

**cadence**<sup>TM</sup>

---

# **Encounter<sup>®</sup> User Guide**

**Product Version 8.1.3  
September 2009**

---

© 2002-2009 Cadence Design Systems, Inc. All rights reserved.  
Printed in the United States of America.

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

**Trademarks:** Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 1-800-862-4522.

All other trademarks are the property of their respective holders.

**Restricted Print Permission:** This publication is protected by copyright and any unauthorized use of this publication may violate copyright, trademark, and other laws. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. This statement grants you permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used solely for personal, informational, and noncommercial purposes;
2. The publication may not be modified in any way;
3. Any copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement; and
4. Cadence reserves the right to revoke this authorization at any time, and any such use shall be discontinued immediately upon written notice from Cadence.

**Disclaimer:** Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

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

# **Contents**

---

<b>About This Manual</b>	37
<u>Audience</u>	37
<u>How This Manual Is Organized</u>	37
<u>Conventions Used in This Manual</u>	38
<u>Related Documents</u>	39
<u>Encounter Foundation Flows Documentation</u>	39
<u>Encounter Product Documentation</u>	40
 <b>1</b>	
<b>Product and Licensing Information</b>	43
<u>Overview</u>	44
<u>About Encounter Products</u>	44
<u>Encounter Digital Implementation System</u>	44
<u>SoC Encounter RTL-to-GDSII System</u>	48
<u>About Encounter Licenses</u>	50
<u>Licensing Terminology</u>	50
<u>Checking Out Licenses for Product Options</u>	51
<u>Checking Out Dynamic Licenses for Product Options</u>	51
 <b>2</b>	
<b>Getting Started</b>	53
<u>Product and Installation Information</u>	54
<u>Setting the Run-Time Environment</u>	54
<u>Supported and Compatible Platforms</u>	54
<u>Specifying the 64-Bit or 32-Bit Version of Encounter Applications</u>	54
<u>Configuring OpenAccess</u>	55
<u>Launching the Console</u>	56
<u>Completing Command Names</u>	56
<u>Command-Line Editing</u>	57
<u>Control (^) Characters</u>	57

## Encounter User Guide

---

<u>Escape Sequences</u>	58
<u>Setting Preferences</u>	59
<u>Initialization Files</u>	60
<u>Starting the Software</u>	62
<u>velocity</u>	63
<u>encounter</u>	68
<u>Using Generic Parameters to Specify 32- or 64-Bit Version</u>	72
<u>Interrupting the Software</u>	73
<u>Interrupting the NanoRoute Router and Timing Optimization (optDesign)</u>	73
<u>Interrupting the Execution of Batch Files</u>	74
<u>Stopping the Software</u>	74
<u>Using the Log File Viewer</u>	75
<u>Integrated Log File Viewer</u>	75
<u>Standalone Log File Viewer</u>	76
<u>Accessing Documentation and Help</u>	77
<u>Launching Cadence Help From the Command Prompt</u>	77
<u>Accessing Documentation and Help From the Encounter GUI</u>	77
<u>Using the Encounter man and help Commands on the Text Command Line</u>	79
<u>Using the Integrated Log File Viewer</u>	81
<u>Other Sources of Information</u>	81

## 3

### Accelerating the Design Process By Using Multiple-CPU Processing

<u>Overview</u>	84
<u>Running Distributed Processing</u>	86
<u>Running Multi-Threading</u>	87
<u>Running Superthreading</u>	88
<u>Setting and Changing the License Check-Out Order</u>	88
<u>Limiting the Multi-CPU License Search to Specific Products</u>	88
<u>Releasing Licenses Before the Session Ends</u>	89
<u>Controlling the Level of Usage Information in the Log File</u>	89
<u>Where to Find More Information on Multi-CPU Licensing</u>	89

### 4

<u>Data Preparation</u> .....	91
<u>Generating a Technology File</u> .....	92
<u>Creating Technology Information Using LEF</u> .....	92
<u>Creating Technology Information Using OpenAccess</u> .....	92
<u>Preparing Physical Libraries</u> .....	92
<u>Using LEF to Create Physical Libraries</u> .....	92
<u>Creating OpenAccess Physical Libraries</u> .....	93
<u>Unsupported LEF and DEF Syntax</u> .....	93
<u>Unsupported LEF 5.6 Syntax</u> .....	93
<u>Unsupported DEF 5.6 Syntax</u> .....	94
<u>Generating the I/O Assignment File</u> .....	96
<u>Creating an I/O Assignment File</u> .....	97
<u>Creating a Rule-Based I/O Assignment File</u> .....	107
<u>I/O Pad and Pin Assignment Examples</u> .....	108
<u>Performing Area I/O Placement</u> .....	111
<u>Preparing Timing Libraries</u> .....	114
<u>Encrypting Libraries</u> .....	114
<u>Preparing Stamp Models</u> .....	115
<u>Preparing Timing Constraints</u> .....	115
<u>Preparing Capacitance Tables</u> .....	116
<u>Preparing Data for Delay Calculation</u> .....	116
<u>Preparing Data for Crosstalk Analysis</u> .....	116
<u>Checking Designs</u> .....	116
<u>Preparing Data in the Timing Closure Design Flow</u> .....	117

### 5

<u>Importing and Exporting Designs</u> .....	119
<u>Overview</u> .....	120
<u>Verifying Data before Importing a Design</u> .....	120
<u>Preparing the Design Netlist</u> .....	120
<u>Creating a Flat Verilog Netlist from a DEF File</u> .....	121
<u>Recommended DEF Import Commands</u> .....	121
<u>Reconciling the Object Names and Creating New DEF File That Can Be Used With the Normal Encounter Flows</u> .....	122

# Encounter User Guide

---

<u>Beginning Designs</u>	123
<u>Beginning a Design with LEF and Verilog</u>	123
<u>Beginning a Design with OpenAccess</u>	124
<u>Loading Previously Saved Configuration Files</u>	125
<u>Loading Configurations Files from the Command Line</u>	125
<u>Loading Configuration Files from the GUI</u>	126
<u>Selecting Files</u>	126
<u>Using Select Files</u>	127
<u>Working with OpenAccess Designs</u>	129
<u>Importing an OpenAccess Design</u>	129
<u>Saving an OpenAccess Design</u>	129
<u>Restoring an OpenAccess Design</u>	129
<u>Transferring OpenAccess Data between Encounter and Virtuoso Chip Editor for ECO</u>	130
<u>Removing Assign Nets from a Verilog Netlist</u>	130
<u>Saving and Restoring Designs</u>	131
<u>Saving Designs</u>	131
<u>Restoring Designs</u>	132
<u>Saving and Restoring OpenAccess Designs</u>	132
<u>Importing and Exporting Design Data</u>	132
<u>Loading a Partition</u>	132
<u>Loading Floorplan Data</u>	133
<u>Placement File Requirement</u>	134
<u>Loading an I/O Assignment File</u>	134
<u>Loading an FSDB File</u>	134
<u>Saving a Partition</u>	134
<u>Saving Floorplan Data</u>	134
<u>Importing and Exporting TDF Files</u>	135
<u>Importing Tile Cell Design Data</u>	136
<u>Converting an Encounter Database to GDSII Stream or OASIS Format</u>	137
<u>Creating Cells and Instances</u>	138
<u>Renaming LEF Vias</u>	138
<u>Merging GDSII Stream or OASIS Files</u>	139
<u>Merge Examples</u>	139
<u>About the GDSII Stream or OASIS Map File</u>	143
<u>Map File Format</u>	143

## Encounter User Guide

---

<u>Map File Columns</u>	144
<u>Specifying Object Subtypes</u>	147
<u>Using Multiple Layers and Data Types</u>	151
<u>Updating Files during an Encounter Session</u>	152
 <b>6</b>	
<u>Flip Chip Methodologies</u>	153
<u>Overview</u>	154
<u>Before You Begin</u>	154
<u>Flip Chip Flow in Encounter</u>	156
<u>Flip Chip Flow Steps</u>	157
<u>SiP Bump Flow</u>	161
<u>Reducing Data Size for SiP Import (Bypass Flow)</u>	161
<u>Splitting Wires in Metal Layers</u>	161
<u>Testing the Package Routing Feasibility</u>	162
<u>Area I/O Flow</u>	163
<u>Area I/O (AIO) Command Flow</u>	164
<u>Routing Bumps to I/O Driver Cells (Hierarchical Area I/O Flow)</u>	164
<u>Flip Chip Routing on Shielded Nets in AIO</u>	165
<u>Example</u>	165
<u>Peripheral I/O Flow</u>	167
<u>Data Preparation</u>	167
<u>Peripheral I/O Flow Steps</u>	168
<u>Peripheral I/O (PIO) Command Flow</u>	169
<u>RDL Planning and Routing</u>	171
<u>Peripheral I/O Extraction</u>	179
<u>SI and Timing Analysis</u>	180
<u>Differentiating Area I/O and Peripheral I/O</u>	181
<u>Point-To-Point Routing</u>	182
<u>Swapping Signals</u>	185
<u>Creating Differential Routing to Signal Bumps</u>	187
<u>Specify Routing Nets</u>	188
<u>Define Differential Pairs</u>	188
<u>Define Nets to Match Tolerance</u>	189
<u>Define a Shield Net</u>	189

## Encounter User Guide

---

<u>Route Multiple Nets with Different Widths</u>	190
<u>Route Nets with Tapering Pin Widths</u>	190
<u>Examples and Report Files</u>	192
<u>Routing and Placement Constraints</u>	192
<u>IO_FILE Example</u>	194
 <b>7</b>	
<b>Using ART in Hierarchical Designs</b>	197
<u>Overview</u>	198
<u>Types of Active Logic Views</u>	198
<u>Flat Top</u>	198
<u>Critical</u>	199
<u>Creating an Active Logic View</u>	200
<u>Example of Active Logic View Creation</u>	200
<u>Applications of ART</u>	200
<u>Timing Budgeting in Hierarchical Flow</u>	201
<u>Timing Optimization After Assembling the Post-Routed Partitioned Design</u>	201
 <b>8</b>	
<b>Using Interface Logic Models in Hierarchical Designs</b>	207
<u>Overview</u>	208
<u>Creating ILMs</u>	209
<u>Example ILM Creation</u>	210
<u>Preserving Selected Instances in ILMs</u>	211
<u>Creating ILMs for Shared Modules</u>	211
<u>Creating ILMs Without Using Encounter Database</u>	211
<u>Specifying ILM Directories at the Top Level</u>	213
<u>Example Top-Level Implementation Flow with ILMs</u>	213
<u>ILMs Supported in MMMC Analysis</u>	215
<u>ILMs Supported in SI</u>	217
<u>Interactive Use of ILMs</u>	217
<u>ILM Limitations</u>	218

### 9

<b>What-If Timing Analysis</b>	221
<b>Performing What-If Timing Analysis</b>	221
<u>Prerequisite</u>	222
<u>Timing Models Supported for What-If Timing Analysis</u>	222
<u>Using the What-If Timing Commands</u>	225

### 10

<b>Bus Planning</b>	229
<u>Overview</u>	230
<u>Bus Planning Flow in Encounter</u>	231
<u>Creating a Bus Guide</u>	232
<u>Using the Edit Bus Guide GUI</u>	232
<u>Using Text Commands</u>	237
<u>Example</u>	238
<u>Customizing the Bus Guide Display</u>	242
<u>Highlighting and Dehighlighting the Bus Guide</u>	242
<u>Saving and Restoring Bus Guide Information</u>	244
<u>Limitations of Bus Planning</u>	245

### 11

<b>Partitioning the Design</b>	247
<u>Overview</u>	248
<u>Flow Methodologies</u>	248
<u>Top-down Methodology</u>	249
<u>Bottom-up Methodology</u>	253
<u>Specifying Partitions and Blackboxes</u>	256
<u>Defining Partitions</u>	257
<u>Defining Partitions as Power Domains</u>	260
<u>Defining Blackboxes</u>	260
<u>Handling of Blackboxes with Non-R0 Orientation</u>	263
<u>Specifying Multiple Instantiated Partitions and Blackboxes</u>	265
<u>Changing Partition Clone Orientation</u>	266
<u>Specifying Rectilinear Partitions and Blackboxes</u>	267

## Encounter User Guide

---

<u>Specifying Core-to-I/O Distance for Partition Cuts</u>	268
<u>Specifying Nested Partitions</u>	269
<u>Assigning Pins</u>	269
<u>Assigning Partition and Blackbox Pins</u>	271
<u>Assigning I/O Pins</u>	293
<u>Performing Congestion-aware Pin Assignment for Channel-based Designs</u>	297
<u>Assigning Pins on Rectilinear Edges</u>	300
<u>Swapping Partition Pins</u>	301
<u>Pin Alignment</u>	301
<u>Snapping Pins to the Grid</u>	302
<u>Assigning Pins for Bus Guides</u>	303
<u>Pin Assignment Limitations</u>	304
<u>Inserting Feedthroughs</u>	305
<u>Inserting Feedthrough Buffers</u>	307
<u>Highlighting the Nets for which Feedthrough Buffers Have been Inserted</u>	318
<u>Utilizing Pre-defined Feedthrough Pins in Custom Macros</u>	318
<u>Inserting Routing Feedthroughs</u>	324
<u>Generating the Wire Crossing Report</u>	326
<u>Interpreting the Wire Crossing Report</u>	327
<u>Estimating the Routing Channel Width</u>	329
<u>Running the Partition Program</u>	331
<u>Pushing Down Signal Routes</u>	332
<u>How Top-level Stripes Are Pushed Down</u>	333
<u>How Bumps, Routes, and Area I/O Cells Are Affected</u>	336
<u>Limitations</u>	342
<u>Restoring the Top-Level Floorplan with Partition Data</u>	347
<u>Concatenating Netlist Files of a Partitioned Design</u>	348
<u>Saving Partitions</u>	349
<u>Loading Partitions</u>	349
<u>Unpartitioning with Routing Data</u>	349
<u>Working with OpenAccess Database</u>	351
<u>Parallel Job Processing</u>	352

## 12

<u>Floorplanning the Design</u> .....	353
<u>Overview</u> .....	354
<u>Common Floorplanning Sequence</u> .....	355
<u>Viewing the Floorplan</u> .....	357
<u>Module Constraint Types</u> .....	360
<u>Target Utilization Display</u> .....	361
<u>Effective Utilization Display</u> .....	363
<u>Calculating Density</u> .....	364
<u>Standard Row Spacing</u> .....	365
<u>Grouping Instances</u> .....	366
<u>Defining the Bounding Box</u> .....	367
<u>Adding Logical Hierarchy Without Creating Additional Hierarchy</u> .....	368
<u>Logical Hierarchy Manipulation</u> .....	369
<u>Creating and Editing Rows</u> .....	373
<u>Using Vertical Rows</u> .....	373
<u>Using Multiple-height Rows</u> .....	375
<u>Using Integer Multiple-height Rows</u> .....	375
<u>Using Non-Integer Multiple-height Rows</u> .....	377
<u>Working with User-defined DEF Files that Contain NIMH Rows or Unaligned Rows</u> .....	380
<u>Merging Hierarchical Floorplans from Partitions</u> .....	382
<u>Performing I/O Row Based Pad Placement</u> .....	387
<u>Prerequisites</u> .....	387
<u>Enabling the I/O Row Flow in Encounter</u> .....	388
<u>Use Models</u> .....	389
<u>Resizing Rectilinear Blocks</u> .....	392
<u>Use Models</u> .....	392
<u>Assumptions</u> .....	393
<u>Results</u> .....	394
<u>Using Blackblobs</u> .....	395
<u>Defining Blackblobs</u> .....	395
<u>Specifying Blackblobs</u> .....	396
<u>Blackblob Useflow</u> .....	400
<u>Blackblob Display</u> .....	403
<u>Blackblob Overlap</u> .....	408

## Encounter User Guide

---

<u>Saving and Restoring Blackblobs</u> .....	411
<u>Editing Pins</u> .....	412
<u>Pin Snapping on Resized Boundaries</u> .....	412
<u>Moving Pins</u> .....	412
<u>Swapping Pins</u> .....	413
<u>Using the Pin Editor</u> .....	413
<u>Running Relative Floorplanning</u> .....	423
<u>Orientation Key</u> .....	423
<u>Instance Place Example</u> .....	424
<u>Pre-Route Examples</u> .....	424
<u>Saving and Restoring Relative Floorplan</u> .....	426
<u>Saving and Loading Floorplan Data</u> .....	426
<u>Resizing the Floorplan</u> .....	427
<u>Resize Floorplan Options</u> .....	428
<u>Setting Resize Lines</u> .....	428
<u>Specifying Resize Directions</u> .....	429
<u>Snapping Resize Values</u> .....	429
<u>Viewing Resize Lines using Color Preferences</u> .....	430
<u>Distributing I/O's using Resize Floorplan</u> .....	432

## 13

<u>Power Planning and Routing</u> .....	435
<u>Overview</u> .....	436
<u>Before You Begin</u> .....	437
<u>Results</u> .....	438
<u>Loading, Saving, and Updating Special Route</u> .....	438
<u>Creating a Ring with User Defined Coordinates</u> .....	438
<u>Global Net Connections</u> .....	439
globalNetConnect Command and Connections for Signal Pins and Power/Ground Pins 440	
<u>Fixing LEF MINIMUMCUT Violations</u> .....	441
<u>Fixing LEF Minimum Spacing Violations</u> .....	441
<u>Adding Stripes to Power Domains</u> .....	441
<u>Automatic Power Planning (APP)</u> .....	443
<u>Creating a Template</u> .....	445

<u>Using the IP Block Page</u>	445
<u>Using the Design Page</u>	446
<u>Specifying Template Parameters</u>	447
<u>Instantiating a Template</u>	448
<u>Template Naming Conventions</u>	448
<u>Using the Synthesize Power Plan Functionality</u>	449
<u>Creating Differential Routing to Signal Bumps</u>	451
 <b>14</b>	
<b>Low Power Design</b>	453
<u>Overview</u>	454
<u>Power Domain Shutdown and Scaling</u>	454
<u>Support for the Common Power Format (CPF)</u>	456
<u>CPF Version Support</u>	456
<u>Encounter Commands Supporting CPF</u>	456
<u>Loading and Committing a CPF File</u>	457
<u>Saving a CPF Database</u>	457
<u>CPF Documentation</u>	458
<u>Multiple Supply Voltage Flat Flow</u>	459
<u>Preparing Data</u>	461
<u>Loading the Configuration File</u>	464
<u>Floorplanning the Design</u>	464
<u>Loading and Committing the CPF File</u>	465
<u>Setting the Power Domain Size</u>	465
<u>Setting the Power Domain mingap</u>	465
<u>Adding Power Switches</u>	466
<u>Verify Power Domains</u>	466
<u>Adding Well Tap Cells</u>	466
<u>Planning Power</u>	466
<u>Placing Standard Cells and Macros</u>	467
<u>Highlight Power Domains (Optional)</u>	469
<u>Adding Tie High/Low cells</u>	470
<u>Routing Power</u>	470
<u>Trial Routing</u>	471
<u>Optimizing Timing</u>	473

# Encounter User Guide

---

<u>Synthesizing Clock Trees</u>	476
<u>Optimizing Timing (Post CTS)</u>	477
<u>Routing the Design</u>	477
<u>Analyzing Timing</u>	477
<u>Analyzing Power</u>	477
<u>Optimizing Timing (Post-Route)</u>	478
<u>Multiple Supply Voltage Top-Down Hierarchical Flow</u>	479
<u>Overview</u>	479
<u>Always-On Feedthrough Handling</u>	480
<u>Chip Partitioning</u>	482
<u>Block-level CPF Generation</u>	482
<u>Top-Level CPF Generation</u>	484
<u>Block-Level Implementation</u>	485
<u>Top-Level Implementation</u>	485
<u>Chip Assembly</u>	485
<u>Example of Block-Level CPF Generated by Encounter</u>	487
<u>Example of Top-Level CPF Generated by Encounter</u>	490
<u>Multiple Supply Voltage Bottom-Up Hierarchical Flow</u>	494
<u>Block-Level Implementation</u>	495
<u>Top-Level Implementation</u>	496
<u>Chip Assembly</u>	496
<u>Leakage Power Optimization Techniques</u>	498
<u>Multi-Vth Optimization</u>	498
<u>Substrate Biasing</u>	499
<u>Power Shutdown Techniques</u>	503
<u>Power Shutdown Commands</u>	503
<u>Data Preparation</u>	504
<u>Buffer Styles</u>	505
<u>Adding Column Switches</u>	506
<u>Attaching the Acknowledge Receiver Pin</u>	507
<u>Enable Chaining</u>	509
<u>Controlling the Maximum Enable Chain Depth</u>	512
<u>Synthesizing Acknowledge Trees</u>	513
<u>Adding Power Switch Rings</u>	515
<u>Ring Conventions</u>	517
<u>Using Pitch Control and Offsets</u>	523

## Encounter User Guide

---

<u>Power Switch Optimization</u>	.....	533
<u>Power Switch Reduction</u>	.....	533
<u>Power Switch ECO</u>	.....	534

## 15

<u>Placing the Design</u>	.....	537
<u>Overview</u>	.....	538
<u>Loading a Design</u>	.....	538
<u>Preparing for Placement</u>	.....	538
<u>Guiding Placement With Blockages</u>	.....	539
<u>Placement Treatment of Preroutes</u>	.....	540
<u>Adding Well-Tap Cells</u>	.....	541
<u>Controlling the Distance Between Well-Tap Cells</u>	.....	541
<u>Adding Well-Tap Cells to MSV Designs</u>	.....	542
<u>Deleting Well-Tap Cells</u>	.....	542
<u>Adding End-Cap Cells</u>	.....	542
<u>Adding End Cap Cells to MSV Designs</u>	.....	543
<u>Deleting End-Cap Cells</u>	.....	543
<u>Placing Spare Cells and Spare Modules</u>	.....	543
<u>Placing Spare Cells That Are Included in the Netlist</u>	.....	543
<u>Placing Spare Cells That Are Not Included in the Netlist</u>	.....	544
<u>Spare Cell Placement Behavior</u>	.....	545
<u>Running Hierarchy-Aware Spare Cell Placement</u>	.....	547
<u>Adding Padding</u>	.....	550
<u>Adding Instance or Module Padding</u>	.....	551
<u>Adding Cell Padding</u>	.....	552
<u>Placing Standard Cells</u>	.....	554
<u>Running Placement in Multi-CPU Mode</u>	.....	555
<u>Multi-Threading Placement Steps</u>	.....	556
<u>Checking Placement</u>	.....	558
<u>Using the Amoeba View</u>	.....	559
<u>Using the Density Map</u>	.....	559
<u>Adding Filler Cells</u>	.....	560
<u>Adding Fillers to MSV Designs</u>	.....	560
<u>Deleting Filler Cells</u>	.....	561

## Encounter User Guide

---

<u>Placing Gate Array Style Filler Cells for Post-Mask ECO</u>	561
<u>Adding Decoupling Capacitance</u>	562
<u>Deleting Decoupling Capacitance</u>	563
<u>Adding Logical Tie-Off Cells</u>	563
<u>Saving Placement Data</u>	564
<u>Specifying and Placing JTAG and Other Cells Close to the I/Os</u>	564
<u>Optimizing and Reordering Scan Chains</u>	565
<u>Specifying Scan Cells</u>	565
<u>About Scan Chains</u>	566
<u>Reordering Scan Chains</u>	566

## 16

<u>Synthesizing Clock Trees</u>	577
<u>Before You Begin</u>	578
<u>Results</u>	578
<u>Understanding CTS Operation Modes</u>	579
<u>Manual CTS Mode</u>	579
<u>Automatic CTS Mode</u>	580
<u>How CTS Calculates Skew Values</u>	584
<u>Improving Postroute Correlation</u>	586
<u>Specifying Macro Model Delays</u>	587
<u>Dynamic Macro Model</u>	587
<u>Grouping Clocks</u>	590
<u>Analyzing Hierarchical Clock Trees</u>	591
<u>Module Placement Utilization</u>	593
<u>Clock Designs with Tight Area</u>	593
<u>Balancing Pins for Macro Models</u>	593
<u>Timing Model Requirement for Cells</u>	593
<u>Delay Variation and OCV</u>	593
<u>Understanding Post-CTS Clock Tree Optimization</u>	594
<u>Using the ckECO Command for Post-CTS Clock Tree Optimization</u>	594
<u>Support for Local Skew Optimization</u>	595
<u>Command Modes for the ckECO Command</u>	595
<u>Using a SPEF File with the ckECO Command for RC Estimation</u>	595
<u>Running Post-CTS Optimization with the ckECO Command</u>	596

## Encounter User Guide

---

<u>Guidelines for Using the ckECO Command</u>	597
<u>Creating a Clock Tree Specification File</u>	598
<u>Using the Automatic Clock Tree Specification File Generator</u>	598
<u>Example of a Clock Tree Specification File</u>	600
<u>Naming Attributes Section</u>	604
<u>NanoRoute Attribute Section</u>	605
<u>Macro Model Data Section</u>	606
<u>Clock Grouping Data Section</u>	610
<u>Clock-Tree Topology Section</u>	610
<u>Automatic Gated CTS Section</u>	610
<u>Log File Headings</u>	628
<u>CTS Report Descriptions</u>	629
<u>General Information</u>	629
<u>Macro Model Information</u>	631
<u>Power Information</u>	631
<u>AC Current Density Violations</u>	632
<u>Supported SDC Constraints</u>	633

## 17

<u>Working with Clock Mesh Structures</u>	635
<u>Overview</u>	636
<u>Clock Meshes Versus Clock Trees</u>	636
<u>Creating Clock Meshes</u>	639
<u>Determining the Mesh Structure</u>	639
<u>Implementing the Clock Mesh</u>	644
<u>Analyzing the Clock Mesh</u>	645

## 18

<u>Editing Wires</u>	649
<u>Overview</u>	651
<u>Before You Begin</u>	652
<u>Results</u>	652
<u>Using Keyboard Shortcuts</u>	652
<u>Keyboard Shortcuts That Open Forms</u>	652
<u>Keyboard Shortcuts That Are Equivalent to Tool Widgets</u>	652

## Encounter User Guide

---

<u>Keyboard Shortcuts Used in Auto Query Mode</u>	653
<u>Keyboard Shortcuts Used in Add Wire Mode</u>	654
<u>Keyboard Shortcuts Used in Stretch Wire Mode</u>	654
<u>Keyboard Shortcuts Used to Change Vias</u>	654
<u>Selecting Wires</u>	655
<u>Deleting Wires</u>	655
<u>Moving Wires</u>	655
<u>Using the Mouse to Move Wires</u>	655
<u>Using Arrow Keys to Move Wires</u>	656
<u>Moving Selected Wires or Vias</u>	657
<u>Adding Wires</u>	657
<u>Adding a Wire for a Single Net</u>	657
<u>Adding Wires for Multiple Nets</u>	659
<u>Adding Wires that Automatically Extend to a Target</u>	660
<u>Using Override to Add Wire Groups with Multiple Widths and Spacing</u>	661
<u>Cutting Shielding Wires</u>	662
<u>Trimming Antennas on Selected Stripes</u>	662
<u>Changing Wire Width</u>	663
<u>Repairing Maximum Wire Width Violations</u>	664
<u>Duplicating Wires</u>	664
<u>Stretching Wires</u>	665
<u>Changing Wire Layers</u>	665
<u>Splitting and Merging Wires</u>	666
<u>Adding Vias</u>	666
<u>Changing Vias</u>	667
<u>Moving Vias</u>	668
<u>Reshaping Routes</u>	668
<u>Controlling Cell Blockage Visibility</u>	669
<u>Editing Wires with Virtuoso Chip Assembly Router</u>	670
<u>Running Virtuoso CAR in Batch Mode</u>	670
<u>Running Virtuoso CAR in Interactive Mode</u>	671
<b>19</b>	
<u>Using Trial Route for Congestion and Timing Analysis</u>	673
<u>Overview</u>	674

## Encounter User Guide

---

<u>Data Preparation</u>	674
<u>Routing A Flat Design</u>	675
<u>Routing a Partitioned Design</u>	676
<u>Routing Two-Metal Layer Designs</u>	678
<u>Loading and Saving Route Data</u>	678
<u>Analyzing Route Data</u>	678
<u>Congestion Markers in the Display</u>	679
<u>Congestion Distribution Report</u>	681
<u>Improving Route Congestion</u>	687
<u>Using Bus Guides</u>	688
<u>Additional Information</u>	689
<u>Wire Overlap</u>	689

## 20

<u>Using the NanoRoute Router</u>	691
<u>About NanoRoute Routing Technology</u>	694
<u>Routing Phases</u>	694
<u>Global Routing</u>	694
<u>Detailed Routing</u>	695
<u>NanoRoute Router in the Encounter Flow</u>	696
<u>Before You Begin</u>	696
<u>Checking Your LEF Files</u>	696
<u>Checking for Problems with Cells, Pins, and Vias</u>	697
<u>Generating Tracks</u>	698
<u>Specifying Routing Layers</u>	698
<u>Interrupting Routing</u>	700
<u>Using the routeDesign Supercommand</u>	700
<u>Results</u>	702
<u>Use Models</u>	703
<u>Running the NanoRoute Router with Encounter Menu Commands and Forms</u>	703
<u>Running the NanoRoute Router with Encounter Text Commands</u>	703
<u>Running the NanoRoute Router in Standalone Mode</u>	704
<u>Using NanoRoute Parameters</u>	705
<u>Using Attributes and Options Together</u>	706
<u>Accelerating Routing with Multi-Threading and Superthreading</u>	708

## Encounter User Guide

---

<u>When to Accelerate Routing</u>	709
<u>Superthreading Log File Excerpts</u>	710
<u>Following a Basic Routing Strategy</u>	712
<u>Using the Encounter Text Commands</u>	712
<u>Using the Encounter GUI</u>	713
<u>Checking Congestion</u>	716
<u>Using the Congestion Analysis Table</u>	716
<u>Using the Congestion Map</u>	718
<u>Resolving Open Nets</u>	722
<u>Log File Examples</u>	722
<u>Diagnosing Problems Using verifyTracks</u>	723
<u>Resolving Additional Open Net Problems</u>	723
<u>Running Timing-Driven Routing</u>	725
<u>Input Files</u>	725
<u>Using the CTE and the NanoRoute Router in Native Mode</u>	725
<u>Using the CTE and Standalone NanoRoute</u>	726
<u>Routing Clocks</u>	728
<u>Setting Attributes for Clock Nets</u>	728
<u>Routing Clock Nets Using the GUI Forms</u>	729
<u>Running Postroute Optimization</u>	729
<u>Preventing and Repairing Crosstalk Problems</u>	730
<u>Crosstalk Prevention Options</u>	732
<u>Running ECO Routing</u>	734
<u>ECO Limitations</u>	734
<u>ECO Flow</u>	735
<u>Evaluating Violations</u>	736
<u>Violations on Upper Metal Layers</u>	740
<u>Violations in Timing-Driven Routing</u>	742
<u>Deleting Violated Nets</u>	743
<u>Using Additional Strategies to Repair Violations</u>	743
<u>Concurrent Routing and Multi-Cut Via Insertion</u>	744
<u>Postroute Via Optimization</u>	744
<u>Optimizing Vias in Selected Nets</u>	745
<u>Via Optimization Options</u>	746
<u>Performing Shielded Routing</u>	747
<u>Shielding Option</u>	747

## Encounter User Guide

---

<u>Performing Shielded Routing Using the GUI</u>	748
<u>Performing Shielded Routing Using Text Commands</u>	749
<u>Interpreting the Shielding Report</u>	749
<u>Routing Wide Wires</u>	750
<u>Using Non-Default Rules</u>	751
<u>Repairing Process Antenna Violations</u>	753
<u>Repairing Violations on Multiple-Pin Nets</u>	753
<u>Changing Layers</u>	754
<u>Using Diodes</u>	754
<u>Deleting and Rerouting Nets with Violations</u>	754
<u>Repairing Violations on Cut Layers</u>	754
<u>Process Antenna Options</u>	755
<u>Examples</u>	755
<u>Using a Design Flow that Includes Astro or Apollo</u>	757
<u>Troubleshooting</u>	759

## 21

<u>Using the Encounter Mixed Signal Router</u>	761
<u>Overview</u>	762
<u>Using the Mixed Signal Router</u>	763
<u>Before You Begin</u>	763
<u>Results</u>	763
<u>Specialized Routing Techniques</u>	764
<u>    Matched Nets</u>	764
<u>    Differential Pair Nets</u>	768
<u>    Bus Routes</u>	769
<u>    Shielded Nets</u>	769
<u>Using Routing Constraints</u>	773
<u>Constraint File Format</u>	773
<u>Generic Constraints</u>	774
<u>Specialized Constraints and Keyword Descriptions</u>	776
<u>    NETS</u>	776
<u>    MATCH</u>	778
<u>    DIFFPAIR</u>	780
<u>    BUS</u>	782

## Encounter User Guide

---

<u>SHIELDING</u>	784
<u>Creating a Constraint File</u>	797
<u>Using the Mixed Signal Constraint Editor</u>	797
<u>Using a Text Editor</u>	801
<u>Loading a Constraint File</u>	802
<u>Using the Mixed Signal Constraint Editor Form</u>	802
<u>Using the Mixed Signal Router Form</u>	802
<u>Using the routeMixedSignal Command</u>	804
<u>Editing a Constraint File</u>	805
<u>Using the Mixed Signal Constraint Editor</u>	805
<u>Using a Text Editor</u>	806
<u>Sample Constraint File</u>	807

## 22

<u>Optimizing Metal Density</u>	811
<u>Overview</u>	812
<u>Before You Begin</u>	813
<u>Adding Metal Fill in Multiple-CPU Processing Mode</u>	813
<u>After You Complete Adding Via and Metal Fill</u>	813
<u>Metal Fill Features</u>	814
<u>Staggered Metal Fill Pattern</u>	814
<u>Connected and Floating Metal Fill</u>	815
<u>Timing-Aware Metal Fill</u>	819
<u>Specifying Metal Fill Parameters</u>	820
<u>Recommendations for Adding Timing-Aware Metal Fill</u>	821
<u>Timing-Aware Examples</u>	823
<u>Specifying the Active Spacing Value</u>	824
<u>Adding Metal Fill Over Macros</u>	824
<u>Recommendations for Power Strapping Mode</u>	825
<u>Adding Via Fill</u>	826
<u>Trimming Metal Fill</u>	827
<u>Verifying Metal Density</u>	827
<u>Adding Metal Fill Using the GUI</u>	829

### 23

<u>Timing Budgeting</u> .....	831
<u>Overview</u> .....	832
<u>Is My Design Ready for Budgeting?</u> .....	834
<u>Deriving Timing Budgets</u> .....	835
<u>Budgeting Using the GUI</u> .....	835
<u>Budgeting Using Text Commands</u> .....	835
<u>Top-Level Budgets Derived by Using Active Logic View</u> .....	836
<u>Deriving Preliminary Budgets in Early Design Phase</u> .....	837
<u>Budgeting Output Files for MMMC Designs</u> .....	839
<u>Corner Cloning</u> .....	839
<u>Mode Cloning</u> .....	840
<u>Setup and View Handling for MMMC Designs</u> .....	841
<u>Constraints Adjustment</u> .....	842
<u>Analyzing Timing Budgets</u> .....	844
<u>Resolving Conflicts with Path-Based Exceptions</u> .....	844
<u>Budgeting Clock Latency in Propagated Mode</u> .....	847
<u>Calculating Timing Budgets</u> .....	849
<u>Customizing Budget Generation</u> .....	852
<u>Verifying Timing Budgets</u> .....	854
<u>Reading the Justify Budget Report</u> .....	855
<u>Design Example</u> .....	857
<u>SDC Constraints for Design Example</u> .....	858
<u>Generated Report for Design Example</u> .....	858
<u>Constraints Support in Budgeting</u> .....	860
<u>Warning Report</u> .....	863
<u>Pin Constraint Values Greater than Available Time</u> .....	863
<u>Warning Report Example</u> .....	863

### 24

<u>RC Extraction</u> .....	865
<u>Overview</u> .....	866
<u>Before You Begin</u> .....	867
<u>Results</u> .....	867

## Encounter User Guide

---

<u>Specifying Temporary File Locations</u>	867
<u>Native RC Extraction</u>	868
<u>CCE RC Extraction</u>	870
<u>Incremental Extraction Support for Design Changes</u>	872
<u>Generating a Capacitance Table</u>	874
<u>Inputs for Generating a Capacitance Table</u>	874
<u>Capacitance Table Generation Flow</u>	875
<u>Generating Capacitance Table With Specified Scaling Factors</u>	880
<u>Reading a Capacitance Table</u>	882
<u>Correlating Native Extraction With Sign-Off Extraction</u>	883
<u>Correlating SPEF Files Using the Ostrich Utility</u>	884
<u>Comparing SPEF Files Using a Perl Script</u>	887
<u>Defining the Scaling Factor</u>	890
<u>Sign-Off Extraction Using QRC</u>	891
<u>Inputs for QRC Sign-Off Extraction</u>	892

## 25

<u>Calculating Delay</u>	893
<u>Overview</u>	894
<u>Data Preparation</u>	895
<u>Operating Conditions</u>	895
<u>ECSM Libraries</u>	895
<u>Delay Calculation Modes and Related Controls</u>	896
<u>Choosing A Delay Calculation Engine</u>	897
<u>Running Delay Calculation</u>	897

## 26

<u>Timing Analysis</u>	899
<u>Overview</u>	900
<u>Timing Analysis Features</u>	901
<u>Before You Begin</u>	902
<u>Reading Timing Libraries</u>	903
<u>Resolving Discrepancies in Timing Libraries</u>	903
<u>Reading Timing Constraints</u>	904
<u>Constraints Quick Reference</u>	904

## Encounter User Guide

---

<u>Timing Analysis Results</u>	906
<u>Setting Operating Conditions</u>	907
<u>Calculating Clock Latency</u>	908
<u>Defining RC Corners</u>	909
<u>Specifying Timing Analysis Modes</u>	911
<u>Definition of Early and Late Paths</u>	911
<u>Single Timing Analysis Mode</u>	913
<u>Best-Case Worst-Case (BC-WC) Timing Analysis Mode</u>	917
<u>On-Chip Variation (OCV) Timing Analysis Mode</u>	922
<u>Clock Path Pessimism Removal</u>	927
<u>Analyzing Timing Problems</u>	933
<u>Resolving Buffer-Related Problems</u>	934

## 27

<u>Debugging Timing Results</u>	937
<u>Overview</u>	938
<u>Timing Debug Flow</u>	939
<u>Generating Timing Debug Report</u>	940
<u>Displaying Violation Report</u>	940
<u>Analyzing Timing Results</u>	941
<u>Viewing Power Domain Information</u>	946
<u>Creating Path Categories</u>	947
<u>Creating Predefined Categories</u>	947
<u>Creating New Categories</u>	948
<u>Creating Sub-Categories</u>	950
<u>Hiding path categories</u>	954
<u>Reporting Path Categories</u>	954
<u>Using Categories to Analyze Timing Results</u>	956
<u>Analyzing MMMC Categories</u>	957
<u>Manual Slack Correction of Categories</u>	960
<u>Editing Table Columns</u>	960
<u>Cell Coloring</u>	962
<u>Viewing Schematics</u>	964
<u>Running Timing Debug with Interface Logic Models</u>	964

### 28

<u>Statistical Static Timing Analysis</u> .....	967
<u>SSTA Overview</u> .....	968
<u>SSTA Inputs</u> .....	972
<u>Libraries with sensitivities</u> .....	973
<u>Statistical Parameter Distribution Format (SPDF) File</u> .....	975
<u>Specifying Global or Die-to-Die Variations in SPDF File</u> .....	975
<u>Specifying Random Variations in SPDF File</u> .....	975
<u>Specifying Spatial Variations in SPDF File</u> .....	976
<u>Sensitivity-Based SPEF (S-SPEF) File</u> .....	977
<u>Loading the S-SPEF File</u> .....	977
<u>SSTA Flows</u> .....	978
<u>Running Block-Based SSTA</u> .....	979
<u>Running Path-Based SSTA</u> .....	980
<u>SSTA Outputs</u> .....	981
<u>Block-Based SSTA Report</u> .....	981
<u>Path-Based SSTA Report</u> .....	983
<u>SSTA Correlation With Monte-Carlo Analysis</u> .....	985

### 29

<u>Extracting Timing Models</u> .....	987
<u>ETM Overview</u> .....	988
<u>Using ETMs in Different Timing Analysis Modes</u> .....	989
<u>Limitation of Timing Models</u> .....	990
<u>ETM Inputs</u> .....	992
<u>Guidelines for Generating ETMs</u> .....	993
<u>ETM Generation Flow</u> .....	995
<u>Validating the Generated Model</u> .....	997
<u>Reducing the Size of GreyBox Models</u> .....	998
<u>ETM Outputs</u> .....	1000
<u>Timing Library File</u> .....	1000
<u>Boundary Nets</u> .....	1000
<u>Internal Nets</u> .....	1001
<u>Timing Paths</u> .....	1001

## Encounter User Guide

---

<u>Minimum Pulse Width and Minimum Period</u>	1002
<u>Path Exceptions</u>	1003
<u>Constants</u>	1003
<u>Gating Checks</u>	1003
<u>Annotated Delays and Slews</u>	1004
<u>Design Rules</u>	1005
<u>Generated Clocks</u>	1005
<u>Timing Constraints Files</u>	1008
<u>set false path and set multicycle path constraints</u>	1008
<u>set disable timing and set case analysis</u>	1008
<u>create clock and create generated clock</u>	1009
<u>set input delay and set output delay</u>	1009
<u>Design Rules</u>	1009
<u>set load, set resistance and set annotated transition</u>	1009
<u>set annotated delay and set annotated check</u>	1010
<u>set input transition and set driving cell</u>	1010
<b>30</b>	
<u>Optimizing Timing</u>	1011
<u>Overview</u>	1012
<u>Before You Begin</u>	1012
<u>Results</u>	1013
<u>Interrupting Timing Optimization</u>	1015
<u>Performing Optimization Before Clock Tree Synthesis</u>	1016
<u>Correcting Violations in Pre-CTS Mode for the First Time</u>	1016
<u>Performing Rapid Timing Optimization for Design Prototyping</u>	1017
<u>Using Additional Pre-CTS Timing Optimization Parameters</u>	1017
<u>Performing Incremental Pre-CTS Optimization</u>	1018
<u>Changing Default Settings in Pre-CTS Mode</u>	1019
<u>Performing Post-CTS Optimization</u>	1020
<u>Correcting Violations in Post-CTS Mode</u>	1020
<u>Using Additional Post-CTS Timing Optimization Parameters</u>	1021
<u>Performing Incremental Post-CTS Optimization</u>	1021
<u>Changing Default Settings in Post-CTS Mode</u>	1022
<u>Performing Postroute Optimization</u>	1023

## Encounter User Guide

---

<u>About Postroute Optimization</u>	1024
<u>Correcting Violations in Postroute Mode</u>	1025
<u>Correcting Signal Integrity Violations</u>	1027
<u>Changing Default Settings in Postroute Mode</u>	1029
<u>Optimizing Power During optDesign</u>	1029
<u>Leakage Power Optimization</u>	1029
<u>Dynamic Power Optimization</u>	1030
<u>Using Useful Skew</u>	1030
<u>Using Useful Skew in Pre-CTS Mode</u>	1030
<u>Using Useful Skew in Post-CTS Mode</u>	1031
<u>Controlling Useful Skew Optimization</u>	1031
<u>Using Active Logic View for Chip-Level Interface Circuit Timing Closure</u>	1033
<u>Optimizing Timing in On-Chip Variation Analysis Mode</u>	1033
<u>Specifying the MMMC Environment</u>	1034
<u>Optimizing Timing in OCV Mode Using the Default Delay Calculator</u>	1036
<u>Optimizing Timing in OCV Mode Using the Sign-Off Delay Calculator</u>	1036
<u>Using Conformal Constraint Designer During Timing Optimization</u>	1037
<u>Post-Processing Approach</u>	1037
<u>Integrated Approach</u>	1038
<u>Optimizing Timing Using a Rule File</u>	1040
<u>Optimizing Timing When the Constraint File Includes the set case analysis Constraint</u>	1040
<u>Using the Footprintless Flow</u>	1040
<u>Using Cell Footprints</u>	1042
<u>Viewing Added Buffers, Instances, and Nets</u>	1042
<u>Default Naming Conventions</u>	1042
<b>31</b>	
<u>Interactive ECO</u>	1045
<u>Overview</u>	1046
<u>Before You Begin</u>	1046
<u>Results</u>	1046
<u>Adding Buffers</u>	1046
<u>Changing the Cell</u>	1049
<u>Deleting Buffers</u>	1051

## Encounter User Guide

---

<u>Displaying Buffer Trees</u>	1052
<u>Running ECO Placement</u>	1054
<u>Naming Conventions for Interactive ECO</u>	1055
<u>Comparing Physical Design Data</u>	1055

## 32

<u>Analyzing and Repairing Crosstalk</u>	1061
<u>Overview</u>	1062
<u>Inputs and Outputs for SI Analysis</u>	1063
<u>Setting Up Encounter for SI Analysis</u>	1064
<u>RC Extraction Settings</u>	1064
<u>Noise Analysis Settings</u>	1066
<u>Static Timing Analysis (STA) Settings</u>	1068
<u>Advanced Settings for SI Analysis</u>	1069
<u>Example of Setting Up Encounter for SI Analysis</u>	1073
<u>Preventing Crosstalk Violations</u>	1074
<u>Fixing Crosstalk Violations</u>	1075
<u>Data Preparation</u>	1075
<u>Using optDesign to Fix Setup Violations with Crosstalk Effects</u>	1076
<u>Using optDesign to Fix Hold Violations with Crosstalk Effects</u>	1078
<u>Using optDesign to Fix Transition Time Violations with Crosstalk Effects</u>	1080
<u>Performing XILM-Based SI Analysis and Fixing</u>	1083

## 33

<u>Power and Rail Analysis</u>	1085
<u>Early Rail Analysis</u>	1086
<u>Early Rail Analysis key features</u>	1086
<u>Prior to running early rail analysis</u>	1086
<u>Setting up and running early rail analysis</u>	1088
<u>Viewing Early Rail Analysis results</u>	1096
<u>Signoff-Rail Analysis</u>	1100
<u>Encounter and EPS menu differences</u>	1101

### 34

<u>Verifying Violations</u> .....	1103
<u>Overview</u> .....	1104
<u>Verifying Connectivity</u> .....	1107
<u>Before You Begin</u> .....	1107
<u>Types of Connectivity Violations Reported</u> .....	1107
<u>Results</u> .....	1108
<u>Verifying Metal Density</u> .....	1109
<u>Before You Begin</u> .....	1109
<u>Results</u> .....	1109
<u>Verifying Geometry</u> .....	1110
<u>Before You Begin</u> .....	1110
<u>Verifying Geometry in Multi-Thread Mode</u> .....	1110
<u>Spacing Violation Checks</u> .....	1111
<u>Types of Antenna Violations Reported</u> .....	1112
<u>Support for Via Rules</u> .....	1113
<u>Results</u> .....	1113
<u>Verifying Process Antennas</u> .....	1114
<u>Before You Begin</u> .....	1114
<u>Verifying PAE</u> .....	1114
<u>Results</u> .....	1114
<u>Sample Process Antenna Report</u> .....	1115
<u>Verifying Maximum Floating Area Violations</u> .....	1117
<u>Verifying AC Limit</u> .....	1118
<u>Before You Begin</u> .....	1118
<u>Results</u> .....	1118
<u>Viewing Violations With the Violation Browser</u> .....	1119
<u>Viewing Geometry or Metal Density Violations</u> .....	1119
<u>Viewing Connectivity, Process Antenna, or AC Limit Violations</u> .....	1119
<u>Viewing Violation Markers From Assura or Calibre</u> .....	1119
<u>Violation Browser Features</u> .....	1120
<u>Clearing Violations</u> .....	1122

### 35

<u>Analyzing Yield</u> .....	1123
<u>Overview</u> .....	1124
<u>What Effects Does reportYield Consider?</u> .....	1124
<u>Calculating Failure Probabilities</u> .....	1125
<u>Critical Area Analysis</u> .....	1126
<u>Defect Data and Cumulative Defect Data Functions</u> .....	1127
<u>Before You Begin</u> .....	1127
<u>Results</u> .....	1127
<u>Interpreting the Yield Map</u> .....	1129
<u>Displaying the Yield Map</u> .....	1129
<u>Interpreting the Yield Report</u> .....	1132
<u>Yield Report</u> .....	1132
<u>Detailed Report</u> .....	1136
<u>Understanding the Yield Technology File</u> .....	1138
<u>File Format</u> .....	1138
<u>File Sections and Keyword Statement Descriptions</u> .....	1140
<u>Yield Technology File Example</u> .....	1150
<u>Formulas and Calculations</u> .....	1153
<u>Calculating the Probability of Failure for a Metal Layer</u> .....	1153
<u>Calculating Defect and Cumulative Defect Data</u> .....	1153
<u>Cost Formulas</u> .....	1156

### 36

<u>Creating An Initial Floorplan Using Masterplan</u> .....	1157
<u>Overview</u> .....	1158
<u>Masterplan Automatic Floorplanner Flow</u> .....	1159
<u>Data Preparation</u> .....	1161
<u>Selecting Seeds</u> .....	1161
<u>Importing the Design</u> .....	1167
<u>Setting Masterplan Global Parameters</u> .....	1168
<u>Creating an Initial Floorplan</u> .....	1168
<u>Creating Floorplan for Hierarchical Design</u> .....	1169
<u>Macro placement</u> .....	1170

## Encounter User Guide

---

<u>Full-chip Floorplan</u>	1171
<u>Power-Domain Aware Floorplan</u>	1172
<u>Creating Multiple Alternative Floorplans</u>	1174
<u>Analyzing the Floorplan</u>	1174
<u>Adjusting Macro Placement</u>	1176
<u>Manual Macro Adjustment</u>	1176
<u>Masterplan Macro Adjustment</u>	1176
<u>Saving the Floorplan</u>	1181

## 37

<u>Performing Multi-Mode Multi-Corner Timing Analysis and Optimization</u>	1183
<u>Overview</u>	1185
<u>Configuring the Setup for Multi-Mode Multi-Corner Analysis</u>	1186
<u>Creating Library Sets</u>	1187
<u>Creating Virtual Operating Conditions</u>	1188
<u>Creating RC Corner Objects</u>	1189
<u>Creating Delay Calculation Corner Objects</u>	1190
<u>Adding A Power Domain Definition To A Delay Calculation Corner</u>	1192
<u>Creating Constraint Mode Objects</u>	1193
<u>Creating Analysis Views</u>	1197
<u>Setting Active Analysis Views</u>	1198
<u>Checking the Multi-Mode Multi-Corner Configuration</u>	1199
<u>Saving Multi-Mode Multi-Corner Configurations</u>	1200
<u>Controlling Multi-Mode Multi-Corner Analysis Through the Flow</u>	1200
<u>Performing Timing Analysis</u>	1202
<u>Generating Timing Reports</u>	1203
<u>Performing Timing Optimization</u>	1203

## 38

<u>Creating the ICT File</u>	1205
<u>Format</u>	1206
<u>Data</u>	1206
<u>Comments</u>	1206

## Encounter User Guide

---

<u>Case Sensitivity</u>	1206
<u>Warnings and Errors</u>	1206
<u>Invalid Layer Names</u>	1206
<u>Commands</u>	1206
<u>Sample ICT File</u>	1219
 <b>39</b>	
<b><u>ECO Flows</u></b>	1231
<u>Overview</u>	1232
<u>Assumptions</u>	1232
<u>Flows</u>	1232
<u>Pre-Mask ECO Changes from a New Verilog File</u>	1234
<u>Preparation</u>	1234
<u>Flow</u>	1234
<u>Steps</u>	1235
<u>Pre-Mask ECO Changes from a New DEF File</u>	1238
<u>Preparation</u>	1238
<u>Flow</u>	1239
<u>Steps</u>	1239
<u>Pre-Mask ECO Changes from an ECO File</u>	1242
<u>Preparation</u>	1242
<u>Flow</u>	1243
<u>Steps</u>	1243
<u>Post-Mask ECO Changes from a New Verilog Netlist</u>	1246
<u>Preparation</u>	1246
<u>Flow</u>	1247
<u>Steps</u>	1247
<u>Post-Mask Gate Array Style ECO from a New Verilog Netlist</u>	1252
<u>Preparation</u>	1252
<u>Steps</u>	1254
 <b>A</b>	
<b><u>ECO Directives</u></b>	1257
<u>ADDHIERINST</u>	1259
<u>ADDINST</u>	1260

## Encounter User Guide

---

<u>ADDMODULEPORT</u>	.....	1262
<u>ADDNET</u>	.....	1264
<u>ATTACHMODULEPORT</u>	.....	1265
<u>ATTACHTERM</u>	.....	1266
<u>CHANGEINSTNAME</u>	.....	1268
<u>DELETEBUFFER</u>	.....	1269
<u>DELETEINST</u>	.....	1271
<u>DELETEMODULEPORT</u>	.....	1272
<u>DELETENET</u>	.....	1273
<u>DETACHMODULEPORT</u>	.....	1274
<u>DETACHTERM</u>	.....	1275
<u>INSERTBUFFER</u>	.....	1276
<u>Example ECO File</u>	.....	1279

## B

<u>Clock Mesh Specification File</u>	.....	1281
<u>Overview</u>	.....	1281
<u>Routing Type Definitions</u>	.....	1282
<u>Cutout Definitions</u>	.....	1282
<u>Clock Mesh Definitions</u>	.....	1283
<u>Timing and Power Constraints Section</u>	.....	1284
<u>Tracing and Analysis Scope Section</u>	.....	1285
<u>Mesh Structure Section</u>	.....	1286
<u>Global Mesh Section</u>	.....	1287
<u>Top Chain Section</u>	.....	1298
<u>Local Tree Section</u>	.....	1300
<u>Clock Mesh Specification File Example</u>	.....	1302

C

Supported CPF 1.0 Commands ..... 1309

D

Supported CPF 1.0e Commands ..... 1319

E

Supported CPF 1.1 Commands ..... 1331

F

CPF 1.0 Script Example ..... 1345

G

CPF 1.0e Script Example ..... 1357

H

CPF 1.1 Script Example ..... 1363

I

Cadence-Specific Liberty Extensions ..... 1369

Overview ..... 1369

Guidelines For Adding ECSM Extensions ..... 1370

Representing ECSM Information in a Library ..... 1370

Defining ECSM Extensions in a Library ..... 1371

ecsm waveform Group ..... 1373

ecsm waveform set Group ..... 1376

ecsm capacitance Group ..... 1378

Example ..... 1382

Index ..... 1391

## **Encounter User Guide**

---

# About This Manual

---

The Cadence® Encounter® family of products provides an integrated solution for an RTL-to-GDSII design flow. This manual describes how to install, configure, and use Encounter to implement digital integrated circuits.

