**Note:** Virtuoso XL does not pass CDF parameters that are not evaluated (are not AEL expressions), and have \texttt{cdfParamRec->storeDefault=\texttt{nil}}. These are master parameters and are expected to match for both schematic and layout master. Virtuoso XL does not pass them to the layout instance even when \texttt{lxUseCell} is used. If you require the parameters to be passed set \texttt{cdfParamRec->storeDefault} to \texttt{t} for the schematic master parameters.
Virtuoso XL Layout Editor User Guide
Properties

lxVcpSubContMaxSpacing

Specifies the maximum spacing after which the placer must place another standard cell substrate contact in order to avoid a violation.

You must specify this property (and the lxVcpSubContMinSpacing property) on the master cellview for the substrate contact.

Form Fields

Table A-29 In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>float</td>
</tr>
<tr>
<td>name</td>
<td>lxVcpSubContMaxSpacing</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>N/A</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Additional Information

A standard cell substrate contact is neither a contact nor a via; it is a standard cell with a component type of STDSUBCONT and two properties defined - lxVcpSubContMaxSpacing and lxVcpSubContMinSpacing - which define the maximum and minimum separation between two adjacent standard cell substrate contacts.

To define and place a standard cell substrate contact,

1. Copy a filler cell to use as the basis for the standard cell substrate contact.
2. Add substrate and well contacts in the cell. Whether these are oaVia, via instance, or polygons is immaterial.
3. Add the lxVcpSubContMaxSpacing and lxVcpSubContMinSpacing properties to the cell.
4. In the Edit Component Types form, define the cell as STDSUBCONT.
5. Run the placer with the *Insert Substrate Contacts* option switched on.

The placer fills any gaps between placed cells with the specified standard cell substrate contacts.

lxVcpSubContMaxSpacing is the spacing after which the placer must place another standard cell substrate contact in order to prevent a violation; 
 lxVcpSubContMinSpacing is the minimum spacing between two consecutive standard cell substrate contacts.
lxVcpSubContMinSpacing

Specifies the minimum spacing allowed between two adjacent standard cell substrate contacts.

You must specify this property (and the lxVcpSubContMaxSpacing property) on the master cellview for the substrate contact.

Form Fields

Table A-30 In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>float</td>
</tr>
<tr>
<td>name</td>
<td>lxVcpSubContMinSpacing</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>N/A</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Additional Information

A standard cell substrate contact is neither a contact nor a via; it is a standard cell with a component type of STDSUBCONT and two properties defined - lxVcpSubContMaxSpacing and lxVcpSubContMinSpacing - which define the maximum and minimum separation between two adjacent standard cell substrate contacts.

To define and place a standard cell substrate contact,

1. Copy a filler cell to use as the basis for the standard cell substrate contact.
2. Add substrate and well contacts in the cell. Whether these are oaVia, via instance, or polygons is immaterial.
3. Add the lxVcpSubContMaxSpacing and lxVcpSubContMinSpacing properties to the cell.
4. In the Edit Component Types form, define the cell as STDSUBCONT.
5. Run the placer with the *Insert Substrate Contacts* option switched on.

The placer fills any gaps between placed cells with the specified standard cell substrate contacts.

*lxVcpSubContMaxSpacing* is the spacing after which the placer must place another standard cell substrate contact in order to prevent a violation;
*lxVcpSubContMinSpacing* is the minimum spacing between two consecutive standard cell substrate contacts.
**lxViewList**

Creates a switch view list that tells Virtuoso XL where to look for the view name of a cell on the lower levels of a hierarchical design.

**Form Fields**

**Table A-31 In the Symbol Master Add Property Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>lxViewList</td>
</tr>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value</td>
<td>schematic netlist symbol layout compacted symbolic</td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Table A-32 In the Add CDF Parameter Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>string</td>
</tr>
<tr>
<td>name</td>
<td>lxViewList</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>schematic netlist symbol layout compacted symbolic</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Additional Information**

The following figure represents the schematic hierarchy and shows which layout views Virtuoso XL chooses under these conditions:

- lxStopList is set to layout layoutS
**Virtuoso XL Layout Editor User Guide**

**Properties**

- lxViewList is set to schematic cmos.sch

The chosen layout views are shown with thick-line boxes.