## Audience

This manual is written for experienced designers of digital integrated circuits. Such designers must be familiar with design planning, placement and routing, block implementation, chip assembly, and design verification. Designers must also have a solid understanding of UNIX and Tcl/Tk programming.

## How This Manual Is Organized

The chapters in this manual are organized to follow the flow of tasks through the design process. Because of variations in design implementations and methodologies, the order of the chapters will not correspond to any specific design flow.

Each chapter focuses on the concepts and tasks related to the particular design phase or topic being discussed.

In addition, the following sections provide prerequisite information for using the Encounter software:

- Chapter 2, “Getting Started”

Describes how to install, set up, and run the Encounter software, and use the online Help system.

- Chapter 4, “Data Preparation”

Describes how to prepare data for import into the Encounter software.

## Conventions Used in This Manual

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

*text* Indicates text that you must type exactly as shown. For example:

```
analyze_connectivity -analyze all
```

*text* Indicates information for which you must substitute a name or value.

In the following example, you must substitute the name of a specific file for *configfile*:

```
wroute -filename configfile
```

*text* Indicates the following:

- Text found in the graphical user interface (GUI), including form names, button labels, and field names
- Terms that are new to the manual, are the subject of discussion, or need special emphasis
- Titles of manuals

[ ] Indicates optional arguments.

In the following example, you can specify none, one, or both of the bracketed arguments:

```
command [-arg1] [arg2 value]
```

[ | ] Indicates an optional choice from a mutually exclusive list.

In the following example, you can specify any of the arguments or none of the arguments, but you cannot specify more than one:

```
command [arg1 | arg2 | arg3 | arg4]
```

{ | } Indicates a required choice from a mutually exclusive list.

In the following example, you must specify one, and only one, of the arguments:

```
command {arg1 | arg2 | arg3}
```

## Encounter User Guide

### About This Manual

---

{ [ ] [ ] }	Indicates a required choice of one or more items in a list.  In the following example, you must choose one argument from the list, but you can choose more than one:  command { [arg1] [arg2] [arg3]}
{ }	Indicates curly braces that must be entered with the command syntax.  In the following example, you must type the curly braces:  command arg1 {x y}
...	Indicates that you can repeat the previous argument.
.	Indicates an omission in an example of computer output or input.
...	
<i>Command – Subcommand</i>	Indicates a command sequence, which shows the order in which you choose commands and subcommands from the GUI menu.  In the following example, you choose <i>Floorplan</i> from the menu, then <i>Power Planning</i> from the submenu, and then <i>Add Rings</i> from the displayed list:  <i>Floorplan – Power Planning – Add Rings</i>  This sequence opens the Add Rings form.

## Related Documents

For more information about the Encounter family of products, see the following documents. You can access these and other Cadence documents with the Cadence Help online documentation system.

### Encounter Foundation Flows Documentation

#### ■ *Encounter Foundation Flows User Guide*

Describes how to use the scripts that represent the recommended implementation flows for digital timing closure with the Encounter software.

#### ■ *Encounter Foundation Flows: Flat Implementation Flow Guide*

Describes the default-effort flat implementation flow, using the Encounter software.

## Encounter User Guide

### About This Manual

---

- *Encounter Foundation Flows: Hierarchical Implementation Flow Guide*  
Describes the default-effort hierarchical implementation flow, using the Encounter software.
- *Encounter CPF-Based Low-Power Implementation Flow Guide*  
Describes the CPF-Based Low Power implementation flow, using the Encounter software.

## Encounter Product Documentation

- *What's New in Encounter*  
Provides information about new and changed features in this release of the Encounter family of products.
- *Encounter Known Problems and Solutions*  
Describes important Cadence Change Requests (CCRs) for the Encounter family of products, including solutions for working around known problems.
- *Encounter Text Command Reference*  
Describes the Encounter text commands, including syntax and examples.
- *Encounter Menu Reference*  
Provides information specific to the forms and commands available from the Encounter graphical user interface.
- *Encounter Database Access Command Reference*  
Lists all of the Encounter database access commands and provides a brief description of syntax and usage.
- *Encounter Library Development Guide*  
Describes library development guidelines for the independent tools that make up the Encounter family of products.
- README file  
Contains installation, compatibility, and other prerequisite information, including a list of Cadence Change Requests (CCRs) that were resolved in this release. You can read this file online at [downloads.cadence.com](http://downloads.cadence.com).

For a complete list of documents provided with this release, see the Cadence Help online documentation system.

## **Encounter User Guide**

### About This Manual

---

9/8/09

## **Encounter User Guide**

### About This Manual

---

---

## **Product and Licensing Information**

---

- [Overview](#) on page 44
- [About Encounter Products](#) on page 44
- [About Encounter Licenses](#) on page 50
  - [Licensing Terminology](#) on page 50
  - [Checking Out Licenses for Product Options](#) on page 51
  - [Checking Out Dynamic Licenses for Product Options](#) on page 51

## Overview

Each Cadence® Encounter® product is sold as part of a product package. Product packages may also include product options. The options provide advanced features and capabilities, such as support for the common power format, the ability to route mixed signal designs or to avoid and correct lithography problems.

Each product and product option has a corresponding license. The software uses licenses to determine the features that are available when the software runs.

## About Encounter Products

This release of the Encounter software includes the following product packages:

- [Encounter Digital Implementation System](#) on page 44
- [SoC Encounter RTL-to-GDSII System](#) on page 48

### Encounter Digital Implementation System

This package includes the products listed in [Table 1-1](#) on page 44. To start any of these products, type the following UNIX/Linux command:

`velocity`

**Table 1-1 Encounter Digital Implementation System Products**

Name	Abbreviation	Prod. Num.	Description
Encounter Digital Implementation System L	EDS-L	EDS100	An automatic digital implementation system for high-performance block-level implementation from RTL synthesis to GDSII with end-to-end multiple-CPU functionality on a configurable, extensible, and scalable platform. Limited to 300,000 instances in the netlist.
Encounter Digital Implementation System XL	EDS-XL	EDS200	Has all the features of EDS-L without the 300,000 instance limitation. In addition, this product supports hierarchical designs.

## Encounter User Guide

### Product and Licensing Information

---

Name	Abbreviation	Prod. Num.	Description
NanoRoute® Ultra SoC Routing Solution	NRU	FE150	Optimized routing and routing verification system with utmost in speed and capacity for signal integrity, timing, and interconnect optimization for manufacturability.
Virtuoso® Digital Implementation	VDI	3002	Has all the features of EDS-L and adds RTL Compiler functionality for logic synthesis. Limited to 50,000 instances in the netlist.

In addition to the products in the preceding table, the Encounter Digital Implementation System includes the following product options, which provide additional features:

**Table 1-2 Encounter Digital Implementation System Product Options**

Name	Abbreviation	Prod. Num.	Description
Encounter Mixed Signal GXL option	ENC-MS Opt.	EDS20	Adds mixed signal functionality by allowing you to transfer design data and routing constraints between the custom (Virtuoso) and digital design environments and route the design with the Encounter mixed signal router.
Encounter Low Power GXL option	ENC-LP Opt.	EDS10	Adds advanced low-power functionality by automating multiple power domain and power-switch-aware floorplan synthesis, implementation, and routing, enabled by full common power format (CPF) support.
Encounter Advanced Node GXL option	ENC-AN Opt.	EDS30	Adds new 40 nm and 32 nm rules support, concurrent design for yield/design for manufacturing capability by preventing lithography hotspots and analyzing and optimizing them if they occur, support for on-chip/off-chip variation mode and statistical static timing analysis (SSTA).

For more information on these products and options, see [Encounter Digital Implementation System Licensing and Packaging](#) on SourceLink®.

## Encounter User Guide

### Product and Licensing Information

---

For more information on 40 nm and 32 nm rules licensing, see [Advanced Node License Required for 40 nm and 32 nm DRC Rules](#).

### Advanced Node License Required for 40 nm and 32 nm DRC Rules

Newly added DRC rules for 40 nm and 32 nm process nodes require an Advanced Node license. The LEF property keywords used for these rules that require an Advanced Node license are highlighted below in **bold** text:

#### Layer (Cut)

```
[ PROPERTY LEF58_TYPE
    "TYPE [TSV | PASSIVATION] ;" ;]

[ PROPERTY LEF58_BACKSIDE
    "BACKSIDE ;" ;]

[ PROPERTY LEF58_CUTCLASS
    "CUTCLASS className WIDTH viaWidth [LENGTH viaLength] [CUTS numCut]
    ;" ;]

[ PROPERTY LEF58_SPACING
    "SPACING cutSpacing
        [MAXXY
        | [CENTERTOCENTER]
        | [SAMENET | SAMEMETAL | SAMEVIA]
        | LAYER secondLayerName [STACK]
        | ADJACENTCUTS {2 | 3 | 4} WITHIN cutWithin [EXCEPTSAMEPGNET]
        | CUTCLASS className
        | PARALLELOVERLAP [ EXCEPTSAMENET | EXCEPTSAMEMETAL | EXCEPTSAMEVIA ]
        | PARALLELWITHIN within [EXCEPTSAMENET]
        | AREA cutArea] ;" ;]

[ PROPERTY LEF58_ENCLOSUREEDGE
    "ENCLOSUREEDGE [CUTCLASS className] [ABOVE | BELOW] overhang
        WIDTH minWidth PARALLEL parLength WITHIN parWithin
        [EXCEPTEXTRACUT [cutWithin]]
        [EXCEPTTWOEDGES]
    ;" ;]

[ PROPERTY LEF58_ENCLOSURE
    "ENCLOSURE [CUTCLASS className] [ABOVE | BELOW] overhang1 overhang2
        [ WIDTH minWidth
        | EXCEPTEXTRACUT cutWithin [NOSHAREDEDGE] ]
        | LENGTH minLength
        | EXTRACUT
        | REDUNDANTCUT cutWithin
    ] ;" ;]
```

# Encounter User Guide

## Product and Licensing Information

---

```
[ PROPERTY LEF58_SPACINGTABLE
  "SPACINGTABLE
   [ORTHOGONAL
    {WITHIN cutWithin SPACING orthoSpacing} ... ;
   | [DEFAULT defaultCutSpacing]
     [SAMENET | SAMEMETAL]
     [LAYER secondLayerName]
     [CENTERTOCENTER { {className1 | ALL} | TO {className2 | ALL}
     }...]
     CUTCLASS { {className1 | ALL} [SIDE | END]}...
      {{className2 | ALL} [SIDE | END] {-|cutSpacing}
     {-|cutSpacing}...}...;
   ]
   ;" ;]

[ PROPERTY LEF58_ARRAYSPACING
  "ARRAYSPACING [CUTCLASS className] [PARALLELOVERLAP]
   [LONGARRAY] [WIDTH viaWidth] CUTSPACING cutSpacing
   {ARRAYCUTS arrayCuts SPACING arraySpacing} ... ;
  ];" ;
```

## Layer (Routing)

```
[ PROPERTY LEF58_BACKSIDE
  "BACKSIDE ;" ;]

[ PROPERTY LEF58_SPACING
  "SPACING eolSpace ENDOFLINE eolWidth [OPPOSITEWIDTH oppositeWidth]
  WITHIN eolWithin
   [ENDTOEND endToEndSpace [OTHERENDWIDTH otherEndWidth]]
   [MAXLENGTH maxLength
   |MINLENGTH minLength [TWOSIDES]]
   [EQUALRECTWIDTH]
   [PARALLELEDGE [SUBTRACTEOLWIDTH] parSpace WITHIN parWithin
    [MINLENGTH minLength] [TWOEDGES]]
   [ENCLOSECUT [BELOW | ABOVE] encloseDist CUTSPACING cutToMetalSpace]
  ;" ;]

[ PROPERTY LEF58_SPACINGTABLE
  "SPACINGTABLE
   PARALLELRUNLENGTH {length} ...
   {WIDTH width {spacing} ...} ... ;
   [SPACINGTABLE
    INFLUENCE {WIDTH width WITHIN distance SPACING spacing} ... ;
   | TWOWIDTHS {WIDTH width [PRL runLength] {spacing} ...} ... ;
   | PARALLELSPANLENGTH PRL runLength {SPANLENGTH spanLength {spacing} ...} ;
  ;"
```

## SoC Encounter RTL-to-GDSII System

This package includes the products listed in [Table 1-3](#) on page 48. To start any of these products, type the following UNIX/Linux command:

encounter

**Table 1-3 SoC™ Encounter RTL-to-GDSII System Products**

Name	Abbreviation	Prod. Num.	Description
First Encounter™ L	FE-L	FE80	An automatic silicon virtual prototyping and hierarchical partitioning solution with built-in power planning and floorplanning.
First Encounter XL	FE-XL	FE100 GPS	Has all the features of First Encounter XL. In addition, this product supports RTL, global physical and clock-tree synthesis.
First Encounter GXL	FE-GXL	FE800	Has all the features of First Encounter XL. In addition, this product supports advanced design for yield (DFY) features.
NanoRoute Ultra SoC Routing Solution	NRU	FE150	Optimized routing and routing verification system with utmost in speed and capacity for signal integrity, timing, and interconnect optimization for manufacturability.
SoC Encounter L	SOCE-L	FE300 DBS	An automatic digital implementation system for block-level implementation from RTL synthesis to GDSII. Limited to 300,000 instances in the netlist.
SoC Encounter XL	SOCE-XL	FE200 GPS	Has all the features of SoC Encounter L without the 300,000 instance limitation. In addition, this product supports hierarchical designs.
SoC Encounter GXL	SOCE-GXL	FE850	Has all the features of SoC Encounter XL. In addition, this product supports advanced manufacturing, yield, and variation-aware design.
Virtuoso Digital Implementation	VDI	3002	Has all the features of EDS-L and adds RTL Compiler functionality for logic synthesis. Limited to 50,000 instances in the netlist.

## **Encounter User Guide**

### Product and Licensing Information

---

For more information on these products, see [SoC Encounter Licensing and Packaging](#) on SourceLink®.

## About Encounter Licenses

When you run a command to invoke an Encounter product or product option, a license is checked out. Each product and product option has a unique license string (also called a license key). The following table lists the product and product option names in alphabetical order and provides the corresponding license strings.

**Table 1-4 Product and product options and corresponding license strings**

<b>Product or option</b>	<b>License string</b>
Encounter Advanced Node GXL option	Encounter_Adv_Node_GXL
Encounter Digital Implementation System L	Encounter_Digital_Impl_Sys_L
Encounter Digital Implementation System XL	Encounter_Digital_Impl_Sys_XL
Encounter Low Power GXL option	Encounter_Low_Power_GXL
Encounter Mixed Signal GXL option	Encounter_Mixed_Signal_GXL
First Encounter GXL	First_Encounter_GXL
First Encounter L	First_Encounter_VIP
First Encounter XL	First_Encounter_GPS
NanoRoute Ultra Routing Solution	NanoRoute_Ultra
SoC Encounter GXL	SOC_Encounter_GXL
SoC Encounter L	Nano_Encounter_DBS
SoC Encounter XL	SOC_Encounter_GPS
Virtuoso Digital Implementation	Virtuoso_Digital_implement

## Licensing Terminology

The following terminology is useful in understanding Encounter licenses.

### **Base license**

The license that is checked out when the software starts. The base license must be a license for a product, not a product option.

## **Dynamic license**

A license for a product option that is not checked out until a feature provided by the product option is needed. You can check out more than one dynamic license per base license. For more information on dynamic licenses, see “[Checking Out Dynamic Licenses for Product Options](#)” on page 51.

## **Multi-CPU license**

A license that enables additional CPUs for multithreading, Superthreading, or distributed processing. Multi-CPU licenses must be product licenses, and can be checked out after the base license is checked out. You can check out more than one multi-CPU license per base license. For more information on multi-CPU licenses, see [SoC Encounter Licensing and Packaging](#) and [Encounter Digital Implementation System Licensing and Packaging](#) on SourceLink®.

## **Checking Out Licenses for Product Options**

Specify the product options to check out immediately by using one of the following methods:

- When you invoke the software by using one of the following UNIX/Linux commands:  
`velocity -checkoutList "option1 option2 ..."`  
`encounter -checkoutList "option1 option2 ..."`
- When the software is already running by using the following text command:  
`setLicenseCheck -checkout option`  
With this command, you can specify only one product option.



You cannot check out a license for a product option if you have not checked out a base license.

## **Checking Out Dynamic Licenses for Product Options**

Specify dynamic licenses to check out by using one of the following methods:

- When you invoke the software by using one of the following UNIX/Linux commands:  
`velocity -optionList "option1 option2 ..."`  
`encounter -optionList "option1 option2 ..."`

Even though you specify the options at startup, the option licenses are not checked out until they are needed.

## **Encounter User Guide**

### Product and Licensing Information

---

- After the software is running by using the following text command:

```
setLicenseCheck -optionList "option1 option2 ..."
```

Even though you specify the options with this command, the option licenses are not checked out until they are needed.

---

## **Getting Started**

---

- [Product and Installation Information](#) on page 54
- [Setting the Run-Time Environment](#) on page 54
- [Configuring OpenAccess](#) on page 55
- [Launching the Console](#) on page 56
- [Completing Command Names](#) on page 56
- [Command-Line Editing](#) on page 57
- [Setting Preferences](#) on page 59
- [Starting the Software](#) on page 62
- [Interrupting the Software](#) on page 73
- [Using the Log File Viewer](#) on page 75
- [Accessing Documentation and Help](#) on page 77

## Product and Installation Information

For product, release, and installation information, see the `README` file at any of the following locations:

- [downloads.cadence.com](http://downloads.cadence.com), where you can review the `README` before you download the software
- In the software installation, where it is also available when you are using or running the software

For information about Encounter® licenses, see [About Encounter Licenses](#) in the “Product and Licensing Information” chapter.

## Setting the Run-Time Environment

- To set the run-time environment, include the following installation directory in your path `install_dir/tools/bin` by using the following command:

```
set path = (<install_dir>/tools/bin $path)
```

## Supported and Compatible Platforms

The `README` file lists the supported and compatible platforms for this release.

## Specifying the 64-Bit or 32-Bit Version of Encounter Applications

You can run the Encounter software in either 32-bit and 64-bit mode. The 32-bit version and 64-bit version of the software are installed in the same tools hierarchy. By default, software runs in 32-bit mode if it is available.

**Note:** 32-bit versions of the software are not available for the following platforms:

- Sun (sol86 and sun4v)
- IBM AIX (ibmrs)

For more information, see the `README` file.

Use one of the following methods the specify the version to use:

- Set the `CDS_AUTO_64BIT` environment variable before starting the software. For more information, see [Using the CDS\\_AUTO\\_64BIT Environment Variable](#) on page 55.

- Use a command parameter when you start the software. For information, see [Using Generic Parameters to Specify 32- or 64-Bit Version](#) on page 72.

### Using the CDS\_AUTO\_64BIT Environment Variable

To run 64-bit versions of all or some applications, complete the following steps before starting the software:

1. If you are using the lnx86 operating system, verify that it supports 64-bit applications.
2. Set the CDS\_AUTO\_64BIT environment variable.

For example,

- ❑ To run all Encounter applications in 64-bit mode, type the following command:  
`setenv CDS_AUTO_64BIT ALL`
- ❑ To run just a few applications in 64-bit mode, such as NanoRoute® and CeltIC®, and all other applications in 32-bit mode, type one of the following commands:  
`setenv CDS_AUTO_64BIT nanoroute:celtic`  
`setenv CDS_AUTO_64BIT nanoroute,celtic`  
`setenv CDS_AUTO_64BIT 'nanoroute;celtic'`

## Configuring OpenAccess

The Encounter software installs OpenAccess in the `<Cadence_install_dir>/` directory. The software creates a symbolic link from `<Cadence_install_dir>/share/oa` to the OpenAccess installation directory.

The software reports the version and data model of OpenAccess with which it was compiled. For example, when you start the Encounter software, it displays a message similar to the following:

INFO: This Encounter release has been compiled with OA data Model 4 and OA version p006.



Cadence recommends that you use the OpenAccess kit that comes with the Encounter software for almost all uses.

However, if you decide to change the kit, use the `OA_HOME` environment variable to override the default OpenAccess installation. Before setting this variable, make sure of the following:

- ❑ The version of the OpenAccess kit you specify must use the same or a newer data model than the one that was included with the Encounter installation.
- ❑ The release data of the OpenAccess kit that you specify must be newer than the release data of the one that was included with the Encounter installation.

To set the variable, type the following command:

```
setenv OA_HOME oa_install_dir
```

Where *oa\_install\_dir* is the path to the OpenAccess installation to use.

For information on the version of OpenAccess supported with this release, see the README file.

## Launching the Console

The window (shell tool, xterm, and so on) where you start the Encounter session is called the Encounter console. You enter all Encounter text commands in the console window, and the software displays messages there. When a session is active, the console displays the following prompt:

```
encounter>
```

**Note:** If you started the software by using the `velocity` command, the console displays the following prompt:

```
velocity>
```

If you use the console for other actions—for example, to use the vi editor—the session suspends until you finish the action.

If you suspend the session by typing Control-z, the `encounter>` prompt is no longer displayed. To return to the Encounter session, type `fg`, which brings the session to the foreground.

## Completing Command Names

Use the Tab key within the software console to complete text command names.

After you type a partial text command name and press the Tab key, the software displays the exact command name that completes or matches the text you typed (if the string is unique to one text command) or all the commands that match the text you typed.

For example, if you type the following text and press the Tab key

```
setPlace
```

The software displays the following command:

`setPlaceMode`

If you type the following text and press the Tab key

`setPl`

The software displays the following commands:

`setPlaceMode`      `setPlanDesignMode`

## Command-Line Editing

The Encounter software provides a GNU Emacs–like editing interface. You can edit a line before it is sent to the calling program by typing control characters or escape sequences. A control character, shown below as a caret followed by a letter, is typed by holding down the Control key when typing the character.

Most editing commands can be given a repeat count, *n*, where *n* is a number. To enter a repeat count, press the Esc key, the number, and then the command to execute. For example, Esc 4 ^f moves forward four characters. If a command can be given a repeat count, the text [ *n* ] is shown at the end of its description.

You can type an editing command anywhere on the line, not just at the beginning. You can press Return anywhere on the line, not just at the end.

**Note:** Editing commands are case sensitive: Esc F is not the same as Esc f.

## Control (^) Characters

- ^A Move to the beginning of the line
- ^B Move left (backwards) [ *n* ]
- ^C Exits from editing mode, returning the console to normal Encounter mode
- ^D Delete character [ *n* ]
- ^E Move to end of line
- ^F Move right (forwards) [ *n* ]
- ^G Ring the bell
- ^H Delete character before cursor (backspace key) [ *n* ]
- ^I Complete filename (Tab key); see below

## Encounter User Guide

### Getting Started

---

<code>^J</code>	Done with line (Return key)
<code>^K</code>	Kill to end of line (or column [ <i>n</i> ])
<code>^L</code>	Redisplay line
<code>^M</code>	Done with line (alternate Return key)
<code>^N</code>	Get next line from history [ <i>n</i> ]
<code>^P</code>	Get previous line from history [ <i>n</i> ]
<code>^R</code>	Search backward (forward if [ <i>n</i> ]) through history for text; must start line if text begins with an up arrow
<code>^T</code>	Transpose characters
<code>^V</code>	Insert next character, even if it is an edit command
<code>^W</code>	Wipe to the mark
<code>^X^X</code>	Exchange current location and mark
<code>^Y</code>	Yank back last killed text
<code>^[</code>	Start an escape sequence (Esc key)
<code>^]c</code>	Move forward to next character <i>c</i>
<code>^?</code>	Delete character before cursor (Delete key) [ <i>n</i> ]

## Escape Sequences

<code>Esc ^H</code>	Delete previous word (Backspace key) [ <i>n</i> ]
<code>Esc Delete</code>	Delete previous word (Delete key) [ <i>n</i> ]
<code>Esc SP</code>	Set the mark (Space bar); see <code>^X^X</code> and <code>^Y</code> above
<code>Esc .</code>	Get the last (or [ <i>n</i> ]'th) word from previous line
<code>Esc &lt;</code>	Move to start of history
<code>Esc &gt;</code>	Move to end of history
<code>Esc b</code>	Move backward a word [ <i>n</i> ]
<code>Esc d</code>	Delete word under cursor [ <i>n</i> ]
<code>Esc f</code>	Move forward a word [ <i>n</i> ]
<code>Esc l</code>	Make word lowercase [ <i>n</i> ]

Esc u	Make word uppercase [n]
Esc Y	Yank back last killed text
Esc v	Show library version
Esc w	Make area up to mark yankable
Esc nn	Set repeat count to the number nn
Esc C	Read from environment variable _C_, where C is an uppercase letter

## Setting Preferences

You set preferences at the beginning of a new design import. You can assign special characters for the design import parser for Verilog®, DEF, and PDEF files, and control the display of the Floorplan and Physical view windows. You can also change the hierarchical delimiter character in the netlist before importing the design, and change the DEF hierarchical default character and the PDEF bus default delimiter before loading the file.

**Note:** If you change the default values for the DEF delimiter or PDEF bus delimiter, these changes become the default delimiters for the DEF and PDEF writers.

You can also change the control defaults while working in the floorplan. These defaults include the snapping of the module guides, minimum module guides, minimum flight line connection width, and route congestion.

For information on setting design preferences, see [Design – Preferences](#) in the *Encounter Menu Reference*.

## Initialization Files

The Encounter software uses the following initialization files for setting preferences:

.encrc	Used for setting Tcl parameters or adding user-defined Tcl commands. If different versions of this file exist in the installation, home, or working directories, the file in the working directory takes precedence.  <b>Note:</b> Usage of this file is no longer recommended, but is allowed for backward compatibility. Use <code>enc.tcl</code> instead. This file is processed before the GUI is created, so it cannot be used to customize the GUI.
enc.tcl	Used for setting Tcl parameters, customizing the GUI, or adding user-defined Tcl commands or global variables. If different versions of this file exist in the installation, home, or working directories, the file in the working directory takes precedence.  <b>Note:</b> The Encounter software does not create or modify this file. You must create the file and then put a copy of the file in the installation directory ( <code>encounter_installation_path/tools/fe/etc</code> ), home directory, or working directory.
enc.pref.tcl	Contains design preferences set using the Design, Display, Floorplan, and Selection tabs in the Preferences form in the GUI (see <a href="#">Design – Preferences</a> in the <i>Encounter Menu Reference</i> ).
.enc	Contains design preferences set using the Windows tab in the Preferences form in the GUI (see <a href="#">Design – Preferences</a> in the <i>Encounter Menu Reference</i> ).

The initialization files are read in the following sequence:

1. .encrc in the home directory
2. .encrc in the working directory
3. enc.pref.tcl in the working directory

## **Encounter User Guide**

### Getting Started

---

4. `.enc` in the home directory
5. `enc.tcl` in the *installation/etc* directory
6. `enc.tcl` in the home directory
7. `enc.tcl` in the working directory

**Note:** If initialization files contain conflicting information, the last file read takes precedence.

## Starting the Software

To start an Encounter session, type one of the following commands with the appropriate parameters on the UNIX/Linux command line. If you type a command without parameters, the software starts in GUI mode and creates a log file and a command file. The system attempts to check out the license with the most functionality, then the license with the next most functionality, and so on.

■ velocity

Starts one of the following products:

- Encounter Digital Implementation System L
- Encounter Digital Implementation System XL

■ encounter

Starts one of the following products:

- First Encounter<sup>TM</sup> L
- First Encounter XL
- First Encounter GXL
- SoC<sup>TM</sup> Encounter
- SoC Encounter
- SoC Encounter
- NanoRoute Ultra
- Virtuoso<sup>®</sup> Digital Implementation

For an overview of the products and product licensing, see [“Product and Licensing Information.”](#)

## velocity

```
velocity
  [-edpl | -edpxl | -eds1 | -edsxl | -nru | -socegxl | -socel | -socexl |
   -vdi [-N{1 | 2}]}
  [-checkoutList "option1 option2..." ]
  [-cmd -cmdFile.cmd]
  [-config configFile.conf]
  [-execute "command_list"]
  [-help]
  [-init initFile.tcl]
  [-libDefFile libDefFile.defs]
  [-log logFile.log]
  [-nowin | -win]
  [-optionList "option1 option2..." ]
  [-overwrite]
  [-version]
  [-wait time_in_minutes]
```

## Parameters

-checkoutList "option1 option2..."

Checks out licenses for the specified product options when the software starts and holds the licenses for the remainder of the session. The product options provide additional features to your base license.

If you specify an option that is not allowed with your base product, or an option without an available license, the software does not check out a license for that option and instead issues a warning message.

If you specify more than one option, begin and end the list with double quotation marks or braces.

Specify one or more of the following parameters:

**Note:** For information on these parameters, see the [“Product and Licensing Information” chapter](#).

cndc	CeltIC Nanometer Delay Calculator
encan	Encounter Advanced Node GXL option
enclp	Encounter Low Power GXL option
encng	Encounter Next Generation
	<b>Note:</b> This license is used to enable beta features.
enccms	Encounter Mixed Signal GXL option

## Encounter User Guide

### Getting Started

---

eps1	Encounter Power System L
epsx1	Encounter Power System XL
etsl	Encounter Timing System L
etsx1	Encounter Timing System XL
nru	NanoRoute Ultra Routing System

`-config configFile.conf`

Specifies the design input configuration file. For information on the configuration file, see “[Configuration File Variables](#)” in the *Encounter Text Command Reference*.

**Note:** The value of this parameter disables the value specified by the `-init` parameter.

`[-edpl | -edpxl | -eds1 | -edsx1 | -nru | -socegxl | -socel | -socexl | -vdi [-N{1 | 2}]]`

Checks out a base license for the specified product.

If no license is available for the product you specify, the software generates an error message and does not start.

**Note:** Use `-optionList` or `-checkoutList` to check out product options.

Specify one of the following products:

**Note:** For information on these parameters, see the “[Product and Licensing Information](#)” chapter.

-edpl	Encounter Design Planner L (This product is not available in the current release.)
-edpxl	Encounter Design Planner XL (This product is not available in the current release.)
-eds1	Encounter Digital Implementation System L
-edsx1	Encounter Digital Implementation System XL
-nru	NanoRoute Ultra SoC Routing System

## Encounter User Guide

### Getting Started

---

`-vdi [-N{1 | 2}]`

Virtuoso® Digital Implementation

VDI is limited to designs with a maximum of 50,000 instances. If you specify `-vdi`, the software checks out one VDI product by default. To check out two VDI licenses, type the following command:

`velocity -vdi -N2`

`-help` Outputs a brief description for each `velocity` parameter.

`-init initFile.tcl`

Specifies the Tcl file to read in at the start of the session and starts the software in non-GUI mode. When the command finishes executing the Tcl file, the software switches to GUI mode.

**Note:** The value of the value of the `-config` parameter overwrites the value of this parameter if both are specified.

`-libDefFile libraryDefinitionFile.defs`

Specifies the path and file name of the library definition file (`lib.defs`) for OpenAccess-based flows. If you do not specify this parameter, all OpenAccess-based operations refer to the `lib.defs` file in the current working directory.

`-log logFile.log`

Specifies a name for the log file. By default, the software saves log files in the run directory and increments them, for example, `encounter.log`, `encounter.log1`, `encounter.log2`, `encounter.log3`, and so on.

*Default:* `encounter.log`

`-nowin | -win`

Specifies whether the software starts in a GUI environment.

*Default:* `-win`

## Encounter User Guide

### Getting Started

---

`-optionList "option1 option2 ..."`

Specifies product options to check out dynamically. Even though you specify options with this parameter, the licenses for the options are not checked out until they are needed. The licenses are held for the duration of the session.

If you specify an product option that is not allowed with your base product, or an option without an available, the software does not check out that license and instead issues a warning message.

If you specify more than one product option, begin and end the list with double quotation marks or braces.

Specify one or more of the following parameters:

**Note:** For information on these parameters, see the [“Product and Licensing Information” chapter.](#)

cndc	CeltIC Nanometer Delay Calculator
encan	Encounter Advanced Node GXL option
enclp	Encounter Low Power GXL option
encnng	Encounter Next Generation

**Note:** This license is used to enable beta features.

enccms	Encounter Mixed Signal GXL option
eps1	Encounter Power System L
epsx1	Encounter Power System XL
etsgxl	Encounter Timing System GXL
etsl	Encounter Timing System L
etsx1	Encounter Timing System XL
nru	NanoRoute Ultra Routing System

`-overwrite`

Overwrites the existing log file.

`-version`

Displays the version of Encounter software installed on the host machine without checking out a license or starting the software.

## **Encounter User Guide**

### Getting Started

---

`-wait time_in_minutes`

Specifies the amount of time the system waits for a license to become available. If the license is available in less than the specified wait time, the system checks out the next needed license without waiting.

*Default:* 0 (no wait time)

*Value range:* 0 to 10,000

## encounter

```
encounter
  [{ -ultra | -nru | -vdi [-N{1 | 2} |
    -fel | -fexl | -fegxl | -socel | -socexl | -socegxl }]
  [-checkoutList "lic1 lic2 ..." ]
  [-cmd file.cmd]
  [-config configFile.conf]
  [-help]
  [-init initFile.tcl]
  [-libDefFile libDefFile.defs]
  [-log logFile.log]
  [-nowin | -win]
  [-optionList "lic1 lic2 ..." ]
  [-overwrite]
  [-version]
  [-wait time_in_minutes]
```

## Parameters

`-checkoutList "option1 option2..."`

Checks out licenses for the specified product options when the software starts and holds the licenses for the remainder of the session. The product options provide additional features to your base license.

If you specify an option that is not allowed with your base product, or an option without an available license, the software does not check out a license for that option and instead issues a warning message.

If you specify more than one option, begin and end the list with double quotation marks or braces.

Specify one or more of the following parameters:

**Note:** For information on these parameters, see the [“Product and Licensing Information” chapter](#).

cndc	CeltIC Nanometer Delay Calculator
encan	Encounter Advanced Node GXL option
enclp	Encounter Low Power GXL option
encms	Encounter Mixed Signal GXL option
encng	Encounter Next Generation

**Note:** This license is used to enable beta features.

## Encounter User Guide

### Getting Started

---

eps1	Encounter Power System L
epsxl	Encounter Power System XL
etsgxl	Encounter Timing System GXL
etsl	Encounter Timing System L
etsxl	Encounter Timing System XL
nru	NanoRoute Ultra Routing Solution

-config *configFile.conf*

Specifies the design input configuration file. For information on the configuration file, see “[Configuration File Variables](#)” in the *Encounter Text Command Reference*.

**Note:** The value of this parameter disables the value specified by the -init parameter, if both are specified.

{-fegxl | -fel | -fexl | -nru | -socegxl | -socel | -socexl | -ultra | -vdi [-N{1 | 2}]} }

Checks out a base license for the specified product.

If the license you specify is not available, the software generates an error message and does not start.

Specify one of the following licenses:

**Note:** For information on these parameters, see the “[Product and Licensing Information](#)” chapter.

-fegxl	First Encounter GXL
-fel	First Encounter L
-fexl	First Encounter XL
-nru	NanoRoute® Ultra
-socegxl	SoC Encounter™ GXL
-socel	SoC Encounter L
-socexl	SoC Encounter XL
-ultra	First Encounter GXL

## Encounter User Guide

### Getting Started

---

`-vdi [-N{1 | 2}]`

Virtuoso® Digital Implementation

The VDI option is limited to designs with a maximum of 50,000 instances. If you specify `-vdi`, the software checks out one VDI option by default. To check out two VDI licenses, type the following command:

`encounter -vdi -N2`

`-help` Outputs a brief description for each `encounter` parameter.

`-init initFile.tcl`

Specifies the Tcl file to read in at the start of the session and starts the software in non-GUI mode. When the command finishes executing the Tcl file, the software switches to GUI mode.

**Note:** The value of the `-config` parameter disables the value specified by this parameter, if both are specified.

`-libDefFile libraryDefinitionFile.defs`

Specifies the path and file name of the library definition file (`lib.defs`) for OpenAccess-based flows. If you do not specify this parameter, all OpenAccess-based operations refer to the `lib.defs` file in the current working directory.

`-log logFile.log`

Specifies a name for the log file. By default, the software saves log files in the run directory and increments them, for example, `encounter.log`, `encounter.log1`, `encounter.log2`, `encounter.log3`, and so on.

*Default:* `encounter.log`

`-nowin | -win`

Specifies whether the software runs in a GUI environment.

*Default:* `-win`

## Encounter User Guide

### Getting Started

---

-optionList "*option1 option2 ...*"

Specifies product options to check out dynamically. Even though you specify options with this parameter, the licenses for the options are not checked out until they are needed. The licenses are held for the duration of the session.

If you specify an product option that is not allowed with your base product, or an option without an available, the software does not check out that license and instead issues a warning message.

If you specify more than one product option, begin and end the list with double quotation marks or braces.

Specify one or more of the following parameters:

**Note:** For information on these parameters, see the [“Product and Licensing Information” chapter](#).

cndc	CeltIC Nanometer Delay Calculator
encan	Encounter Advanced Node GXL option
enclp	Encounter Low Power GXL option
encms	Encounter Mixed Signal GXL option
encng	Encounter Next Generation
	<b>Note:</b> This license is used to enable beta features.
eps1	Encounter Power System L
epsxl	Encounter Power System XL
etsl	Encounter Timing System L
etsxl	Encounter Timing System XL
nru	NanoRoute Ultra Routing Solution
-overwrite	Overwrites the existing log file.
-version	Displays the version of Encounter software installed on the host machine without checking out a license or starting the software.

`-wait time_in_minutes`

Specifies the amount of time the system waits for a license to become available. If the license is available in less than the specified wait time, the system checks out the next needed license without waiting.

*Default:* 0 (no wait time)

*Value range:* 0 to 10,000

## Using Generic Parameters to Specify 32- or 64-Bit Version

When you start the software, complete the following steps:

1. Specify one of the following parameters:

{ -32 | -64 | -32only | -64only | -3264 | -6432 }

-32	Tries to run the 32-bit version of the application. If the 32-bit version is not available, prints a warning and tries to run the 64-bit version.
-64	Tries to run the 64-bit version of the application. If the 64-bit version is not available, prints a warning and tries to run the 32-bit version.
-32only	Tries to run the 32-bit version of the application. If the 32-bit version is not available, prints an error and exits with an error.
-64only	Tries to run the 64-bit version of the application. If the 64-bit version is not available, prints an error and exits with an error.
-3264	Tries to run the 32-bit version of the application. If the 32-bit version is not available, prints an info and tries to run the 64-bit version.
-6432	Tries to run the 64-bit version of the application. If the 64-bit version is not available, prints an info and tries to run the 32-bit version.

**Note:** If the CDS\_AUTO\_64BIT environment variable is not set and one of the following parameters is specified, the wrapper sets CDS\_AUTO\_64BIT to NONE :

- 32
- 32only
- 3264

If the CDS\_AUTO\_64BIT environment variable is not set and one of the following parameters is specified, the wrapper sets CDS\_AUTO\_64BIT to ALL:

- 64
- 64only
- 6432

**2. Optionally, specify one or more of the following parameters:**

`[-quiet3264] [-debug3264] [-plat platform] [-v3264] [-help3264]`

`-debug3264` Prints the environment, updated by the wrapper and the command launched.

`-plat platform` Allows you to override the default platform selection when you launch the tool from the following directory:  
*install\_root/bin*

`-quiet3264` Suppresses warning, error, and info messages generated by the -32, -32only, -3264, -64, -64only, or -6432 parameters.

`-v3264` Prints the wrapper's version string.

## Interrupting the Software

You can interrupt an Encounter session by using the Ctrl-C key combination. For most commands, Ctrl-C exits the session and causes the software to issue the following message:

Interrupt—one more Ctrl-C to exit First Encounter ...

- If you do not press Ctrl-C again, the software proceeds as normal.
- If you do press Ctrl-C again, the software stops and the session ends.

## Interrupting the NanoRoute Router and Timing Optimization (optDesign)

The behavior of the software when you use Ctrl-C differs for the following long-running commands:

- `routeDesign`

- `globalDetailRoute`

For information, see [Interrupting Routing](#) in the “Using the NanoRoute Router” chapter.

- `optDesign`

For information, see [Interrupting Timing Optimization](#) in the “Optimizing Timing” chapter.

## Interrupting the Execution of Batch Files

The behavior of the software when you use `Ctrl-C` differs if you interrupt the execution of a batch script.

When you press `Ctrl-C` during the execution of a batch script, the command that is running when you press `Ctrl-C` continues to completion. The software then stops and prompts you to confirm whether to interrupt the script.

- To confirm that you want to interrupt script, type `Y`.  
In this case, you can save the design and proceed with the flow.
- To continue running the script, type `N`.

## Stopping the Software

Use one of the following methods to stop the software:

- In the main Encounter window, select *Design – Exit*.
- On the text command line, type the following command:  
`exit`

## Using the Log File Viewer

The Encounter software provides the following methods to view the log file:

- [Integrated Log File Viewer](#) on page 75
- [Standalone Log File Viewer](#) on page 76

### Integrated Log File Viewer

You can use the integrated log file viewer when the software is running. It has the following features:

- Ability to expand and collapse command information.
- Ability to view multiple log files in separate console windows simultaneously.
- Color coding of error, warning, and information messages.
- String matching through the *Edit – Find>Select Object* menu.

For more information, see [Find>Select Object](#) in the “Edit Menu” chapter of the *Encounter Menu Reference*.

- Access to the documentation in the *Encounter Text Command Reference*.

When a log file is displayed, click on any of the underlined commands to open an HTML window that displays the documentation for that command.

Use one of the following methods to use the viewer:

- Select *Tools – Log Viewer* on the main Encounter menu.

The *Log File* window is displayed. Select the log file to view. The software opens a separate console window and displays the log file.

For more information, see [Tools – Log Viewer](#) in the “Tools Menu” chapter of the *Encounter Menu Reference*.

- On the text command line, type the following command in the console window where the Encounter software is running:

`viewLog [-file logFileName]`

This command opens the log file in a separate window. It opens the most recently created log file unless you specify a different log file with the `-file` parameter.

## **Standalone Log File Viewer**

You can use the standalone viewer even if the software is not running. It provides most of the same functionality as the viewer that is run within the Encounter software but does not provide access to the documentation.

To use the standalone viewer, type the following UNIX/Linux command in the console window:

```
viewlog [-file logFileName]
```

The viewer opens the most recently created log file unless you specify a different file with the *-file* parameter.

## Accessing Documentation and Help

You can access the Encounter documentation and help system by using the following methods:

- [Launching Cadence Help From the Command Prompt](#) on page 77
- [Accessing Documentation and Help From the Encounter GUI](#) on page 77
- [Using the Encounter man and help Commands on the Text Command Line](#) on page 79
- [Using the Integrated Log File Viewer](#) on page 81
- [Other Sources of Information](#) on page 81

### Launching Cadence Help From the Command Prompt

1. Change to the following directory:

*installation\_dir/tools/bin*

2. Enter the following command:

`./cdnshelp`

After launching Cadence® Help, press F1 or choose *Help – Contents* to display the help page for Cadence Help.

For more information see the [Cadence Help](#) manual.

### Accessing Documentation and Help From the Encounter GUI

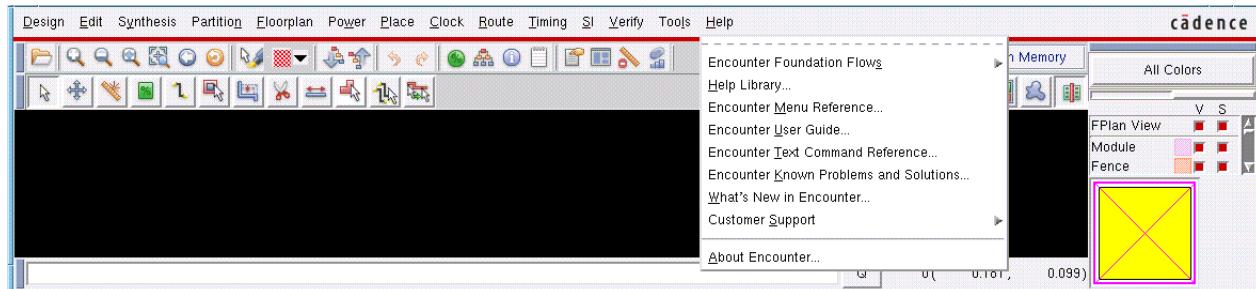
The software provides the following two methods to access documentation and help from the GUI:

- [Select Help on the Main Encounter Menu](#) on page 78
- [Select Help on an Encounter Form](#) on page 79

# Encounter User Guide

## Getting Started

### Select Help on the Main Encounter Menu



- Select *Help*, and then any of the following options:

- Encounter Foundation Flows*

Provides access to the following documents:

- Encounter Foundation Flows User Guide*
  - Encounter Flat Implementation Flow Guide*
  - Encounter Hierarchical Implementation Flow Guide*

Click on any of the document titles to go to the document's Table of Contents page.

- Help Library*

Opens the Cadence Help window, which provides access to all the documentation shipped with the release.

- Encounter Menu Reference*

Opens the Table of Contents page of the menu reference.

- Encounter User Guide*

Opens the Table of Contents page of the user guide.

- Encounter Text Command Reference*

Opens the Table of Contents page of the text command reference.

- Encounter Known Problems and Solutions*

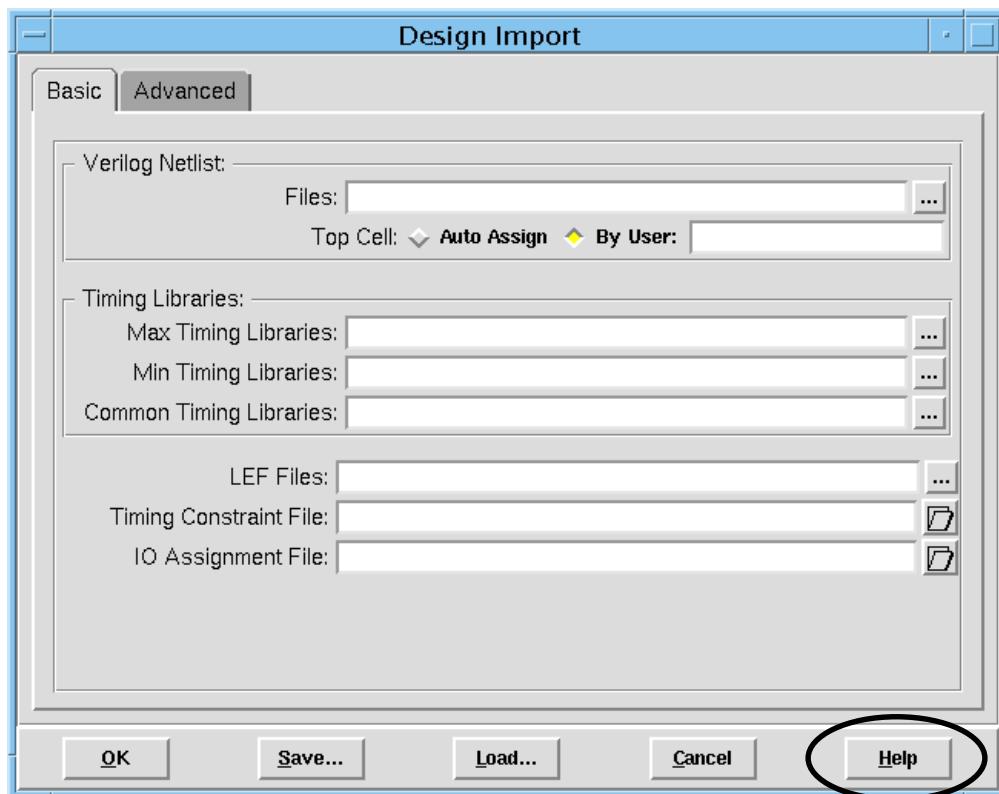
Opens the Table of Contents page of the known problems and solutions document.

- What's New in Encounter*

Opens the Table of Contents page of the what's new document.

### Select Help on an Encounter Form

- Click the *Help* button in the bottom right corner of a form.



Clicking the *Help* button opens the [\*Encounter Menu Reference\*](#) entry for the form in the Cadence Help window.

### Using the Encounter man and help Commands on the Text Command Line

#### Using the help Command

- To see syntax information for a command, type the following command in the software console:

```
help command_name
```

For example, to see syntax information for the `loadConfig` command, type the following command:

```
help loadConfig
```

The software displays the following text:

# Encounter User Guide

## Getting Started

---

```
Usage: loadConfig <fileName> [0 | 1]
```

- To see the entire list of Encounter commands and their syntax, type the following command in the software console:

```
help
```

### Using the man Command

- To see the complete set of information for an Encounter command, type the following command in the software console:

```
man command_name
```

For example, to see the complete set of information for the `loadConfig` command, type the following command:

```
man loadConfig
```

The software displays the following text:

User Commands

`loadConfig(1)`

NAME

`loadConfig`

SYNTAX

```
loadConfig <fileName> [0 | 1]
```

DESCRIPTION

Loads a configuration file. If you use this command in batch mode and specify a filename, the file is loaded and the design is imported. If you specify a filename and the 0 parameter, the software loads the file, but does not import the design. To synchronize all relative paths in the configuration file to the current working directory, you can precede the `loadConfig` command with the `setImportMode -syncRelativePath true` command. You can load this configuration file from any directory without first changing your current working directory to the previous working directory where the configuration file was saved. You can use the `loadconfig` command to import the design when used in batch mode. You can use it only once in a design session. For more information on the variables that can be set in the configuration file, see Configuration File Variables.

Parameters

`<fileName>`

Specifies the configuration file to load.

## Encounter User Guide

### Getting Started

---

[ 0 | 1 ]

Specifies whether to apply settings in the configuration file. 1 loads the configuration file and imports the design. 0 loads the configuration file and does not import the design.

Default: If you do not specify this parameter, 1 is selected.

#### Example

- The following command loads mydesign.conf file:

```
loadConfig mydesign.conf
```

- The following command loads mydesign.conf, but does not apply the settings in the file:

```
loadConfig mydesign.conf 0
```

This populates the fields in the Design Import form.

Click OK to commit settings.

## Using the Integrated Log File Viewer

You can also access the command documentation by using the integrated log file viewer. The command to start the viewer is `viewLog`. For more information see “[Integrated Log File Viewer](#)” on page 75 or `viewLog` in the “General Commands” chapter of the *Encounter Text Command Reference*.

## Other Sources of Information

You can also get help on Cadence products by selecting *Customer Support* on the *Help* menu. The *Customer Support* submenu provides access to the following Cadence resources:

- SourceLink®
  - Opens SourceLink in your browser.
- Web Collaboration
  - Opens SpaceCruiser in your browser.
- Education Services

## **Encounter User Guide**

### Getting Started

---

Opens the Education Services website in your browser.

---

## Accelerating the Design Process By Using Multiple-CPU Processing

---

- [Overview](#) on page 84
- [Running Distributed Processing](#) on page 86
- [Running Multi-Threading](#) on page 87
- [Running Superthreading](#) on page 88
- [Setting and Changing the License Check-Out Order](#) on page 88
- [Limiting the Multi-CPU License Search to Specific Products](#) on page 88
- [Releasing Licenses Before the Session Ends](#) on page 89
- [Controlling the Level of Usage Information in the Log File](#) on page 89
- [Where to Find More Information on Multi-CPU Licensing](#) on page 89

## Encounter User Guide

Accelerating the Design Process By Using Multiple-CPU Processing

---

## Overview

You can accelerate portions of the design flow by using multiple-CPU processing. The Encounter® software has the following multiple-CPU modes:

- Multi-threading

In this mode, a job is divided into several threads, and multiple processors in a single machine process them concurrently.

- Distributed processing

In this mode, a job is processed by two or more networked computers running concurrently.

- Superthreading

In this mode, a job runs in both multi-threading and distributed processing mode; that is, two or more networked computers, each with multiple processors, work concurrently to complete a job.

You configure multiple-CPU processing by using the commands described in the [Multiple-CPU Processing Commands](#) chapter of the *Encounter Text Command Reference* or the [Multiple CPU Processing](#) form on the *Design* menu.

[Table 3-1](#) on page 84 shows the Encounter features that support multiple-CPU processing. In the table, DP stands for distributed processing, MT stands for multi-threading, and ST stands for Superthreading.

**Table 3-1 Encounter features that support multiple-CPU processing**

Feature	Command	DP	MT	ST	Limitations/Notes
Capacitance table generation	<a href="#">generateCapTbl</a>	x			For more information, see <a href="#">Capacitance Table Generation Flow</a> in the “RC Extraction” chapter.

## Encounter User Guide

Accelerating the Design Process By Using Multiple-CPU Processing

---

Feature	Command	DP	MT	ST	Limitations/Notes
Global placement	<a href="#"><u>placeDesign</u></a>		x		<ul style="list-style-type: none"> <li>■ Supported in default mode only (-modulePlan is true)</li> </ul> <p>For more information, see <a href="#"><u>Running Placement in Multi-CPU Mode</u></a> in the “Placing the Design” chapter.</p>
Automatic floorplan synthesis	<a href="#"><u>multiPlanDesign</u></a>		x		For more information, see <a href="#"><u>Creating Multiple Alternative Floorplans</u></a> in the “Creating an Initial Floorplan Using Masterplan” chapter.
Metal fill	<a href="#"><u>addMetalFill</u></a>		x		For more information, see <a href="#"><u>Adding Metal Fill in Multiple-CPU Processing Mode</u></a> in the “Optimizing Metal Density” chapter.
NanoRoute router	<a href="#"><u>globalRoute</u></a> <a href="#"><u>globalDetailRoute</u></a> <a href="#"><u>detailRoute</u></a> <a href="#"><u>routeDesign</u></a> <a href="#"><u>ecoRoute</u></a>	x	x	x	<ul style="list-style-type: none"> <li>■ Superthreading is supported for detailed routing only.</li> <li>■ Superthreading options take precedence over multi-threading options.</li> </ul> <p>For more information, see <a href="#"><u>Accelerating Routing with Multi-Threading and Superthreading</u></a> in the “Using the NanoRoute Router” chapter.</p>
QRC	<a href="#"><u>runQRC</u></a>		x		

## Encounter User Guide

Accelerating the Design Process By Using Multiple-CPU Processing

Feature	Command	DP	MT	ST	Limitations/Notes
Signal integrity analysis	<code>optDesign -postRoute -si</code> <code>timeDesign -si</code> ■ In MMMC mode ■ In non-MMMC mode	x	x	x	■ For backward compatibility, distributed processing options take precedence. ■ Superthreading options take precedence over multi-threading options.
Verify geometry	<code>verifyGeometry</code>	x			For more information, see <a href="#">Multi-CPU Processing Settings</a> in the “Analyzing and Repairing Crosstalk” chapter. For more information, see <a href="#">Verifying Geometry in Multi-Thread Mode</a> in the “Verifying Violations” chapter.

### Related Topics

- [setup.tcl](#) in the *Encounter Flat Implementation Flow Guide*.
- [mode.tcl](#) in the *Encounter Flat Implementation Flow Guide*.
- [metal\\_fill.tcl](#) in the *Encounter Flat Implementation Flow Guide*.
- [Route the Design and Run Postroute Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

## Running Distributed Processing

To run the software in distributed processing mode, the following two commands are required:

- [setDistributeHost](#)

## Encounter User Guide

### Accelerating the Design Process By Using Multiple-CPU Processing

---

Use this command to specify a configuration file for distributed processing or create the configuration for the remote shell, secure shell, or load-sharing facility queue to use for distributed processing. If you request more machines than are available, most applications wait until all requested machines are available.

To display the current setting for `setDistributeHost`, use the [getDistributeHost](#) command.

- [setMultiCpuUsage](#)

Use this command to specify the maximum number of computers to use for processing.

To display the current setting for `setMultiCpuUsage`, use the [getMultiCpuUsage](#) command.

For example, to run the `multiPlanDesign` command in distributed processing mode on four machines with a in an existing LSF environment on machines that have 4 GB of memory, specify the following commands:

```
setDistributeHost -lsf -queue mem4G  
setMultiCpuUsage -numHosts 4  
multiPlanDesign -autoTrials 4
```

## Running Multi-Threading

To run the software in multi-threading mode, the following command is required:

- [setMultiCpuUsage](#)

Use this command to specify the number of threads to use. Upon completion, the log file generated by each thread is appended to the main log file.

**Note:** The `-numThreads` parameter limits the number of threads running concurrently. Although the software can create additional threaded jobs during run time, depending on the application in use, only the number of threads specified with this parameter are run at a given time.

If you ask for more threads than are available, the software issues a warning and runs with the maximum number of available threads.

For example, to run placement with four threads, specify the following commands:

```
setMultiCpuUsage -numThreads 4  
placeDesign
```

## Running Superthreading

To run the Encounter software in Superthreading mode, the following two commands are required:

- setDistributeHost
- setMultiCpuUsage

Because Superthreading is multi-threading plus distributed processing, you must specify the number of threads and the number of hosts. If you request more machines than are available, most applications wait until all requested machines are available.

For example, to run the NanoRoute router in Superthreading mode, using a load-sharing facility queue with two machines and three processors each, specify the following commands:

```
setDistributeHost -lsf -queue myQueue -resource "mem>4000 OS=RH4"  
setMultiCpuUsage -superThreadsNumHosts 2 -superThreadsNumThreads 6  
detailRoute
```

## Setting and Changing the License Check-Out Order

To change the license check-out order, use the following command:

getMultiCpuLicense

For information on the default check-out order, see [SoC Encounter Licensing and Packaging](#) and [Encounter Digital Implementation System Licensing and Packaging](#) on SourceLink®.

## Limiting the Multi-CPU License Search to Specific Products

By default, the software checks for any available license for multiple-CPU processing. You can limit the license search to specific products.

For example, if the base license is for First Encounter® GXL and you do not want to consume any SOC Encounter™ GXL licenses for multi-CPU applications, use the following command:

getMultiCpuLicense -licOptions "fel fexl soce socel socexl"

## Releasing Licenses Before the Session Ends

By default, the software holds multi-CPU licenses for the duration of the current session. To release the multi-CPU licenses before the Encounter session ends, complete one of the following steps:

- Before running any multi-CPU applications, specify the following command:

```
setReleaseMultiCpuLicense true
```

To display the current setting for `setReleaseMultiCpuLicense`, use the `getReleaseMultiCpuLicense` command.

- At the point when you want to release the multi-CPU licenses (for example, when global placement finishes), specify the following command:

```
releaseMultiCpuLicense
```

## Controlling the Level of Usage Information in the Log File

Use the following command to set the level of usage information in the log file:

```
setMultiCpuUsage -threadInfo {0 | 1 | 2}
```

By default, the software does not write starting and ending information for threads or timing details to the log file, but you can change this behavior by specifying 1 or 2 for the `-threadInfo` parameter.

- Specify 1 to report the starting and ending information.
- Specify 2 to report the information above, plus the elapsed time, processor time, and additional details for each thread and the totals for all threads.

## Where to Find More Information on Multi-CPU Licensing

See [Encounter Digital Implementation System Licensing and Packaging](#) and [SoC Encounter Licensing and Packaging](#) on SourceLink®.

## **Encounter User Guide**

Accelerating the Design Process By Using Multiple-CPU Processing

---

---

## **Data Preparation**

---

- [Generating a Technology File](#) on page 92
- [Preparing Physical Libraries](#) on page 92
- [Unsupported LEF and DEF Syntax](#) on page 93
- [Generating the I/O Assignment File](#) on page 96
- [Preparing Timing Libraries](#) on page 114
- [Encrypting Libraries](#) on page 114
- [Preparing Stamp Models](#) on page 115
- [Preparing Timing Constraints](#) on page 115
- [Preparing Capacitance Tables](#) on page 116
- [Preparing Data for Delay Calculation](#) on page 116
- [Preparing Data for Crosstalk Analysis](#) on page 116
- [Checking Designs](#) on page 116
- [Preparing Data in the Timing Closure Design Flow](#) on page 117

## Generating a Technology File

The technology file provides the software with design rules for placement and routing, and interconnect resistance and capacitance data for generating RC values and wireload models for the design. The technology file also contains process information for the metal interconnect layers, including metal thickness, metal resistance, and line-to-line capacitance values of metal layers, for determining coupling capacitance.

### Creating Technology Information Using LEF

You can use the Library Exchange Format (LEF) to specify technology information. If you do not have LEF technology information, refer to the *LEF/DEF Language Reference* for details on specifying the information manually.

### Creating Technology Information Using OpenAccess

You can also create technology information equivalent to the information you specify in LEF, but in an OpenAccess database format. This allows you to share technology information easily among tools that support the OpenAccess standard.

## Preparing Physical Libraries

To run the Encounter software, you must create physical libraries (cells and macros).

If you have a complete LEF file that contains all cells in the design, and process technology information, then you can import a LEF file.

### Using LEF to Create Physical Libraries

You can use the following methods for creating abstracts for each leaf cell in the design.

- Use the Abstract Generator.

For more information, see the *Cadence Abstract Generator User Guide*.

- Create LEF MACROS manually.

For more information, see the *LEF/DEF Language Reference*.

## Creating OpenAccess Physical Libraries

You can translate the LEF MACROS to OpenAccess format by using a LEF-to-OpenAccess translator. This allows you to share libraries easily among tools supporting OpenAccess standard.

## Unsupported LEF and DEF Syntax

The Encounter software supports most of the syntax statements in the 5.6 versions of LEF and DEF with the exception of the ones listed below.

### Unsupported LEF 5.6 Syntax

The Encounter software parses but ignores the following LEF 5.6 syntax:

LEF Statement	Unsupported Syntax
Layer (Routing)	[DIAGWIDTH <i>diagWidth</i> ;] [DIAGSPACING <i>diagSpacing</i> ;] [DIAGMINEDGELENGTH <i>diagLength</i> ;] [SLOTWIREWIDTH <i>minWidth</i> ;] [SLOTWIRELENGTH <i>minLength</i> ;] [SLOTWIDTH <i>minWidth</i> ;] [SLOTLLENGTH <i>minLength</i> ;] [MAXADJACENTSLOTSPACING <i>spacing</i> ;] [MAXCOAXIALSLOTSPACING <i>spacing</i> ;] [MAXEDGESLOTSPACING <i>spacing</i> ;] [SPLITWIREWIDTH <i>minWidth</i> ;] [HEIGHT <i>distance</i> ;] [SHRINKAGE <i>distance</i> ;] [CAPMULTIPLIER <i>value</i> ;]
Macro Pin	[TAPERRULE <i>ruleName</i> ;] [NETEXPR “ <i>netExprPropName defaultNetName</i> ” ;] [SUPPLYSENSITIVITY <i>powerPinName</i> ;] [GROUNDSENSITIVITY <i>groundPinName</i> ;]
Nondefault Rule	[DIAGWIDTH <i>diagWidth</i> ;] [HARDSPACING ;] [MINCUTS <i>cutLayer numCuts</i> ;]

## Encounter User Guide

### Data Preparation

The following LEF 5.6 syntax causes an error message in the Encounter software:

LEF Statement	Unsupported Syntax
Layer (Routing)	DIRECTION {DIAG45   DIAG135} ;
Nondefault Rule	[USEVIARULE <i>viaRuleName</i> ; ]
Via	POLYGON <i>pt pt pt ...</i> ;
Via Rule Generate	[DEFAULT]

## Unsupported DEF 5.6 Syntax

The Encounter software parses but ignores the following DEF 5.6 syntax:

DEF Statement	Unsupported Syntax
Blockages	[+ SLOTS]
Groups	[+ PROPERTY { <i>propName propValue</i> }...]
Extensions	All BEGINEXT syntax
History	All HISTORY syntax
Nets	<p>[+ SYNTHESIZED]</p> <p>[+ VPIN <i>vpinName</i> [LAYER <i>layerName</i>] <i>pt pt</i> [PLACED <i>pt orient</i>   FIXED <i>pt orient</i>   COVER <i>pt orient</i>]]</p> <p>[+ SUBNET <i>subNetName</i> [ ( {<i>compName pinName</i>   PIN <i>pinName</i>   VPIN <i>vpinName</i>} ) ] [NONDEFAULTRULE <i>ruleName</i> ]]</p> <p><b>Note:</b> SUBNET NONDEFAULTRULE is ignored; routing uses rule for NET.</p> <p>[+ USE {RESET   SCAN   TIEOFF}]</p> <p><b>Note:</b> Supports ANALOG, CLOCK, GROUND, POWER, and SIGNAL.</p> <p>[+ PATTERN {STEINER   WIREDLOGIC}]</p> <p>[+ ESTCAP <i>wireCapacitance</i>]</p> <p>[+ SOURCE {DIST   NETLIST   TEST   USER}]</p>

## Encounter User Guide

### Data Preparation

---

<b>DEF Statement</b>	<b>Unsupported Syntax</b>
Pins	<p>[ + USE {TIEOFF   SCAN   RESET}]</p> <p><b>Note:</b> Supports SIGNAL, POWER, GROUND, ANALOG, and CLOCK.</p> <p>[ + DIRECTION FEEDTHRU]</p> <p>[ + NETEXPR "netExprPropName defaultNetName"]</p> <p>[ + SUPPLYSENSITIVITY powerPinName ]</p> <p>[ + GROUNDSENSITIVITY groundPinName ]</p> <p>[ + COVER <i>pt orient</i> ]</p> <p><b>Note:</b> + COVER is ignored and converted to + FIXED</p>
Pin Properties	All PINPROPERTIES syntax
Property Definitions	The object types: GROUP, REGION, and ROW
Regions	[ + PROPERTY { <i>propName propVal</i> }... ]
Rows	[ + PROPERTY { <i>propName propVal</i> }... ]
Slots	All SLOTS syntax
Special Nets	<p>[ + SYNTHESIZED]</p> <p>[ + VOLTAGE <i>volts</i>]</p> <p>[ + SOURCE {DIST   NETLIST   USER} ]</p> <p>[ + USE {RESET   SCAN   TIEOFF} ]</p> <p><b>Note:</b> Supports ANALOG, CLOCK, GROUND, POWER, and SIGNAL.</p> <p>[ + PATTERN {STEINER   WIREDLOGIC} ]</p> <p>[ + ESTCAP <i>wireCapacitance</i> ]</p> <p>[ + WEIGHT <i>weight</i> ]</p> <p><b>Note:</b> + WEIGHT only supported in NETS section.</p> <p>Special Wiring Statement:</p> <p>[ + STYLE <i>styleNum</i> ]</p> <p><b>Note:</b> If included in the DEF file, the software displays an error message stating that only the default style is supported, ignores the specified style, and replaces it with the default one.</p>
Styles	All STYLES syntax

## Encounter User Guide

### Data Preparation

---

The following syntax causes an error message in the Encounter software:

DEF Statement	Unsupported Syntax
Blockages	[ + COMPONENT <i>compName</i> ]
Nets (Regular Wiring Statement)	[ <i>orient</i> ] [STYLE <i>styleNum</i> ]
Vias	+ POLYGON <i>layerName</i> <i>pt pt pt...</i>

## Generating the I/O Assignment File

The I/O assignment file defines the rules that determine how the I/O instances (pad cells and area I/O), I/O pins, bumps, and bump arrays are organized. The file is rule-based to specify exact location, global spacing, individual spacing, skip, offset, keep clear, and corner information. You can specify detailed rules to control the locations, or you can specify minimal or no rules to allow Encounter to determine the locations automatically.

Encounter does not require you to create an I/O assignment file to run the software. If you do not specify an I/O assignment file when you import a design, I/Os are assigned randomly.

If you do not specify an I/O assignment file, but you want to set I/O pin or pad placement, use a DEF file. Load the DEF file after importing the design, then save the floorplan. You can also save the I/O file to write a sequence file for rule-based work.

If you provide an I/O assignment file, you are not required to specify the exact location of all I/O pads. You can specify the I/O row name to place the I/O pads in a specific I/O row. Also, if you do not provide offset values, Encounter spaces the I/O pads evenly along the specified row. The spacing between the corners and adjacent pads is the same as the spacing between the other pads.

This section discusses the following topics:

- [Creating an I/O Assignment File](#) on page 97
- [Creating a Rule-Based I/O Assignment File](#) on page 107
- [I/O Pad and Pin Assignment Examples](#) on page 108
- [Performing Area I/O Placement](#) on page 111

## **Creating an I/O Assignment File**

You manually create an I/O assignment file using the following template:

```
(globals
    version = 3
    io_order = clockwise
    total_edge = 10
    space = 1.06
)
(row_margin
    (top | north | left | west | right | east | bottom | south
        (io_row ring_number = 1 margin = 0.0000)
        (io_row ring_number = 2 margin = 94.0000)
        (io_row ring_number = 3 margin = 181.0000)
    )
    ( ...
    )
)
(iopad
    (top | north | left | west | right | east | bottom | south | row
    (locals
        row_name = name_of_row
        space = 1.2
        ring_number = 1
        io_order = counterclockwise
    )
)
(inst
    name = ioinst_1
    skip = 2.2
    space = 1.2
    offset = 10.2
```

## Encounter User Guide

### Data Preparation

---

```
    indent = 10.2
    orientation = r180
    place_status = fixed
)
(keepclear begin = 10.0 end = 12.0)
(inst
    name = ioinst_2
    orientation = r180
    skip = 2.2
    cell = mymaster
)
(endspace gap = 1.2)
)

# corner io cell
(topright | northeast | topleft | northwest | bottomright | southeast |
bottomleft | southwest | row
(locals
    row_name = name_of_row
    ring_number = 1
)
(inst
    name = corner_1
    orientation = r180
    cell = corner_master
)
)
(inst
    name = ioinst_2
    x = 100.0
    y = 200.0
    orientation = r180
    place_status = fixed
)
)

(iopin
    (top | north | edge num = 0

        (locals
            space = 1.2
            io_order = counterclockwise
        )
    )
)
```

## Encounter User Guide

### Data Preparation

---

```
(pin name = "din [0]"
layer = 3
width = 0.28
depth = 0.28
skip = 2.2
space = 1.2
offset = 10.2
place_status = fixed
)
)
(left | west | edge #
)
(right | east | edge #
)
(bottom | south | edge #
)
)
(bump
(array
    name = array_1
    cell = bcl
    llx = 100
    lly = 100
    urx = 100
    ury = 100
    x = 100.0
    y = 200.0
    xpitch = 20
    ypitch = 20
    xspace = 10
    yspace = 10
    row = 6
    column = 6
    out_rings = 3
    style = stagger | full
    center_column = 4
    center_row = 4
    center_style = stagger | full
)
)
(bump
```

## Encounter User Guide

### Data Preparation

---

```
name = lvdsov33v12_ca_sref_i42_r1c1
cell = bcl
x = 100.0
y = 200.0
signal = vdd12
type = power | ground
assignment = fixed
array = array_1
orientation = r180
)
)
```

The following entries are included in the template:

#### globals

version = 3	Specifies the beginning of a new I/O format.
<i>io_order</i>	Specifies the order of the I/O pads and pins. This can be: <ul style="list-style-type: none"><li>■ clockwise</li><li>■ counterclockwise</li><li>■ default</li></ul>
	<b>Note:</b> The default I/O order for a vertical edge is from the bottom to the top, and for a horizontal edge, it is from the left to the right.
<i>total_edge</i>	Specifies the number of edges for the rectilinear block design.  The edges are numbered starting from 0. For example, if the <i>total_edge</i> is 4, then the edges are numbered as edge 0, edge 1, edge 2, and edge 3.
	<b>Note:</b> You must verify that the total number of edges that you specify matches with the value in the destination design.
<i>space</i>	Specifies the global I/O pin spacing, in $\mu$ meters.

#### iopad locals

## Encounter User Guide

### Data Preparation

---

	<i>space</i>	Specifies the local I/O pad spacing, in $\mu$ meters.
		<b>Note:</b> This space setting is honored by the first cell on one edge, when xy or offset is not specified.
	<i>ring_number</i>	Specifies the ring number in which the I/O pad is placed.
	<i>row_name</i>	Specifies the I/O <i>row</i> name.
iopad instance		
	<i>name</i>	Specifies the name of the I/O instance.
	<i>x, Y</i>	Specifies the absolute <i>x, y</i> location of the I/O pad instance, starting from the lower left corner.
		<b>Note:</b> Specifying <i>x,y</i> location for sides and edges of I/O pads is not supported in the I/O file.
	<i>skip</i>	Specifies the distance, in $\mu$ meters, of the I/O pad from the previously defined I/O pad.  The value that you specify here is valid only for this cell.

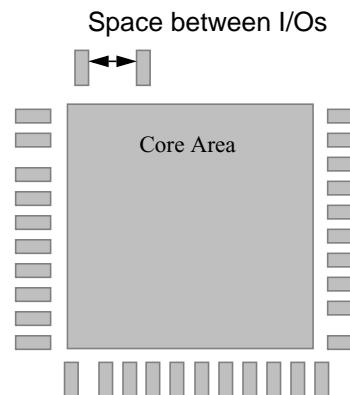
## Encounter User Guide

### Data Preparation

*space*

Specifies the spacing, in  $\mu$ meters, between the pad being defined and the previously defined pad.

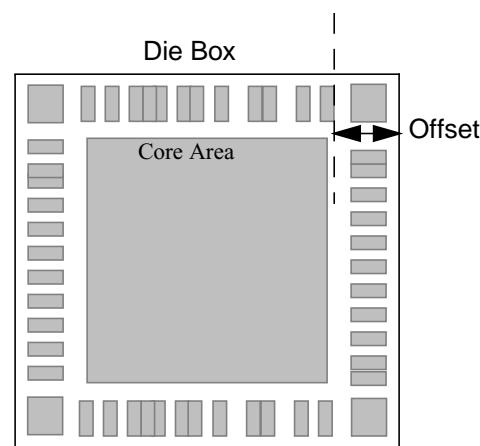
The value that you specify here, overrides the global space setting.



*offset*

Specifies the offset in  $\mu$ meters. The offset of a pad is the offset from the die boundary, based on the order direction.

The value that you specify here is valid only for this cell.



## Encounter User Guide

### Data Preparation

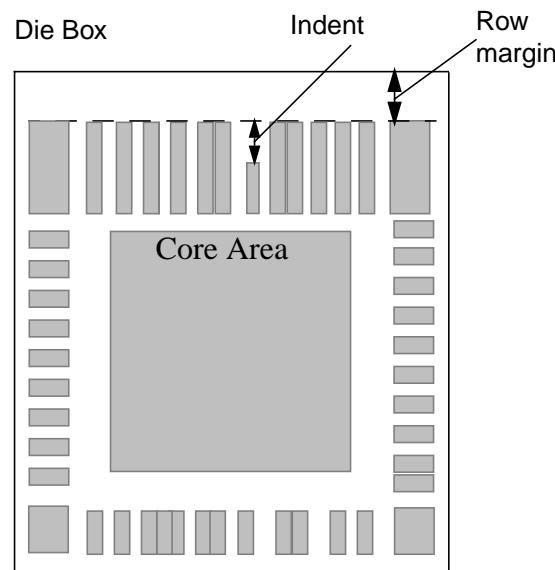
**Note:** For one I/O pad, you can specify only one of the following parameters:

- skip
- space
- offset

If you specify all the three parameters, only the last parameter that you define, is considered for I/O pad placement.

*indent*

Specifies the offset, in  $\mu$ meters, from the row margin.



However, for designs with single I/O ring, row margin is 0. Hence, indent is the offset of the I/O pad from the die boundary.

*orientation*

Specifies the orientation of the I/O.

*place\_status*

Specifies the placement status of the I/O pad instance. This can be:

- placed
- covered
- fixed

*Default:* fixed.

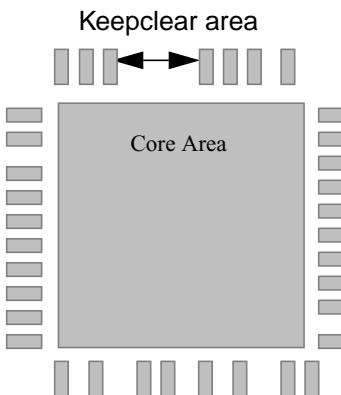
## Encounter User Guide

### Data Preparation

*keepclear*

Specifies an area on the chip where you cannot place pins or pads. Specify a range between *begin* and *end*, in  $\mu$ meters, on the chip side in which pins and pads cannot be placed.

**Note:** You must define pad cells in the order in which they appear in the design.

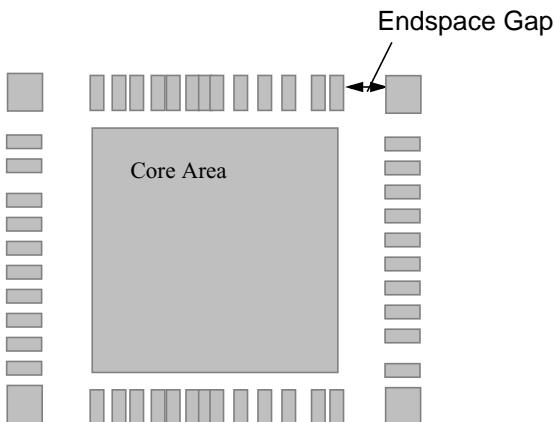


*cell*

Specifies the physical I/O cell.

*endspace gap*

Specifies the space, in  $\mu$ meters, between the corner pad and the last I/O pad for the specified side of the design.

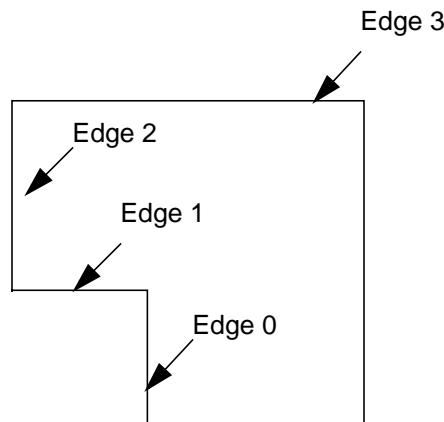


## Encounter User Guide

### Data Preparation

iopin locals

<i>side</i>	Specifies the side of the I/O pin. This can be: <ul style="list-style-type: none"><li>■ top   north</li><li>■ left   west</li><li>■ right   east</li><li>■ bottom   south</li></ul>
<i>edge num = 0</i>	Specifies the edge number of the I/O pin, with <i>edge num = 0</i> starting from the left side of the lowest y coordinate and the left most corner, in the clockwise direction.



<i>space</i>	Specifies the spacing, in $\mu$ meters, between the previously defined pin and the pin being defined.
	The value that you specify here, sets the global space setting.

iopin

<i>pin name</i>	Specifies the name of a pin. Specify I/Os as pins for block designs.
<i>layer</i>	Specifies the metal layer on which the pin must be placed.
<i>width</i>	Specifies the width of the pin in $\mu$ meters. It is the length of the edge that is centered at the reference point.

## Encounter User Guide

### Data Preparation

---

*depth*                      Specifies the length of the pin in  $\mu$ meters.

#                              Specifies the incremented I/O pin edge number.

**Note:** The I/O file does not support specifying xy location for I/O pins.

The following commands allow you to create multiple I/O rows on multiple rings:

#### Row Margin

*side*                      Specifies the side of the I/O row margin. This can be:

- top
- north
- left
- west
- right
- east
- bottom
- south

*ring\_number*              Specifies the I/O ring number on which the I/O rows are placed, with ring 1 being the outer most ring.

*margin*                      Specifies the distance, in microns, from the die boundary edge to the I/O row edge.

**Note:** When creating the I/O assignment file, start comment lines with a pound (#) sign.

#### Specifying Area I/O Information

You can also define the following objects in the I/O assignment file for area I/O placement:

- Bump

## Encounter User Guide

### Data Preparation

---

A bump is a piece of metal that works as a bonding pad to the package. When defining a bump, you must specify its master bump cell and its physical location. You can generate one bump, or an array of bumps of the same bump cell type.

- ❑ To define an individual signal bump, use the following syntax:

```
Bump: bump_name bumpCell_name x y signal_type assignment array orientation
```

For example,

```
Bump: A3 BUMP 300 100 vdd12 power fixed array_1 r180
```

- ❑ To define an array of bumps, use the following syntax:

```
Bump: bump_name bumpCell_llx_lly_urx_ury_x_y_xpitch_ypitch_xspace_yspace_row_column_out_rings_style_center_column_center_row_center_style
```

For example,

```
Bump: myBumpArray myBumpCell 100 100 100 100 300 100 20 20 10 10 6 6 3 full 4 4 full
```

### ■ IOInst

This section specifies the preplaced area I/O instances. Define area I/O instances using the following format:

```
IOInst: inst_name [x y [orientation] [place_status]]
```

For example,

```
IOInst: A1/B1/BUF1 200 200 r180 fixed
```

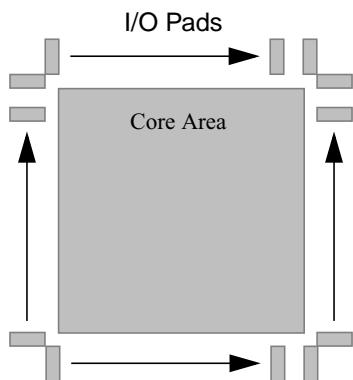
## Creating a Rule-Based I/O Assignment File

To create a rule-based I/O assignment file,

1. Create an I/O assignment file with I/O pads in the proper sequence.  
This file can include VDD and VSS filler pads.
2. Import the design.
3. After reviewing the I/O pads, choose *Design – Save – IO File*.
4. On the Save IO File form, select *sequence*.
5. Edit the new file for reimporting, or use the `loadIoFile` command.
6. Save the floorplan to a file.

## I/O Pad and Pin Assignment Examples

The following example shows statements in a sample I/O assignment file for I/O pads as shown in the figure below:

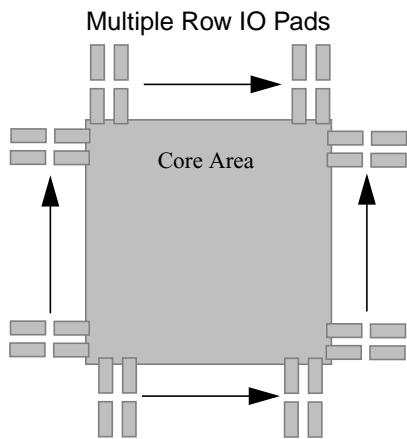


```
version = 3
io_order = clockwise
total_edge = 4
space = 1.06

(inst
    name = IOPADS_INST/pad1 W
    offset = 235.0000
    orientation = R0
    place_status = fixed
)
(inst
    name = IOPADS_INST/pad2 W
    offset = 296.1250
    orientation = R0
    place_status = fixed
)
```

## Assigning Pads for Multiple Rows

The following example shows statements in a sample I/O assignment file for multiple rows of I/O pads as shown in the figure below:



```
version = 3
io_order = clockwise
total_edge = 4
space = 1.06

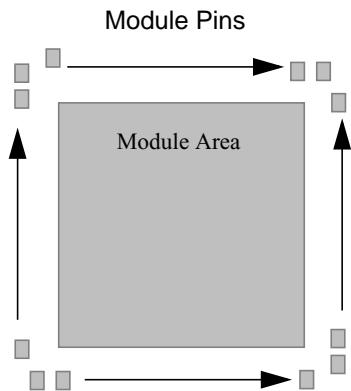
iopad
(topleft
(locals
    ring_number = 1
)
(
    instname = IOPADS_INST/pad1 W
    offset = 235.0000
)
(locals
    ring_number = 2
)
(
    instname = IOPADS_INST/pad2 W
    offset = 296.1250
)
)
```

# Encounter User Guide

## Data Preparation

## Assigning Module Pins

The following example shows an I/O assignment file for module pins as shown in the figure below:



```
version = 3

(iopin
  (top | north | edge num = 0
    (locals
      space = 1.2
    )
    (pin name = address[14] N
      layer = 3
      width = 0.28
      depth = 0.28
      offset = 19.4700
      place_status = fixed
    )
    (pin name = address[14] N
      layer = 4
      width = 0.38
      depth = 0.38
      offset = 39.2700
      place_status = fixed
    )
  )
)
```

## Performing Area I/O Placement

Before you begin area I/O placement, you must first specify CLASS PAD AREAIO in a LEF file. Additionally, a SITE or region must be defined for the placeAIO command to place the CLASS PAD AREAIO macro in the required location. The SITE must be referenced in the AREAIO macro.

The following example shows a SITE definition followed by a CLASS PAD AREAIO macro which refers to the SITE.

```
SITE IO CLASS PAD ; SIZE 210 BY 100.8 ; END IO
MACRO INBUF
  CLASS PAD AREAIO ;
  FOREIGN INBUF 0.00 0.00 ;
  ORIGIN 0 0 ;
  SIZE 210 BY 100.8 ;
  SYMMETRY X Y R90 ;
  SITE 10 ;
  PIN PAD
    DIRECTION INPUT ;
    USE SIGNAL ;
    PORT ;
    LAYER M6 ;
    RECT 95.0 40.0 115.0 60.0 ;
  END
END PAD
```

**Note:** The CLASS PAD AREAIO saves bump status defined in the DEF file only if the bump status is FIXED or COVER. See [Defining BUMP CELL Placement Status](#) on page 112.

## Defining the Connection between a Bump and P/G Pin Shape

The flip chip router (area I/O) determines which power/ground pin shape on the I/O driver cell must be connected to a bump. The following MACRO PIN statement added in the LEF 5.7 file specifies that the port is a bump connection point for multiple pins.

```
MACRO PVDD1DGZ
  CLASS PAD AREAIO ;
  FOREIGN PVDD1DGZ 0.000 0.000 ;
  ORIGIN 0.000 0.000 ;
  SIZE 40.000 BY 35.280 ;
  SYMMETRY x y r90 ;
  SITE IO1 ;

  PIN VDD
    DIRECTION OUTPUT ;
    USE POWER ;
```

## Encounter User Guide

### Data Preparation

---

```
PORT

CLASS BUMP ;
LAYER METAL8 ;
RECT 5.0 25.0 15.0 35.0 ;
END
END VDD
END PVDD1DGZ
```

For more information, see “[Macro Pin Statement](#)” in the *LEF/DEF Language Reference*.

### Defining BUMP CELL in LEF

Bumps must also be defined in a LEF file. The following example shows a BUMPCELL macro.

```
MACRO BUMPCELL
  CLASS COVER BUMP ;
  ORIGIN 0 0 ;
  SIZE 80.0 BY 80.0 ;
  SYMMETRY X Y ;
  PIN PAD
    DIRECTION INPUT ;
    USE SIGNAL ;
    PORT
      LAYER M6 ;
      RECT 0.0 0.0 80.0 80.0 ;
      #POLYGON 23.0 0.057.0 0.0 80.0 2
    END
  END PAD
END BUMPCELL
```

### Defining BUMP CELL Placement Status

You can define the bump cell placement status, `FIXED` | `COVER` for a bump object in the design, in a DEF/IN file or using the [Attribute Editor](#) in Encounter. The `CLASS PAD AREAIO` saves the bump placement status— `FIXED` or `COVER`.

**Note:** The default bump placement status is `PLACED`.

The following example shows the BUMP CELL placement status defined in the DEF file:

```
Bump: Bump_83_2_8 BUMPCELL 255.720 855.720 refclk -fixed -bumpArray array_0 -
placeStatus placed
Bump: Bump_82_1_8 BUMPCELL 155.720 855.720 pllrst -fixed -bumpArray array_0 -
placeStatus cover
Bump: Bump_81_0_8 BUMPCELL 55.720 855.720 ibias -fixed -bumpArray array_0 -
placeStatus fixed
```

## Importing LEF Files

To import the LEF files, use the following procedure:

1. Select *Design – Design Import*.

The Design Import form appears.

2. On the Design page, enter the names of the Verilog and ILM files, and choose a top cell assignment option.
3. In the LEF Files field, type the LEF file names to import, and include the file that contains the CLASS PAD AREAIO statement. Or, you can click on the ... icon to the right of the field to select files.
4. Click *OK*.

The Design Import form closes and Encounter imports the data.

To load the floorplan and I/O assignment files separately, use the following procedure:

1. Select *Design – Load – Floorplan* or run the `loadFPlan` text command.
2. Select *Design – Load – I/O file* or run the `loadIoFile` text command.

As an alternative, you can include the I/O assignment file in the floorplan file, add the following statement to your floorplan file before loading your floorplan.

`IOfile: iofile_name`

**Note:** You can also specify area I/O rows in DEF or PDEF files.

For more information on the I/O assignment file, see “[Creating an I/O Assignment File](#)” on page 97.

To save your floorplan and I/O assignment files, use the following procedure:

1. Select *Design – Save – Floorplan*. Fill out the form and click *Save*.

As an alternative, you can specify the `saveFPlan` text command.

2. Select *Design – Save – I/O File*. Fill out the form and click *Save*.

As an alternative, you can specify the `saveIoFile` text command.

To place area I/Os, use either the GUI or command line:

- To place area I/Os from the GUI, select *Place – Flipchip – Place Area I/O*. Fill out the form and click *OK*.
- To place area I/Os from the command line, use the `placeAIO` text command.

Specify the `-onlyAIO` argument to place only the area I/Os on the area I/O rows. If you do not specify this argument, all standard cell instances and blocks are also placed.

Specify the `-assignBump` argument if you have unassigned bumps for area I/O instance connections. If you specify this argument, area I/O instances are connected to the nearest unassigned bumps.

**Note:** You can also assign bumps after area I/O placement by using the `assignBump` command.

## Preparing Timing Libraries

Timing library files contain timing information in ASCII format for all of the standard cells, blocks and I/O pad cells. The Encounter software reads timing library format files (`.tlf`) or Synopsys Technology Library format files (`.lib`). You do not need to translate timing library files before reading them into the software.

## Encrypting Libraries

To protect proprietary data, you can encrypt the ASCII library files. Use the `lib_encrypt` utility to perform the encryption. The `lib_encrypt` utility is installed along with the Encounter software. To encrypt the ASCII library file, use the following command:

```
lib_encrypt [-ogz] [-help] in_file out_file
```

## Parameters

-help	Displays the syntax of the lib_encrypt command.
<i>in_file</i>	Specifies the name of library file to be encrypted.
-ogz	Creates a gzip file of the encrypted output library file.
<i>out_file</i>	Specifies the name of the output file.

## Preparing Stamp Models

Stamp models contain timing information for a module, such as an instantiated module, block, or partitioned module. It describes the timing models of large blocks, such as RAM blocks, microprocessor cores, DSP, and others that have not been synthesized into gate-level netlists.

A stamp model consists of two files:

- Model file that describes the ports, timing arcs, and other attributes of the block.
- Data file that provides technology specific data, including values for timing arcs, port capacitance, and maximum transitions.

You do not need to translate these files before reading them into the software.

**Note:** Stamp models supersede timing library models.

## Preparing Timing Constraints

To import timing constraints, use the `write_script` or `write_sdc` command from within Design Compiler, PrimeTime, or Physical Compiler. These commands eliminate any variable substitution confusion, making them easier for the user and the software to read.

Use the `write_script` command on the design inside `dc_shell` or `pt_shell` for the best results, for example:

```
write_script -format {ptsh | dcsh | dctcl} -output fileName
```

Or, you can use the following command:

```
write_sdc
```

You do not need to translate either DC or PT constraints before reading them into the software.

**Note:** When reading in constraints, only read in one format type in a session.

## Preparing Capacitance Tables

For accurate extraction results, use capacitance tables. You can generate and use separate capacitance tables for different process corners.

For more information on preparing capacitance tables, see [Chapter 18, “RC Extraction.”](#)

## Preparing Data for Delay Calculation

If you want to use the SignalStorm® nanometer delay calculator, see [Chapter 25, “Calculating Delay”](#) for information about preparing ECSM libraries.

## Preparing Data for Crosstalk Analysis

For information on preparing data for crosstalk analysis, see [Chapter 32, “Analyzing and Repairing Crosstalk.”](#) For more information on preparing cdB noise libraries using the `make_cdB` utility, see the “*make\_cdB Noise Characterizer User Guide*.”

## Checking Designs

Before importing the design or running Encounter at various stages of the design process, you can check for missing or inconsistent library and design data.

To perform these checks, use the following command:

`checkDesign`

You can check for the following data:

- Physical library
- Timing library
- Netlist
- I/Os
- Tie-high and tie-low pins
- Power and ground pins

Cadence recommends that you check libraries and data as follows:

- Perform I/O checking at any time. I/O problems might not impede any tool, but they might add to design problems.
- Perform netlist checking at any time after the design has been loaded.
- Perform physical library checking before floorplanning.
- Perform power and ground checking before routing and extraction, and verifying geometry and connectivity.
- Perform timing library checking before any timing-related operation (for example, timing-driven placement or routing, timing optimization, clock-tree synthesis, and static timing analysis).
- Perform tie-high and tie-low checking before routing and extraction.