In this figure, as Virtuoso XL builds the layout, it looks at each symbol in the Top Cell.

The mux cell has no view listed in the stop list; therefore, Virtuoso XL goes to the view list and switches into the Mux schematic. All devices in the Mux schematic have views that are in the stop list; therefore, Virtuoso XL places those views.

The AND cell has a view listed in the stop list; therefore, Virtuoso XL places that view and never switches into the view list.
**mfactor**

Defines a one-to-many parallel relationship between a device in a schematic and multiple instances of the device in the layout.

**Form Fields**

**Table A-33 In the Symbol Master Add Property Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>mfactor</td>
</tr>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Table A-34 In the Add CDF Parameter Form**

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>string</td>
</tr>
<tr>
<td>name</td>
<td>mfactor</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>N/A</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Additional Information**

The multiplication factor (*mfactor*) is a parameter that defines a one-to-many parallel relationship between a device in a schematic and multiple instances of the device in the layout.
The \texttt{mfactor} property and \texttt{mfactorSplit} environment variable both work hierarchically. Any instance in the connectivity source hierarchy can have these properties, and the multiplicity is computed from the hierarchical path.

You can use the \texttt{mfactorSplit} environment variable to control whether the \texttt{mfactor} produces multiple layout devices or not. Set the \texttt{IxMfactorSplit} property on a given instance to override the global value set by the \texttt{mfactorSplit} environment variable.

\textbf{Note}: You cannot use multiplication factors with components that cannot be used in parallel, such as voltage sources.

\textbf{Naming Convention}

Devices specified using the \texttt{mfactor} property in the schematic (and those that are folded in the layout) are named \texttt{instName}\texttt{.integer} in the layout.

For example, if you have a schematic instance named \texttt{P0} with

\begin{verbatim}
  mfactor = 3
\end{verbatim}
the layout instances generated are named as follows.

<table>
<thead>
<tr>
<th>Schematic</th>
<th>Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>P0.1</td>
</tr>
<tr>
<td></td>
<td>P0.2</td>
</tr>
<tr>
<td></td>
<td>P0.3</td>
</tr>
</tbody>
</table>
permuteRule

Makes the instance pins or terminals of a device permutable. The syntax is the same as is used by the Assura verification tools.

- \((p \ E1 \ E2)\) specifies that \(E1\) and \(E2\) are permutable.
- \((f \ E1 \ E2)\) specifies that \(E1\) and \(E2\) are fixed (i.e., not permutable).

All pins that are not listed in the permuteRule property for a device are considered fixed. If you define the permuteRule property on a schematic symbol, that setting overrides any other rule defined in the layout for that instance.

You define the permuteRule in the Edit Component CDF form in either the Component Parameters section or the Simulation Information section. If it is defined in both sections, Virtuoso XL uses the definition in the Component Parameters section.

Tip

If you want Virtuoso XL to permute pins automatically, switch on the Auto Permute Pins option in the Layout XL Options form. If this option is switched off, you can only permute pins manually.

Form Fields

### Table A-35 In the Symbol Master Add Property Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>permuteRule</td>
</tr>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Value</td>
<td>((p \ 1 \ 2))</td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Table A-36 In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>string</td>
</tr>
</tbody>
</table>
Table A-36 In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>permuteRule</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>((p \ 1 \ 2))</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Examples

- To specify that pins E1 and E2 are permutable, use the following `permuteRule`.
  \((p \ E1 \ E2)\)

- To specify that pins E1, E2, and E3 are permutable, use the following `permuteRule`.
  \((p \ E1 \ E2 \ E3)\)

- You can also specify a hierarchical `permuteRule`. For example, the value below defines two sets of permutable pins (E1, E2, E3 and E4, E5, E6) and specifies that the two sets are permutable with each other.
The following `permuteRule` value defines two sets of permutable pins (E1, E2, E3 and E4, E5, E6) and specifies that the two sets are not permutable with each other.