## Preparing Data in the Timing Closure Design Flow

For information on preparing data for the timing closure design flow, see the *Encounter Timing Closure Guide*.

## **Encounter User Guide**

### Data Preparation

---

---

## Importing and Exporting Designs

---

- [Overview](#) on page 120
- [Verifying Data before Importing a Design](#) on page 120
- [Preparing the Design Netlist](#) on page 120
- [Creating a Flat Verilog Netlist from a DEF File](#) on page 121
- [Beginning Designs](#) on page 123
- [Loading Previously Saved Configuration Files](#) on page 125
- [Selecting Files](#) on page 126
- [Working with OpenAccess Designs](#) on page 129
- [Removing Assign Nets from a Verilog Netlist](#) on page 130n
- [Saving and Restoring Designs](#) on page 131
- [Importing and Exporting Design Data](#) on page 132
- [Converting an Encounter Database to GDSII Stream or OASIS Format](#) on page 137
- [About the GDSII Stream or OASIS Map File](#) on page 143
- [Updating Files during an Encounter Session](#) on page 152

## Overview

The Encounter™ software provides the following options for saving, restoring, importing, and exporting design data:

Starting (importing) designs	Allows you to specify data for starting a design or load existing configuration files.
Saving designs	Allows you to save the work you complete on designs during a design session for access at a later date.
Restoring designs	Allows you to load saved data from a previous design session.
Loading design data	Allows you to load design data saved in various stages of the design process, and to bring data from specific formats (DEF, TDEF, PDEF, SPEF, SDF, and OpenAccess) into the Encounter environment.
Saving and exporting design data	Allows you to save design data in various stages of the design process, and to export data in specific formats (DEF, TDEF, PDEF, GDS, and OASIS) from the Encounter environment.

## Verifying Data before Importing a Design

To check that Verilog, LEF, and .lib files are available at the beginning of an Encounter session, use the following command:

```
setCheckMode -netlist true -library true
```

Encounter performs this check by default. To report the current checking mode, use the following command:

```
getCheckMode
```

## Preparing the Design Netlist

The Encounter software requires that your Verilog® design netlist or OpenAccess netlist be unique so that you can run Clock Tree Synthesis (CTS), Scan Reorder, and timing optimization features.

- To ensure that the names of all instantiated cell types are unique in a Verilog netlist, use the following command:

### uniqualifyNetlist

The `uniqualifyNetlist` command tests all levels of intermediate modules. It does not test leaf cells.

There is no equivalent command for unquifying OpenAccess netlists. You must manually ensure that the names of all instantiated cell types are unique.

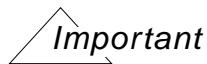
## Creating a Flat Verilog Netlist from a DEF File

Cadence requires a Verilog netlist for design import. There is an exception: if you have a DEF file that contains connectivity information, you can import this file. This is not the recommended approach.



***After loading the DEF netlist, you can perform floorplanning, non-timing driven placement and routing, wire editing, and verification. You cannot use the DEF netlist flow for parasitic extraction, delay calculation, and timing-driven placement and routing effectively because the DEF names do not properly match the Verilog names used in timing constraints and timing analysis.***

## Recommended DEF Import Commands



Cadence highly recommends using these commands instead of the alternative DEF import flow.

To import a DEF file that contains connectivity information, use either of the following commands:

- `defToVerilog defFile verilogFile`

The `defToVerilog` command loads the DEF netlist, saves the netlist as a Verilog file, and frees the design, enabling to you continue in the Encounter environment.

- `loadDefFile defFile`

The `loadDefFile` command loads a DEF file to build Encounter's in-memory database. In order to use this command, the library data must be present in memory (use the `loadLefFile` command).

For more information, see the `defToVerilog` and `loadDefFile` commands in the *Encounter Text Command Reference*.

## **Reconciling the Object Names and Creating New DEF File That Can Be Used With the Normal Encounter Flows**

The following procedure imports the Verilog file generated by the `saveNetlist` command in the previous encounter session, and reconciles names in the DEF and Verilog files. This procedure is required if you want to retrieve more information from the original `example.def` file.

1. Start Encounter.

```
encounter
```

2. Use a configuration file containing commands to load the LEF file and the Verilog file generated by the `saveNetlist` command from the first session.

```
loadConfig output.conf
```

3. Import the DEF file.

```
defIn -verilog_from_def_netlist_flow example.def
```

This command reads all DEF constructs, not just connectivity.

**Note:** The `-verilog_from_def_netlist_flow` parameter is used in this flow only. The `defIn` operation uses this parameter to correct special characters so that the names in the `output.def` file match the names in the new Verilog file.

**4.** Write the DEF file.

```
defOut [other-options] output.def
```

The `output.def` file is equivalent to the `example.def` file, but with the Verilog names resolved. It can be used in future encounter sessions without the `-verilog_from_def_netlist_flow` parameter.

**5.** Exit the current session.

```
exit
```

Now, you can use `output.conf`, `output.v`, and `output.def` in any encounter flow.

## Beginning Designs

Before you begin a design, you must first prepare the data. For more information, see “[Data Preparation](#)” in the *Encounter User Guide*.

### Beginning a Design with LEF and Verilog

To begin a LEF and Verilog design, complete the following steps:

- 1.** Select *Design – Import Design*.
- 2.** Select the *Basic* tab if it is not already selected.
- 3.** Specify the gate-level Verilog netlist files to import in the *Files* text field.
- 4.** Select one of the following options to specify the top cell:
  - Auto Assign*  
(Default) Automatically extracts the top cell name from the netlist, provided the netlist contains only one design.
  - By User*  
Specifies the name of the top cell when a netlist contains more than one design (more than one top design name). The top cell name specified is the design the software imports and processes.
- 5.** Specify the LEf files to import. You must specify the technology LEF file first, then specify the standard cell LEF and block LEF in any order.

The LEF file provides technology information, such as metal layer and via layer information and via generation rules, which is used in the Add Rings and Add Stripes forms. The router also uses the technology information contained in the LEF file.

If a cell is defined multiple times, Encounter reads the geometry information only from the first definition. For subsequent definitions, Encounter reads the antenna information only.

**Note:** If the LEF file contains all the physical information for the design, no other files are required for the *Technology Information/Physical Libraries* panel.

**6.** Click Save or OK.

- Save* saves your settings to a configuration file. The design is not imported.
- OK* uses the current settings to import the design. The configuration file is not updated.

## Beginning a Design with OpenAccess

To begin an OpenAccess design, complete the following steps:

- 1.** Select *Design – Import Design*.
- 2.** Click the *Advanced* tab, then select *OpenAccess*.
- 3.** Select *Use OA Netlist*, then specify the following information:
  - Library*  
Specifies the OpenAccess database library.
  - Cell*  
Specifies the OpenAccess database cell.
  - View*  
Specifies the OpenAccess database view.
- 4.** Specify the following OpenAccess technology and physical library information:
  - Reference Libraries*  
Specifies the OpenAccess reference libraries to import. The first OpenAccess reference library listed in this field must contain the technology information for the leaf cells.

Each reference library is processed using the abstract view name list (*Abstract View Names*).

For example, if the reference library is “lib1 lib2”, and the abstract view name list is “abstract abstract2”, LEF MACRO information is processed for lib1 with the abstract view. Then, for any cells in lib1 that didn’t have abstract, but did have abstract2, that view is processed for MACRO information. If a cell has both views, the first one is used. The process then is repeated for lib2.

*Abstract View Names*

Specifies the OpenAccess view names that the software should examine to find the equivalent LEF MACRO information (for example, PINS, OBS, FOREIGN).

*Layout View Names*

Specifies the layout view names (separated by spaces) to import.

**5. Click Save or OK.**

- Save* saves your settings to a configuration file. The design is not imported.
- OK* uses the current settings to import the design. The configuration file is not updated.

## Loading Previously Saved Configuration Files

You can use either the command line or GUI to load previously saved configuration files.

### Loading Configurations Files from the Command Line

To load a previously saved configuration file, use the following command:

```
loadConfig fileName [0 | 1]
```

If you use this command in batch mode and specify a *filename*, the file is loaded and the design is imported. If you specify a filename and the 0 parameter, the software loads the file, but does not import the design.

To apply settings specified in the current configuration file and import the design, use the following command:

```
commitConfig
```

## Related Topics

- [Configuration File Variables](#) in the *Encounter Text Command Reference*.
- [Load and Check Data](#) in the *Encounter Flat Implementation Flow Guide*.

## Loading Configuration Files from the GUI

To load a previously saved configuration file from the GUI, complete the following steps:

1. Select *Design – Import Design*.
2. Select the *Basic* tab if it is not already selected.
3. Click *Load*.

The Load Import Configuration form is displayed.

4. Select the directory of the file you want to load.
5. Select *Input config file (\*.conf\*)* in the *Files of type* field.
6. Specify a file name or click on the filename in the window. The filename suffix is .conf.
7. Click *Open*.

The Load Import Configuration form closes.

The configuration file is loaded.

8. In the Design Import form, continue to specify data you want to import into the design.
9. Click *Save* or *OK*.
  - Save* saves your settings to a configuration file. The design is not imported.
  - OK* saves your settings to a configuration file and starts the design import process. This might take several minutes to complete, depending on the size of your design. When the design is loaded, the Design Import form closes and the design displays in the Encounter main window.

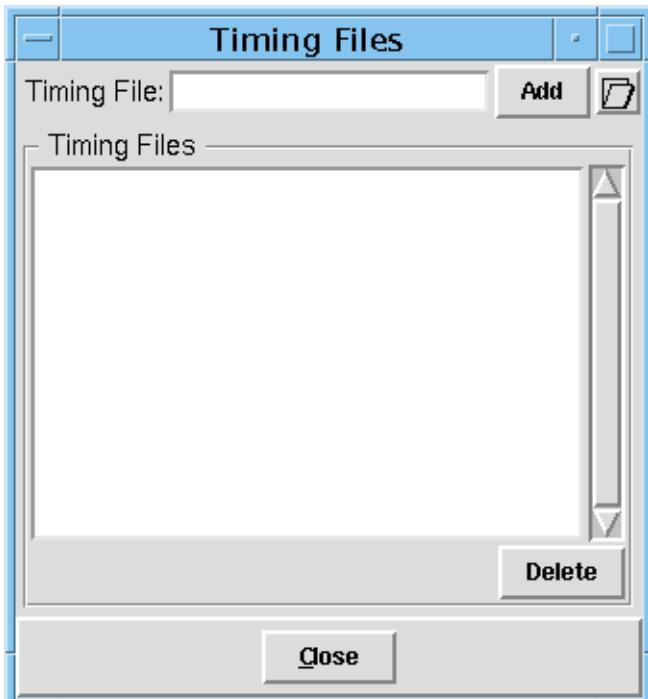
## Selecting Files

Many of the text fields on the Design Import form contain a browse (...) button that opens a separate form for selecting files. The name of the form corresponds to the specific file you are selecting; for example, Netlist Files, ILM Files, or Timing Files. These forms are provided for easier file management.

## Using Select Files

1. On the Design Import form, click the browse (...) button next to the text field of the file type in which you are interested.

This opens the Files form for that file type. For example, clicking the browse button next to the *Max Timing Libraries* field opens the Timing Files form.



2. To type in a specific filename, do the following:

- a. In the first text field, type one or more filenames, specify wildcards, or select a directory. Use spaces to separate multiple filenames.
- b. Click *Add*.

The filenames appear in the *Files* list of this form and in the specific *Files* field of the Design Import form.

- c. Click *close*.

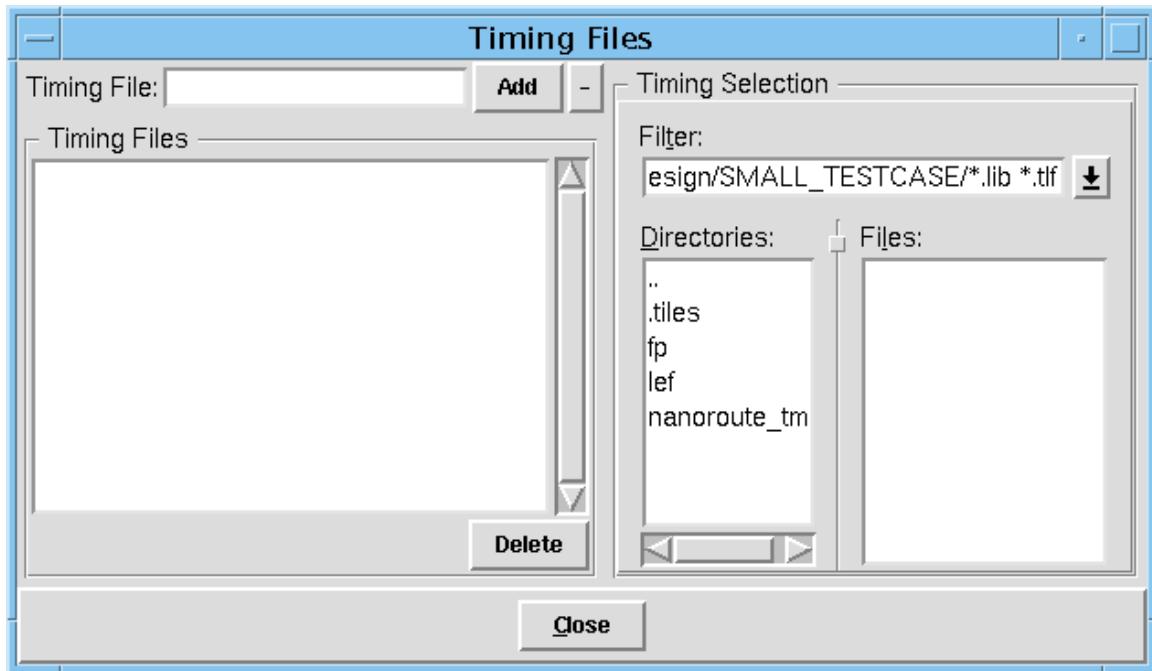
3. To choose a file from a directory, do the following:

## Encounter User Guide

### Importing and Exporting Designs

- a. Click the file folder icon.

This expands the form and displays a list of directories and files.



- b. Select one or more files in the *Files* list.
  - c. Click *Add*.
- The filenames appear in the *Files* list of this form and in the specific *Files* field of the Design Import form.
- d. Click close.
4. To delete files, select the file(s) to be deleted in the *Files* list and click *Delete*.
- The files are deleted from the both this form and the Design Import form.

## Working with OpenAccess Designs

- [Importing an OpenAccess Design](#) on page 129
- [Saving an OpenAccess Design](#) on page 129
- [Restoring an OpenAccess Design](#) on page 129
- [Transferring OpenAccess Data between Encounter and Virtuoso Chip Editor for ECO](#) on page 130

### Importing an OpenAccess Design

For information on importing OpenAccess designs, see “[Beginning Designs](#)” on page 123.

### Saving an OpenAccess Design

Before you attempt to save an OpenAccess design for the first time, you must set the *Reference Libraries* and *Abstract View* values on the Design Import form.

Then, after you run *Design – Restore Design – OA* in a new session, the Encounter software restores the design state to the same as state it was in when you used *Design – Save Design As – OA* in the previous session.

### Restoring an OpenAccess Design

To restore an OpenAccess design, use one of the following methods:

- To load the netlist information from an Open Access database, select *Design – Import Design*, click the *Advanced* tab, then select *OpenAccess*. Specify the information needed in the *Library*, *Cell*, and *View* fields.
- After you have imported the design, if the OpenAccess database contains physical information, select *Design – Load – OA Database*.
- To restore a design that you saved previously by using *Design – Save Design As – OA*, select *Design – Restore Design – OA*.

**Note:** If you saved the design in Encounter, then edited and saved it under a different name using another tool, you must run the `oaCopyRestoreFiles` command to copy the required information from the original Library/Cell/View that was saved.

## Transferring OpenAccess Data between Encounter and Virtuoso Chip Editor for ECO

1. From an Encounter session, save the OpenAccess design.

```
saveOaDesign lib cell view
```

2. Exit the Encounter session.

3. Open the OpenAccess database in the Virtuoso Chip Editor tool and edit the design.

**Note:** You must use Virtuoso Chip Editor rather than the Virtuoso Layout Editor.

4. Save the design.

5. Exit the VCE tool.

6. Start an Encounter session.

7. Restore the OpenAccess design.

```
restoreOaDesign lib cell view
```

## Removing Assign Nets from a Verilog Netlist

You can use the [setDoAssign](#) command to remove assign statements from the Verilog netlist for a design. Encounter has the following approaches to process assign statements, each with its own limitation:

- Encounter removes assign statements and does not change the netlist. Use the following command:

```
setDoAssign off
```

- Encounter removes assign statements by merging the nets in Assign statements, changing the netlist. Use the following command:

```
setDoAssign on
```

- Encounter replaces Assign statement with buffers during design import. Use the following command:

```
setDoAssign on -buffer buffer_cell
```

The following conditions arise from using the `setDoAssign` command.

- Timing constraints applied to hierarchical pins associated with assign statements are lost in some cases. To retain constraints, use the following command:

```
setDoAssign on -buffer buffer_name
```

- Encounter cannot remove Assign statements associated with I/O pins. To do this manually, use the following command:  
`setDoAssign on -buffer buffer_name`
- Encounter does not handle Assign statements associated with bussed nets. You must process assign statements using your front-end software.

## Saving and Restoring Designs

This section contains the following general guidelines for saving and restoring your design data:

- [Saving Designs](#) on page 131
- [Restoring Designs](#) on page 132
- [Saving and Restoring OpenAccess Designs](#) on page 132

### Saving Designs

To save a design, you can use the text command or menu command.

- Use the text command as follows:  
`saveDesign sessionName`  
or
- From the Encounter GUI, use the menu command as follows:  
[Design – Save Design As – SoCE](#)

The design files you save depend on the work completed during an Encounter session. For example, if you did not perform Trial Route on an imported design, the saved design data will not include a route file.



You can save a netlist file *only* if you made a design change during the Encounter session. If you make no changes, Encounter references the original netlist when it saves the design. Do *not* use the Save Design form to save a partition.

## Restoring Designs

To restore a design, you can use the text command or menu command.

- Use the text command as follows:

`restoreDesign sessionName.dat`

or

- From the Encounter GUI, use the menu command as follows:

*Design – Restore Design – SoCE*

## Saving and Restoring OpenAccess Designs

For information on saving and restoring OpenAccess designs, see “[Working with OpenAccess Designs](#)” on page 129

## Importing and Exporting Design Data

This section contains some general suggestions for importing design data into the Encounter environment and exporting data out of the Encounter environment.

### Loading a Partition

To load a partition, you can use the menu command as follows:

*Design – Load – Partition*

Before you load a partition, perform the following tasks:

1. Import the design
2. Load the full chip (flat) floorplan, including partition specifications
3. Commit the partition without pin assignment or a timing budget
4. Place and route each of the partitions

When you load a partition design, the Encounter software rebuilds the individual partition and the top level, so that the entire chip can be analyzed. When you load a saved partition, the software loads all the files that are selected in the Load Partition File form.



The netlist and routing must be consistent when you load a partition that contains routing data. For example, if your netlist was modified after in-place optimization (IPO) or after running NanoRoute, you should make sure that the loaded routing results correctly correspond to the new netlist.

## Loading Floorplan Data

To load floorplan data, use the following menu command:

### Design - Load - Floorplan

When you load a floorplan, the Encounter software treats the following items as floorplan data:

- Floorplan dimensions
- Standard cell rows
- Floorplan guides
- Hard blocks (macros)
- Blackboxes
- Power structures
- Density screens
- Placement blockages
- Routing blockages
- Pin blockages
- Partition pin cuts
- Feedthrough guides



Blocks and instances that you load with the Load Floorplan command are set as preplaced.

## Placement File Requirement

Before you load the floorplan file that was used to generate the placement file, make sure the placement file is in Encounter format.

## Loading an I/O Assignment File

If you do not read an I/O assignment file into your Encounter session, and if no I/O pad instances are preplaced, the Encounter software randomly places I/O pad instances.

## Loading an FSDB File

Before you begin, run a simulation-based power analysis with VCD input. Load an FSDB file for detailed power analysis using the Debussy Waveform (nWave) tool.

## Saving a Partition

You can save import configuration, netlist, floorplan, special route, and vendor-specific files for each partition, including the top level.

**Note:** Regardless of your choice of output file, the Verilog® netlist, configuration file, and floorplan file are always saved.



You can specify a timing constraint output format for each partition only if you selected *Derive Timing Budget* when you ran the Partition program.

## Saving Floorplan Data

When you save a floorplan, the Encounter software treats the following items as floorplan data:

- Floorplan dimensions
- Standard cell rows
- Floorplan guides
- Hard blocks (macros)
- Blackboxes
- Power structures

- Density screens
- Placement blockages
- Routing blockages
- Pin blockages
- Partition pin cuts
- Feedthrough guides

After you save a floorplan, the Encounter software creates the following files:

- A general floorplanning file with the extension .fp
- A power route data file with the extension .fp.spr

If there is an entry in the *IO Cell Libraries* field in the Design Import form, a third file is created with the extension .fp.areaio.

## Importing and Exporting TDF Files

To load TDF files, use the following menu command:

Design – Load – TDF

Before you can exchange data between the Encounter software and the Astro third-party tool, you must provide information on the layer and contact mapping. Before you can use the tdfIn and tdfOut commands, you must add the following information to your LEF technology file:

```
BEGINEXT "mapName"
LAYER layerName LAYEREXTID
VIA contactName CONTACTID
VIA viaName CONTACTID
VIARULE ruleName CONTACTID
ENDEXT
```

The following examples shows a layer and contact mapping:

```
BEGINEXT "FE_TF_LAYEREXTID"
LAYER MET1 31
LAYER MET2 32
LAYER MET3 33
LAYER MET4 34
LAYER MET5 35
VIA polyCont 1
VIA vial 2
VIA via2 3
VIA via3 4
VIA via4 5
```

```
VIARULE vial_array 2
VIARULE via2_array 3
VIARULE via3_array 4
VIARULE via4_array 5
ENDEXT
```

## Importing Tile Cell Design Data

You can use text commands or the GUI to import a DEF file containing tile cell design data.

To import tile cell data by using a script, complete the following steps:

1. Load the configuration file.

```
loadConfig
```

2. Set the relative path from the working directory.

```
set rda_ciop(tile:rootdir) [file join [pwd] "tile_cell_lib_dir"]
```

3. Find all LEF files in the specified path.

```
ciopLoadTileLib
```

4. Read the DEF file.

```
defIn tile_cell_def_file
```

To import the tile cell data by using the GUI, complete the following steps:

1. Provide a valid path to the tile library.

*FlipChip – Select Tile – Edit*

2. Load tiles from the library.

*FlipChip – Select Tile – Load*

3. Read the DEF file.

```
defIn tile_cell_def_file
```

## Converting an Encounter Database to GDSII Stream or OASIS Format

To convert an Encounter database to GDSII Stream or OASIS format at any stage of the design flow, use the following commands:

■ For GDSII Stream format:

- [setStreamOutMode](#)
- [streamOut](#)

Create GZIP files by appending .gz to the filename. The `streamOut -merge` command can read files with the .gz extension.

**Note:** You can also use the following GUI forms:

- [Design – Mode Setup – StreamOut](#)
- [Design – Save – GDS/OASIS](#)

■ For OASIS format:

- [setOasisOutMode](#)
- [oasisOut](#)

**Note:** You can also use the following GUI forms:

- [Design – Mode Setup – OasisOut](#)
- [Design – Save – GDS/OASIS](#)

If the database is partitioned into hierarchical blocks, create a file that includes all cells by completing the following steps:

1. Generate GDSII Stream or OASIS files for the hierarchical blocks.
2. Merge the block-level GDSII Stream or OASIS files to make a top-level file for the whole design.

### Related Topics

To see where this step fits in the design flow, see [Analyze SI, Run Post-SI Optimizatin and Physical Verification and Generate GDSII Stream File](#) in the *Encounter Flat Implementation Flow Guide*.

For more information, see “[Merging GDSII Stream or OASIS Files](#)” on page 139.

## Creating Cells and Instances

When it converts the database, the software creates instances according to following cases:

- If a LEF MACRO does not have any FOREIGN statements, or if a MACRO name and FOREIGN name are the same, the software creates one top-level instance that has the same name as the MACRO. At the cell level, a cell with the same name as the MACRO already exists, so the software does not create any new cells.
- If a LEF MACRO has multiple FOREIGN statements, or if the MACRO name and FOREIGN name are different, the software also creates one top-level instance that has the same name as the MACRO. However, at the cell level there is no cell with the same name as the MACRO, so the software creates one. This cell contains pointers to the data for each FOREIGN structure in the LEF MACRO.

## Renaming LEF Vias

To force the `streamOut` or `oasisOut` commands to give unique names to LEF vias, type one of the following commands before running the `streamOut` or `oasisOut` command:

- `setStreamOutMode -SEVianames true`
- `setOasisOutMode -SEVianames true`

These commands rename all LEF vias, and all generated vias, using the following naming convention:

*topStructureName\_VIA index*

Examples of renamed vias are `chip_VIA1` and `bigDesign_VIA23`.

For more information, see [`setStreamOutMode`](#) or [`setOasisOutMode`](#) in the *Encounter Text Command Reference*.

## Merging GDSII Stream or OASIS Files

The software allows you to merge several GDSII Stream or OASIS files into a single file for hierarchical designs. It merges cells that are either referenced (instantiated) in the design or can be referenced in a recursive search from any child cell that is referenced in the design. For example, if a merge file contains cells A, B, C, X, Y, and Z, and C has a reference to X, and X has a reference to Y, and the design references cells A, B, and C (but not directly X, Y, or Z), the software merges cells A, B, C, X, and Y, but not Z.

The software creates a file in the highest version number of all the merge files.

### Merging Files Using the Command Line

1. Create the block-level GDSII Stream or OASIS files by using one of the following commands:

```
streamOut -merge list_of_GDS_files [-uniquifyCellNames]  
oasisOut -merge list_of_OASIS_files [-uniquifyCellNames]
```

If you specify the `-uniquifyCellNames` parameter, you must list the top-level file first, as the software uses the first name in the search path when renaming cells. For more information, see [“Merge Examples”](#) on page 139.

2. Create the top-level GDSII Stream or OASIS file by using the block-level files as the merge files.

The software issues warning messages if any of the files, including the block-level files, contain structures with the same name or if it renames any cells.

The top-level GDSII Stream or OASIS file contains the following structures:

- Top structure (the design data from the Encounter software)
- Via structures (output from the Encounter design data)
- Leaf cell structures and their children (copied from the merge files)
- Intermediate structures from the FOREIGN structure

### Merge Examples

The following examples show the order dependency in merge files. In the examples, the COMMON cells may be the same or different. If the cells are different, or if you are not sure whether they are the same or different, use the `-uniquifyCellNames` parameter in addition to the `-merge` parameter.

**Note:** In the examples, for simplicity GDS and streamOut are used. If you are merging OASIS format files, substitute OASIS for GDS and oasisOut for streamOut.

### Case 1

Most cases are similar to the following:

- GDS1 contains cells X, COMMON (COMMON is instantiated in X).
- GDS2 contains cell Y, COMMON (COMMON is instantiated in Y).

The design instantiates cells X and Y.

- For examples of cases where hierarchical cells are involved and the contents of a hierarchical cell is different from another cell with the same name, see "[Case 2](#)" on page 141.

#### **Example 1**

```
streamOut -merge {GDS1 GDS2}
```

- GDS1 processed: X and COMMON are copied from GDS1.
- GDS2 processed: Y is copied from GDS2, COMMON is assumed to be the same, so it is not copied, Y references the version of COMMON that was copied from GDS1.

#### **Example 2**

```
streamOut -merge {GDS2 GDS1}
```

- GDS2 processed: Y and COMMON are copied from GDS2.
- GDS1 processed: X is copied from GDS1, COMMON is assumed to be the same, so it is not copied, X references the version of COMMON that was copied from GDS2.

#### **Example 3**

```
streamOut -merge {GDS1 GDS2} -uniquifyCellNames
```

- GDS1 processed: X and COMMON are copied from GDS1.
- GDS2 processed: Y is copied from GDS2, COMMON is copied from GDS2 but renamed COMMON\_GDS2 due to unification, reference from Y to COMMON is changed to COMMON\_GDS2.

### **Example 4**

```
streamOut -merge {GDS2 GDS1} -uniquifyCellNames
```

- GDS2 processed: Y and COMMON are copied from GDS2.
- GDS1 processed: X is copied from GDS2, COMMON is copied from GDS2 but renamed to COMMON\_GDS1 due to uniquification, reference from X to COMMON is changed to COMMON\_GDS1.

### **Results**

Assuming the COMMON cells are copies of the same cell, the results of Example 1 and Example 2 are the same. Example 3 and Example 4 are geometrically equivalent, but have duplicate copies of the COMMON cell (with one copy with a different name).

Assuming the COMMON cells are different, the results of Example 1 and Example 2 are not correct. In this case, the results of Example 3 and Example 4 are both correct, but yield different cell names depending on the order.

### **Case 2**

In some cases, hierarchical cells are involved and the contents of a hierarchical cell is different from another cell with the same name. The following examples show the results of order dependency of merge files in these cases.

- GDS1 contains cells X, Y (Y is instantiated in X).
- GDS2 contains cell Y.

The Y cells in the files contain different information.

The design instantiates cells X and Y.

### **Example 5**

```
streamOut -merge {GDS1 GDS2}
```

- GDS1 processed: X and Y are copied from GDS1.
- GDS2 processed: Y from GDS2 is dropped.

**Example 6**

```
streamOut -merge {GDS2 GDS1}
```

- GDS2 processed: Y from GDS2 is copied from GDS2.
- GDS1 processed: X is copied from GDS1, Y is dropped (references from X to Y now use the one copied from GDS2).

**Example 7**

```
streamOut -merge {GDS1 GDS2} -uniquifyCellNames
```

- GDS1 processed: X and Y are copied from GDS1.
- GDS2 processed: Y from GDS2 is dropped.

**Example 8**

```
streamOut -merge {GDS2 GDS1} -uniquifyCellNames
```

- GDS2 processed: Y from GDS2 is copied from GDS2.
- GDS1 processed: X is copied from GDS1, Y is copied from GDS1 but renamed to Y\_GDS1 due to unification, reference from X to Y changed to Y\_GDS1.

**Results**

Assuming the Y cells are copies of the same cell, the results of Example 5, Example 6, and Example 7 are the same. The results of Example 8 are geometrically equivalent, but have two copies of the Y cell, and one copy has a different name.

Assuming the Y cells are different, you must know whether the design is supposed to have its Y cell from GDS1 or GDS2. If the correct version of Y is from GDS1, then Example 5 and Example 7 give the correct results. If the correct version of Y is from GDS2, then only Example 8 gives the correct results.

For more information, see the following commands:

- [streamOut](#)
- [oasisOut](#)

## Merging GDS/OASIS Files Using the GUI

Use the GDS/OASIS Export form.

1. Choose *Design – Save – GDS/OASIS*.
2. Fill in the appropriate fields on the form.

For more information, see Save – GDS/OASIS in the “Design Menu” chapter of the *Encounter Menu Reference*.

## About the GDSII Stream or OASIS Map File

When the software converts an Encounter database to GDSII Stream or OASIS format, it creates a file for mapping the layers in the Encounter database to a GDSII Stream or OASIS database. The file can handle up to 1000 GDSII Stream or OASIS layers. In the file each layer is assigned a unique number and is described on a separate line. You must customize the file to make it appropriate for your design.

### Related Topics

- Flat Implementation Flow chapter in the *Encounter Flat Implementation Flow Guide*
  - “Results”

## Map File Format

The file has the following four columns, and may contain comments:

- Layer object name (layerObjName)
- Layer object type (layerObjType)
- Layer number (layerNumber)
- Data type (dataType)

Each comment starts and ends with a hash mark (#) and must be the first or last argument on a line. It can be preceded by spaces or tabs.

Following is a short example of a map file with comments:

```
#This comment is the first argument on a line#
METAL1    NET          1      0
METAL1    SPNET        999    0
#This comment is preceded by white space#
```

## Encounter User Guide

### Importing and Exporting Designs

---

```
METAL1    PIN        1000      0
        #This comment is preceded by a tab#
METAL1    LEFPIN     2000      0
METAL1    FILL       3000      0
METAL1    VIA        4000      0 #This comment is at the end of a line#
METAL1    VIAFILL    5000      0
METAL1    LEFOBS     10000     0
NAME      METAL1/NET 20000     0
```

## Map File Columns

<i>layerObjName</i>	Specifies one of the following objects:
<i>LEF_layer_name</i>	Specifies a LEF layer from the LAYER statement in the LEF technology file.  If the <i>layerObjName</i> is a LEF layer name, the <i>layerObjType</i> must specify the DEF object type.
COMP	Specifies component outlines.  If the <i>layerObjName</i> is COMP, the <i>layerObjType</i> must specify ALL.
DIEAREA	Specifies the chip boundary.  If <i>layerObjName</i> is DIEAREA, the <i>layerObjType</i> must specify ALL.

## Encounter User Guide

### Importing and Exporting Designs

---

**NAME**      Specifies a text label for the layer name and associated object type. If you do not want to output text labels, remove the NAME lines from the file.

There is no limit on the length of a structure (cell) name. Because some GDS/OASIS readers have a 32-character limit, the Encounter software issues a warning message when a structure name is longer than 32 characters.

If *layerObjName* is NAME, *layerObjType* can be a composite layer name/object type (LEFPIN, NET, PIN, or SPNET), or COMP.

- LEFPIN places the label on the LEF MACRO PIN shape. (Applies only when the *-outPutMacros* parameter is specified. For more information, see [streamOut](#) or [oasisOut](#).)
- NET places the label on the NET.
- PIN places the label on the PIN or I/O abstract pad.
- SPNET places the label on the SPECIALNET.
- COMP places the label on the placed DEF component.

*layerObjType*      Specifies an object type.

You can specify a subtype for some *layerObjTypes*. For more information, see [“Specifying Object Subtypes”](#) on page 147.

ALL

- In routing layers, ALL is equivalent to NET, SPNET, VIA, PIN, LEFPIN, FILL, FILLOPC, LEFOBS, VIAFILL, and VIAFILLOPC.
- In cut layers, ALL is equivalent to VIA, VIAFILL, and VIAFILLOPC.

## Encounter User Guide

### Importing and Exporting Designs

---

BLOCKAGE	Equivalent to DEF BLOCKAGES without + FILLS.
BLOCKAGEFILL	Equivalent to DEF BLOCKAGES with + FILLS.
CUSTOM	Applies to addCustomText and addCustomBox information only.  For more information, see <a href="#">addCustomText</a> and <a href="#">addCustomBox</a> .
FILL	Equivalent to DEF FILLS without + OPC or DEF SPECIALNETS with + SHAPE FILLWIRE.
	You can separate FILL into floating and connected fill by specifying the FLOATING subtype. For more information, see “ <a href="#">Fill Subtype</a> ” on page 148.
FILLOPC	Equivalent to DEF FILLS with + OPC or DEF SPECIALNETS + SHAPE FILLWIREOPC.
	You can separate FILLOPC into floating and connected fill by specifying the FLOATING subtype. For more information, see “ <a href="#">Fill Subtype</a> ” on page 148.
	<b>Note:</b> DEF 5.6 does not support this object type.
LEFOBS	Equivalent to LEF OBS. (Applies only when the –outPutMacros parameter is specified. For more information, see <a href="#">streamOut</a> or <a href="#">oasisOut</a> .)
LEFPIN	Equivalent to LEF PIN. (Applies only when the –outPutMacros parameter is specified. For more information, see <a href="#">streamOut</a> or <a href="#">oasisOut</a> .)
NET	Equivalent to DEF NETS wiring. For more information, see “ <a href="#">Net Name Subtype</a> ” on page 149.
PIN	Equivalent to DEF PINS.

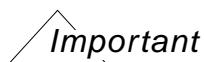
## Encounter User Guide

### Importing and Exporting Designs

---

	SPNET	Equivalent to DEF SPECIALNETS without + SHAPE FILLWIRE or + SHAPE FILLWIREOPC. For more information, see “ <a href="#">Net Name Subtype</a> ” on page 149.
	TEXT	Applies to strip box information. For more information, see <a href="#">dbCreateText</a>
	VIA	For via master creation for regular vias.
	VIAFILL	You can separate VIAFILL into floating and connected fill by specifying the FLOATING subtype. For more information, see “ <a href="#">Fill Subtype</a> ” on page 148.
	VIAFILLOPC	You can separate VIAFILLOPC into floating and connected fill by specifying the FLOATING subtype. For more information, see “ <a href="#">Fill Subtype</a> ” on page 148.  <b>Note:</b> DEF 5.6 does not support this object type. “ <a href="#">Fill Syntax</a> ” on page 107
<i>layerNumber</i>		Specifies the GDSII Stream/OASIS layer number or numbers. The number must be an integer between 1 and 65535.
<i>dataType</i>		Specifies the GDSII Stream/OASIS data type or data types. The data type must be an integer between 0 and 65535.

See the “[DEF Syntax](#)” chapter in the *LEF/DEF Language Reference* for more information on the object types.



Layer names or object types that exist in the Encounter database but are not specified in the map file are not output to the GDSII Stream or OASIS file.

## Specifying Object Subtypes

You can specify subtypes for some *layerObjTypes*. Specifying a subtype allows you to split the data from a *layerObjType*, so that part of it is output to one *layerName/dataType* and part of it is output to another *layerName/dataType*, or to copy it, so it is output to more than one *layerName/dataType*. For example, if you use the FLOATING subtype for FILL, you can divide the output for FILL so that FILL that is FLOATING is output to one *layerName/dataType* and FILL that is not FLOATING is output to a different

*layerName*/*dataType*, or you can output FILL that is FLOATING to a specified *layerName*/*dataType* and also output it to the same *layerName*/*dataType* as FILL that is not FLOATING.

You can specify the following subtypes:

- Floating and non-floating metal and via fill
  - For more information, see [“Fill Subtype”](#) on page 148.
- Net names
  - For more information, see [“Net Name Subtype”](#) on page 149.
- Voltage levels
  - For more information, see [“Voltage Subtype”](#) on page 149.
- VIA cut sizes
  - For more information, see [“SIZE Subtype”](#) on page 150.

## **Fill Subtype**

Use the following syntax to specify metal and via fill:

```
layerObjName layerObjType[:FLOATING] layerNumber dataType
```

:FLOATING is optional. It specifies unconnected fill. Use this syntax for FILL, FILLOPC, VIAFILL, and VIAFILLOPC shapes.

In the map file, FLOATING shapes can be output to a different *layerNumber*/*dataType* than the non-FLOATING (connected) shapes, or they can be output to the same *layerNumber*/*dataType* and to a different *layerNumber*/*dataType*.

For example, to divide the output of metal fill shapes, so that non-floating fill on M1 is output to *layerNumber* 8 *dataType* 0 and floating fill to *layerNumber* 8 *dataType* 51, the map file would have the following statements:

```
M1 FILL 8 0
M1 FILL:FLOATING 8 51
```

To output the connected metal fill shapes on M1 to *layerNumber* 8 *dataType* 0 and floating fill to both *layerNumber* 8 *dataType* 0 and to *layerNumber* 8 *dataType* 51, the map file would have the following statements:

```
M1 FILL 8 0
M1 FILL:FLOATING 8 0,51
```

## Net Name Subtype

Use the following syntax to specify layers for nets. The syntax affects wires only, not vias.

For special nets, use the following syntax:

```
layerObjName SPNET[:netName] layerNumber dataType
```

For regular nets, use the following syntax:

```
layerObjName NET[:netName] layerNumber dataType
```

*:netName* is optional. Use the whole net name of any net.

For example, to output special nets named VDD on LEF layer *M1* to GDS layer 41, and all other special nets on LEF layer *M1* to GDS layer 31, include the following lines in the map file:

```
M1 SPNET:VDD 41 0  
M1 SPNET 31 0
```

## Voltage Subtype

Use the following syntax to specify the voltage level for nets, special nets, pins, and vias:

```
layerObjName layerObjType:VOLTAGE:minVoltage[:maxVoltage] layerNumber  
dataType
```

For example, to output nets on LEF layer *M1* with a minimum voltage of 1.8 to *layerNumber* 31 *dataType* 3, use the following syntax:

```
M1 NET:VOLTAGE:1.8 31 3
```

To output nets on LEF layer *M1* with a minimum voltage of 1.8 and a maximum voltage of 2.499 to *layerNumber* 31 *dataType* 3, use the following syntax:

```
M1 NET:VOLTAGE:1.8:2.499 31 3
```

If you specify both net names and voltages in the file, the net name overrides the voltage (because the net name is more specific than the voltage). In the following example, VDD nets are output to *layerName*/*dataType* 31 4, even whose voltage is between 1.8 and 2.499.

```
M1 NET:VDD 31 4  
M1 NET:VOLTAGE:1.8:2.499 31 1
```

As with other subtypes, you can output objects with different voltages to different *layerNames*/*dataTypes*, or you can copy the output, so that it appears in more than one *layerName*/*dataType* in the map file. In the following example, nets whose voltage is between 1.8 and 2.499 are output to both *layerName*/*dataType* 31 0 and *layerName*/*dataType* 31 1.

```
M1 NET 31 0
M1 NET:VOLTAGE:1.8:2.499 31 0,1
```

## **SIZE Subtype**

You can use the **SIZE** attribute to specify the size of cuts to be checked. The **SIZE** attribute applies only to VIA object types (VIA, VIAFILL, and VIAFILLOPC) and to their cut layers. A warning message is displayed if the **SIZE** attribute is applied to a non-cut layer or a non-VIA object.

The map file syntax is as follows:

```
layer VIA:SIZE:value1xvalue2 gdsLayer gdsDatatype
layer VIAFILL:SIZE:value1xvalue2 gdsLayer gdsDatatype
layer VIAFILLOPC:SIZE:value1xvalue2 gdsLayer gdsDatatype
```

The cut size values **value1** and **value2** are specified in microns.

Examples of usage of **SIZE** attribute are given below:

```
V12 VIA:SIZE:0.1x0.1 41 0
V12 VIA:SIZE:0.1x0.2 41 1
V12 VIA:SIZE:0.2x0.2 41 2
```

For rectangles both the cut orientations are checked using one statement. For example, cuts 0.1x0.2 and 0.2x0.1 are checked using the following statement:

```
VIA12 VIA:SIZE:0.1x0.2 41 1
```

It is recommended to define a via without using the **SIZE** attribute. For example,

```
VIA12 VIA 41 0
VIA12 VIA:SIZE:0.1x0.1 41 0
VIA12 VIA:SIZE:0.1x0.2 41 1
VIA12 VIA SIZE:0.2x0.2 41 2
```

In this case, all the possible cut sizes are checked. If, say, three standard cut sizes are specified, the “default” size is picked and not the one specified using the **SIZE** attribute. The “unsized” construct is used to check cuts that do not have standard sizes.

For 0.1x0.1 VIA defined without a **SIZE** attribute, you can also specify a simpler usage, such as,

```
VIA12 VIA 41 0
VIA12 VIA:SIZE:0.1x0.2 41 1
VIA12 VIA SIZE:0.2x0.2 41 2
```

## Using Multiple Layers and Data Types

The following examples show the use of multiple layers and data types.

To output ...	To ...	Use ...
M1 NET 31 0	Single layer, single data type	31:0
M1 NET 31 0,1	Single layer, two data types	31:0, 31:1
M1 NET 31,32 0	Two layers, single data type	31:0, 32:0
M1 NET 31,32 0,1	Two layers, two data types	31:0, 31:1, 32:0, 32:1

## Updating Files during an Encounter Session

The following table lists the files you can replace or update incrementally during an Encounter session:

Type	Replace	Update	How
ILM	N	N	
LEF	N	Y	<code>loadLefFile -incremental</code>
Encounter Tech File	N	N	
Timing Libraries	N	N	
Timing Constraints	Y	Y	<code>loadTimingCon -incr</code>
Stamp Models	N	N	
I/O Assignment File	Y	N	<code>loadIoFile</code>
Partition File	Y	N	<code>specifyPartition</code>
Floorplan File	Y	N	<code>loadFPlan</code>
Placement File	Y	N	<code>restorePlace</code>
Routing File	Y	N	<code>restoreRoute</code>
Special Route File	Y	Y	Use <code>loadSpecialRoute</code> to replace
DEF	Y	Y	<code>defIn</code> (use <code>-scanChain</code> option to update scan chains)
TDF	Y	Y	<code>tdfIn</code>
PDEF	Y	Y	<code>pdefIn</code>

\* The Encounter software loads information for display only. You cannot edit it.

---

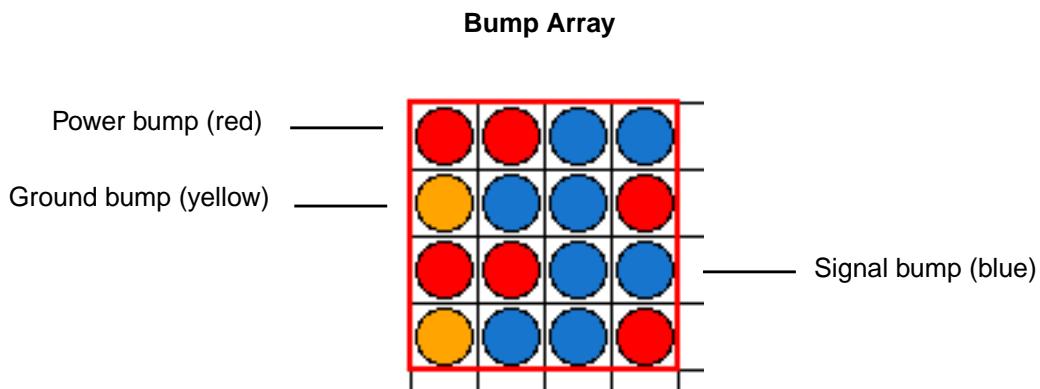
## Flip Chip Methodologies

---

- [Overview](#) on page 154
- [Flip Chip Flow in Encounter](#) on page 156
- [SiP Bump Flow](#) on page 161
- [Area I/O Flow](#) on page 163
- [Peripheral I/O Flow](#) on page 167
- [Differentiating Area I/O and Peripheral I/O](#) on page 181
- [Point-To-Point Routing](#) on page 182
- [Swapping Signals](#) on page 185
- [Creating Differential Routing to Signal Bumps](#) on page 187
- [Examples and Report Files](#) on page 192

## Overview

Flip chip is a methodology for placing I/O bumps and driver cells over the entire chip area in either a boundary (peripheral I/O) or core (area I/O) configuration. The Encounter flip chip design handles bump arrays, I/O drivers, electrostatic discharge cells (ESDs), and routing information. Power, ground, and signal assignments are made after the bumps are placed.



**Note:** Flip chip is sometimes referred to as area I/O placement in Encounter documentation. Area I/O placement is a subset of flip chip.

### Related Packaging Tools

Allegro® Package Designer (APD) and Allegro® SiP Digital Layout are related packaging tools that interface with flip chip. You must have a separate license to run APD. The documentation for APD is provided in the *Allegro® Package Designer User Guide* available on SourceLink.

To check the package routing from the bump array, use the APD/SiP tool.

### Before You Begin

Before using flip chip, the following information is required:

- Parameter data for:
  - Bumps
  - I/O drivers

## Using this Chapter

The flows in this chapter include steps with examples of how to use flip chip.

- For general flip chip flow information, see “[Flip Chip Flow in Encounter](#)” on page 156.
- For information on a specific type of flow, see one of the following sections:
  - [SiP Bump Flow](#) on page 161
  - [Area I/O Flow](#) on page 163
  - [Peripheral I/O Flow](#) on page 167

## Related Flip Chip Information

- Text commands

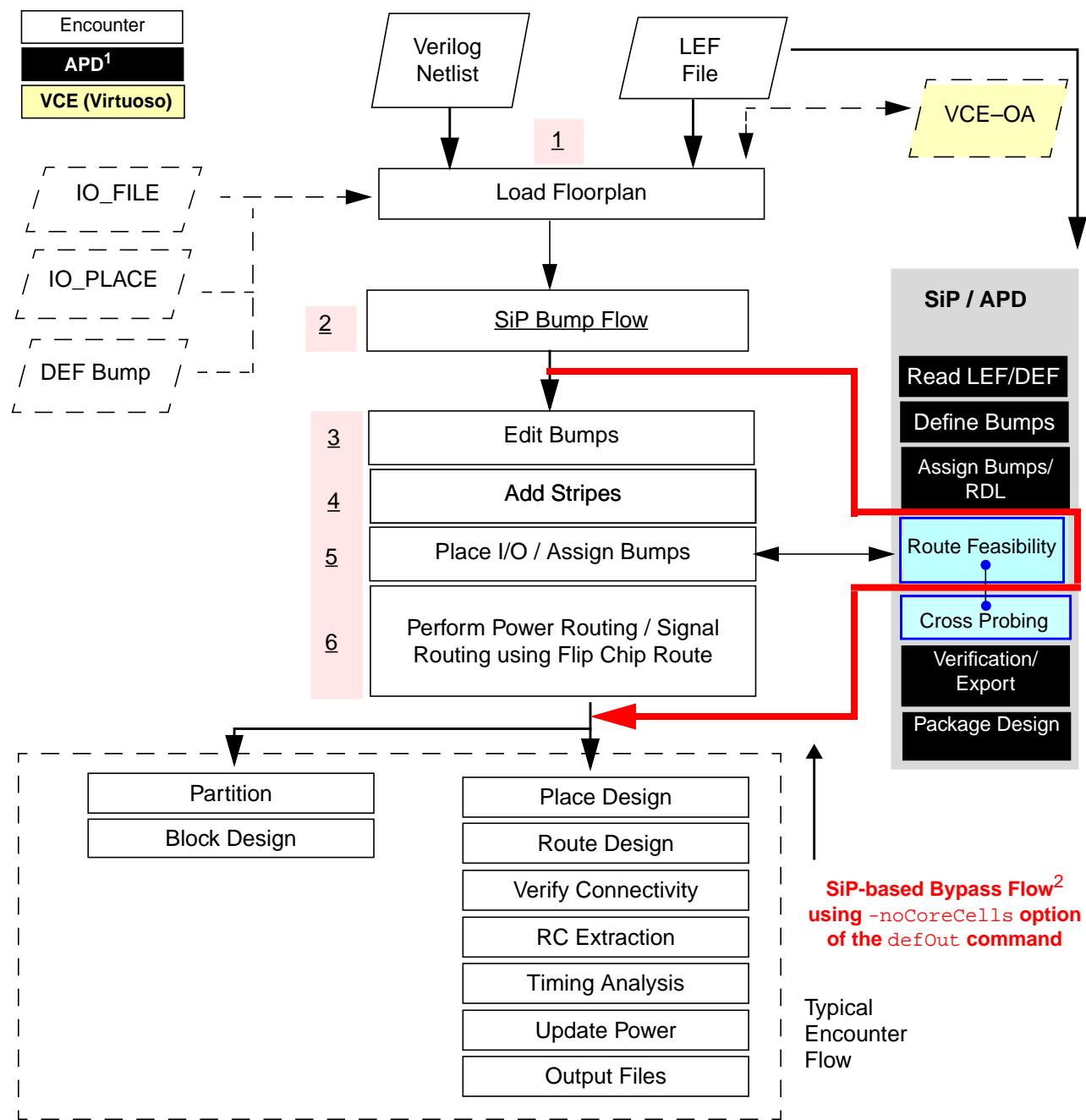
For information on the flip chip commands, see the “[Flip Chip Commands](#)” chapter of the [\*Encounter Text Command Reference\*](#).

- Menu Commands and Forms

For information on the flip chip menu forms, see the “[FlipChip Menu](#)” chapter of the [\*Encounter Menu Reference\*](#).

## Flip Chip Flow in Encounter

The following figure shows the general Encounter flip chip flow including sub flows.



<sup>1</sup> License Required

<sup>2</sup> Bypasses *FlipChip* menu (see [Reducing Data Size for SiP Import \(Bypass Flow\)](#) on page 161)

## Flip Chip Flow Steps

### 1. Load the floorplan.

Load the floorplan as in a typical Encounter flow.

**Note:** The floorplan information can be passed to SiP through the DEF file.

The following files are imported during this step:

- Verilog netlist

A Verilog structural netlist is required for the design connectivity. No bumps are allowed in the netlist since they are physical cells.

- LEF File

The LEF input files must contain the normal technology information, standard cell macros plus the IO PAD, and bump LEFS.

- The LEF BUMP MACRO must contain CLASS COVER BUMP.
  - The LEF IO Driver cells must contain CLASS PAD (peripheral I/O) or CLASS PAD AREAIO (area I/O).

For more information, see [Differentiating Area I/O and Peripheral I/O](#) on page 181.

*Text Command:* [loadLefFile](#)

- OA database via Virtuoso (VCE)

The Virtuoso Chip Editor (VCE) can be used through the OpenAccess (OA) 2.0 database.

**Note:** This is a specialized flow. The VCE data should be imported as flat so the routes can be extracted.

- IO\_FILE, IO\_PLACE, or DEF Bump file

Import either the IO\_FILE, IO\_PLACE, or DEF Bump file.

- The IO\_FILE contains bumps, I/O rows (optional), and I/O instances (optional).  
For an example IO\_FILE, see "[IO\\_FILE Example](#)" on page 194.

*Text Command:* [loadIoFile](#)

- The IO\_PLACE file can be used for specific placement of peripheral I/Os or double I/O rows.

*Text Command:* [loadIoFile](#)

- The DEF file can be used to import bumps.

*Text Command:* defIn

**2.** Define the bumps using the bump flow.

- Bump Flow— See “SiP Bump Flow” on page 161.

The area I/O flow supports several methods to define the bumps:

- Bump Matrix Generation  
Use the bump matrix generator. These bumps will be assigned later in step 5.
- IO\_FILE Generation  
Generate an IO\_FILE that contains the x and y locations of the bumps along with the x and y locations of the I/O rows. The I/O rows are the rows or sites into which the I/O driver cells are placed. These bumps may or may not be assigned to signals at this time.
- APD Bump Generation  
Use APD to generate the bump matrix or other DEF input file, and pass the bumps via a DEF instance.

**3.** Edit the bumps.

Use the following flip chip forms to edit bumps:

- Edit Bump Array
- Add Bump to Bump Array
- Unassign Bump
- Swap Signals

**4.** Add stripes.

Generate the power stripes on the chip using the addStripe text command.

**5.** Place driver cells and assign bumps.

Use the placeAIO -onlyAIO -assignBump command and options to place the area I/O driver cells into the rows/sites closest to the corresponding bumps. If the bumps are not assigned at this time, this command assigns the bumps and also place all of the standard cells, if requested.

Use the placePIO command to perform initial peripheral I/O pad placement. After you run the assignBump command to assign the signal and power/ground bumps, use the

`placePIO -assignBump -noRandomPlacement` command and options to optimize the initial bump assignment.

**6. Connect the power and ground bumps / signal bumps.**

Use the `fcroute` -type power command and option to connect the power and ground bumps to stripes.

Use the `fcroute` -type signal command and option to connect the signal bumps to the I/O driver cell specified in the netlist.

If required, use the `routePointToPoint` command for SPECIALNETS, to connect any remaining I/O pad pins and bumps, or wires and bumps, or bumps and stripes that were not routed correctly during `fcroute`.

 **Important**

Before running the `placePIO` and `fcroute` commands, you must specify the flip chip constraints using the `setFlipChipMode` command which loads data for `placePIO` and `fcroute` commands.

**Note:** If you want to view the flight lines before you route the bumps, you must first be in the Floorplan view. Then, use the left mouse button to click on the bump.

The remainder of the flow is similar to the typical Encounter flow.

**7. Partition the design.**

Bumps, bump routing, power routing, and I/O driver cells can be pushed down as blockages into the partition. See `specifyPartition` and `handlePtnAreaIo` commands for more information.

**8. Place the design.**

Place the design using the `placeAIO` command.

**9. Route the design.**

NanoRoute (`globalDetailRouteBatch` command) can be used to connect the regular nets in the design.

**10. Verify the connectivity.**

Verify the bump (physical cells) connections to the logical cells using the `verifyConnectivity` command.

**11. Run extraction.**

Extract the RC data using the `runQRC` command or the `extractRC` command and then generate a SPEF file. The `runQRC` command input is the DEF output file which contains the bumps that are not present in the original Verilog file. You can create a Verilog output file containing bumps to match the `runQRC` command SPEF.

**Note:** You can also create a defout file and convert the bumps to pins so you do not have to create a physical verilog.

**12.** Do a timing analysis.

The timing analysis report is the same as in the normal Encounter flow. See [report timing](#) command.

**13.** Update power.

Update power using the `updatePower` command. A flip chip design can have multiple power sources.

**14.** Output the files.

Write out the DEF, Verilog, OpenAccess, SPEF, and GDSII files. The `defOut` command contains the `-noCoreCells` option to reduce the data sent to APD. For more information, see the “[Reducing Data Size for SiP Import \(Bypass Flow\)](#)” on page 161.

## SiP Bump Flow

For information on the SiP bump flow, see System-in-Package Flow Guide available on SourceLink or in the SiP Product Help.

### Reducing Data Size for SiP Import (Bypass Flow)

You can use the `-noCoreCells` option of the `defOut` command to reduce data size for import into SiP. The syntax is as follows:

```
defOut -noCoreCells
```

This flow bypasses the bump flow (see [Flip Chip Flow in Encounter](#) on page 156).



You should use the `-noCoreCells` option whenever you are creating a DEF file for SiP.

### Splitting Wires in Metal Layers

If wires that route the bumps are wider than the LEF MAXWIDTH parameter, you can use the `editFixWideWires` command to split them.

For wires splitting in specific metal layers, you can modify a LEF layer with a specific MAXWIDTH parameter as shown in the following example for LAYER M5.

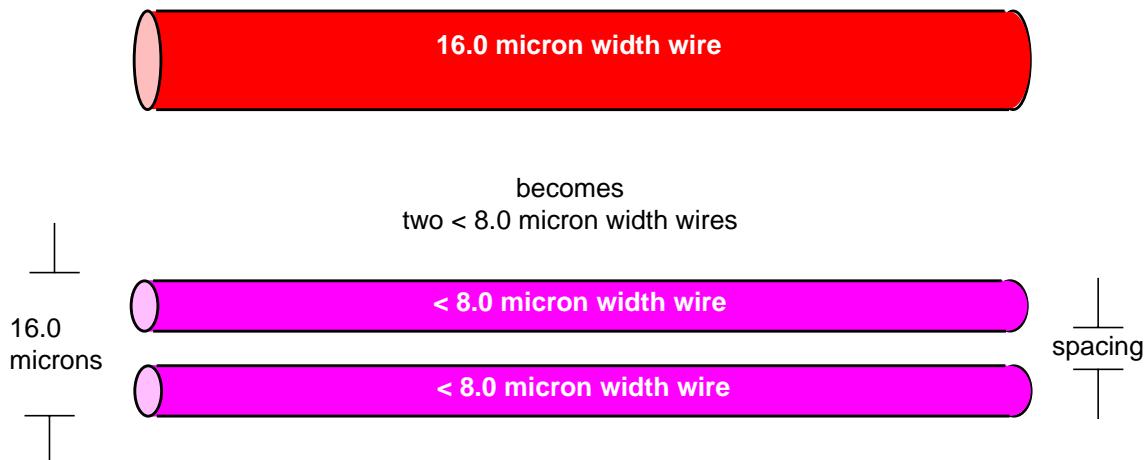
```
LAYER M5 TYPE ROUTING ; DIRECTION VERTICAL ;
    WIDTH 0.70 ; SPACING 0.70 ; PITCH 1.4 ;
    CAPACITANCE CPERSQDIST 0.0001000 ; RESISTANCE RPERSQ 0.010000 ;
    MAXWIDTH 8.0 ;
END M5
```

After running the `editFixWideWires` command, wires in this layer are split to satisfy the MAXWIDTH value in the LEF file.

The following figure shows how a 16.0 micron wire is split using this LEF layer code and the `editFixWideWires` command. The resulting split wires will be slightly less than 8.0 microns each. There will be a split spacing between the wires such that the total width is 16.0 microns.

The split spacing is automatically determined by considering the MINSPACING, PARALLEL RUNLENGTH SPACING, and DENSITY constraints. The split spacing will be greater than or

equal to the MINSPACING constraint. There is no manual control for the split spacing parameter.



## Testing the Package Routing Feasibility

You can test the package routing feasibility of the design using APD / SiP.

For more information, see the *Cadence Chip I/O Planner User Guide* or the *SiP Digital Architect/Layout User Guide* on SourceLink.

## Area I/O Flow

For Area I/O designs, bumps are placed within the core area of the design, and the bonding pads are not built into the bump cells. This means that the bump cells require routing to the pads.

To create an Area I/O design, complete the following steps:

1. Load the floorplan.

Use the [Load FPlan File](#) form to load the floorplan file.

2. Define the bumps.

Use the [Create Bump Array](#) form to set up the bump array.

3. Create area I/O driver rows.

Use the [Create Area IO Row](#) form to set up the area I/O rows.

4. Place Area I/O pads and standard cells.

Use the [Place Area I/O](#) form to place I/O driver cells.

5. Assign signals, power, and ground to the bumps.

Use the [Assign Signals](#) form to assign the signals to the bumps. Signal bumps are blue-filled squares.

Use the [Assign Power/Ground Bumps](#) form to assign power and ground to bumps. Power bumps are red-filled squares. Ground bumps are yellow-filled squares.

6. Add area I/O filler cells in the blank sites of the specified area I/O row clusters using [addAIOFiller](#) command.

7. Create power rings and stripes.

Use the [Add Rings](#) form to create rings around the core area and around the power and ground bumps.

Use the [Add Stripes](#) form to create stripes that connect to the power and ground bumps.

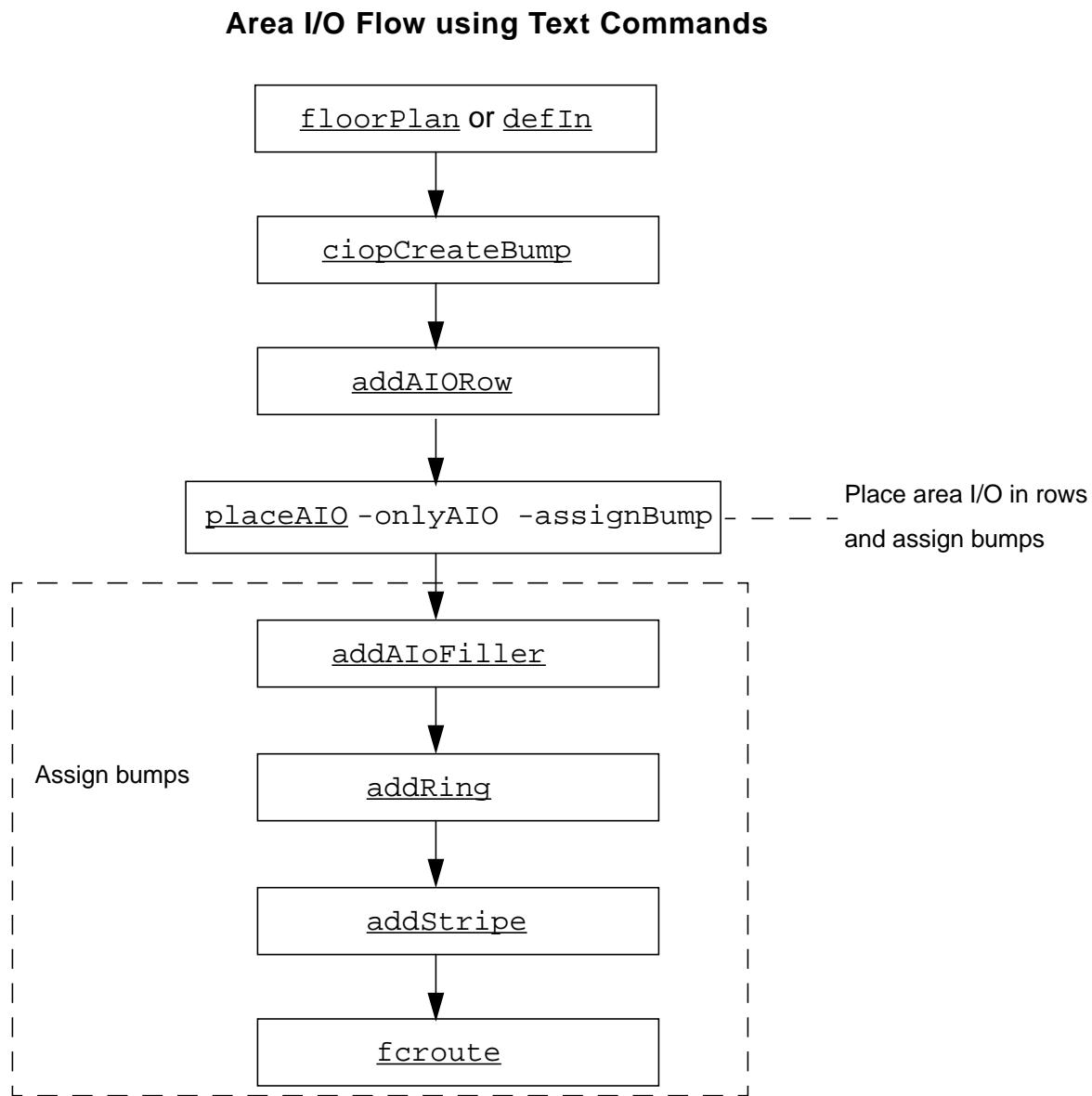
8. Connect power, from bumps to I/O cells or from bumps to rings/stripes.

Use the [Route Flip Chip - Advanced - Routing Style](#) form to establish the power connections.

**Note:** The remainder of this flow is similar to the typical Encounter flow.

## Area I/O (AIO) Command Flow

The area I/O command flow is described as follows:



## Routing Bumps to I/O Driver Cells (Hierarchical Area I/O Flow)

The hierarchical area I/O flow allows you to route the bumps, using the fcroute command, to I/O driver cells and then push down (partition) this data into a lower level.

The text command is:

```
handlePtnAreaIo buffer_name
```

This command pushes down data in the partition as follows:

- Bumps become routing blockages
- I/O cells become placement and routing blockages
- An internal pin is created over the I/O cell pin
- A boundary pin is created
- A buffer is created between the internal pin and the boundary pin

**Note:** If you want to view the flight lines before you route the bumps, you must first be in the Floorplan view. Then, use the left mouse button to click on the bump.

## Flip Chip Routing on Shielded Nets in AIO

When using the `fcroute` shielding option in the AIO mode with manhattan (90 degree) routing style, the `defOut` marks the shielded nets as 'SHIELD', while displaying the `SHAPE` and `ROUTED` status of the metal shield wire.

**Note:** Shielding nets is not supported in PIO mode.

## Example

Consider the following example in which the `fcroute` command connects signal bumps to I/O cells using 90 degree signal routing for AIO; The command adds a side shield (`VSS`) on both sides of the signal route.

## Command

```
fcroute -type signal -designStyle aio -routeStyle manhattan -layerChangeTopLayer 8  
-layerChangeBotLayer 7 -routeWidth 8 -constraintFile CFG/aio.constr
```

## Constraint File CFG/aio.constr: Shield Net Description

SHIELDING

```
SHIELDBUMP true  
SHIELDWIDTH 0.4  
SHIELDLAYERS abc  
SHIELDNET VSS  
scan_out_2  
port_pad_data_out[15]
```

## Encounter User Guide

### Flip Chip Methodologies

---

```
END SHIELDING
```

## DEF Syntax

The defOut contains the SHIELD syntax as follows:

```
-scan_out_2 ( Bump_27_6_2 PAD ) (IOPADS_INST/Pscanout2op PAD)
+ ROUTED METAL8 16000 + SHAPE IOWIRE (1255310 541920 ) ( 1369310 * )
NEW METAL8 16000 + SHAPE IOWIRE (1263310 533920 ) ( * 695760 )
+ PROPERTY BUMP_ASSIGNMENT "ASSIGNED"
;
-VSS ( * VSS )

+ SHIELD scan_out_2 METAL8 800 + SHAPE IOWIRE ( 1275310 554320 ) ( 1315310 * )
NET METAL7 16000 + SHAPE IOWIRE ( 1255310 541920 ) ( 1315310 * )
NET METAL8 800 + SHAPE IOWIRE ( 1250510 529520 ) ( 1315310 * )
+ ROUTED METAL6 16000 + SHAPE STRIPE ( 1553200 109600 ) ( * 186800 )
NET METAL6 16000 + SHAPE STRIPE ( 1753200 109600 ) ( * 186800 )
+ SHIELD scan_out_2 METAL8 800 + SHAPE IOWIRE ( 1275710 553920 ) ( * 675760 )
METAL7 16000 + SHAPE IOWIRE ( 1263310 533920 ) ( * 695760 )
METAL8 800 + SHAPE IOWIRE ( 1250910 529120 ) ( * 675760 )
```

## Peripheral I/O Flow

The peripheral I/O approach to flip chip methodology places I/O driver cells at the edges of the core area of the design. This means that the bump cells require routing to the I/O driver cells using the top-most layer with or without one extra layer below. This layer is called the redistribution layer (RDL), and is used to connect the bumps to the I/O pads. The procedures and examples in this section use the two-layer approach.

The peripheral I/O flow is similar to area I/O, wherein you can use I/O rows (regions/sites) to place the I/O driver cells since they remain on the boundary. The peripheral I/O flow also includes non-orthogonal (45-degree) RDL routing, I/O cell optimization, and bump reassignment for better single layer routing.

Since the top two layers are used for RDL routes, and RDL routes are wider than regular routes, coupling effects from the RDL routes to regular routes can be significant. To avoid huge coupling effects, avoid regular routing in one layer below the RDL.

There are three major aspects of the peripheral I/O flow:

- RDL planning and routing
  - The automatic placement of the I/O cells on the edge of the design
  - The optimization of the I/O cells and the reassignment of bumps to enhance single layer routing.
  - Non-orthogonal routing on the redistribution layer (RDL).
- RC extraction
- Signal integrity and timing analysis

## Data Preparation

The LEF CLASS statements for I/O pad cells and bump cells must contain the following classes for the peripheral I/O flow to work.

I/O cell: CLASS PAD AREAIO

Bump cell: CLASS COVER BUMP

These are the LEF properties used for connecting power/signal bumps to power/signal I/O cells.

Normally, the bump to I/O pad connection is defined in the Verilog file. The signal names are specified in the Verilog top module port list, and the I/O cells are connected to these ports.

For I/O power pads which are not defined in the Verilog file, you can define the connection of the I/O pads to bumps using the command:

- `fcroute -connectPowerCellToBump`

The MACRO `PIN` statement added in LEF 5.7 tells which power/ground pin shape on the I/O driver cell must be connected to a bump. See [Defining the Connection between a Bump and P/G Pin Shape](#) in the “[Data Preparation](#)” chapter of the *Encounter User Guide*.

## Peripheral I/O Flow Steps

The peripheral I/O implementation flow is similar to the traditional physical implementation flow, except for the handling of bump cells and RDL routing.

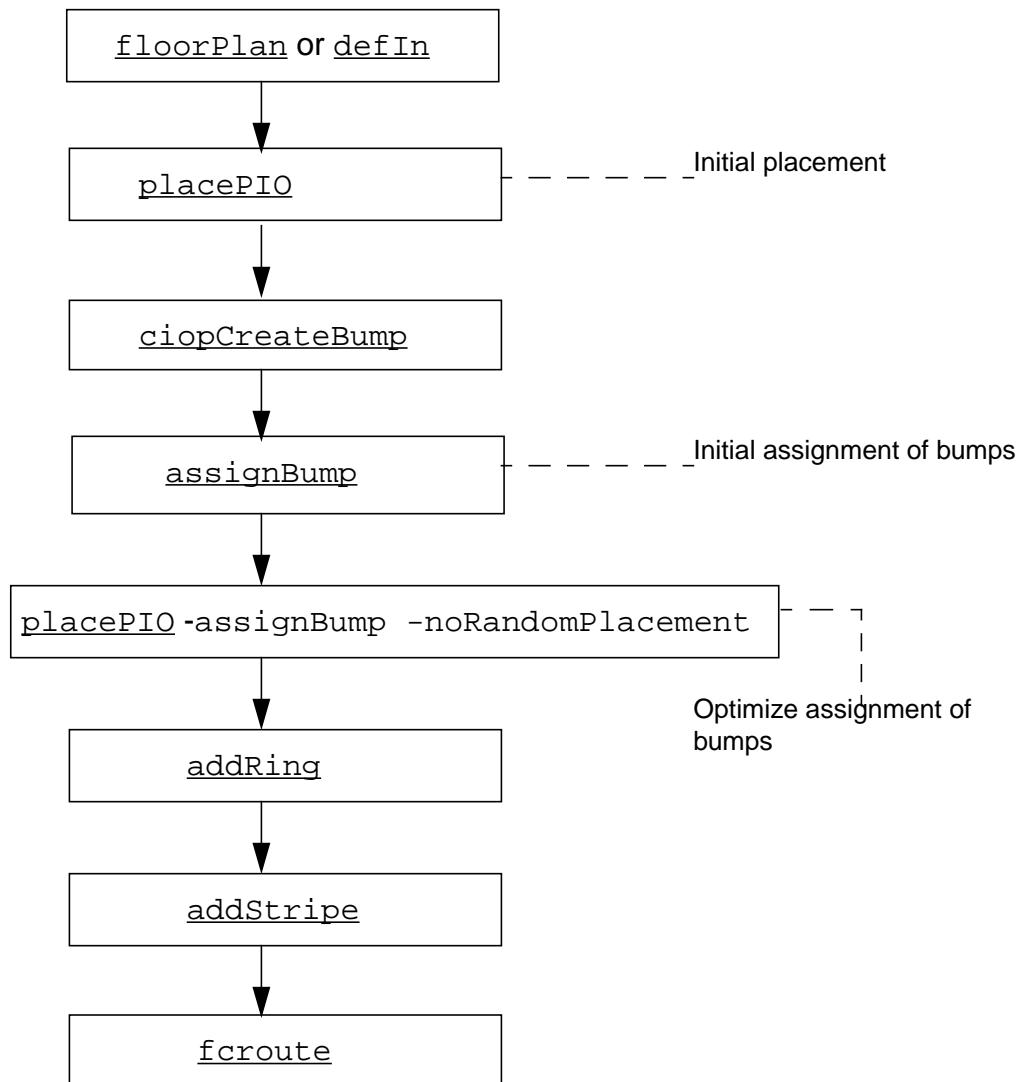
There are four major elements of the flow:

- After the initial floorplanning stage (set die and area and place I/O driver cells), the RDL implementation flow includes bump placement and assignment, optimization of I/O driver cell placement, and RDL routing.
- The bump placement and assignment is passed to APD (Allegro® Package Designer) for package design. You can determine the route feasibility by using APD to check the bump routability to the package. This can be invoked from the Encounter user interface.
- The RDL-routed design is then ready for power planning / QRC / other placement and routing operations.
- Initial parasitics can be extracted in Encounter using the `extractRC` command. If more accurate parasitics are required, the signal-routed design can be streamed out in GDSII format and sent to Assura™ RCX for extracting RC parasitics, which can be used for timing and SI analysis with the RDL effects.

## Peripheral I/O (PIO) Command Flow

The peripheral I/O command flow is described as follows:

### Peripheral I/O Flow using Text Commands

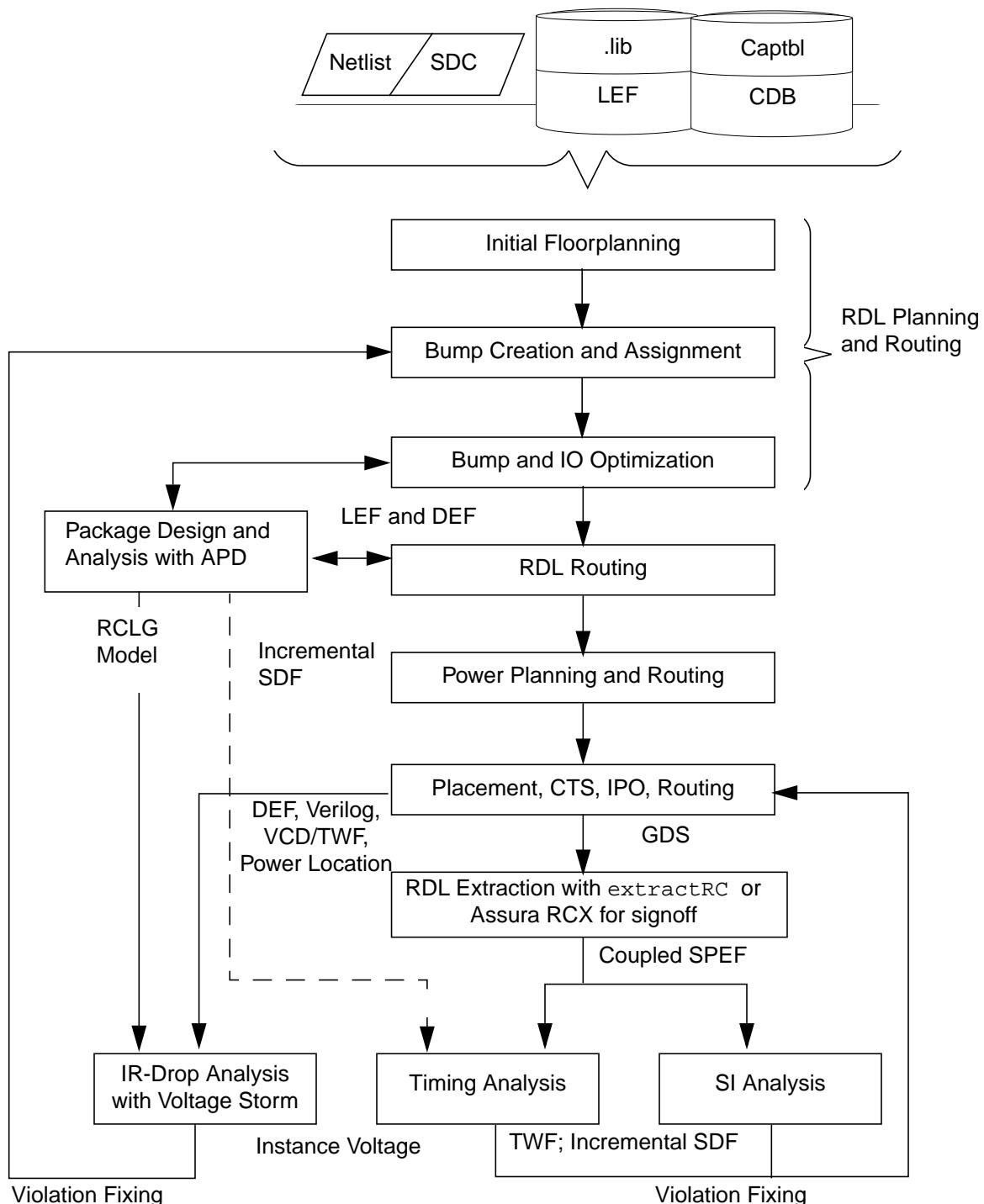


## Encounter User Guide

### Flip Chip Methodologies

---

The following flow diagram shows the major flow components for implementing an RDL design.



## RDL Planning and Routing

The following are the basic steps for planning and routing in a peripheral I/O flow.

**1. Load the floorplan.**

Use the [Load FPlan File](#) form to load the floorplan file.

**2. Define the bumps.**

Create a bump matrix based on bump pitch and other parameters by using the Encounter bump selection and assignment user interface or the `ciopCreateBump` text command.

From the Encounter user interface, select *FloorPlan -> FlipChip -> Create Bump Array*.

**3. Place the peripheral I/Os.** See “[Place peripheral I/O pads](#)” on page 173.

**4. Assign the power and ground bumps either by loading a predefined I/O File using the `loadIoFile` command or by using the Encounter bump selection and assignment user interface or the text command, `assignPGBumps`.**

From the Encounter user interface, select *FloorPlan->Flip-Chip->Assign Power/Ground*.

**5. Assign the signal bumps by either loading a predefined I/O File using the `loadIoFile` command or by using the Encounter bump selection and assignment user interface or the text command, `assignBump`.**

From the Encounter user interface, select *FloorPlan->Flip-Chip->Assign Signal*.

The `assignBump` command uses the signal names (ports) in the Verilog top module list and assigns them to the closest I/O cell. The `assignBump` command assumes the I/O cells have been preplaced.

**6. Route the signal and power/ground bumps to the I/O driver cells or power/ground stripes using the `fcroute` command.** See “[Route bumps](#)” on page 177.

**7. If the routing is not optimal, either reassign the bumps or change the I/O cell placement using the `placePIO` command.** See “[Reassign bumps](#)” on page 176.

**8. Snap or split route.**

Use the `snapRoute` command to snap the 45-degree routes created by APD to the manufacturing grid.

Use the `splitRoute` command to split 45-degree routes that are wider than the maximum width.

**9.** Create power rings and stripes.

- Use the [Add Rings](#) form to create rings around the core area and around the power and ground bumps.
- Use the [Add Stripes](#) form to create stripes that connect to the power and ground bumps.

**10.** Connect power, from bumps to I/O cells or from bumps to rings/stripes.

Use the [Route Flip Chip - Advanced - Routing Style](#) form to establish the power connections.

**Note:** The remainder of this flow is similar to the typical Encounter flow.

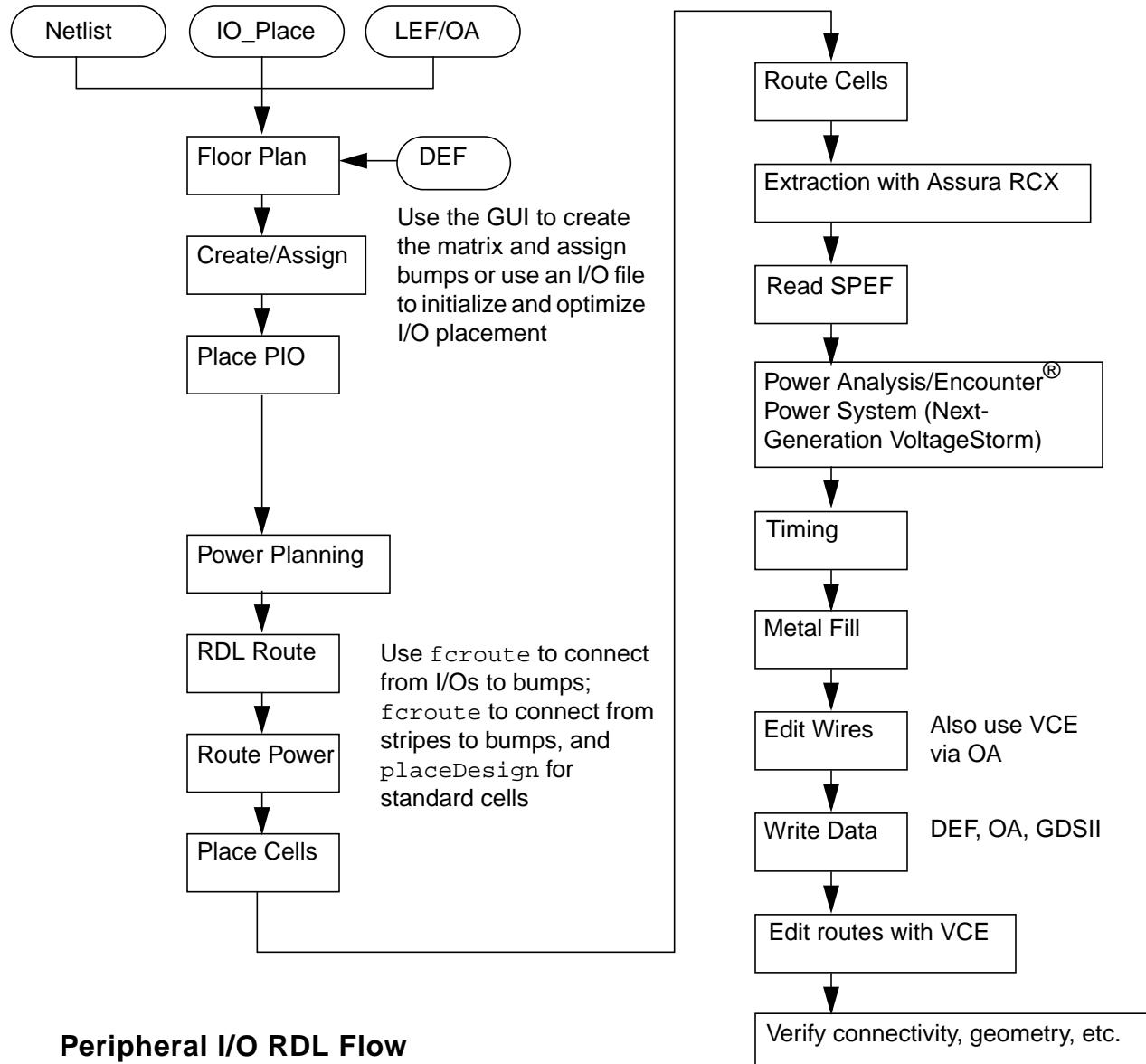
The Encounter log file displays the routing status of each bump-pad pair in the fcroute PIO mode.

## Encounter User Guide

### Flip Chip Methodologies

---

The following diagram shows the peripheral I/O task flow.



### Place peripheral I/O pads

Since CLASS PAD AREAI/O cells are not automatically placed, you must invoke a command to randomly place the I/Os on the periphery. This command has an option to specify the number of peripheral I/O rows (rings).

## Encounter User Guide

### Flip Chip Methodologies

---

From the Encounter user interface, select *Place* -> *Flip-Chip* -> *Place Peripheral IO*.

The syntax for the placePIO command is

```
placePIO [-assignBump] [-optIOs] [-overflowMap] [-maxIOHeight value] [-ioFile  
fileName] [-rdlConstraintFile fileName] [-noRandomPlacement] [-extraConfig  
filename] [-cellList {cellList}] [-instList {instList}] [-fixPadSide] [-  
fixNetPadSide]
```

**Note:** The placePIO command also reads flip chip options from setFlipChipMode command.

Another method for creating the initial I/O placement is to read in an I/O file specifying the Pad: or IOInst: syntax with the instance name and side of the design or XY location.

You can refine the initial placement or reassign bumps by using various command options. The initial placement can be modified in two ways:

- Fixed Bumps

If the bumps have been assigned, the I/O cells can be moved using the placePIO command to help ensure a one-layer route.

- Fixed I/Os

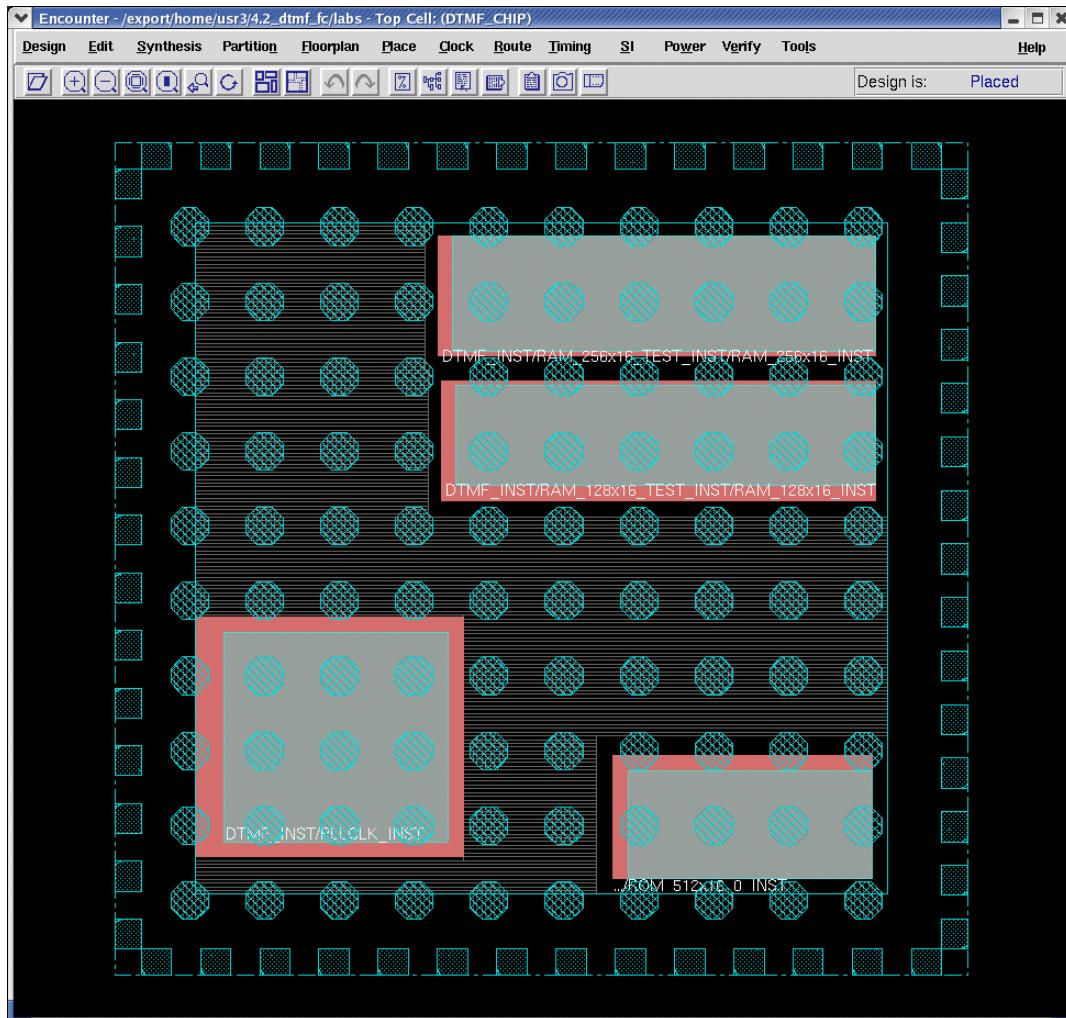
The bumps can be reassigned to improve the routing if the I/O cells have been fixed.

Once the placement is finished, the data can be stored in the floor plan file and restored. The I/O cells can also be moved manually with the move command since there are no specified I/O rows.

## Encounter User Guide

### Flip Chip Methodologies

The following figure shows the results of the `placePIO` command.



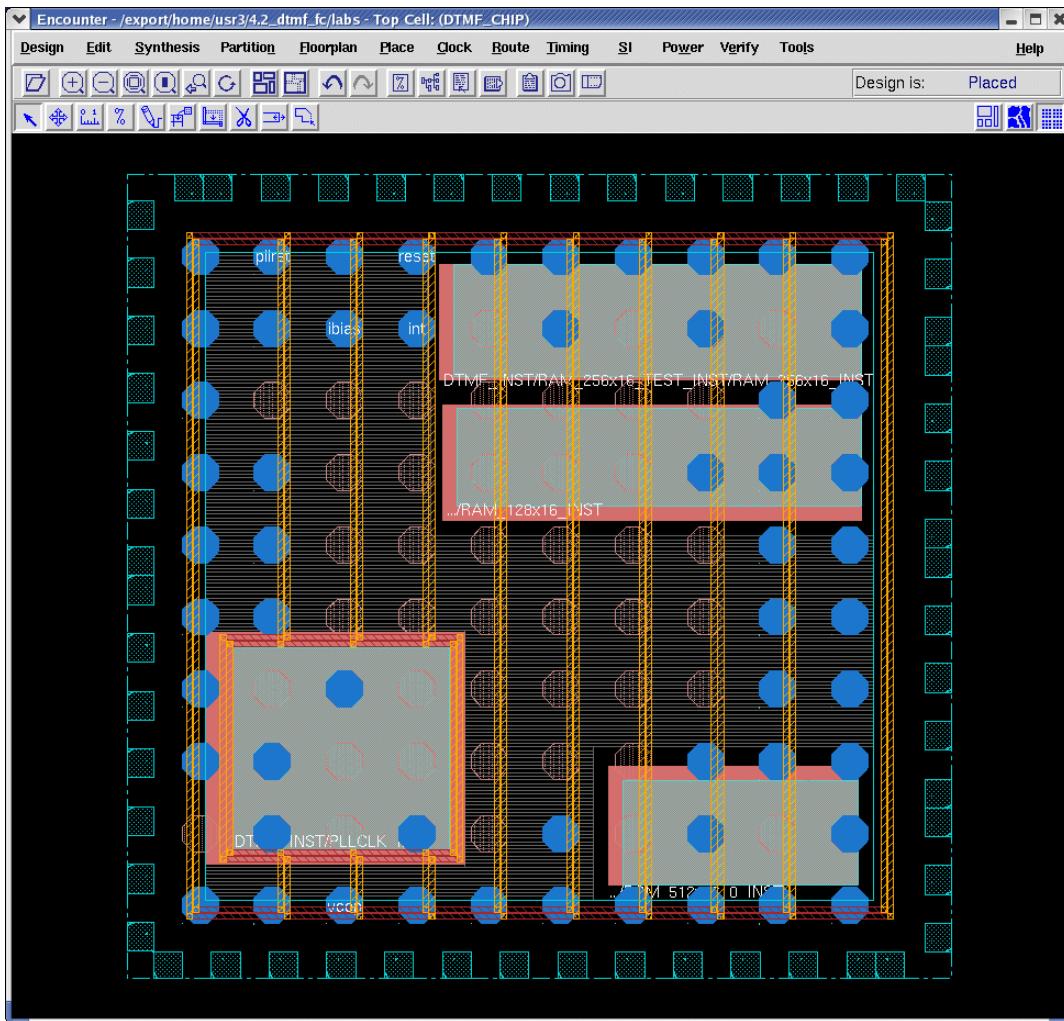
### Optimize peripheral I/O placement

During peripheral I/O placement, you can specify a constraint file which controls certain features of the optimization. For an example of a constraint file, see [Routing and Placement Constraints](#) on page 192.

## Encounter User Guide

### Flip Chip Methodologies

The following figure shows the results of the `placePIO` command.



**Note:** `placePIO` has two features. The first is to randomly place the I/O cells on the periphery, and the second is to optimize the I/O pad cells. If you specify an I/O file with the `-ioFile` option or use the `-noRandomPlacement` option, `placePIO` does not do a random placement.

### Reassign bumps

To reassign bumps without using the `placePIO` command, use the `assignBump -pio` command. If you use the `assignBump` command without the `-pio` option, it uses a placement algorithm; with the `-pio` option, it uses the global routing algorithm which improves single-layer routing.

## Route bumps

This command routes the bumps to I/O cells using 45-degree routing.

1. In Encounter, select signal routing type, using `fcroute -type signal`  
By default, 45-degree routing style is selected.
2. Select the peripheral I/O routing style using the command `fcroute -style pio`.  
This option calls both the detail and global routers to route the bumps.
3. Set the minimum escape distance from the bump to where the route can proceed at an angle. Specify either the Minimum Escape Distance constraint on the form or use the command `fcroute -minEscapeDistance unit` to specify the distance.
4. To specify the minimum distance before the route can turn, you must create a configuration file (`fcroute.config`) and include the following command:

```
srouteMinlength value
```

Use the command `-fcroute -extraConfig fcroute.config` to specify the file or specify the file from the *Route Flip Chip Advanced* form.

5. You can specify the routing constraints by using the `-constraintFile` option.

```
fcroute -constraintFile file_name
```

For example:

```
fcroute -constraintFile fcroute.constr
```

For an example of an `fcroute` constraint file, see [Routing and Placement Constraints](#) on page 192.

Alternatively, you can specify basic and advanced routing and placement constraints using the *Flip Chip Route* form in the Encounter GUI.

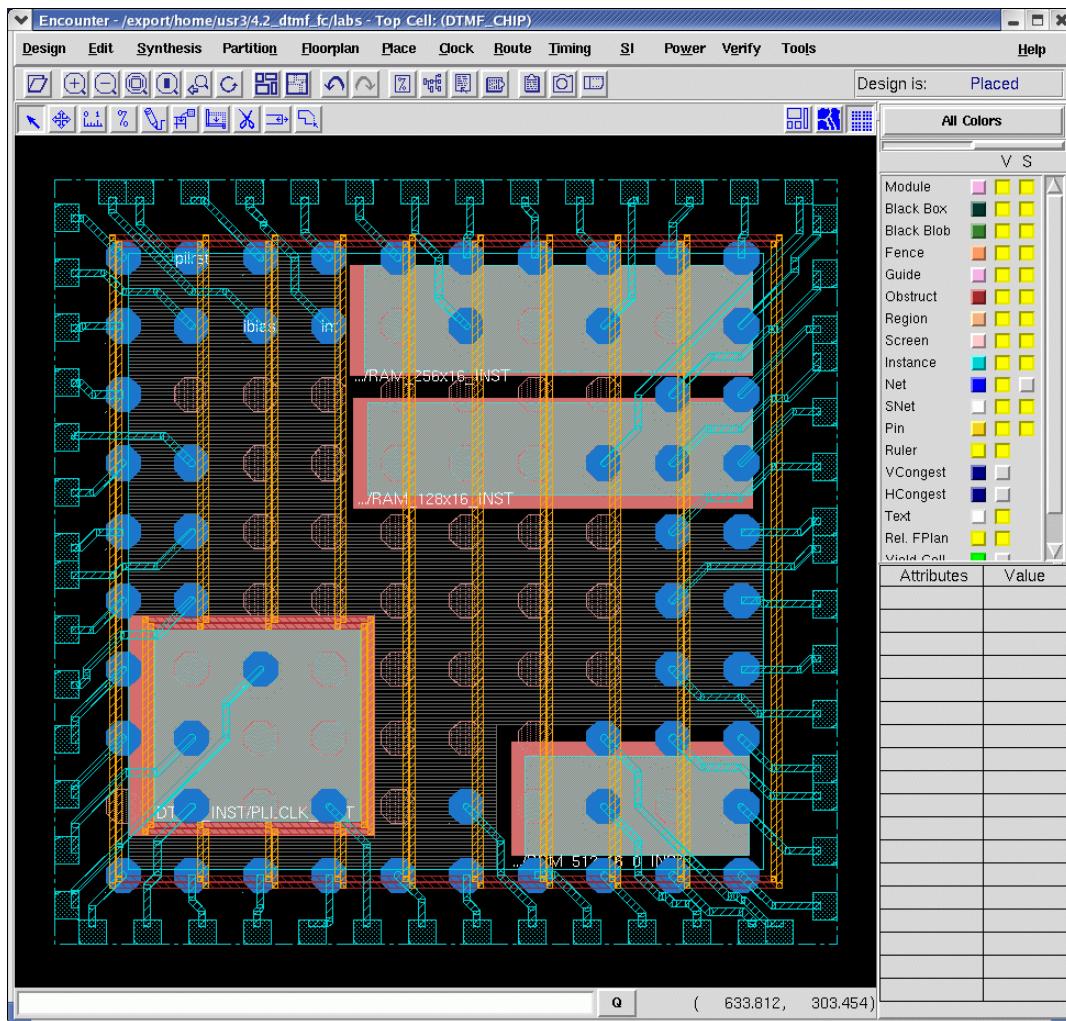
For more information, see the following topics in the “Route Menu” chapter of the *Encounter Menu Reference*:

- [Flip Chip Route – Basic](#)
- [Flip Chip Route – Advanced](#)

## Encounter User Guide

### Flip Chip Methodologies

The following figure shows the results of the `fcroute` command.