(f (p E1 E2 E3) (p E4 E5 E6))

**Macros**

To avoid entering long lists of permutable pins for the `permuteRule` property, you can use the following macro notations.

- (f ALL), which means that all of the pins in the cell are fixed
- (p ALL), which means that all of the pins in the cell are permutable
- Range indications, where pins are numbered or sorted as in bus notations.

For example, the notation

(p A<0:3> B<4:7>)

is equivalent to

(p A<0> A<1> A<2> A<3> B<4> B<5> B<6> B<7>)

The notation

(p (f A<0:3>) (f B<4:7>))

defines two sets of fixed pins (A<0:3> and B<4:7>) which are permutable with each other.

You can also express range indications in descending order

(p A<3:0> B<7:4>)
sfactor

Defines a one-to-many relationship between a device in a schematic and multiple series-connected instances in the layout.

Form Fields

Table A-37 In the Symbol Master Add Property Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>sfactor</td>
</tr>
<tr>
<td>Type</td>
<td>integer, string, floating point number, or expression,</td>
</tr>
<tr>
<td>Value</td>
<td>user defined</td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table A-38 In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>integer, string, floating point number, or expression</td>
</tr>
<tr>
<td>name</td>
<td>sfactor</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>user defined</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Additional Information

The series-connection factor (sfactor) is a parameter that can be used to define a one-to-many relationship between a device in a schematic and multiple instances in series connection in the layout. The sfactor property can be applied to two or three terminal resistors, capacitors, and inductors. You set the value of the sfactor property (called s or S, unless you change it in the Layout XL Options form) to the number of layout devices you
want to generate. When you run the *Gen from Source* or *Update Components and Nets* command, Virtuoso XL generates the number of layout devices specified as the property value. It creates all the devices identical in size.

The *sfactor* can be a property only at the layout stop level. Any property above or below the layout stop level is ignored.

You can use an environment variable, *sfactorParam*, to tell Virtuoso XL which parameter of the device is to be split. The default names of the parameters to be split are “r R” for resistor, “C c” for capacitor, and “l L” for inductor. You can set this environment variable from the *Layout XL Options* form.

**Note:** If both *mfactor* and *sfactor* are defined on the same instance, *sfactor* is ignored.

The *lxSeriesTerms* property must be defined on the instance in the schematic in order for Virtuoso XL to understand the connectivity of the devices with *sfactor* >1 and instance terminals>2.

**Naming Convention**

Devices specified using the *sfactor* property in the schematic are named *instName.sinteger* in the layout.

For example, if you have a schematic instance named *R0* with

```
    sfactor = 3
```
the layout instances generated are named as follows.

<table>
<thead>
<tr>
<th>Schematic</th>
<th>Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R0.s1</td>
</tr>
<tr>
<td></td>
<td>R0.s2</td>
</tr>
<tr>
<td></td>
<td>R0.s3</td>
</tr>
</tbody>
</table>
vxlInstSpacingDir

Assigns automatic spacing properties to pins of instances. If these pins are on different nets or the pins cannot abut for any reason (for example, they are not assigned abutment properties), the software automatically separates the instances by the distance and in the direction you specify.

Form Fields

Table A-39 In the Symbol Master Add Property Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>vxlInstSpacingDir</td>
</tr>
<tr>
<td>Type</td>
<td>list</td>
</tr>
<tr>
<td>Value</td>
<td>left right top bottom</td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table A-40 In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>list</td>
</tr>
<tr>
<td>name</td>
<td>vxlInstSpacingDir</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>left right top bottom</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Additional Information

If you want automatic spacing to apply to Cadence® SKILL language or technology file parameterized cells (pcells), you can enter these properties in SKILL to add the spacing properties to the pins.
If you want automatic spacing to apply to standard cells, you can also use the Virtuoso layout editor to enter properties on the pins.