### Splitting wires

You can use the `splitRoute` command to split the RDL layer after it has been routed if you do not use the native `fcroute` splitting.

```
splitRoute [-absWidth value1] [-maxWidth value2] [-minSpacing value3]
```

The `fcroute` command splits the route during the routing process. The `splitRoute` command is used after routing is complete, most often when working with an APD-routed DEF tile where the route was not split.

You can invoke the wire splitting function during `fcroute` by specifying the `MAXWIDTH` value in the LEF layers section.

```
LAYER METAL7  
....  
MAXWIDTH 10.0 ;
```

### **Adding power stripes**

You can use the `addStripe` command can add a power stripe over or between power bumps without specifying the exact xy locations. If the stripe is on a different layer than the bump layer, `addStripe` will automatically drop a via array.

From the Encounter user interface, select Floorplan->Custom Power Planning->`addStripe`

The syntax for the `addStripe` command is

```
addStripe
```

### **Routing the power bumps**

- Route the power routes to the stripes by using the Encounter user interface or the `fcroute` text command.

From the Encounter user interface, select Route -> FlipChip -> Power

The syntax for the `fcroute` command is

```
fcroute -type power
```

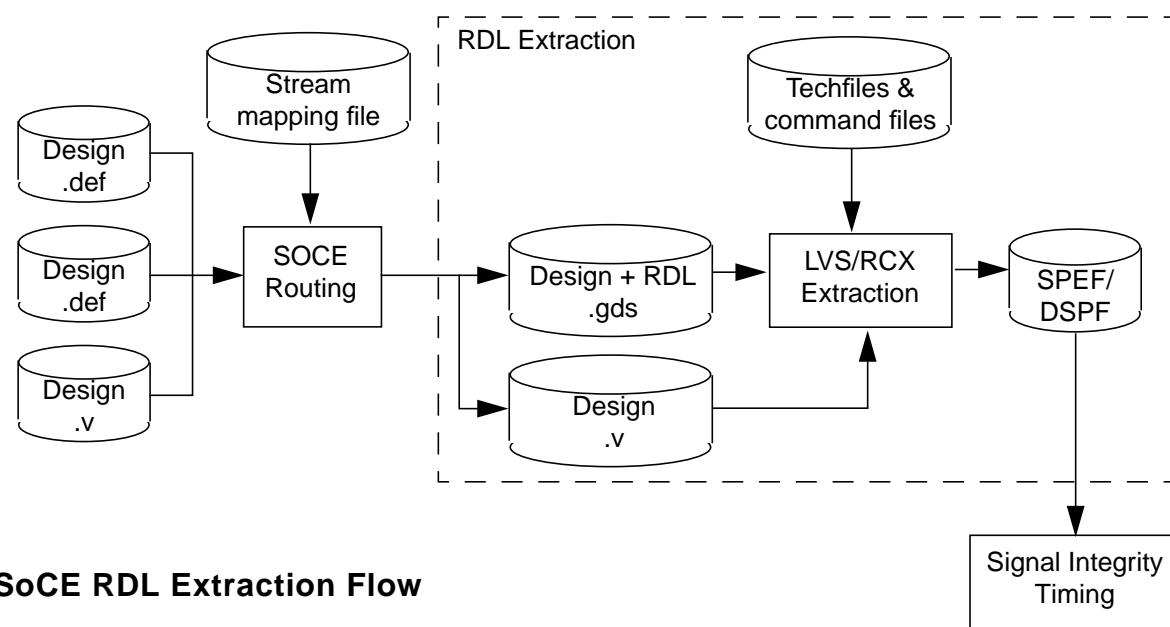
### **Peripheral I/O Extraction**

In the RDL extraction flow for designs using peripheral I/O methodology, Encounter outputs the design with the RDL routing into a GDS file that is fed into RCX for parasitic extraction at the cell-level. RCX generates a cell-level SPEF/DSPF file that is used for timing and signal integrity analysis.

There are two steps involved in parasitic extraction with RCX.

- LVS is run to perform connectivity extraction.
- RCX is run to perform parasitic extraction.

The following diagram illustrates this flow.



#### Inputs to Extraction

- Verilog netlist for annotation, generated by SoCE
- GDS of design with RDL, generated by SoCE
- RCX technology data

#### Outputs from Extraction

- Cell-level SPEF/DSPF for SI/Timing analysis
- Includes coupling RDL nets to signal nets

## SI and Timing Analysis

The following procedure describes the signal integrity and timing analysis flow for an RDL design using the coupled SPEF file generated by the RCX extraction tool.

### 1. Restore the design.

```
restoreDesign routedSession.dat designname
```

This command restores the routed view of the design including the regular routing and RDL routing.

**2.** Import the coupled SPEF file from RCX.

```
spefIn rcx_couple.spef
```

Make sure all the parasitics of the SPEF are back annotated in Encounter. If all the nets are back annotated, Encounter displays the following message:

```
0 nets are missing in SPEF file.
```

**3.** Perform timing analysis in Encounter.

- Run timing analysis using the `timeDesign` command.

```
timeDesign -postRoute -reportOnly
```

This command reports worst and total negative slack as well as register-to-register, input-to-register, and input-to-output port slacks.

**4.** Analyze signal integrity using CeltIC™ for the SI analysis in Encounter. CeltIC analyzes the design for glitch and SI violations. It generates incremental sdf for the delay induced due to SI. The incremental sdf is used to analyze timing with SI effects.

```
runCeltic -cdb ../../library/cdb/stdcells.cdb -process 90nm
```

This command analyzes the design for SI and creates the CeltIC report. Later, the command uses an incremental sdf file for timing analysis and reports the worst negative slack path with SI-induced delay.

The following listing is a sample script for signal integrity and timing analysis in Encounter.

```
restoreDesign ./routed-design.enc.dat designName
spefIn rcx_couple.spef
timeDesign -postRoute -reportOnly
runCeltic -cdb ../../library/cdb/stdcells_wc.cdb -process 90nm
```

## Differentiating Area I/O and Peripheral I/O

The LEF I/O Driver cells must contain CLASS PAD (for peripheral I/O) or CLASS PAD AREAIO (for area I/O).

**Note:** Depending on your design style, you may need to modify the LEF macro CLASS statement.

■ Area I/O

CLASS PAD AREAIO = I/O cell without bump.

CLASS PAD AREAIO is used by the `assignBump` and `placeAIO` commands.

Additionally, the SITE must be defined and referenced in the LEF macro. See [Performing Area I/O Placement](#) on page 111 in the Data Preparation chapter for more information and example.

■ Peripheral I/O

CLASS PAD = I/O cell with bound pad.

CLASS PAD is used by the `io_placer` to place the pads along the boundary.

By default, the CLASS PAD macro is automatically placed along the boundary when the configuration file is read. You can also load a file with the `loadIoFile` command.

The normal wire bound I/O cells are CLASS PAD, however, to use the `assignBump` and `placePIO` commands, they must be CLASS PAD AREAIO even on the periphery.

## LEF MACRO CLASS PAD and PAD AREAIO

To support a peripheral I/O-driver with flip-chip bumps flow, PAD AREAIO cells are allowed outside the core box.

- LEF MACRO CLASS PAD has the bonding pad built into the cell.
- LEF MACRO PAD AREAIO has no bonding pad built-in, so it requires routing to the bump.

## Point-To-Point Routing

The Point-To-Point routing in flip chip enables routing between any two DEF SPECIALNET objects such as a bump and an I/O pad pin, a wire and a bump, or a bump and a stripe. The point-to-point router can point any location in the chip area.

Use the point-to-point router any time on special nets, especially after you run `fcroute` and you find an area where routing is not complete or an area which contains a problem route. In such cases, delete the problem route and reroute using the point-to-point router.

To perform point-to-point routing:

1. In the `Flip Chip – Point-To-Point` form, specify the minimum *width*.
2. In the tools area, click the *Point-To-Point Route*  widget.
3. Select 2 points in the design, an I/O pad pin and a bump (or a wire). View the routing that occurs between the 2 selected points.

To view the point-to-point routing, ensure that you are in the physical view in Encounter. If the point-to-point route is not complete, check the `encounter.log` file or check for any error message on the screen.

## **Encounter User Guide**

### Flip Chip Methodologies

---

The point-to-point router connects any two objects defined in the [Point-To-Point](#) form only. The router automatically selects the net name when you point the two objects — bump and I/O pad.

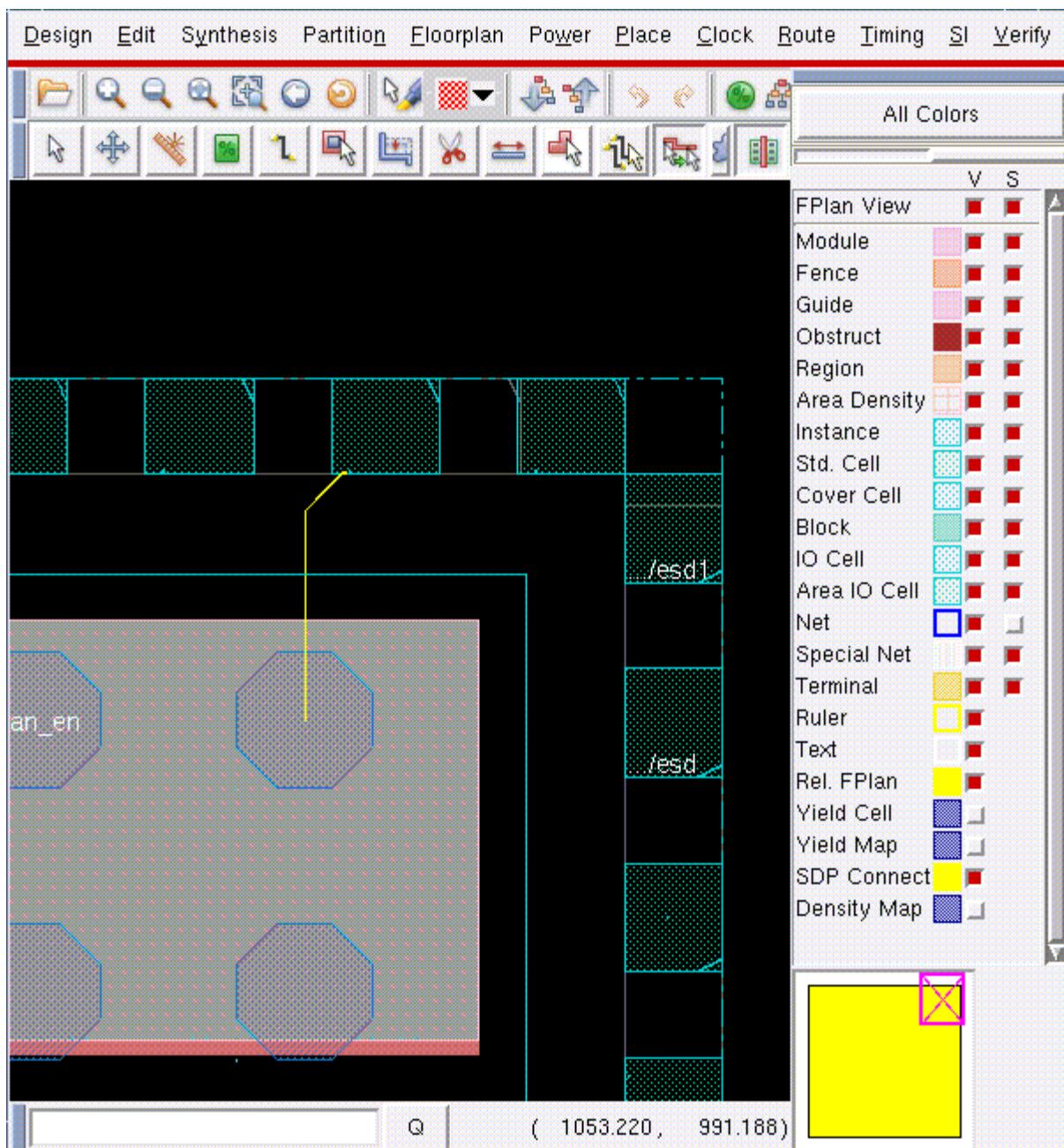
Alternatively, you can run the `routePointToPoint` command to perform point-to-point routing.

## Encounter User Guide

### Flip Chip Methodologies

The following example displays the results of the `routePointToPoint` command:

```
routePointToPoint -routeLayer M8:M8 -width 0.44 -spacing 0.46 -routeStyle  
doubleBend -pin {IOPADS_INST/Ptdspip01 PAD (944.4765 1013.278)} -pin  
{Bump_81_8_8 port_pad_data_in[1] (933.832 917.4755)}
```



## Swapping Signals

Signal swapping allows you to swap signals between bumps. Signals must be assigned to either one or both of the bumps to be swapped.

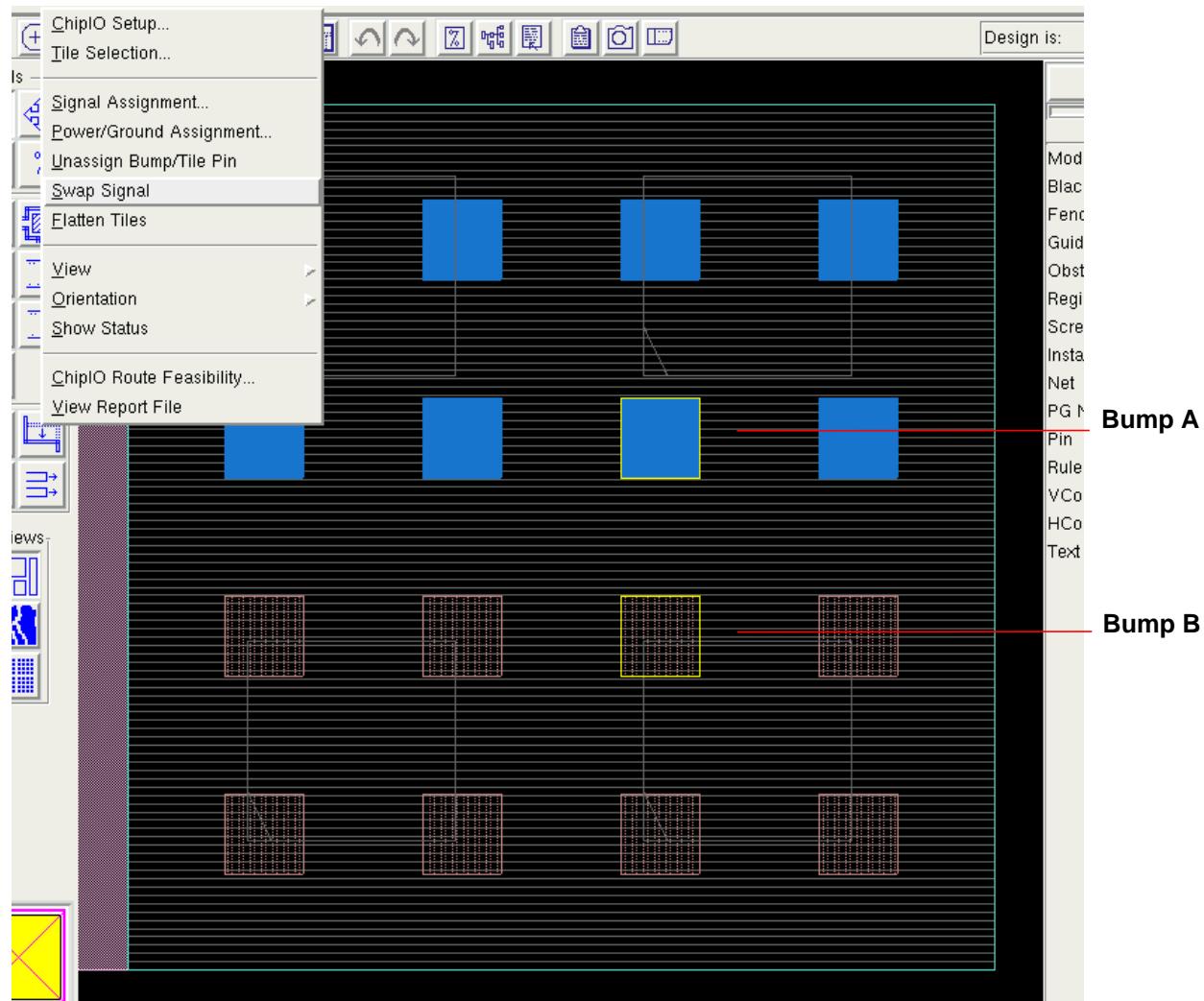
1. Click on the two bumps for the signals you want to swap.
2. Select *Place – FlipChip – Swap Signals*.

The figures below show signal swapping as follows:

- [Highlight the Bumps](#) on page 186
- [Signals Swapped](#) on page 187

## Highlight the Bumps

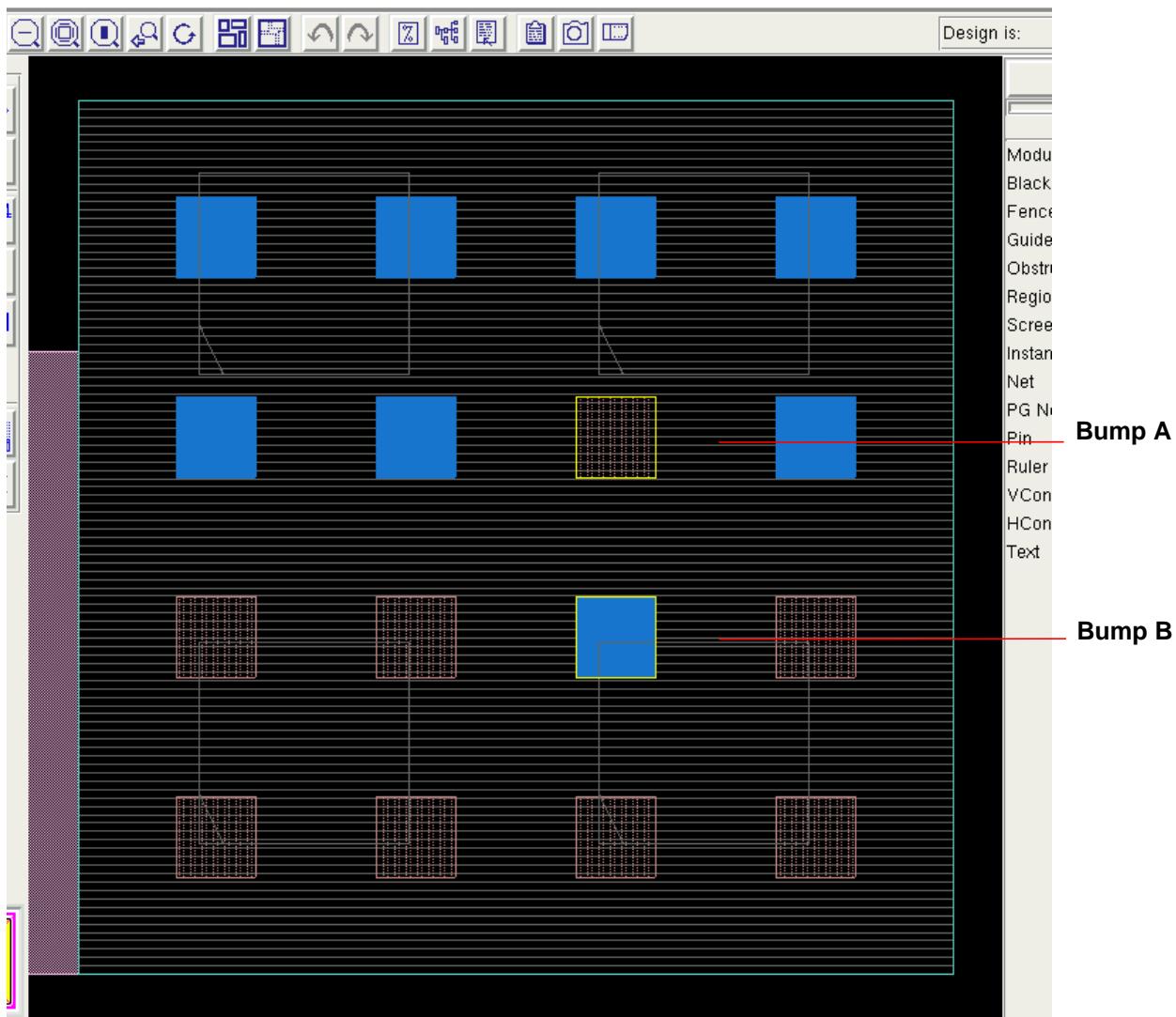
The following figure shows two highlighted bumps with signals to be swapped (bumps A and B).



**Note:** If you want to view the flight lines before you swap signals, you must first be in the Floorplan view. Then, use the left mouse button to click on the bump.

## Signals Swapped

The following figure shows the signals after swapping.



## Creating Differential Routing to Signal Bumps

Differential routing creates wires of the same length or configuration between a set of sources and targets. Use the [Route Flip Chip - Advanced -Routing Constraints](#) form to specify differential routing parameters.

You can create a constraint file to define differential pairs, shield nets, and nets to match tolerance. The following information provides the syntax and examples for creating a constraint file.

## Specify Routing Nets

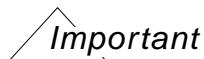
### Syntax

```
NETS
    WIDTHRANGE
    WIDTHSTEP
    SPACING
    PINSPACING dbUnitSpacing
    <nets>
END NETS
```

### Example

```
NETS
    WIDTHRANGE 5:10
    WIDTHSTEP 1
    SPACING 0.1
    PINSPACING 0.2
    out[10] out[11] out[14] out[19] out[8] out[9]
    out[15] out[16] out[17] out[18] resetn
    clk out[12] out[12] out[13] out[15] out[3] out[3]
END NETS
```

## Define Differential Pairs



DIFFPAIR is supported only in the froute AIO mode.

### Syntax

```
DIFFPAIR
    THRESHOLD
    <2 nets>
END DIFFPAIR
```

## Example

```
DIFFPAIR
  THRESHOLD 0.2
  port_pad_data_out[7]
  port_pad_data_out[8]
END DIFFPAIR
```

## Define Nets to Match Tolerance

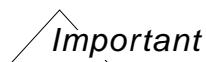
### Syntax

```
MATCH
  TOLERANCE
  <2 or more nets>
END MATCH
```

## Example

```
MATCH
  TOLERANCE 0.2
  tdigit[1] tdigit[2] tdigit[3]
END MATCH
```

## Define a Shield Net



Shielding is supported only in the froute AIO mode.

### Syntax 1

```
SHIELDING
  SHIELDBUMP true | false
  SHIELDWIDTH
  SHIELDGAP
  SHIELDLAYERS Above | Below | Common
  SHIELDNET
  <nets>
END SHIELDING
```

## Example 1

```
SHIELDING
  SHIELDBUMP true
  SHIELDWIDTH 0.4
  SHIELDGAP 0.1
  SHIELDLAYERS Above
  SHIELDNET VSS
  scan_out_2
  port_pad_data_out[15]
END SHIELDING
```

## Route Multiple Nets with Different Widths

The following shows a constraint syntax that allows one `fcroute` command to route multiple routes with different widths.

```
fcroute -constraintFile file_name
```

## Example Constraints File

```
NETS
  WIDTH 24.0
  ROUTELAYERS 7:7
  SPACING 0.1
  ## Net Definition ##
  VDDPST          #apply two nets only
  VSSPST
END NETS
NETS
  WIDTH 20.0
  ROUTELAYERS 8:7
  SPACING 0.1
  ## Net Definition ##
  ~VDDPST         #negation - for all other nets
  ~VSSPST         #negation
END NETS
```

## Route Nets with Tapering Pin Widths

### Syntax 1

```
NETS
  TAPERSTEP stepsizevalue
  TAPERWIDTH PINWIDTH | value
  <nets>
```

```
END NETS
```

## Example 1

```
NETS
  TAPERSTEP 1          # 1 enables tapering, 0 disables tapering
  TAPERWIDTH PINWIDTH | value      # If PINWIDTH is specified, the router
                                    fetches the pin width automatically as the
                                    starting routing width; If value is specified,
                                    the router starts routing with the width
                                    value.
  vssx_0              # Specifies the net name
  vddcx_1              # Specifies the net name
END NETS
```

## Examples and Report Files

### Routing and Placement Constraints

The following listing is the constraint file that is used for both `fcroute` and `placePIO`.

For individual constraint descriptions, see [Route Flip Chip - Advanced -Routing Constraints](#) page in the “[Route Menu](#)” chapter of the *Encounter Menu Reference*.

```
*****
Routing constraints: fcroute -designStyle aio | pio
*****
#One constraint file is used for fcroute and placePIO
VERSION 2
    WIDTH 10          ; global constraint
    SHIELDBUMP        ; global constraint
NETS
    out[10]
END NETS
DIFFPAIR
    WIDTH 20          ; can't accept, because of global constraint
    MAXLENGTH 1000     ; can't accept, because of global constraint
    SHIELDWIDTH 0.5    ; local constraint
    SHIELDLAYERS abc   ; local constraint
    out[111] out[114] SHIELDNET VDD
END DIFFPAIR
DIFFPAIR
    out[119] out[120]
END DIFFPAIR
SHIELDING
    WIDTH 20          ; can't accept, because of global constraint
    SHIELDWIDTH 0.5    ; local constraint
    SHIELDLAYERS abc   ; local constraint
    out[18] out[19] out[115] out[116] out[117] out[118] resetn
END SHIELDING
# DIFFPAIR and MATCH results maybe different in -designStyle aio | pio
DIFFPAIR
    out[10] out[11] SHIELDNET VDD
    out[14] out[19]
END DIFFPAIR

#SHIELDING only works with fcroute -designStyle aio
```

# Encounter User Guide

## Flip Chip Methodologies

---

```
SHIELDING
  SHIELDNET VDD (width spacing)
    in1 net2
    out1 out2
END SHIELDING
#NETS only work with fcroute -designStyle pio
NETS
  WIDTH 24.0
  ROUTELAYERS 8:7
  SPACING 0.1
  VDDPT          # apply two nets only
  VSSPT
END NETS
NETS
  WIDTH 20.0
  ROUTELAYERS 8:7
  SPACING 0.1
  ~VDDPT         # negation - for all other nets.
  ~VSSPT         # negation
END NETS
BUMPREGION
  AREA 3942.0 3545.0 3903.0 -3979.0 -3868.0 -3932.0 3774.0 -3521.0
  VDD*
  END AREA
END BUMPREGION
```

```
*****
```

```
Placement constraints: Only used with placePIO command
```

```
*****
```

```
FIXNETPAD
  net_name_list
END FIXNETPAD           ;(All the pads associated with given nets are fixed)
FIXPAD
  pad_name_list
END FIXPAD               ;(All the pads in the list are fixed)
FIXNETPADSIDE {EAST WEST SOUTH NORTH}
  net_name_list
END FIXNETPADSIDE
FIXPADSIDE {EAST WEST SOUTH NORTH}
  pad_name_list
END FIXPADSIDE
GROUPNET
  net_name_list
END GROUPNET
```

## Encounter User Guide

### Flip Chip Methodologies

---

```
GROUP
    pad_name_list
END GROUP
FIXBUMP
    net1 net2 net3
END FIXBUMP
*****
Resistance constraints for all (MAXRES) and/or individual nets (RESTABLE)
used by placePIO command
*****

NETS
    WIDTHRANGE
    ROUTELAYERS
    MAXRES <resistance(ohms)>
        NET_1
        NET_2
    END NETS

RESTABLE
#<Netname> <resistance(ohms)>
    NET_1 0.1
    NET_2 0.2
END RESTABLE
```

## IO\_FILE Example

The following sample is an IO\_FILE file showing bumps, I/O rows, and I/O instances. Format definitions follow the sample.

```
BumpCell: BUMPCELL Rect 1 Layer 6 0.000 0.000 80.000 80.000
Bump: bumpAry_16_3_3 BUMPCELL 697.440 696.800 DI[1]
Bump: bumpAry_15_2_3 BUMPCELL 497.440 696.800 DO[1]
Bump: bumpAry_14_1_3 BUMPCELL 297.440 696.800 DO[0]
Bump: bumpAry_13_0_3 BUMPCELL 97.440 696.800 SO
Bump: bumpAry_12_3_2 BUMPCELL 697.440 496.800
Bump: bumpAry_11_2_2 BUMPCELL 497.440 496.800
Bump: bumpAry_10_1_2 BUMPCELL 297.440 496.800
Bump: bumpAry_9_0_2 BUMPCELL 97.440 496.800
Bump: bumpAry_8_3_1 BUMPCELL 697.440 296.800 DI[0]
Bump: bumpAry_7_2_1 BUMPCELL 497.440 296.800
Bump: bumpAry_6_1_1 BUMPCELL 297.440 296.800 SI
Bump: bumpAry_5_0_1 BUMPCELL 97.440 296.800
Bump: bumpAry_4_3_0 BUMPCELL 697.440 96.800 CLK
Bump: bumpAry_3_2_0 BUMPCELL 497.440 96.800
```

## Encounter User Guide

### Flip Chip Methodologies

---

```
Bump: bumpAry_2_1_0 BUMPCELL 297.440 96.800 SM  
Bump: bumpAry_1_0_0 BUMPCELL 97.440 96.800
```

```
IORow: IOROW_1 520.100 596.400 IO1 R0 V 100.800 2  
IORow: IOROW_2 520.100 126.000 IO1 R0 V 100.800 2  
IORow: IOROW_3 119.700 596.400 IO1 R0 V 100.800 2  
IORow: IOROW_4 119.700 126.000 IO1 R0 V 100.800 2
```

```
IOInst: test_clk/clk/inbuf 520.100 126.000 R0 -fixed  
IOInst: test_clk/test/smbuf 119.700 126.000 R0 -fixed  
IOInst: test_clk/test/sibuf 119.700 226.800 R0 -fixed  
IOInst: test_clk/test/sobuf 119.700 697.200 R0 -fixed  
IOInst: ioall/io_A/inbuf_0/inbuf 520.100 226.800 R0 -fixed  
IOInst: ioall/io_A/inbuf_1/inbuf 520.100 596.400 R0 -fixed  
IOInst: ioall/io_B/outbuf_0/outbuf 119.700 596.400 R0 -fixed  
IOInst: ioall/io_B/outbuf_1/outbuf 520.100 697.200 R0 -fixed
```

### **Format Definitions**

#### ■ I/O Rows:

```
IORow: iorow_name x y site_name [orient] [ [H | V] step num]
```

iorow_name	Specifies the row name.
x y	Specifies the x and y coordinates, in microns, of the origin.
site_name	Specifies the site name. This must be defined in the LEF file.
orient	Specifies the row orientation.
H   V	Specifies either a <b>Horizontal</b> or a <b>Vertical</b> row.
step	Specifies the site width or height (depending on orientation), in microns, of the row.
num	Specifies the number of sites in the row (multiply by step for row length).

#### ■ I/O instances:

```
IOInst: inst_name [x y [orient] [-fixed]]
```

inst_name	Specifies the instance name.
x y	Specifies the x and y coordinates, in microns, of the origin.
orient	Specifies the instance orientation.
-fixed	Sets the placement status to fixed.

## **Encounter User Guide**

### Flip Chip Methodologies

---

For more information, see the `addAIORow` command in the “Flip Chip Commands” chapter of the *Encounter Text Command Reference*.

---

## Using ART in Hierarchical Designs

---

- [Overview](#) on page 198
- [Types of Active Logic Views](#) on page 198
- [Creating an Active Logic View](#) on page 200
- [Applications of ART](#) on page 200
  - [Timing Budgeting in Hierarchical Flow](#) on page 201
  - [Timing Optimization After Assembling the Post-Routed Partitioned Design](#) on page 201

## Overview

Active-logic Reduction Technology (ART) is a technique that is used to activate certain portion of a logic in a design and masking the other logic, while maintaining full physical design database in memory. In ART, an active logic view contains only the active portion of the logic.

ART can be applied to any timing-related command, such as timing budgeting or timing optimization to reduce run time and memory usage. In timing operations, an active logic view contains only the set of timing paths exposed to the specific operation. When applied to timing optimization, active logic views enable cross-hierarchical optimization while preserving the full hierarchical view of the design after optimization is complete.

## Types of Active Logic Views

The tool creates an active logic view based on the partition boundaries, set of critical timing paths, block module boundaries, and physical area. There are two types of active logic views:

- Flat Top
- Critical

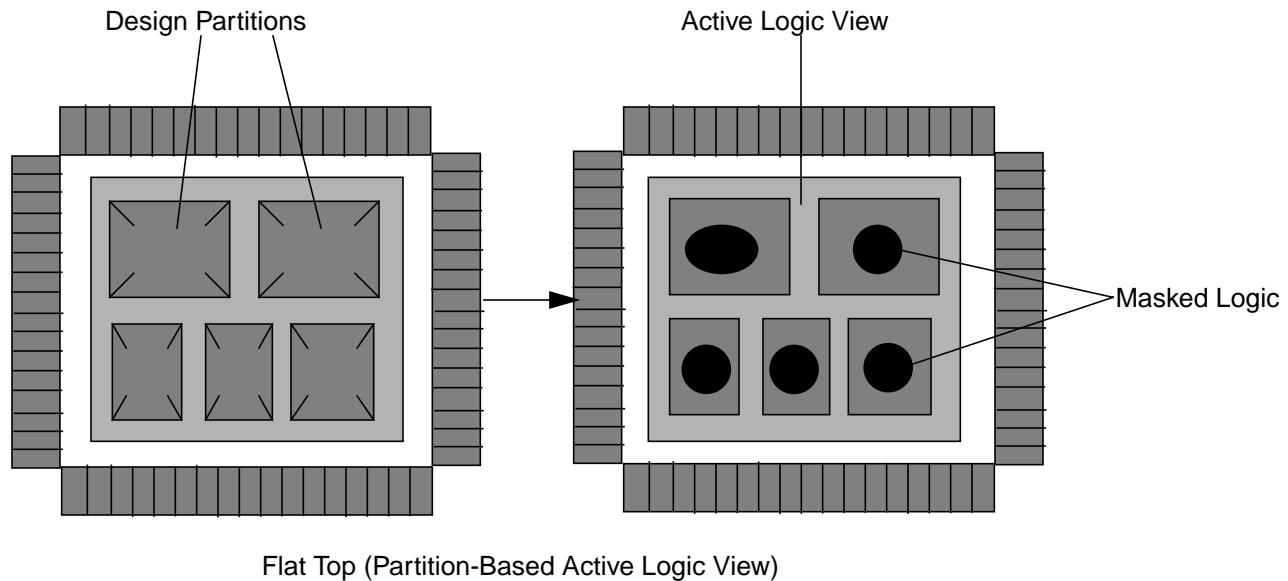
### Flat Top

A flat top is a partition-based active logic view that activates the top-level paths and the interface path of partition blocks. The logic inside the partition blocks is excluded from the timing database.

## Encounter User Guide

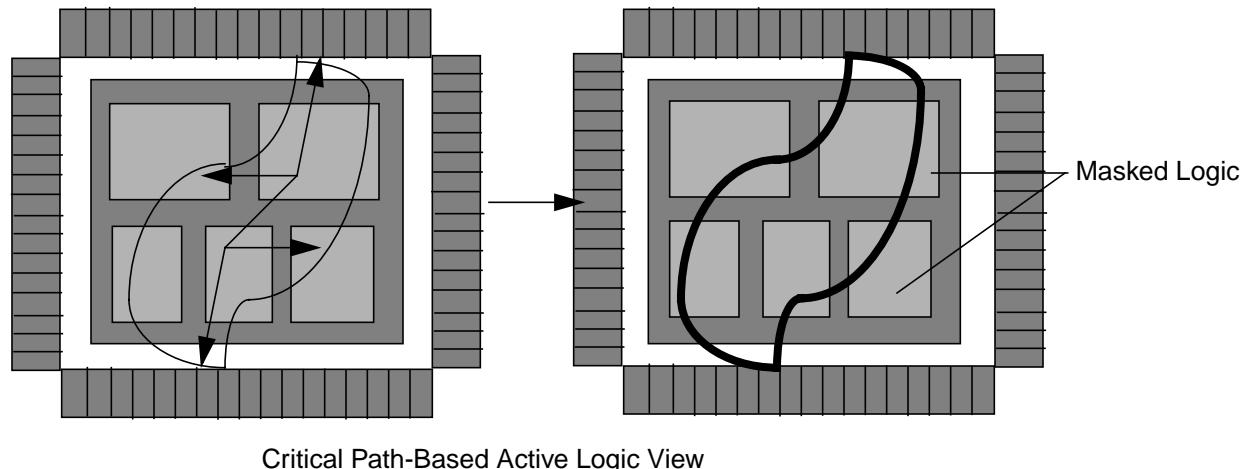
### Using ART in Hierarchical Designs

The following figure shows the flat top active logic view:



## Critical

The critical active logic view activates all paths in a design that have a negative slack. All other logic in the design is masked.



## Creating an Active Logic View

To create an active logic view, load the entire chip as a design in the Encounter software, specify the partition, and then run the `createActiveLogicView` command with an appropriate option.



An active logic view cannot be saved as a database or a file. Run the `createActiveLogicView` command to create an active logic view.

**Note:** The Encounter software considers MMMC settings while creating an active logic view.

### Example of Active Logic View Creation

The following method shows time budgeting using active logic view in a hierarchical design:

To create an active logic view:

1. Mark the top-level timing graph to mask all logic inside the interface logic of each partition.

```
createActiveLogicView -type flatTop
```

2. Derive timing budget.

```
deriveTimingBudget
```

3. Clear ART marking.

```
clearActiveLogicView
```

**Note:** Timing database will be rebuilt when the next timing command is called.

## Applications of ART

ART helps reduce run time and memory snapshots for big designs. It brings the active portion of a design to a size that is manageable for flat analysis.

This section provides information about the use of ART in timing budgeting and timing optimization during the post-route stage.

## Timing Budgeting in Hierarchical Flow

In a hierarchical flow, the `deriveTimingBudget` command creates timing constraints for the partition blocks based on timing budgeting with full-chip timing conditions. During timing budgeting, it is not essential to analyze the paths that are enclosed in partitions. In such a situation, the `flatTop` type of an active logic view provides an exact condition that activates the top-level logic and interface logic of the partition blocks. It takes less memory and run time.

For more information, see [Top-Level Budgets Derived by Using Active Logic View](#) section of the Timing Budgeting chapter.

## Timing Optimization After Assembling the Post-Routed Partitioned Design

After assembling the design, you might see timing issues at the top-level logic, the interface paths of the partition blocks, or the internal paths of partition blocks as you have a full-chip view. In the traditional hierarchical flow, you might need to go back to the partition level to solve the timing issues that might be time consuming.

The ART-based post-route optimization flow helps reducing the overhead and works effectively for the timing issues because it contains the full-chip view for solving the timing issues.

### ART-based Post-Route Optimization

When you perform ART-based post-route optimization, the top-level timing paths and interface paths of partition blocks are activated by ART as active logic views and then they are optimized. The other internal paths of the partition blocks are masked and not optimized. This technique saves memory usage and run time for large designs.

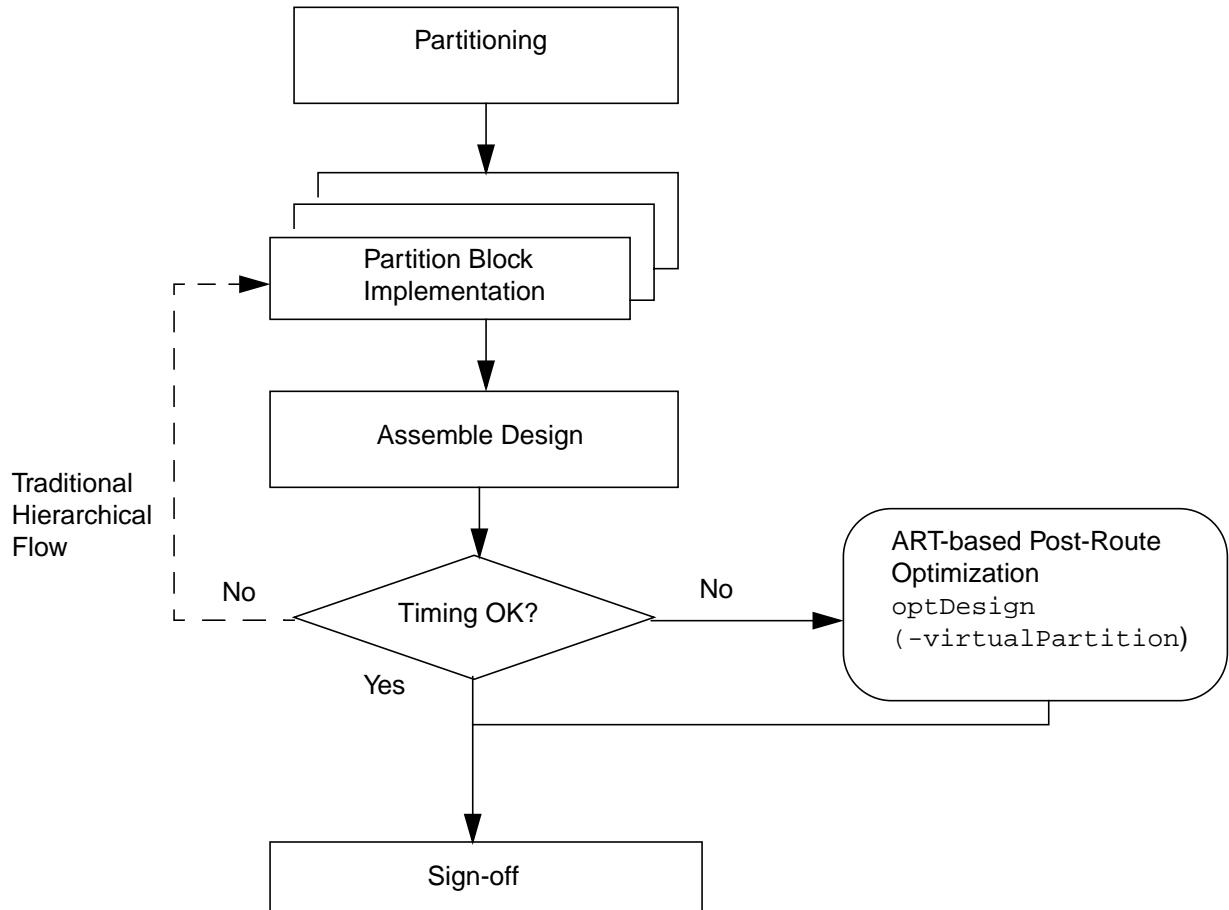


ART-based post-route optimization is partition aware. Therefore, after completing ART-based post-route optimization, you can still partition your design and sign it off at the block level or convert these blocks to IPs.

## Encounter User Guide

### Using ART in Hierarchical Designs

The following figure shows timing optimization using active logic views in the Encounter hierarchical flow:



To use ART-based flow, run the following commands:

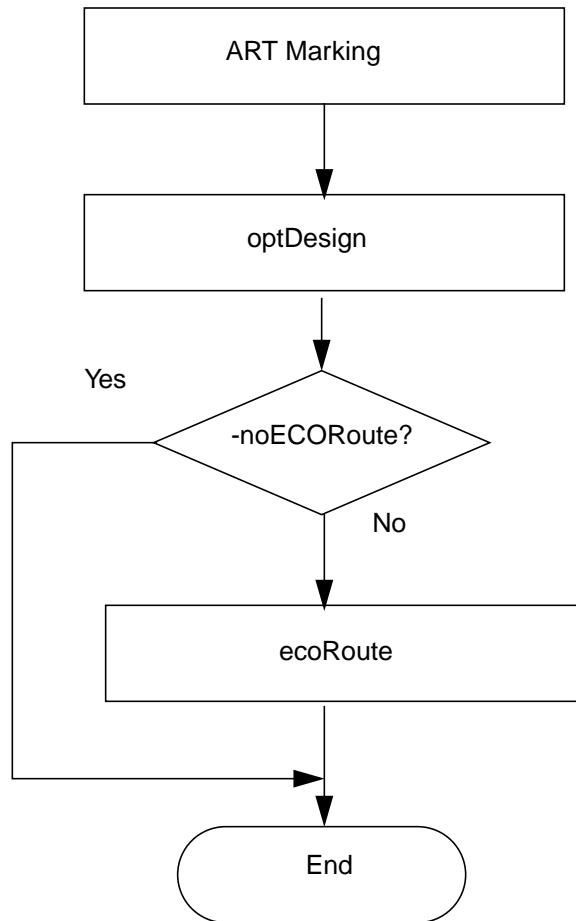
```
setOptMode -virtualPartition true  
optDesign -postRoute [-hold][-noECORoute]  
setOptMode -virtualPartition false
```

The `setOptMode -virtualPartition` command runs the ART-based post-route optimization. The `optDesign` command internally applies active logic view for the design.

## Encounter User Guide

### Using ART in Hierarchical Designs

The following figure depicts the `setOptMode -virtualPartition` command flow details:



### Steps to Run ART-Based Post-Route Optimization

#### 1. Assemble the partition blocks and the top partition.

```
assembleDesign -fe -topDir partition1.enc.dat \
               -blockDir Block1.enc.dat \
               ...
               -blockDir Block10.enc.dat \
               -saveEcoRef -ecoRefDir partitions_eco
```

The `assembleDesign` command creates full-chip data from the partition blocks and the top-level partition data to apply ART-based post-route optimization. Using the following options, `assembleDesign` creates an ECO reference file for hierarchical ECO routing after `optDesign`:

- ❑ The `-saveEcoRef` parameter creates a directory for every partition block that is specified in the `assembleDesign` command. It also creates the ECO reference files for these partition blocks. For more information about the `-saveEcoRef`

## Encounter User Guide

### Using ART in Hierarchical Designs

---

parameter, see the `assembleDesign` command in “Partition Commands” chapter of the *Encounter Text Command Reference*.

- ❑ The `-ecoRefDir directoryName` parameter is used to specify the directory where the files generated by `-saveEcoRef` parameter are saved. If you do not specify this parameter, the files generated by `-saveEcoRef` are saved in the current working directory.
- ❑ The ECO reference files are used during hierarchical ECO routing after `optDesign`. The files are placed in the directory which is created by the `-saveEcoRef` parameter or the `-ecoRefDir` parameter. In the process of hierarchical ECO routing, an ECO reference file is used to identify ECO nets in the design to make ECO routing more efficient.

The ECO reference file gets updated after routing.



***The ECO reference file and the directory has been designed for an efficient hierarchical ECO routing. To avoid any unexpected issues within the flow, do not modify the ECO reference file.***



During `assembleDesign`, if a DEF file and a Verilog netlist are used, there can be a run-time penalty at `ecoRoute` because the routing results have been imported from the DEF file.

Contact your Cadence representative if you require help.



***The ART-based post-route optimization resolves timing issues for top-level paths and interface paths of partition blocks. Therefore, ensure that the internal paths of partition blocks have no timing issues and are DRC checked for routing before optimization.***

## 2. Perform optimization using ART and execute `optDesign` for the assembled design.

```
setOptMode -virtualPartition true  
optDesign -postRoute  
setOptMode -virtualPartition false
```

or

```
setOptMode -virtualPartition true  
optDesign -postRoute -noECORoute  
setOptMode -virtualPartition false
```

## Encounter User Guide

### Using ART in Hierarchical Designs

---

You can also use this flow for hold violation. By default, the hierarchical ecoRoute is called to ensure that design can be partitioned and it uses the ECO reference file which was generated earlier using the assembleDesign command.

The hierarchical ecoRoute might reset some settings that have been set using the command file, especially the global Encounter variables. Cadence recommends that you reinitialize the global variables after completing ART-based post-route optimization.



Disable the ART controls after ART-based portion of the flow is completed to ensure that there are no side effects on other parts of the flow.

Run the following command after ART-enabled portion of the flow is completed:

```
setOptMode -virtualPartition false
```



***SI fixing is not supported in this flow.***

### 3. Run ECO routing.

This is an optional step if you have used -noECORoute earlier. To execute the hierarchical ecoRoute, run the following command:

```
ecoRoute -handlePartition
```

The ecoRoute -handlePartition parameter enables hierarchical ECO routing that maintains the partition structure of the design and performs routing with reasonable run time for a large design.

The hierarchical ecoRoute performs routing by maintaining the partition-pin location and number of pins of the partition for ECO nets which have been changed by the IPO operation using optDesign in the flow. An ECO reference file is used in the process.

#### **Note:**

- ❑ Use ecoRoute -handlePartition together with assembleDesign - saveEcoRef as a part of the flow. This is similar to the ART-based post-route optimization which is done for a hierarchical design. To use this command apart from this flow, contact your Cadence representative.
- ❑ You can apply multiple-CPU processing for ecoRoute -handlePartition. For help regarding its usage, contact your Cadence representative.

## Encounter User Guide

### Using ART in Hierarchical Designs

---



***You can still use the ecoRoute command without the -handlePartition parameter. However, you will be unable to partition your design if you do not use this parameter.***

---

## Using Interface Logic Models in Hierarchical Designs

---

- [Overview](#) on page 208
- [Creating ILMs](#) on page 209
  - [Example ILM Creation](#) on page 210
  - [Preserving Selected Instances in ILMs](#) on page 211
  - [Creating ILMs for Shared Modules](#) on page 211
- [Specifying ILM Directories at the Top Level](#) on page 213
  - [Example Top-Level Implementation Flow with ILMs](#) on page 213
- [ILMs Supported in MMMC Analysis](#) on page 215
- [ILMs Supported in SI](#) on page 217
- [Interactive Use of ILMs](#) on page 217
- [ILM Limitations](#) on page 218

## Overview

Models are compact and accurate representations of timing characteristics of a block. An Interface Logic Model (ILM) is a structural representation of a block, specifically a subset of the block's structure including instances along the I/O timing paths, clock-tree instances, and instances or net coupling affecting the signal integrity (SI) on I/O timing paths.

Instead of using a blackbox at the top level, you create an ILM at the block level and use it as you would use a blackbox.

The advantages of using ILMs are as follows:

- More accurate analysis than a black box flow
  - More SI aware than combined .lib or .cdb approach
  - Can model clock generator inside block
  - More accurate timing and SI reduces the number of design iterations to close timing and SI.
- No need to characterize blocks
  - Works on actual design data
- Can be used in the initial prototyping stage for very big designs. when loading full design data is not feasible.
  - Allows you to modify only top-level data
  - Fully preserves implemented partitions
- Uses the original constraint file for top-level analysis
  - No abstraction for timing exceptions

## Creating ILMs

In the hierarchical design flow, you create a detailed block-level implementation of a block, then specify the `createInterfaceLogic` command to create an ILM for the block. This command creates the specified directory containing ILM files.

You can also create ILMs for blocks that are in an intermediate stage of design, then use the data at the top level of the design for preliminary timing optimization.



An ILM created for an incomplete block is not as accurate as an ILM created for a complete block. Always use ILMs for complete blocks to complete the top-level design.

The software generates ILM data for CTS, signal integrity, and other design stages (pre-CTS, post-CTS, post-route)

- ILM data for pre-CTS, post-CTS, and post-route

The model contains the netlist of the circuitry leading from the I/O ports to interface sequential instances (that is, registers or latches), and from interface sequential instances to I/O ports. The clock tree leading to the interface registers is preserved.

ILMs do not contain information about the following:

- Internal register-to-register paths, if internal logic is not part of the interface path
- Internal paths (if `-noInterClockPath` is used): Internal paths controlled by different clock, or clocks connected to the ILM module through different ports.

If the logic between the I/O ports is pure combinational, it is preserved in an ILM.

- ILM data for SI

The model includes all of the above, plus aggressor drivers or nets which affect I/O paths. It also includes the timing window files in the ILM model directory.

- ILM data for CTS

The model includes all clocked instances (clock sinks), and clock tree instances and nets.

Use `createInterfaceLogic -writeSDC` to generate block level constraints which can be used:

## Encounter User Guide

### Using Interface Logic Models in Hierarchical Designs

---

- During a bottom-up design flow to manually build a top-level constraint file from the block constraints. The generated block-level .sdc file contains references to the block instances or pins or nets which made it into the ILM model netlist.
- To validate a model at the block level. For example, an ILM netlist and the .sdc file can be read in a separate Encounter session and timing analysis can be run on all paths. Then, the results can be compared against timing for the same path during full-block implementation.

**Note:** When `createInterfaceLogic` is called, all views are generated for multi-corner, multi-mode (MMMC) analysis.

### Example ILM Creation

The following method creates a model that can be used in the top-level implementation flow by both `timeDesign` and `optDesign` for both setup and hold efforts, including post-route SI optimization. This model is also used during `clockDesign`.

```
createInterfaceLogic -hold -dir block_A.ILM
```

### Sample Summary Report

The following is a sample summary report generated at the end of the `createInterfaceLogic` command:

---

createInterfaceLogicSummary		
Model	Reduced Instances	Reduced Registers
ilm_data	7153/7621 (93%)	174/285 (61%)
cts_data	7254/7621 (95%)	0/285 (0%)
si_ilm_data	6793/7621 (89%)	160/285 (56%)

---

In this report, the reduction ratio in the `ilm_data` model is 93 percent which means that 7153 out the total 7621 instances for this block have been eliminated. Only 468 instances are written to the Verilog netlist for the `ilm_data` model out of which 111 instances are registers.

This summary report applies to a block using MMMC. Therefore, views with worst reduction ratio are displayed for each model.

**Note:** You can run the following commands for improving the reduction ratio:

- ❑ `setILmMode -highFanoutPort false`
- ❑ `createInterfaceLogic -noInterClockPath`

## Preserving Selected Instances in ILMs

You can force the selected instances and nets to be included in the ILM model by using the `createInterfaceLogic -keepSelected` parameter.

1. Select instances or nets using the `selectInst` or `selectNet` commands.
2. Specify `createInterfaceLogic -keepSelected`.

## Creating ILMs for Shared Modules

You can use the same sub-block module in different ILM blocks, enabling reuse of versatile modules. The `createInterfaceLogic` command considers constant propagate, so that only the enabled parts of a module are considered when creating ILMs for the reused modules. Because the Encounter database cannot handle the same module name in different circuits, the software automatically modifies the module names with the following rule:

`topModuleName+timestamp+$+moduleName`

As an example, one ILM block (`ModuleA`) uses an ALU module (`ALU`) as an unsigned ALU, and a second block (`ModuleB`) uses the ALU as a signed ALU. You can change the input signal to use the ALU differently, setting one ALU as sign enabled and the other to off. When you run the `createInterfaceLogic` command, the software considers only the enabled parts of the ALU when creating ILMs for `ModuleA` and `ModuleB`. The software also ensures that the name of the ALU module in `ModuleA` and the name of the ALU module in `ModuleB` are different.

## Creating ILMs Without Using Encounter Database

If you do not have Encounter database for an implemented block but have a Verilog netlist, constraints, and SPEF for that block, then use the `createILMDATADir` command to store data in the ILM format.

Following is the usage of the `createILMDATADir` command:

```
createILMDATADir -cts -si -dir block_A.ILM -cell block_A -mmmc -verilog myfile.v  
createILMDATADir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \  
-spef max.spef.gz -rcCorner rcMax  
createILMDATADir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \  
-spef typ.spef.gz -rcCorner rcTyp  
createILMDATADir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \  
-spef min.spef.gz -rcCorner rcMin  
createILMDATADir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \  
-sdc funMaxMax.sdc -viewName funct-devSlow-rcMax
```

## **Encounter User Guide**

### Using Interface Logic Models in Hierarchical Designs

---

```
createILMDataDir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc funMaxTyp.sdc -viewName funct-devSlow-rcTyp
```

```
createILMDataDir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc tstMaxMax.sdc -viewName test-devSlow-rcMax
```

```
createILMDataDir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc funMinMin.sdc -viewName funct-devFast-rcMin
```

```
createILMDataDir -cts -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc tstMinMin.sdc -viewName test-devFast-rcMin
```

## Specifying ILM Directories at the Top Level

Use `specifyILm` to use the ILM data for a block at the top partition level rather than using the default `.lib` model. You can run `specifyILm` multiple times in the same session. Each time you run this command, the software overwrites the previous setting for the block. If master/clones exist in the design, the cell name will have the name of the master partition.

**Note:** You can use this command (and `unSpecifyILm`) only if the ILMs are unflattened (`unflattenILm`). You cannot change ILM settings in flattened or ILM view.

Use `unSpecifyILm` to revert to using the `.lib` model for the block.

- The following form enables you to specify and unspecify ILM directories:
  - *Design Import – Advanced – Specify ILM*

### Example Top-Level Implementation Flow with ILMs

1. Before you start the Encounter tool, prepare the top-level Verilog file, if needed.

If you use the Encounter hierarchical flow in a previous Encounter session, then the `savePartition` command automatically creates the top-level data. Else, you need the following in the top-level directory:

- A Verilog netlist that includes dummy modules for the blocks (ILM or Liberty) in the design.
- A view definition file since ILMs are supported only in the MMMC mode. If you have a non-MMMC design, create or load a view definition file that contains the following:  
`set_analysis_views -setup {model_slowCorner} -hold {model_fastCorner}`

2. Start an Encounter session from the top-level module directory within the directory where the partitions are saved.
3. Load the config file, including the top-level netlist, ILM directory name, `ilm_blocks.lib` (optional if using ILM), `stdcells.lib`, and `.lef` for the blocks, top-level constraints, and chip-level constraints.

```
loadConfig fileName
specifyILm -cell block_A -dir ../block_A/block_A.ILM
specifyILm -cell block_B -dir ../block_B/block_B.ILM
```

As an alternative, you can use the GUI to specify the ILM directories.

*Design Import – Advanced – Specify ILM*

Specify the directory for each module, and the timing constraints file.

## Encounter User Guide

### Using Interface Logic Models in Hierarchical Designs

---

**4.** Load the floorplan.

```
loadFPlan top_floorplan
```

**5.** Place the design.

```
placeDesign
```

**6.** Run pre-CTS timing optimization.

```
optDesign -preCTS
```

**7.** Build the clock tree.

```
clockDesign
```

**8.** Run post-CTS timing optimization.

```
optDesign -postCTS
```

or

```
optDesign -postCTS -hold      ;#optional
```

**9.** Route the design.

```
routeDesign
```

**10.** Run post-route optimization for setup.

```
optDesign -postRoute
```

**11.** Run post-route optimization for setup and hold.

```
optDesign -postRoute -hold
```

**12.** Run post-route optimization for SI.

```
optDesign -postRoute -si
```

If you want to create an ILM of the resulting block for use in the next level up in the hierarchy, run the following steps with the above-mentioned flow:

**1.** Flatten the design as creating ILM calls timing analysis.

```
setILmType -model si  
flattenILm
```

**2.** Perform timing analysis.

```
timeDesign -postRoute -si
```

**3.** Create ILM.

```
createInterfaceLogic -dir block_parent
```

## ILMs Supported in MMMC Analysis

Cadence strongly recommends that you use ILMs in the MMMC mode. If you have a non-MMMC design, create and load a view definition file that contains the following:

```
set_analysis_views -setup {model_slowCorner} -hold {model_fastCorner}
```

The MMMC analysis for designs including ILMs is identical to MMMC analysis for black box designs except for the following considerations:

1. Views, modes, and corners at the top and partition levels must have same names.
2. When you use `create_constraint_mode` to specify constraints for MMMC, you must specify the ILM constraints using the `-ilm_sdc_files` parameter (that is, timing in the presence of ILMs get constraints from the `-ilm_sdc_files` parameter, not the `-sdc_files` parameter). The `.sdc` files specified with the `-ilm_sdc_files` parameter are allowed to reference nets or pins internal to the ILM model.
3. The interactive constraint commands are currently not supported when using ILMs. Use the `update_constraint_mode -ilm_sdc_files` to change the current constraints files. When using ILMs, the `-ilm_sdc_files` is used. It allows references to nets or pins internal to the ILM model.

**Note:** In the current ILM flow, the SDC constraints (originally specified against the complete flat netlist for the design) that reference parts of the design that were pruned cause warnings and errors during constraint loading. In this release, you can set the temporary `timing_suppress_ilm_constraint_mismatches` global variable to `true` to suppress all error and warning messages related to the unfound objects. Note that this command might also suppress error messages that might be of use (that is, where the top-level pins or nets or instances cannot be found).

Currently, constraints are used during timing in the flattened mode. So, the internal ILM instances are seen instead of the LEF pins of the ILMs. Therefore, reading the bounding box constraints causes errors without using the `timing_suppress_ilm_constraint_mismatches` variable.

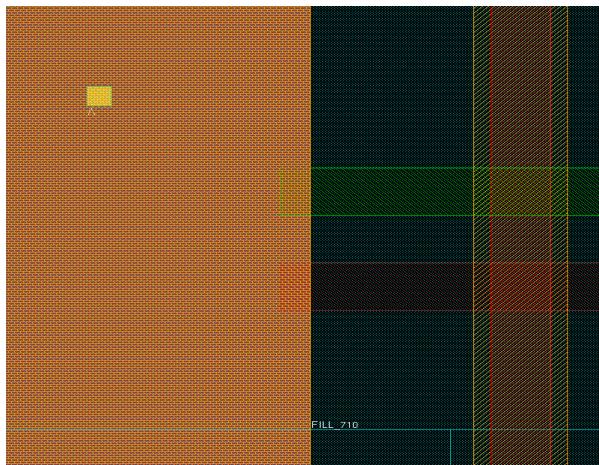
If you want to see the LEF pins of the ILM in GUI, the design must be in the unflattened mode.

## Encounter User Guide

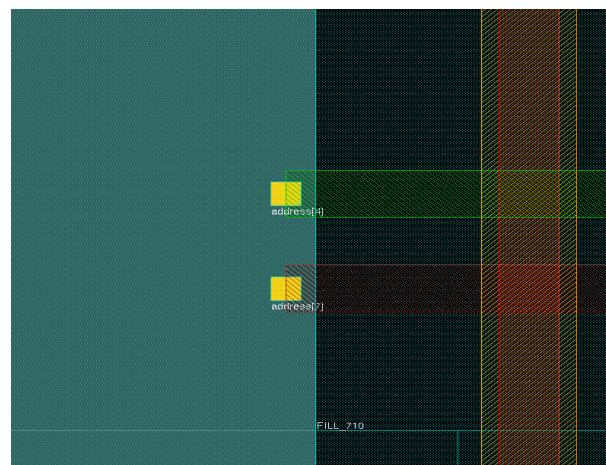
### Using Interface Logic Models in Hierarchical Designs

---

The following figure shows the flattened and unflattened ILM. The LEF pins of the ILM are visible after unflattening the ILM.



Flattened ILM



Unflattened ILM

## ILMs Supported in SI

ILM supports the `-si` parameter for `optDesign` and `timeDesign`. These commands automatically run `setILmType -model si` before calling `flattenILm` such that the SI ILM model is used. Therefore, your present post-route optimization scripts should run successfully in the presence ILMs (without any additional changes).

The following command can be used to get timing reports containing the SI push-out delays on nets using the `-setILmType -model` command:

```
setILmType -model si  
  
# Flattens to the timing model  
  
flattenILm  
  
# Reflattens to SI model, then does not unflatten (All other design  
# commands unflatten upon exit, regardless of the flattened/unflattened  
# state before invocation)  
  
timeDesign -postroute -si  
  
# Adds incremental delay column (for SI push-out delays) in timing output:  
set_global_report_timing_format {instance arc cell fanout load slew delay  
incr_delay arrival}  
  
# Minimizes the width of the report such that it easily fits into the screen  
# without wrapping  
  
set_table_style -name report_timing -no_frame -indent 0  
  
report_timing
```

**Note:** You can also invoke the Global Timing Debugger (Timing – Debug Timing – Generate)

## Interactive Use of ILMs

- Commands such as `optDesign`, `timeDesign`, `clockDesign`, and so on automatically take care of flattening and upon completion, leaves the design in an unflattened state.
- Timing commands require you to run `flattenILm` first so that the nets and instances internal to ILM are exposed to the timing engine.

```
encounter> flattenILm  
ilmView> report_timing
```

Notice that the prompt changes to `ilmView` after `flattenILm`.

- The Global Timing Debugger (GTD) also requires the design to be in a flattened state. GTD displays rows with instances or nets which are internal to the ILM as grayed out.

## Encounter User Guide

### Using Interface Logic Models in Hierarchical Designs

---

- The non-timing commands require the design to be in an unflattened state before invocation:

```
ilmView> unflattenIIm  
encounter>verifyGeometry
```

## ILM Limitations

When ILMs are present in a design, ensure that you set the following variable in the `~/enc.tcl` file in your home directory or `./enc.tcl` file in the run directory before loading the design:

```
set socelImEnableCommandControl 2
```

In the present release, all previously entered interactive constraints are lost while running `unflattenIIm`, which is automatically called at the end of running design commands, such as `placeDesign`, `optDesign`, `clockDesign`, `routeDesign`, `saveDesign` and so on. Therefore, it does not save the interactive constraints.

If the design is in the flattened mode, `timeDesign` does not run `unflattenIIm` while exiting and leaves the interactive constraints intact. Therefore, you can use `timeDesign` (and `report_timing`, and Timing Debug GUI) to debug interactive constraints in the presence of ILMs.

If you want the design commands (including `saveDesign`) to honor new constraints when ILMs are present in the design, perform either of the following:

- Edit one of the existing `-ilm_sdc_files` (as defined in the `create_constraint_mode` command in the `viewdefinition.tcl` file) and then run the following in the MMMC mode:

```
flattenIIm  
set_interactive_constraint_modes {yourListOfViews}  
set_analysis_view -setup "[all_setup_analysis_views]"  
-hold "[all_hold_analysis_views]"
```

In non-MMMC mode, edit the existing `.sdc` file and then run the following commands:

```
flattenIIm  
unloadTimingCon  
loadTimingCon -ilm previous.sdc
```

The above commands force reading of this changed constraint file (this is the `set_analysis_view` command found in this design's `viewdefinition.tcl` file) again.

## Encounter User Guide

### Using Interface Logic Models in Hierarchical Designs

---

- Create a file with additional constraints and then run the following commands in the MMMC mode:

```
flattenIlm  
  
set_interactive_constraint_modes {yourListOfViews}  
  
set previousSDCs [get_constraint_mode constraintName -ilm_sdc_files]  
  
update_constraint_mode -name constraintName -ilm_sdc_files  
[concat $previousSDCs additional.sdc]  
  
set_analysis_view -setup "[all_setup_analysis_views]"  
-hold "[all_hold_analysis_views]"
```

In the non-MMMC mode, run the following commands:

```
flattenIlm  
  
loadTimingCon -ilm additional.sdc -incr
```

The above commands automatically read in these additional constraints (as well as the previous constraints).

**Encounter User Guide**  
Using Interface Logic Models in Hierarchical Designs

---

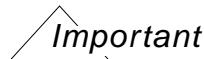
---

## What-If Timing Analysis

---

### Performing What-If Timing Analysis

You use blackboxes or blackblobs in large designs containing hierarchical flows when gate-level details are not available at the beginning of the design cycle. You can easily modify the timing model of a blackbox or blackblob at the top level because it is not a hard macro. Using the Encounter software, you can make quick modifications to the timing model of a blackbox or blackblob, and run timing analysis to check the impact of the modifications. This feature is known as what-if timing budgeting. The Encounter software provides what-if timing commands to support what-if timing budgeting. For more information on what-if timing commands, see the chapter ["What-if Timing Commands."](#) in the *Encounter Text Command Reference*.

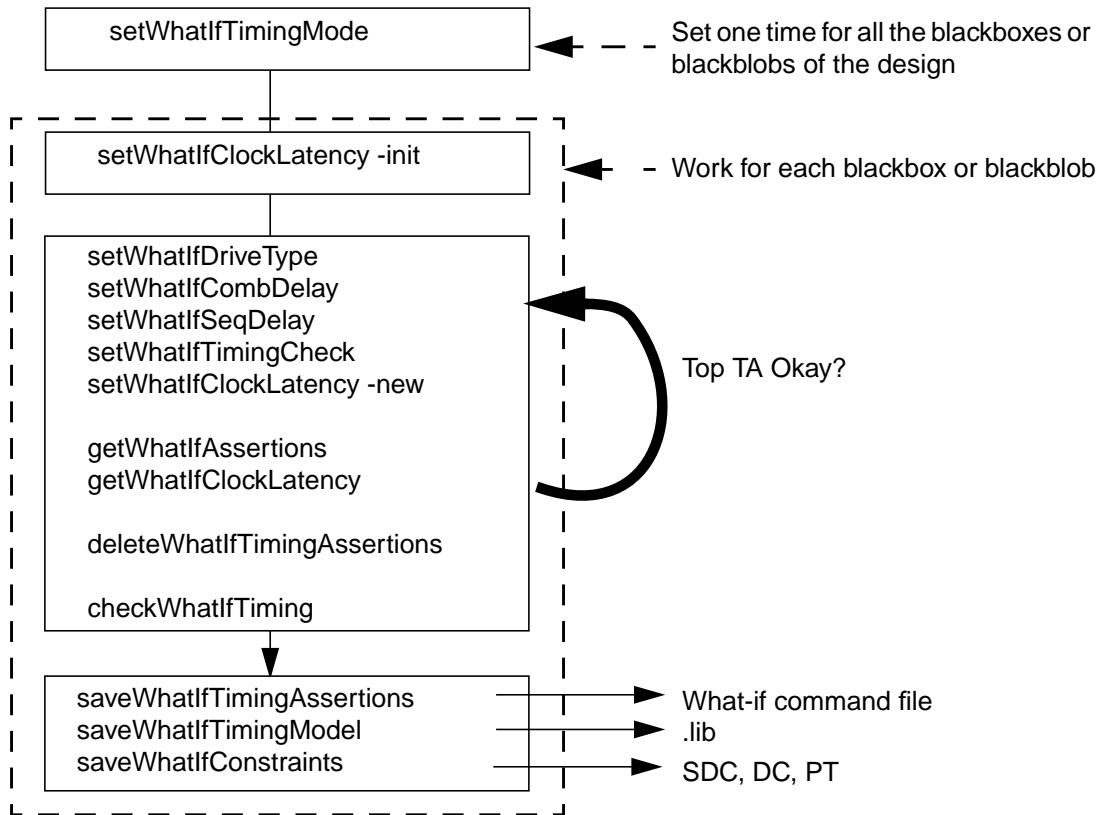


The what-if timing analysis commands do not support the Multi-Mode Multi-Corner (MMMC) feature.

## Encounter User Guide

### What-If Timing Analysis

The following diagram shows the what-if budgeting flow.



## Prerequisite

Prior to using what-if timing commands, you must load the what-if timing models into the database because the what-if timing commands simulate the modifications of the timing arcs.

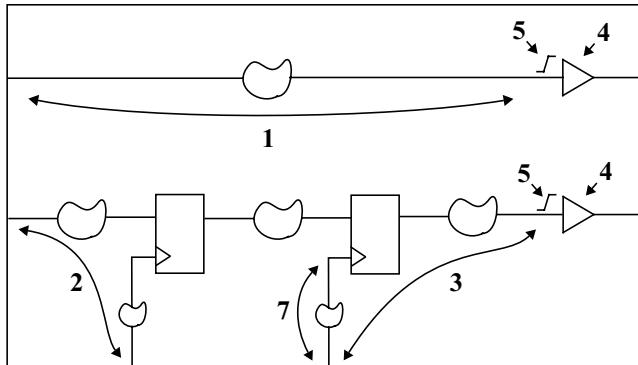
If you do not have timing models in the early design phase, you can use the `setWhatIfClockPort` command to create clock ports. You can then use the clock port to create timing arcs.

## Timing Models Supported for What-If Timing Analysis

The Encounter software supports two timing models for what-if timing analysis: intrinsic and normalized. You can select only one mode at a time.

Figure 9-1 shows the intrinsic timing model.

**Figure 9-1 Intrinsic Timing Model**



The data types associated with the numbers in the Figure 9-1 and the corresponding commands that you use to specify that data are as follows:

#	Data Type	Command
1	Combinational delay from an input port to the input of the driver	<a href="#">setWhatIfCombDelay</a>
2	Delay from the clock input port to the data input port	<a href="#">setWhatIfTimingCheck</a>
3	Sequential delay from the clock input port to the input of the driver	<a href="#">setWhatIfSeqDelay</a>
4	Type of Driver	<a href="#">setWhatIfDriveType</a>
5	Driver input slew	
7	Clock insertion delay to internal registers	<a href="#">setWhatIfClockLatency</a>

An intrinsic timing model uses the following formula for timing arcs ending on output ports:

Delay = constant delay + driver delay (look-up table)

If you do not use slew specifications in an intrinsic timing model, the timing arc is a 2-D timing table containing input slew and output capacitance dependencies. With slew specifications, the timing arc is only load dependent.

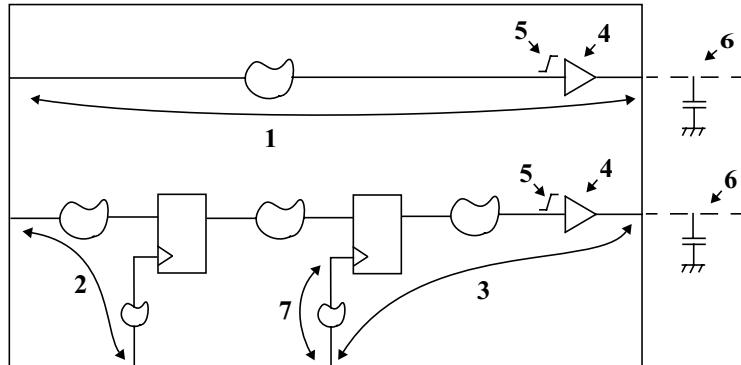
Figure 9-2 shows the normalized timing model.

## Encounter User Guide

### What-If Timing Analysis

---

**Figure 9-2 Normalized Timing Model**



The data types associated with the numbers in Figure 9-2, and the corresponding commands that you use to specify that data is as follows:

#	Data Type	Command
1	Combinational delay from an input port to the output port. It includes the driver delay	<a href="#">setWhatIfCombDelay</a>
2	Delay from the clock input port to the data input port	<a href="#">setWhatIfTimingCheck</a>
3	Sequential delay from the clock input port to the data output port. It includes the driver delay	<a href="#">setWhatIfSeqDelay</a>
4	Driver type	<a href="#">setWhatIfDriveType</a>
5	Driver input slew	
6	Total driver output net capacitance	
7	Clock insertion delay to internal registers	<a href="#">setWhatIfClockLatency</a>

A normalized timing model uses the following formula for timing arcs ending on output ports:

Delay = constant delay - driver delay<sup>\*</sup> + driver delay (look-up table)

Where,

*constant delay* = Timing arc delay including driver delay

*driver delay*<sup>\*</sup> = Constant delay considering an input slew and an output capacitance

*constant delay - clock latency* must be greater than *driver delay*<sup>\*</sup>

In a normalized timing model mode driver input slew is always required. In this mode, timing arcs are only load dependant. If you do not specify the driver total output net capacitance, the software takes real net capacitance into account.

## Using the What-If Timing Commands

You can perform the following tasks with the what-if timing commands:

- Selecting Timing Model

Use the following command to select the timing mode:

- [setWhatIfTimingMode](#)

- Defining generated clocks on internal pins:

Use the following command to create an internal pin and to define a generated clock on the pin.

- [createWhatIfInternalGeneratedClock](#)

- Set the following values on the what-if ports, if required:

- Capacitance
  - Maximum capacitance
  - Maximum transition
  - Maximum fanout

Use the following command to set these values on the what-if ports:

- [setWhatIfPortParameters](#)

By default, the parameters specified with the [setWhatIfPortParameters](#) command are applied to all ports in the what-if timing analysis model. If you want to apply the values for a particular port, specify the port name with the [setWhatIfPortParameters](#) -port parameter.

- Selecting the precedence between the values set by [setWhatIfDriveType](#) command and the values set by the [setWhatIfPortParameters](#) command

On output ports, parameters such as capacitance value, maximum capacitance values, maximum transition value, or the maximum fanout value can come from the driver ([setWhatIfDriveType](#) command) or they can be set through the [setWhatIfPortParameters](#) command.

Use the following command to define which of these values will take precedence in case of a conflict.

- [setWhatIfTimingMode](#)

## ■ Modifying Timing Arcs

While what-if commands are the same for both intrinsic and normalized timing models, the delay value specified in the commands for the combinatorial and the sequential timing arcs has different meaning. The driver output net capacitance is a characteristic of the normalized timing model only. Whenever you create or modify a timing arc, the timing graph is updated automatically. The Encounter software recomputes the entire timing arc whenever any of the parameter such as clock insertion delay, timing arc delay or driver type is modified.

**Note:** The timing sense of the driver is taken into account in the combinatorial what-if timing arc description—while applying the drive type, the timing sense of the combinatorial arc is replaced by the timing sense of the driver's timing arc. For sequential arcs, the timing sense is always set to non\_unate.

Use the following commands to modify timing arcs:

- [setWhatIfDriveType](#)
- [setWhatIfCombDelay](#)
- [setWhatIfSeqDelay](#)
- [setWhatIfTimingCheck](#)
- [setWhatIfClockPort](#)
- [setWhatIfClockLatency](#)

## ■ Getting Timing Arcs Assertions

Use the following command to get what-if timing arc assertions:

- [getWhatIfTimingAssertions](#)

## ■ Saving Timing Arcs Assertions

Use the following command to save what-if timing arc assertions:

- [saveWhatIfTimingAssertions](#)

## ■ Deleting Timing Arcs Assertions

Use the following command to delete the what-if timing arc assertions:

- [deleteWhatIfTimingAssertions](#)

## Encounter User Guide

### What-If Timing Analysis

---

#### ■ Checking Timing Assertions

Use the following command to check the what-if timing assertions:

- [checkWhatIfTiming](#)

#### ■ Generating what-if timing Models

After modifying the what-if timing model (in memory) using the what-if command, you can generate an updated timing model (.lib).

Use the following command to generate an updated .lib file:

- [saveWhatIfTimingModel](#)

#### ■ Generating What-If SDC constraints

The Encounter software generates the what-if timing constraints considering the top-level environment of the blackbox or blackblob. It provides a higher convergence for a top-down flow. The software generates drive, load and transition as IN context. The software generates the input and output delays as OUT context taking into account the last modifications done when you use the what-if commands.

Use the following command to save the What-If constraints:

- [saveWhatIfConstraints](#)

**Encounter User Guide**  
What-If Timing Analysis

---

---

## **Bus Planning**

---

- [Overview](#) on page 230
- [Bus Planning Flow in Encounter](#) on page 231
- [Creating a Bus Guide](#) on page 232
  - [Using the Edit Bus Guide GUI](#) on page 232
  - [Using Text Commands](#) on page 237
  - [Example](#) on page 238
- [Customizing the Bus Guide Display](#) on page 242
  - [Highlighting and Dehighlighting the Bus Guide](#) on page 242
- [Saving and Restoring Bus Guide Information](#) on page 244
- [Limitations of Bus Planning](#) on page 245

## Overview

The Bus Planning feature in the Encounter software enables you to plan and create bus guides which are used to guide the path of busses for floorplanning, partition pin optimization, feedthrough insertion, congestion prediction in trialroute, and final routing in nanoroute.

Most designs need bus planning for estimating the design size and routing channel widths. Without bus guides, the routers do not route all the bus bits together on the desired path. Routing the bus bits outside the desired path can have high cost implications. Hence it is very important to accurately plan the bus guide layouts.

Bus planning is critical in the prototyping stage of the hierarchical flow. Use the bus planning capability to guide the path of bus routing for feedthrough insertion, partition pin optimization, and congestion prediction. If you are in the implementation stage, use bus planning to guide the path of busses for detailed routing.

## Bus Planning Flow in Encounter

For hierarchical designs, you create bus guides before or after assigning the partition/black box pins. For flat or top-level designs, you create bus guides before routing. Normally, you create bus guides before pin assignment.

The following steps describe the bus planning flow in Encounter:

1. Importing the design

Import the design into the Encounter environment.

2. Floorplanning the design

If the design is a partition design then specify partitions. For more information, see [Specifying Partitions and Blackboxes](#) in the “Partitioning the Design” chapter of the *Encounter User Guide*.

If it is a black box design then define black boxes and specify their sizes. You can manually preplace black boxes/macros or run `planDesign` to automatically place them. Further, adjust the floorplan if needed.

3. Defining net groups

Group the bus bit nets together as net groups using `createNetGroup` and/or `addNetToNetGroup` commands.

4. Creating bus guides

Create bus guides associated with the net groups, to guide routing for all the nets of the specified net group. Bus guides can be created using the [Edit Bus Guide](#) GUI and/or the `createBusGuide` command. See [Creating a Bus Guide](#) on page 232.

5. Placing the design

Place the standard cells. If you do not want the Encounter placer (`placeDesign`) to move your macros and/or black boxes, set their placement status to `fixed` before running placement.

**Note:** This is an optional step for designs that do not have standard cells at full-chip level.

6. (Optional) Routing the design

Run `trialRoute` to route the design.

7. (Optional) Inserting feedthrough buffers

Feedthrough can be inserted based on routing or placement. If `trialRoute` was run before this step, then feedthroughs are inserted based on routing.

For more information, see [Inserting Routing Feedthroughs](#) in the “Partitioning the Design” chapter of the *Encounter User Guide*.

8. Assigning pins

Assign pins using [`assignPtnPin`](#) command.

9. Committing partition

Commit partitions using [`partition`](#) command.

10. Saving Partition

Save the partition information using [`savePartition`](#) command.

11. Running NanoRoute/Mixed Signal Route at the top-level design

Perform detailed routing using [`NanoRoute router`](#) / Mixed Signal Route ([`routeMixedSignal`](#)) at the top-level design.

## Creating a Bus Guide

A bus guide consists of one or more overlapping segments. It must always be associated with a net group. So, before creating a bus guide you must define a [`net group`](#). Remember that a net group can either be assigned to a bus guide or a pin guide, but not to both. For each bus guide segment that you create, you must specify a layer or a layer range.

You can create a bus guide [Using the Edit Bus Guide GUI](#) and/or [Using Text Commands](#).

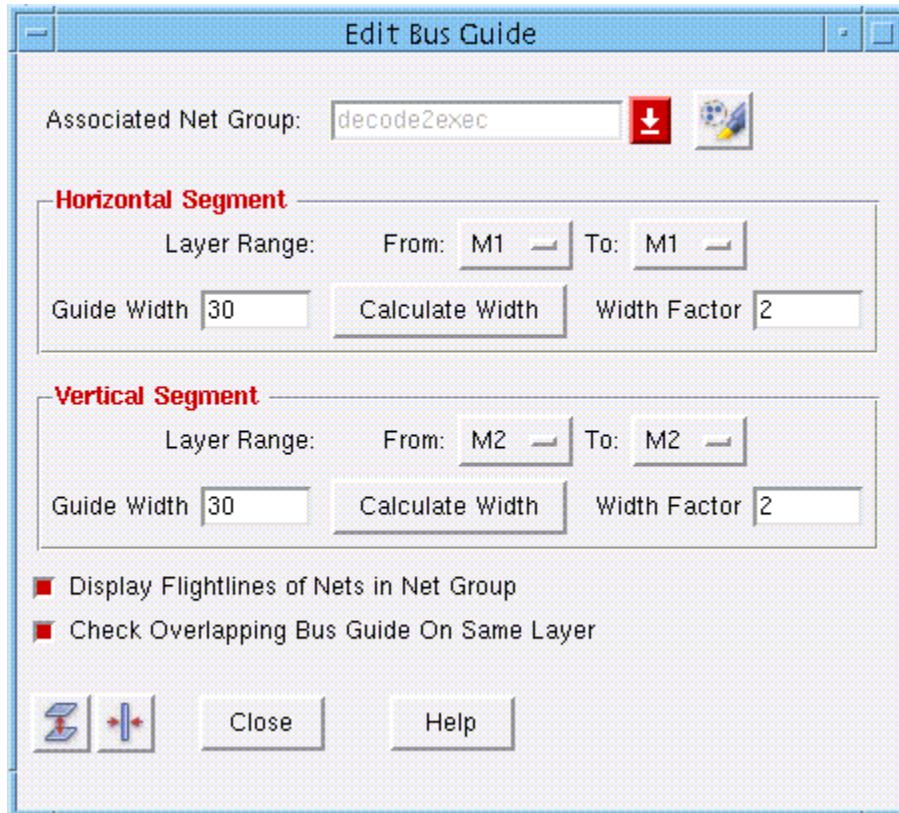
### Using the Edit Bus Guide GUI

The bus guide editor in Encounter, allows you to create bus guides before or after assigning the bus pins. Using the *Edit Bus Guide* form, you can edit the bus guide properties and interactively create the bus guide. You can specify the net group associated with the bus guide, layer or layer range on which the bus guide is to be created, and the width of the bus guide segment. By default, the bus guide editor derives the default minimum guide width required to hold all the nets assigned to the bus guide. If the bus guide connects to placed pins on block edges, the bus guide editor automatically adjusts the width of the guide segment to cover all the pins of nets in the net group. The bus guide editor provides options

## Encounter User Guide

### Bus Planning

to enable overlapping check for bus guides created on a specific layer and display flight lines of nets in the net group, when creating the bus guides.



For more information on the *Edit Bus Guide* form, see [Edit - Object - Edit Bus Guide](#) in the “[Edit Menu](#)” chapter of the *Encounter Menu Reference*.

### Drawing a Bus Guide

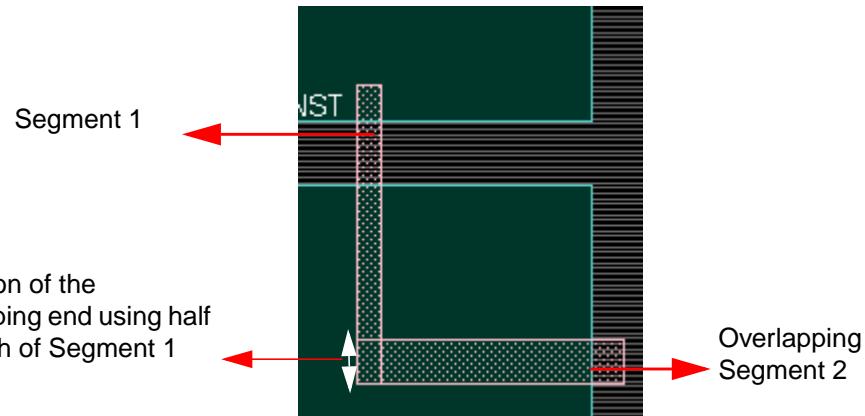
To draw a bus guide in Encounter, you must first click the *Add Bus Guide* icon in the toolbar.

Once you are in the bus planning mode, you can draw the bus guide segment by clicking the left mouse button and dragging it along the points of center line for the guide segment. To end a bus guide segment, double-click the left mouse button. By default, the bus guide extends half width for the overlapping end of the created segment. However, if the guide segment overlaps with another segment that has bigger or smaller width, the bus guide

## Encounter User Guide

### Bus Planning

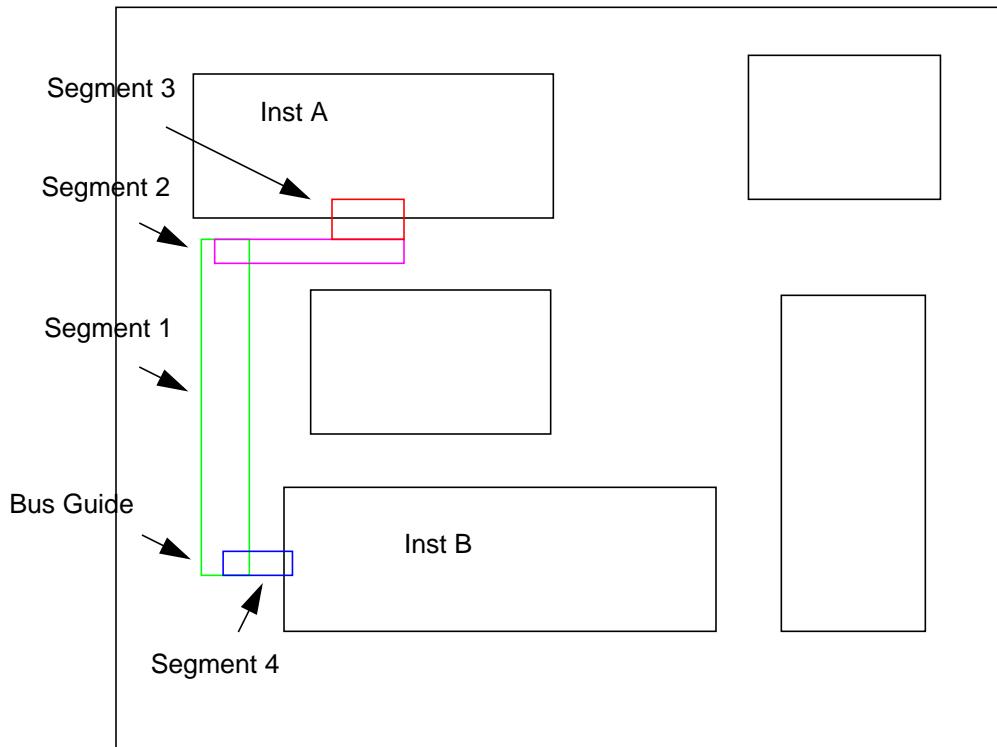
editor uses half the width of the other segment for the extension of the overlapping end.



**Note:** All the segments of the bus guide should overlap to ensure continuity; Otherwise, the router (nanoroute) may create routing problems or may take longer time to run.

You can specify a new segment connected to an existing segment as shown in the following image where segment 4 overlaps with segment 1:

**Figure 10-1**

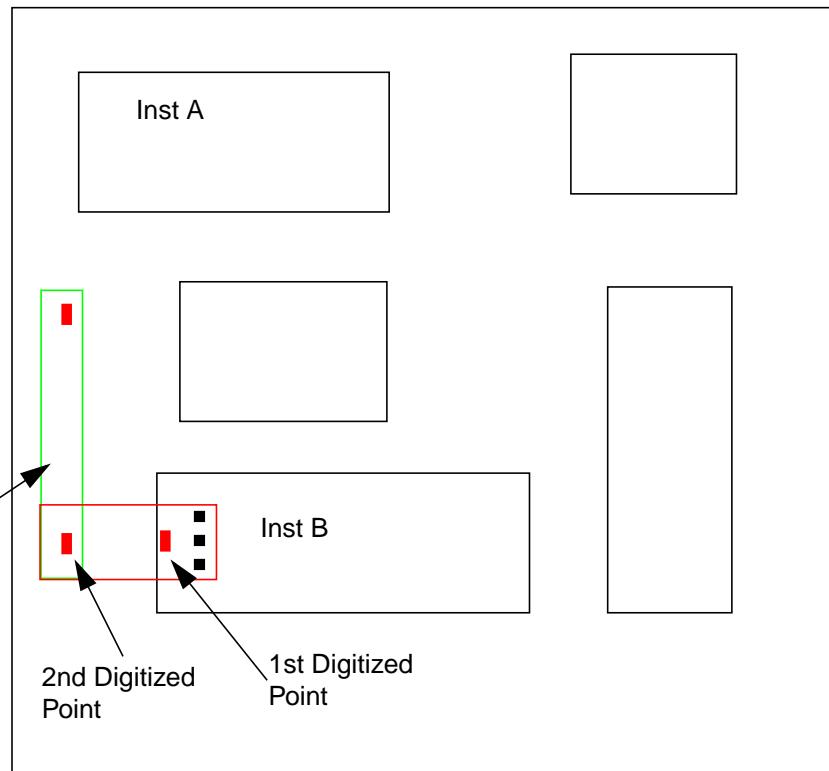


## Encounter User Guide

### Bus Planning

You can also draw a bus guide segment that connects to the placed pins of the associated net group.

**Figure 10-2**



If you click on the partition boundary side where the pins are placed, the bus guide editor automatically snaps to these pins. If the width value specified in the bus guide editor is smaller than the width required to fully cover all these pins, the bus guide editor derives new width for the guide segment such that all the associated physical pin geometries are covered. If the width value is bigger than the width that needs to cover all pins, the editor will use the current width value without adjusting it.

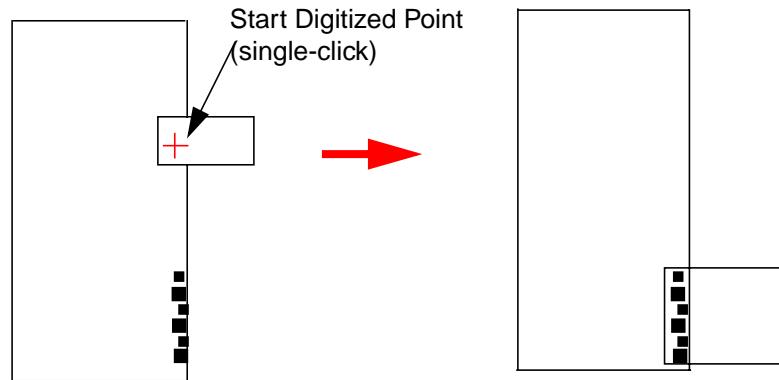
In the [Figure 10-2](#) on page 235, the width of the segment defined by the first and the second digitized points is derived based on the placed pin information such that the segment width can fully cover the all the pins. The width of the next segment (defined by second and third points) is the width that is specified in the bus guide editor.

The snapping of bus guides to pins (partition or black box pins) occur at the start or at the end of the bus guide, when you double-click to end the bus guide.

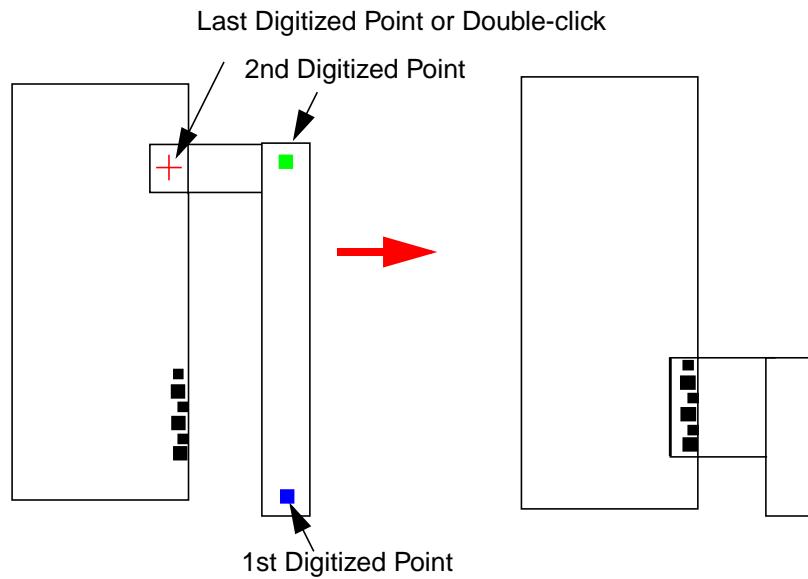
## Encounter User Guide

### Bus Planning

The following example illustrates the snapping behavior at the starting digitized point. The snapping occurs before you specify the second point:



The following example illustrates the snapping behavior at the end of a bus guide

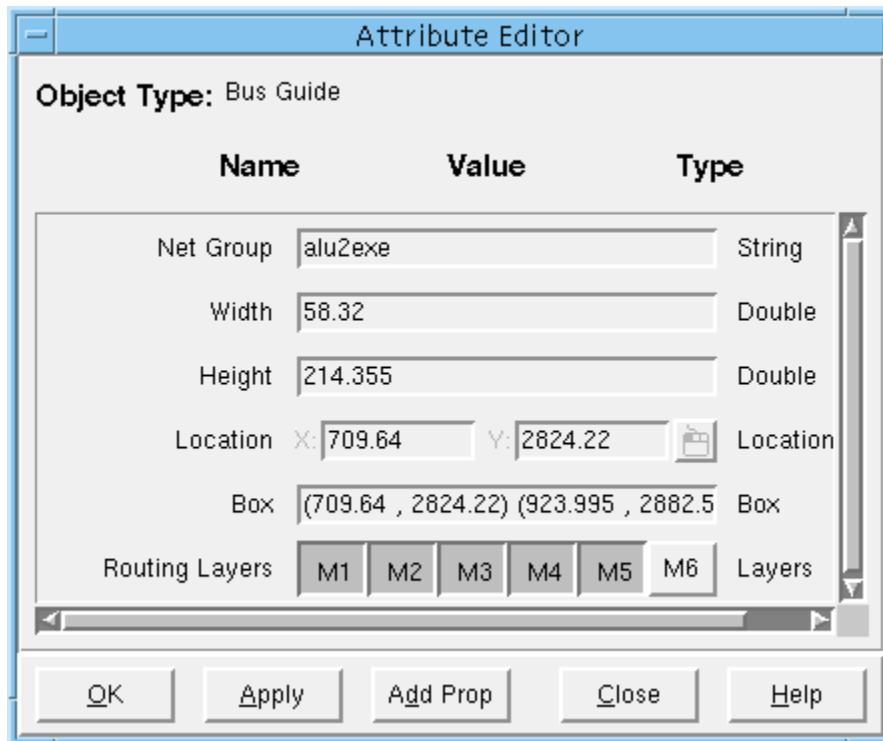


To view the attributes of a bus guide that you created, double-click the bus guide segment to

## Encounter User Guide

### Bus Planning

display the Attribute Editor as shown in the following example:



A bus guide gets deleted when you delete its associated net group.