If the `vxlInstSpacingRule` value is different for two pins affected by *Auto Space*, the larger of the two values is used; if the value is defined for only one of two pins affected by *Auto Space*, the defined value is used.

To activate automatic spacing, turn on *Auto Space* in the *Layout XL Options* form (*on* is the default).

You can use the Virtuoso layout editor user interface to enter properties on the pins in the same way that you enter the `permuteRule` properties.

To add the access direction to the pin:

<table>
<thead>
<tr>
<th>property</th>
<th>value type</th>
<th>valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vxlInstSpacingDir</code></td>
<td>skill list</td>
<td>one or more of the following strings in the list: &quot;left&quot; &quot;right&quot; &quot;top&quot; &quot;bottom&quot;</td>
</tr>
</tbody>
</table>
vxlInstSpacingRule

Defines a spacing offset when pins are on a different net.

Form Fields

Table A-41  In the Symbol Master Add Property Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>vxlInstSpacingRule</td>
</tr>
<tr>
<td>Type</td>
<td>float</td>
</tr>
<tr>
<td>Value</td>
<td>number greater than 0</td>
</tr>
<tr>
<td>Choices</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table A-42  In the Add CDF Parameter Form

<table>
<thead>
<tr>
<th>Form Field</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add After Parameter</td>
<td>As First Parameter</td>
</tr>
<tr>
<td>paramType</td>
<td>float</td>
</tr>
<tr>
<td>name</td>
<td>vxlInstSpacingRule</td>
</tr>
<tr>
<td>prompt</td>
<td>N/A</td>
</tr>
<tr>
<td>defValue</td>
<td>number greater than 0</td>
</tr>
<tr>
<td>use</td>
<td>N/A</td>
</tr>
<tr>
<td>display</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Additional Information

A user defined function, called abutFunction, can be used to define a spacing offset when pins are on a different net. When you set the vxlInstSpacingRule to float, two pins of the same abutment class and different nets are pushed apart a distance equal to the maximum values of the two vxlInstSpacingRule pins. To control the abutment spacing set the vxlInstSpacingRule to string abutFunction.

Note: The abutFunction is not the name of your custom abutment function, but is the property that is set to the name of your abutment function.
You can write your own spacing function for the two instances involved. For example, `myCustomSpacingFunc()` is to be called in the body of the user defined abutment function `abutFunction`. The `myCustomSpacingFunc()` must return a float value to be effective.

There are two numbers passed to the abutment function, the connection condition (1, 2, or 3) and the abutment event (1, 2, 3, or 4). The connection condition (3) is for different nets. The sequence for nonabuttable pins should be the same as for abuttable. The calling of the abutment function for adjusting the pcells (event 2) and then one to calculate the spacing of the resultant reference edges (event 4). This allows you to adjust the well or isolation on the pcell and then space the devices based on the reference edges.

To get the spacing offset when pins are on different nets add the following case in your custom abutment function.

```plaintext
(4
  if(instA->w > 3.0 && instB->w > 3.0
    then
      result = 3
    else
      result = 1
  )
);; end of case 4
```

You can use the layout editor user interface to enter properties on the pins in the same way that you enter the `permuteRule` properties.

If you need to add a variable spacing rule to the pin, set the `vxlInstSpacingRule` to `abutFunction` as shown below;

```plaintext
dbReplaceProp(sObj->dbId "vxlInstSpacingRule" "string" "abutFunction")
```

The properties to set automatic spacing are listed below.

To add the spacing rule to the pin:

<table>
<thead>
<tr>
<th>property</th>
<th>value type</th>
<th>valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>vxlInstSpacingRule</td>
<td>floating</td>
<td>number greater than 0</td>
</tr>
<tr>
<td></td>
<td>pt number</td>
<td></td>
</tr>
</tbody>
</table>