## Using Text Commands

You can create and edit bus guides using the following text commands:

Commands	Usage
<u>createBusGuide</u>	Creates a bus guide segment.
<u>deleteBusGuide</u>	Deletes a bus guide.  <b>Note:</b> You can also delete a bus guide segment by selecting the segment and pressing the Del key on the keyboard.
<u>deselectBusGuide</u>	Deselects a bus guide segment.
<u>selectBusGuide</u>	Selects a bus guide segment.

## Encounter User Guide

### Bus Planning

---

<b>Commands</b>	<b>Usage</b>
<code><u>selectBusGuideSegment</u></code>	Selects a bus guide segment with its specified bounding box.

For more information on the commands, see the “[Bus Plan Commands](#)” chapter in the *Encounter Text Command Reference*.

The following [Example](#) describes the steps to create bus guides using text commands.

### **Example**

This sample script creates 2 bus guides for 2 bus nets, abcBusNet and cdeBusNet. The abcBusNet bus has 32 bus bits and cdeBusNet has 100 bus bits. 2 net groups, abcNetGroup and cdeNetGroup are defined for abcBusNet and cdeBusNet busses, respectively. 2 bus guides are used to guide routing for these 2 busses for feedthrough insertion:

```
#Restore the bBoxFP.enc.dat design of top cell Test that is already being floorplanned  
restoreDesign bBoxFP.enc.dat Test
```

```
#Create net groups for busses abcBusNet and cdeBusNet
```

```
createNetGroup abcNetGroup -net abcBus*  
createNetGroup cdeNetGroup -net cdeBus*
```

```
#Create bus guide for bus net abcBusNet[0..31]. This bus guide has 4 segments.
```

```
createBusGuide -netGroup abcNetGroup -centerLine 4421.8 10749.36 4960.8 10749.36 -  
width 90 -layer 4:8  
createBusGuide -netGroup abcNetGroup -centerLine 4900.8 10809.36 4900.8 9470 -width  
90 -layer 3:7  
createBusGuide -netGroup abcNetGroup -centerLine 4840.8 9530.0 11525.4 9530.0 -  
width 90 -layer 4:8  
createBusGuide -netGroup abcNetGroup -centerLine 11465.4 9590.0 11465.4 9203.5 -  
width 90 -layer 3:7
```

```
#Create bus guide for net cdeBusNet[0..99] that has only one vertical segment.
```

```
createBusGuide -netGroup cdeNetGroup -centerLine 15300.7 7061 15300.7 11230 -width  
300 -layer 5:7
```

```
# Place the design since design has some top-level cells
```

```
placeDesign
```

```
#Run trialRoute with option -printWiresOutsideBusguide to report any nets that  
are routed outside specified bus guide areas
```

## **Encounter User Guide**

### Bus Planning

---

```
trialRoute -printWiresOutsideBusguide
```

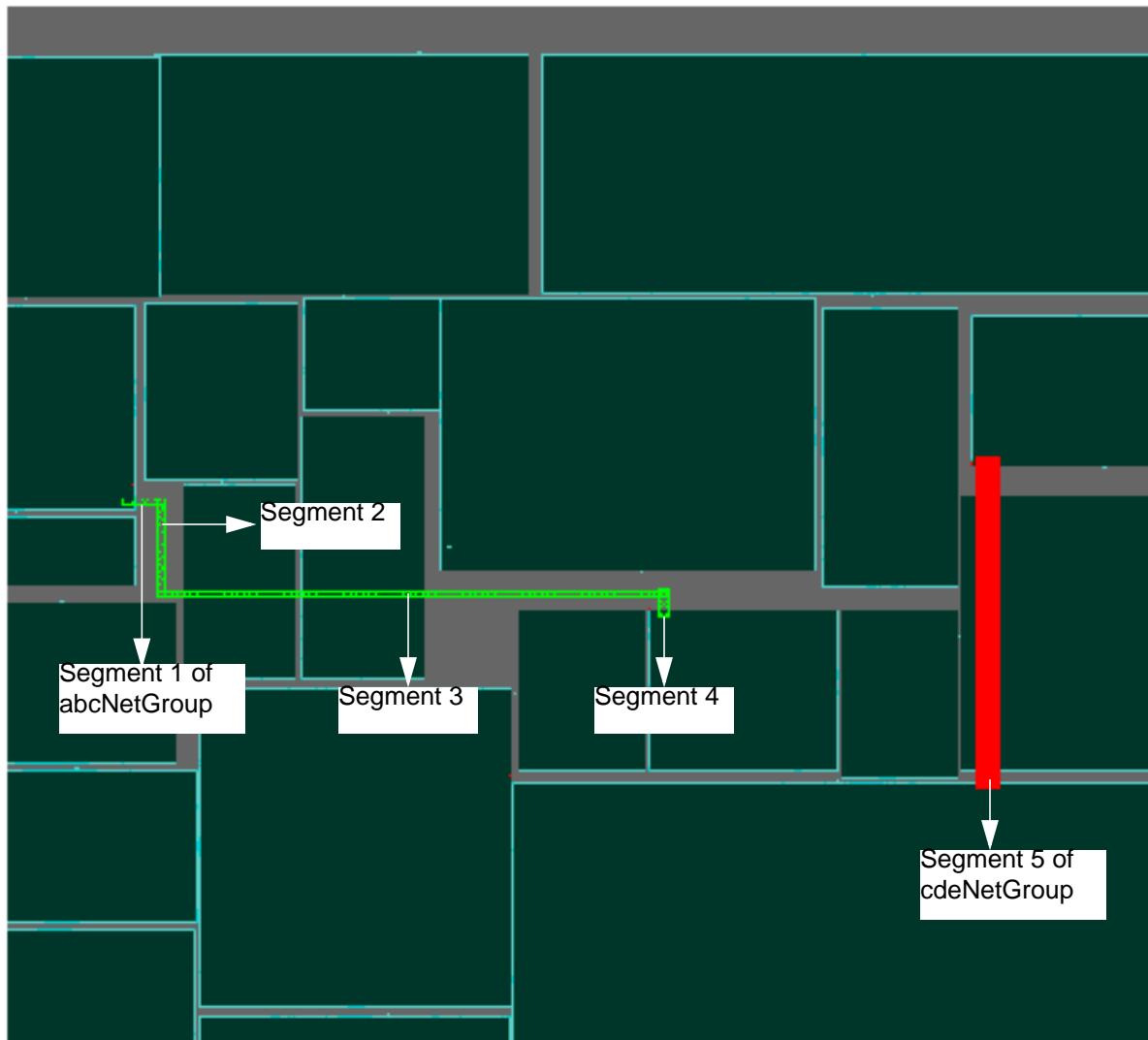
#Continue with the normal flow, invoking feedthrough insertion, pin assignment, and so on...

## Encounter User Guide

### Bus Planning

The following figure displays the bus guide associated with the net group `abcNetGroup`, highlighted in *green*, and the bus guide associated with the net group `cdeNetGroup`, highlighted in *red*:

After running `createBusGuide` to create 5 segments



Segments 1, 2, 3, and 4 belong to `abcNetGroup`  
Segment 5 belongs to `cdeNetGroup`

## Encounter User Guide

### Bus Planning

The following figure displays the routing of the bus `abcBusNet[0...31]`, routed within the bus guide area:

After running the `placeDesign` and `trialRoute`



All the 32-bus bits of `abcBusNet` group are routed within the bus guide area.

## Customizing the Bus Guide Display

You can specify multiple colors for bus guide objects in the design, using the *Bus Guide Color Selection* form. (*Color Preferences — Objects — Bus Guide — Bus Guide Color Selection*)

### Highlighting and Dehighlighting the Bus Guide

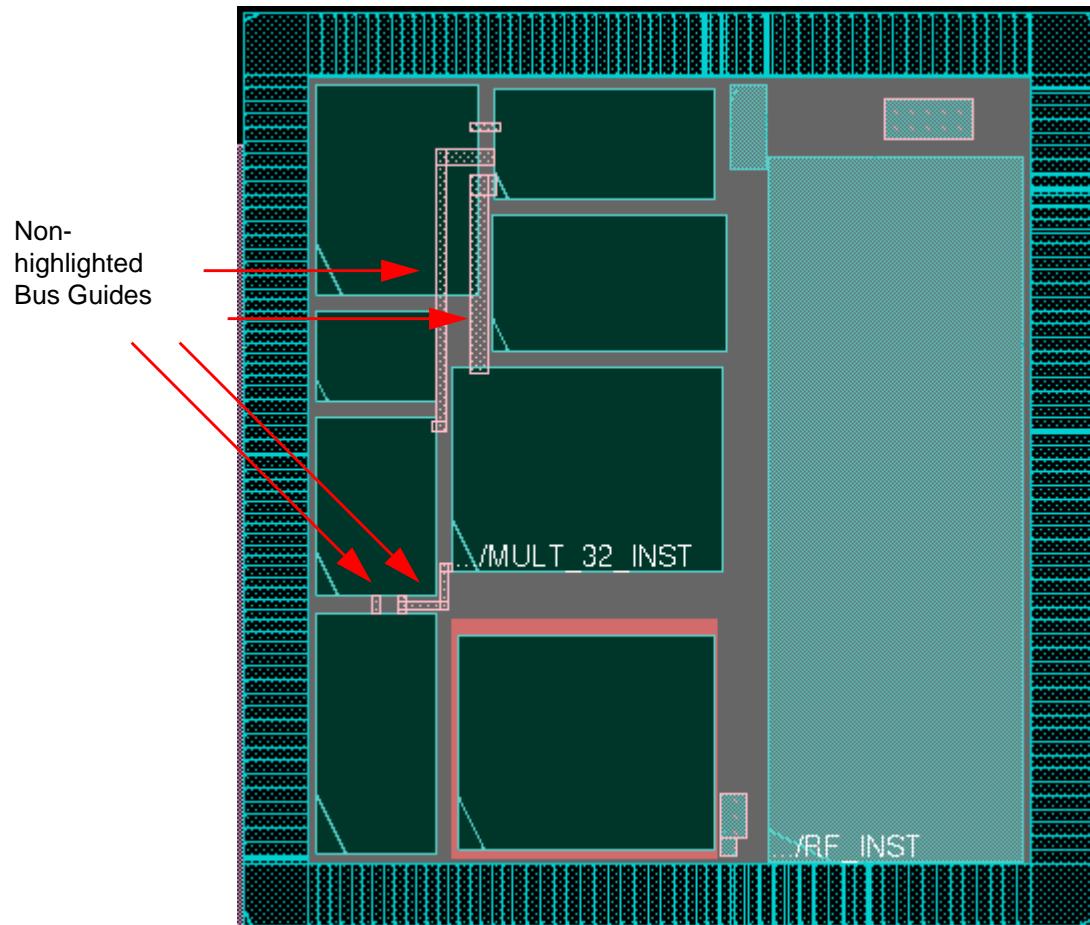
After specifying colors for bus guides, you can highlight the bus guides in the design using the *Edit — Objects — Highlight Bus Guide Colors* menu command.

Alternatively, you can run the `setBusGuideMultiColors` command to color the bus guides and `resetBusGuideMultiColors` command to clear the bus guide colors.

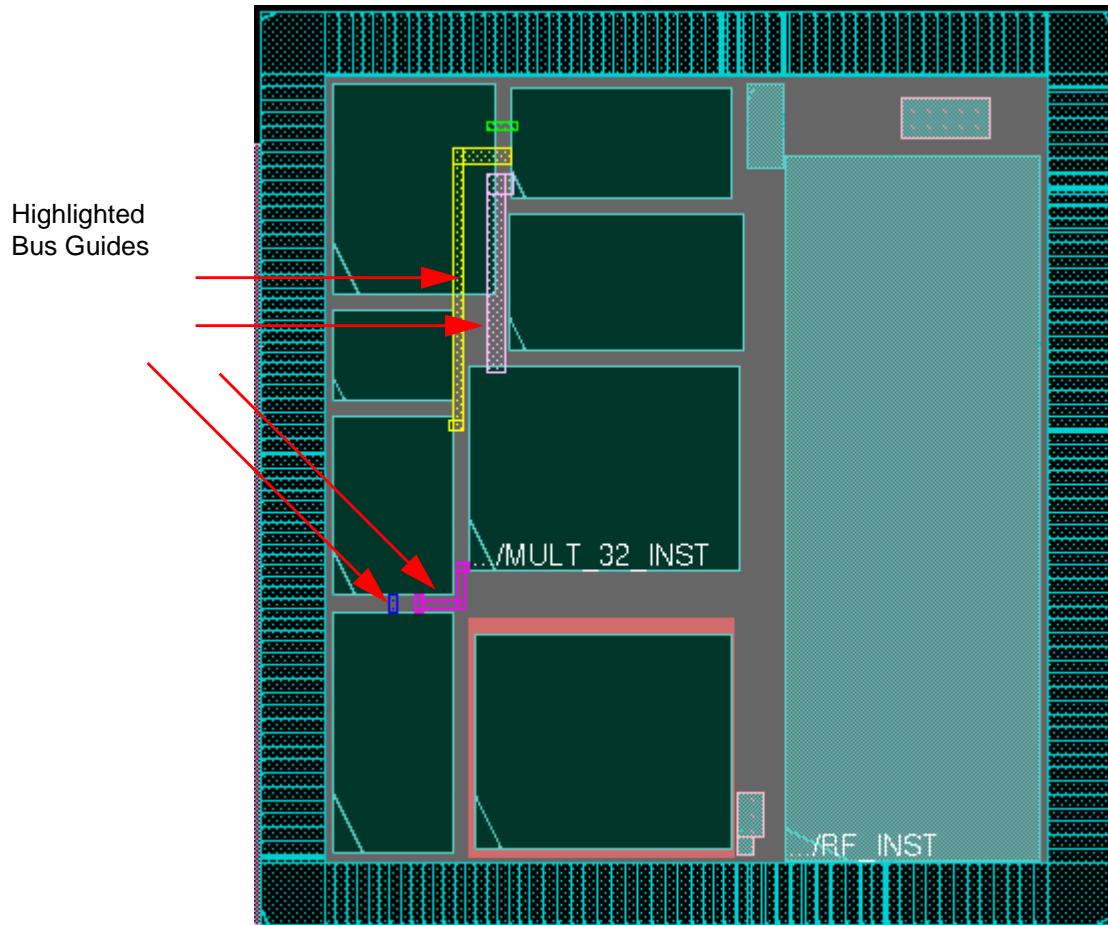
## Encounter User Guide

### Bus Planning

The following example displays the bus guides before you run the [setBusGuideMultiColors](#) command:



The following example displays the bus guides after you ran the [setBusGuideMultiColors](#) command:



## Saving and Restoring Bus Guide Information

The bus guide data is stored in the floorplan spr file (.fp.spr file). You can save and restore this information using the [saveFPlan](#) and [loadFPlan](#) commands.

However, you cannot load the .fp.spr file having bus guide information from the 8.1 version, into an older version of Encounter.

## **Limitations of Bus Planning**

- Feedthrough insertion does not honor bus planning. Once you insert feedthroughs in the design, the existing bus guides will no longer be valid.
- The software currently does not provide checks to detect the following:
  - Overlapping bus guide segments on different layers
  - Complete bus guide coverage from source to sink
  - Complete coverage of placed pins
  - Enough room for routing.

## **Encounter User Guide**

### Bus Planning

---

---

## Partitioning the Design

---

- [Overview](#) on page 248
- [Flow Methodologies](#) on page 248
- [Specifying Partitions and Blackboxes](#) on page 256
- [Inserting Feedthroughs](#) on page 305
- [Generating the Wire Crossing Report](#) on page 326
- [Estimating the Routing Channel Width](#) on page 329
- [Running the Partition Program](#) on page 331
- [Restoring the Top-Level Floorplan with Partition Data](#) on page 347
- [Concatenating Netlist Files of a Partitioned Design](#) on page 348
- [Saving Partitions](#) on page 349
- [Loading Partitions](#) on page 349
- [Working with OpenAccess Database](#) on page 351
- [Parallel Job Processing](#) on page 352

## Overview

Most of the system-on-a-chip devices are designed in a traditional flat flow that avoids the effort to set up a design hierarchy. However, in multi-million gate designs, this could result in memory limitations and long run time. Design teams can develop and adopt a hierarchical flow to shorten the turnaround time on large designs. Designs can be divided into manageable partitions; each partition can be independently assigned to different design groups to be developed in parallel.

## Flow Methodologies

Hierarchical design can be divided into three general stages: chip planning, implementation, and chip assembly.

- Chip Planning
  - Breaks down a design into block-level designs to be implemented separately.
- Implementation
  - This stage consists of two sub-stages: block implementation for a block-level design, and top-level implementation for a design based on block-level design abstracts and timing models.
- Chip Assembly
  - Connects all block-level designs into the final chip.

This chapter covers the following methodologies in the partitioning area:

- [Top-down Methodology](#) on page 249
- [Bottom-up Methodology](#) on page 253

## Top-down Methodology

The top-down methodology usually consists of top-down planning, implementation, and chip assembly stages. Use this methodology to create a top-level or hierarchical floorplan from a flat floorplan based on fenced modules. In this approach, the die size, shape, and I/O pads locations will drive block and partition placement. Block-level design size and pins will be generated based on the top-level floorplan.

### Chip Planning

The following steps describe the most common flow for chip planning, which includes specifying partitions and blackboxes:

1. Import the entire design to be partitioned.

Import the design into the Encounter environment. You can also include blackboxes.

2. (Optional) Define the blackboxes.

If your design has blackboxes that are not specified in step 1, you can define them after reading in the netlist. You can also adjust the size of the blackboxes. For more information, see [Saving Blackboxes](#) on page 262.

3. Lay out the floorplan.

Manually pre-place all modules that will become partitions or blackboxes. You can also generate an initial floorplan by running plandesign to place Macros, then place standard cells and/or bring all modules inside the core by generating the floorplan guide (using the *Floorplan – Generate FP Guide* menu command or the `generateGuide` text command).

4. Run power planning.

5. Specify the modules and blackboxes that will become partitions.

You can further adjust blackbox size, if necessary. For more information, see [Specifying Partitions and Blackboxes](#) on page 256.

6. Run placement.

7. (Optional) Insert feedthrough buffers.

Insert feedthrough buffers into partitions to avoid routing nets over partition areas. This step is necessary for channelless or mixed designs. For more information, see [Inserting Feedthroughs](#) on page 305.

Run Trial Route before this step if you want to run route-based feedthrough insertion. You must also run Trial Route if you want to display and generate a list of all nets that cross over the top of each partition (using the *Partition – Show Wire Crossing* menu command or the `showPtnWireX` text command).

## 8. Run Trial Route

Depending on what stage of the design is in, such as prototyping, intermediate, tapeout, use the appropriate option of the `trialRoute` command. For example, the `-floorplanMode` option should be used for prototyping and the `-highEffort` option should be used for tapeout mode. Use the `-handlePartition` or the `-handlePartitionComplex` parameter for channel-based designs. Use the `-handlePartitionComplex` parameter for channelless designs only after the feedthrough insertion step.

For designs that have multiple partitions, are congested, and/or have thin channels, use the `ptnAwareRouteForPA` command or `trialRoute -fastRouteForPinAssign`.

This route command generates routing topology similar to `trialRoute -handlePartitionComplex` but with lesser run time because it routes only the inter-partition and top-level nets.

If your design has blackboxes, you can run the `trialRoute` command with the `-routeBasedBBPin` parameter. With this parameter, the `trialRoute` command determines near-optimal location for blackbox pins with respect to top channel congestion and places blackbox pins at these locations. The `trialRoute` command then creates routes to the blackbox pins without crossing over blackboxes.

The results give the first-order location of aligning the partition pins.

9. Assign partition pins and blackbox pins using the `assignPtnPin` command.
10. Regenerate the routes that follow assign pins using the `trialRoute -honorPin` command.
11. Validate pin assignment result
12. If needed, refine the pin assignment results or perform incremental pin assignment.  
If pin placement results need to be improved, you can further refine pin placement manually or automatically. After re-adjusting pins, verify pin placement again.
13. Budget the timing for blocks using the `deriveTimingBudget` command.
14. Partition the design using the `partition` command.

If your design has multiple instantiated partitions, run the `alignPtnClone` command before the pin assignment step to make sure that all partition clones are well aligned with

the master partition on a power mesh so you will not have any problems when flattening the partitions. For more information, see [Specifying Multiple Instantiated Partitions and Blackboxes](#) on page 265.

**15. Save the partition.**

This creates a directory for each block, and saves its netlist, floorplan, and budgeted constraints to this directory. For top-level designs, this also creates a directory containing the top-level netlist, floorplan, simple timing model, and physical abstract for each partition block or blackbox. Subsequent work should be done in these block-level and top-level directories for implementing the block-level and top-level designs, respectively.



*Tip*  
You should do all design work in each saved partition directory, including the top-level directory.

## Implementation

After the chip planning is complete, the next stage is to implement the individual blocks. The detail of each block is implemented using the constraints for timing, size, and pin assignment determined during the planning stage. Block implementation should be done at a block directory that was generated by the [savePartition](#) step. At the completion of this step, block abstracts, timing models, a DEF file, and a GDSII file should be generated to be used in top-level implementation and chip-assembly.

The next step is to implement the top-level designs with block model data, such as LEF, timing model, power model, and noise model.

## Chip Assembly

Chip assembly is the last stage in the top-down process and consists of bringing together the detailed information for the top-level and all of the blocks for full chip extraction, power, timing, and crosstalk analysis. Chip assembly is done using the [assembleDesign](#) command.

**Note:** Before using the [assembleDesign](#) command, for each design, save the top-level and the block-level designs using the [saveDesign -def](#) command.

As an example, consider a design called `dtmpf` that has two partitions: `arb` and `tdsp`. After running the [partition](#) command, the partition directories are saved under the PTN directory. You would, therefore, implement the following:

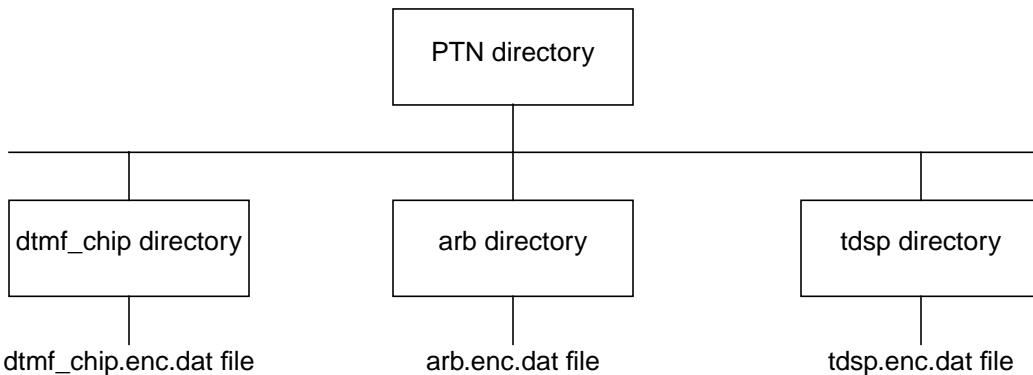
- top-level design `dtmpf_chip`

## Encounter User Guide

### Partitioning the Design

- arb block
- tdsp block

The design files are `arb.enc.dat` and `tdsp.enc.dat` for the arb and tdsp blocks respectively. The following figure shows the directory structure:



You can now perform chip assembly using the `assembleDesign` command. This command does the following:

- Concatenates the Verilog netlist files from the partitions back to the top level  
**Note:** The partition netlists and top level netlist are changed from the time the save partition step was performed.
- Merges the design data with the original top design level. By default, data from DEF files is used. However, you can use the `-fe` parameter to specify that Encounter data should be used. You can also use data in the OpenAccess database format.
- Brings back the row information if the `-row` parameter is specified.

Run this command from the directory that contains the full chip-level floorplan for the top-down hierarchical flow.

For details of the syntax and the parameters, see the description of the `assembleDesign` command in the *Encounter Text Command Reference*.

For this example, you would run the `assembleDesign` command as follows:

```
assembleDesign -topDir PTN/dtmf_chip/dtmf_chip.enc.dat -blockDir PTN/arb/arb.enc.dat -blockDir PTN/tdsp/tdsp.enc.dat -topFP fullChip.fp
```

This assembles the entire design.

You can also use the `assembleDesign` command to bring back specified block data from OpenAccess database. Here is an example:

```
assembleDesign -topDesign testOALib DTMF layout -block testOALib ptn1 layout -block testOALib ptn2 layout
```

In this example, the OpenAccess database top-level library is `testOALib`, the top-level cell name is `DTMF`, and the top-level view is `layout`. Two blocks, `ptn1` and `ptn2`, have been specified.

**Note:** The `assembleDesign` command supports rectilinear partitions. It also supports nested blackboxes for the place-and-route data (-fe parameter) and the OpenAccess database. However, because blackbox information cannot be specified in a block-level DEF file, nested blackboxes are not supported for the DEF flow.

## **Bottom-up Methodology**

The bottom-up methodology consists of implementation and assembly stages. In the bottom-up methodology, the size, shape, and pin position of block-level designs will drive the top-level floorplanning.

### **Implementation**

Each block in the design must be fully implemented. This includes place and route as well as clock, power, and I/O.

This section covers the following topics:

- [Block Implementation](#) on page 253
- [Top-level Implementation](#) on page 255

### ***Block Implementation***

The size of a block-level design can be derived or adjusted using the *Floorplan – Specify Floorplan* menu command or the `floorPlan` text command. The Encounter software can support a rectilinear block level design. You can use the same procedure to create a rectilinear partition to create a rectilinear block-level design using the following steps:

1. Click on the *Cut Rectilinear* widget from the *Tools* area.
2. Move the mouse to an edge or corner of the design.
3. Left click and drag over the area.

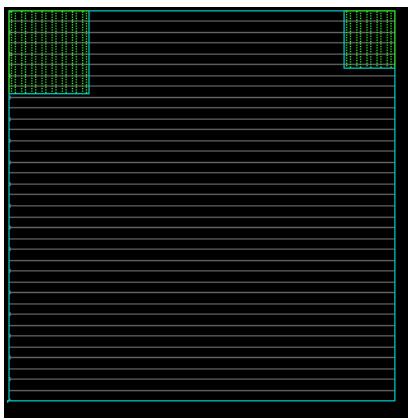
## Encounter User Guide

### Partitioning the Design

4. Left click again to complete the cut.

At a block level design the rectilinear information will be stored in a floorplan file as a CellPtnCutList syntax, for example:

```
CellPtnCutList: execute_i 2
    0.0000 142.5100 37.1200 181.4400
    156.3800 154.9350 180.1800 181.4400
```



You can run the [assignIoPins](#) text command to assign I/O pins based on placement information.

You can specify initial I/O pin placement in an I/O constraint file. For more information, see the Generating the I/O Assignment File section in the “[Data Preparation](#)” chapter of the *Encounter User Guide*. You can read in the I/O constraint file into the Encounter environment during the design import step, or use the `loadIoFile` text command after reading in the netlist.

If an I/O constraint file does not exist, an initial I/O pin placement can be derived from cell placement. After placing macros and standard cells, the placer can internally call the `assignIoPins` text command to place I/O pins based on current cell placement. By default, pins are placed under power areas on different layers. Use the `-pinOffStripes` or `-noPinBelowStripe` option of the `assignIoPins` command to disable the default behavior.

**Note:** Use the `setPlaceMode -placeIoPins` option to disable I/O pin assignment during placement.

After I/O pins have been assigned, you can further refine the current I/O pin assignment by doing either of the following:

- Adjust pins (using the Pin Editor or the `editPin` text command). You can also use direct pin manipulation to manually move selected pins to different locations.

- Run incremental pin assignment by running the `assignIoPins` text command. This command honors fixed pins and re-assigns only the ones that have a *placed* or *unplaced* status.

**Note:** The `loadIoFile` text command automatically sets the I/O pin placement status to fixed. For the pins that need to be re-assigned, you must change their pin placement status.

You can use the `legalizePin` text command to resolve pin overlaps or pins off-grid.

### ***Top-level Implementation***

After block implementation, an abstract should be developed for each block-level design that will be used in the top-level implementation.

For the bottom-up approach, create a top-level floorplan where block-level abstracts would be referenced in the top-level design.

### **Chip Assembly**

For the bottom-up approach, see “[Chip Assembly](#)” on page 251, to bring together all the top-level and block-level netlists and routing information.

**Note:** For the bottom-up approach, do not use the `-topFP` option of the `assembleDesign` command.

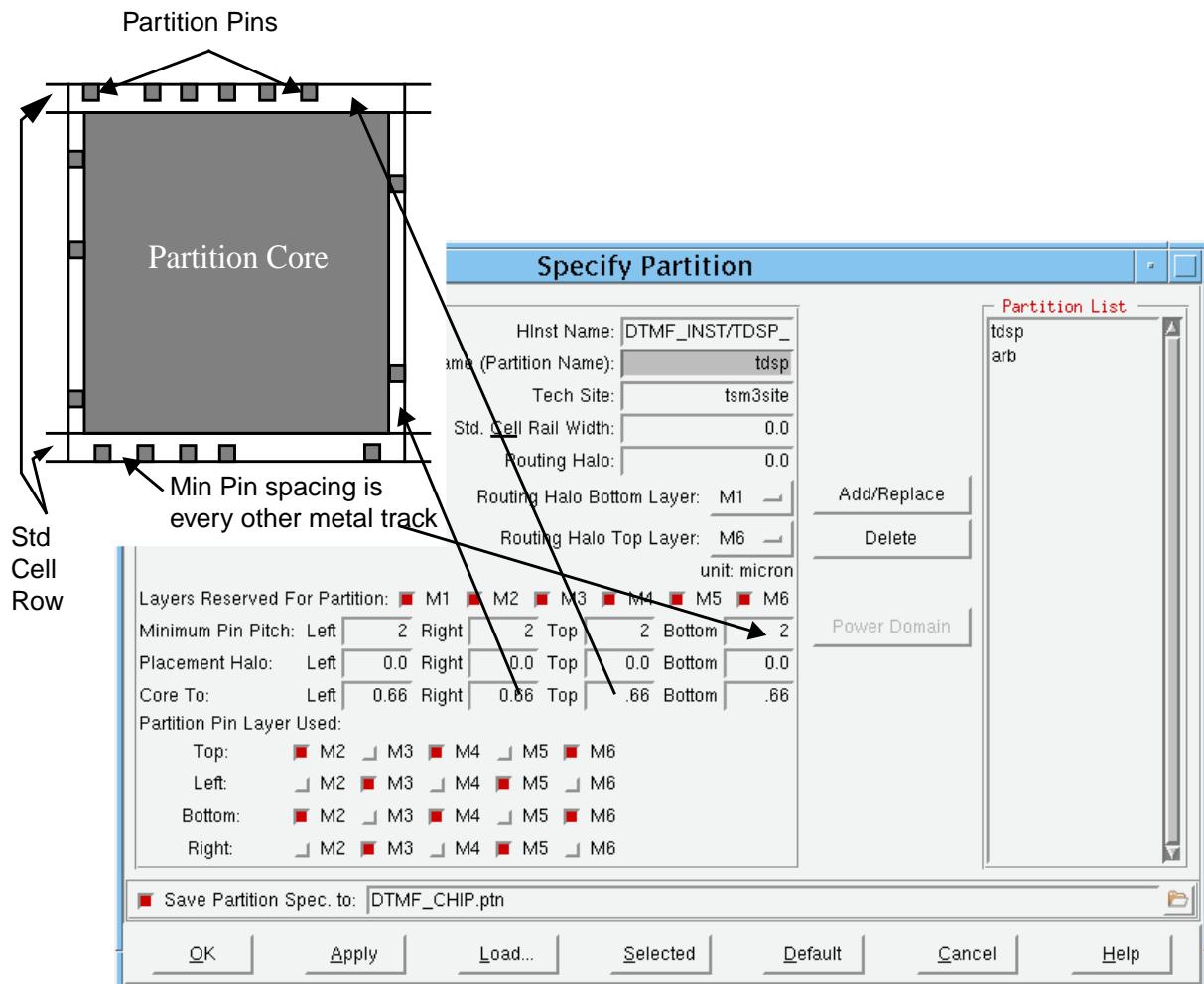
## Specifying Partitions and Blackboxes

- [Defining Partitions](#) on page 257
- [Defining Partitions as Power Domains](#) on page 260
- [Defining Blackboxes](#) on page 260
- [Saving Blackboxes](#) on page 262
- [Handling of Blackboxes with Non-R0 Orientation](#) on page 263
- [Specifying Multiple Instantiated Partitions and Blackboxes](#) on page 265
- [Changing Partition Clone Orientation](#) on page 266
- [Specifying Rectilinear Partitions and Blackboxes](#) on page 267
- [Specifying Core-to-I/O Distance for Partition Cuts](#) on page 268
- [Specifying Nested Partitions](#) on page 269
- [Assigning Pins](#) on page 269

## Defining Partitions

To designate partitions, use the `definePartition` and `specifyPartition` text commands and the Specify Partition form.

The following figure shows an example of how some of the fields in the Specify Partition form relate to the partition. For a description of all the fields, see Specify Partition in the *Encounter Menu Reference*.



To specify a module as a partition, complete the following steps:

1. Move the module inside the core area.

You can manually move a module, or use the `setFPlanObjBox` text command, to define a new module boundary with its coordinates in the core area.

**Note:** A blackbox is a special partition where this restriction does not apply.

2. Specify the name of the partition.
3. Specify the instance name of a module that is to become a partition.

**Note:** A partition cannot have another partition as its ancestor or descendant. For the case where more than one module is instantiated with the same cell type, see [Specifying Multiple Instantiated Partitions and Blackboxes](#) on page 265.

4. Specify the space, in micrometers, between the module boundary and core design area of the partition module.
5. (Optional) If the partition row height is different than specified in the *Core Spec* page of the Design Import form, specify the row height, in micrometers.
6. (Optional) To account for wide wires at the top-level design, specify the extra spacing, in micrometers, around the partition.

At the top-level design, this information is saved as part of the partition section in a floorplan file. This information is also saved in a partition floorplan file when saving partitions. By default, this value is 0; the top-level router uses minimum wire spacing.

7. Specify the selected metal layers that are used for routing in the partition and generating partition pins.

A normal six-metal layer selection process is M1, M2, M3, M4 and M5 selected, and M6 unselected. When saving the partition, the LEF generated for this partition will have routing blockages on their layers so that the top-level router is aware of which metal layers are being used in the partition.

The selected metal layers generate a file which is saved in the top-level design directory. This file notifies the top-level which metal layers are being used in the partition. In addition, the floorplan file generated by saving partition will include the routing blockage for the partition. To customize routing interconnects over a partition, use the *Add Partition Feedthrough* widget.

8. (Optional) Specify the pin pitch dimension for the partition sides.
9. (Optional) Select or deselect the metal layers from the defaults.

Deselecting all metal layers for a side of a partition prevents pins from being created for the entire side of that partition.

The selection of partition pin metal layers works in conjunction with the Partition Pin Guide floorplan object. The partition pin guide object specifies exactly where the pins are to be created. When partition pin guide objects are not used, the partition pins are created where the top-level routing connects with the partition.

## **Encounter User Guide**

### Partitioning the Design

---

- 10.** Add the partition information to the *Partition List* field and save the partition specification file.

## Defining Partitions as Power Domains

If a block-level design has different row structures than a top-level design, you will need to define a partition as a power domain. The power domain must be a hierarchical instance. The power domain will have the same size as the partition fence.

To specify a partition as a power domain, complete the following steps:

1. Import the design.
2. Create the power domain.
3. Floorplan the design.

In this step you would normally place the I/Os, place the power domain, and so on.

4. Assign a partition to a power domain by specifying the same power domain hierarchical instance as the partition.
5. Continue with the normal partition flow (see [Defining Partitions](#) on page 257).

## Defining Blackboxes

Normally a blackbox is a module with content that is not well defined. However, a well-defined module can also be defined as a blackbox. A blackbox is similar to a hard block, but like a fence, a blackbox can be resized, reshaped, and have pins assigned. After a blackbox has its pins assigned and is partitioned, it behaves like a hard block. The blackbox feature can be used only with a partitioned design.

After the netlist has been loaded, you can further specify which modules or cells will be regarded as blackboxes, or modify the existing blackbox sizes. A blackbox size can be specified in terms of an estimated area (an actual value or an area value in terms of gate count), or a fixed block width and height.

You can define a blackbox in the following ways:

- Use the `setImportMode -treatUndefinedCellsAsBbox False -keepEmptyModule True` command before importing a design. Once the design is imported, specify a module or hard macro as blackbox using the [specifyBlackBox](#) command or the [Partition-Specify Black Box](#) GUI form.

**Note:** Converting a hard macro into a blackbox will not update the blockage definitions when you change the blackbox size.

- Define LEF abstracts for blackboxes. You can specify a blackbox library in the *LEF Files* field of the Design Import form.

If a blackbox LEF abstract is specified in the *LEF Files* field, the LEF abstract should have CLASS type as BLOCK BLACKBOX to indicate it is a blackbox.

The following is an example of a blackbox LEF abstract:

```
MACRO amba_dsp
    CLASS BLOCK BLACKBOX ;
    ORIGIN 0 0 ;
    SIZE 4411.8600 BY 5697.3600 ;
END amba_dsp
```

After defining a blackbox with any of the above methods, you can further modify an existing blackbox size with the [specifyBlackBox](#) command.

**Note:** You can use the [getBlackBoxArea](#) command to retrieve the standard cell area, macro area, and cell utilization value for the specified blackbox.



***If you convert a hard macro into a blackbox or define a blackbox with a LEF abstract that has obstructions, the obstructions size will not be updated with a new blackbox size. Due to this limitation, obstructions may be intruded outside of the new blackbox boundary.***

## Blackbox Flow

**Note:** Even though there are more than one ways to define a black box, it is recommended that you define a black box by using the [specifyBlackBox](#) command.

The following flow specifies blackboxes with an original netlist that has modules with content that is not well-defined:

1. Import the design. By default, the Encounter software keeps empty modules ([setImportMode](#)  
`-treatUndefinedCellAsBbox false -keepEmptyModule true`)
2. Specify the blackboxes or load a floorplan file with blackbox information.
3. Floorplan the design.
4. (Optional) Save the design, which saves the blackbox information.
5. Run placement.
6. (Optional) Run Trial Route with or without the `-routeBasedBBPin` parameter. When you run Trial Route with this parameter, Trial Route determines near-optimal location for

blackbox pins with respect to top channel congestion and places blackbox pins at these locations. Trial Route then creates routes to the blackbox pins without crossing over blackboxes.

**7.** Proceed with the normal hierarchical flow for the design.

There is no separate step required for assigning blackbox pins or committing the blackbox.

After the blackbox pins are placed at near-optimal location by running Trial Route with the `-routeBasedBBPin` parameter, use the `assignPtnPin` command to finally place blackbox pins to honor user-specified constraints.

When you partition the design, blackboxes as well as regular partitions are committed.

The following flow is an ECO flow where the contents of the black box are now well defined.

- 1.** Restore the design (or import the design and load a floorplan with the black box information)
- 2.** Run the `loadBlackBoxNetlist` command to incrementally load the netlist for the blackbox. You can run this command without exiting the current session of the Encounter software.
- 3.** Run the `convertBlackBoxToFence` command to convert the blackbox to a fence.

**Note:** To convert the fence back to a blackbox, run the `convertFenceToBlackBox` command.

Continue with the following steps to finalize pin assignment for the black box:

- 4.** Proceed with the normal hierarchical flow for the design.

## Saving Blackboxes

To save blackbox information, use the `saveDesign` command or the *Design – Save Design* menu command.

## Deleting Blackboxes

If a blackbox is an empty module in the netlist, or a cell without a physical macro definition, you must modify the netlist before you can delete it.



*Tip*

You should not delete a blackbox that was originally defined as a macro in the technology file; otherwise, you might have problems with loosely integrated applications because these application interfaces automatically generate only macro definitions for blackboxes. You should only use the delete capability to try out different floorplan.

## Handling of Blackboxes with Non-R0 Orientation

The partitioning- and blackbox-related commands in Encounter support only those blackboxes whose master instances have an R0 orientation. Clones with a non-R0 orientation clones are, however, supported.

Partitioning-related commands such as [assignPtnPin](#), [partition](#), [assembleDesign](#), [flattenPartition](#), [convertBlackBoxToFence](#), and [editPin](#) work only with those blackboxes whose master instances have an R0 orientation.

Several commands in the Encounter software automatically convert the orientation of master blackboxes to R0.

In addition, you can also run the [changeBBoxMasterToR0](#) command to convert the orientation of the master blackboxes to R0. This would be useful for example, you restore a design and want to convert the orientation of all the master blackboxes to R0.

The following sections provide addition information about automatic conversion of orientation and about the [changeBBoxMasterToR0](#) command.

- [Automatic Conversion of Orientation](#) on page 263
- [Performing R0 Transformation](#) on page 264

### Automatic Conversion of Orientation

When the following commands change the orientation of a master instance blackbox to non-R0, the commands automatically convert the new orientation to R0:

- [specifyBlackBox](#)
- [flipInst](#)
- [multiPlanDesign](#)
- [orientateInst](#)

- [placeInstance](#)
- [planDesign](#)
- [rotateInst](#)

In addition:

- opening the Attribute Editor for such a master blackbox automatically converts the orientation to R0.
- using the Flip or the Rotate options from the context menu (the menu that appears when you click the middle mouse button on an object) automatically converts the orientation to R0.
- using the Flip or the Rotate options on the Floorplan toolbox automatically converts the orientation to R0. For more information, see [Floorplan Toolbox](#) in the “Tools Menu” chapter of *Encounter Menu Reference*.

The conversion includes the following:

- Cell blackbox geometries (PORT, OBS, and so on) are transformed.
- Master instances are converted to R0 orientation. The clone instances are oriented accordingly.  
**Note:** The placement location remains unchanged.
- Any pin guides, pin blockages, and pin constraints associated with transformed blackboxes are deleted.

 *Important*

There is no change in the design physically as a result of these transformations. Only the cell orientation and the instance representation are modified.

As an example, if the blackbox master instance is MX, then after the transformation:

- cell geometries are transformed to MX
- The orientation of the master instance is changed to R0.

## **Performing R0 Transformation**

For designs that contain blackboxes whose master instances have a non-R0 orientation, you can use the [changeBBoxMasterToR0](#) command to convert the orientation of the master blackboxes to R0. The syntax of the command is as follows:

## Encounter User Guide

### Partitioning the Design

---

```
changeBBoxMasterToR0 [-checkOnly] [{cellName | cellNameList}]
```

If a cell name, or a list of cell names, is not specified, the command converts the orientation of all the non-R0 master blackboxes to R0.

If the `-checkOnly` parameter is specified, the command does not actually convert the orientation of any master blackbox; it only displays the number of master blackboxes whose orientation would have been changed had the command been run without the `-checkOnly` parameter.

For more information, see the description of the [changeBBoxMasterToR0](#) command in the *Encounter Text Command Reference*.



*Tip*

When you are ready to run a loosely integrated application, complete the following steps:

1. Run the [saveDesign](#) command to make sure that you have updated the size and pin information.
2. Exit the Encounter software, or use the [freeDesign](#) text command.
3. Rerun the Encounter software with the updated macro information.

To delete all the blackboxes in the design, use the [unspecifyBlackBox](#) `-all` command.

## Specifying Multiple Instantiated Partitions and Blackboxes

When a module with multiple instantiations (also known as repeated partitions) of the same cell type is assigned to become a partition, you can specify either one of the multiple instantiated hierarchical instances to be partitions. The name of a hierarchical instance used for partition specification becomes the master partition, and the other instantiations are clones of this master partition.

**Note:** All the master and clone hierarchical instances should be placed inside the core before you specify the partition. This restriction does not apply to blackboxes.

When working with repeated partitions, you should be aware of the following:

- You can only specify one instance as a master partition. The Encounter software will treat the other instances as partition clones.

- For the top-down hierarchical flow, where the top-level design is implemented first, the instance must have a R0 orientation; otherwise, you will run into problems with the pin assignment, feedthrough buffer insertion, and commit partition steps.
- For the bottom-up hierarchical flow, where the block is implemented first, the partition master can have a non-R0 orientation. Make sure all the non-uniquified instances are placed inside the core before you specify the partition.
- For non-uniquified blackboxes, the Encounter software automatically converts all hierarchical instances of a same module as repeated blackboxes. The hierarchical instance that is first instantiated in the netlist is treated as the master blackbox.
- Partition and blackbox clones can be rotated and flipped if the design only has regular square vias, and flipped if regular vias used in the design are symmetry in the flip directions.
- Partition clones share the same pin assignment and pushed-down data as their partition master, you must run the `alignPtnClone` command before the commit partition step to make sure all the partition clones are well aligned with the master on power mesh so you do not run into problems when flattening the partitions.
- You cannot use inserted partition feedthrough buffers for partitions that have clones. The Encounter software does not support this capability.
- For master and clones partitions, the Encounter software automatically snaps the clone partitions such that clones will have the same row structure and pattern as their master. To disable this snapping capability, use the `-noEqualizePtnHInst` option of the `loadFPlan` command.

## Changing Partition Clone Orientation

After specifying the partition, you can change the partition clones' orientation by using the `setClonePtnOrient` command or through Attribute Editor during floorplanning.

For routing purposes, the Encounter software automatically stitches regular wires and rotates vias correctly for non-R0 orientations, such as MX, MY, R90, R180, and R270.

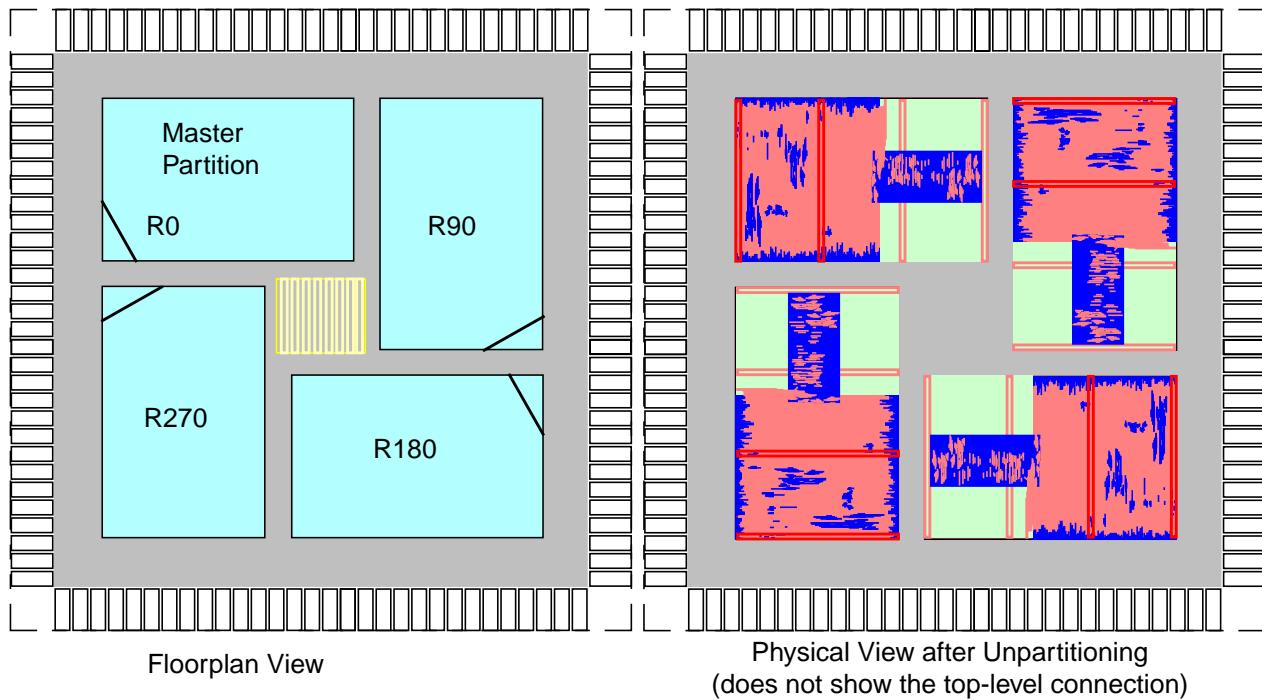
For example, there is a case where some of the clones follow the orientation of the master instance (R0), and some are placed with R180 orientation. After chip assembly, the Encounter software flips and places the clone instances' standard cells to match the R180 clone orientation, and repositions the routing according to the R180 orientation.

Because R90 and R270 orientation clones have vertical rows, all the cell placement, routing, and IPO should be done at the top-level before flattening step. After flattening the design, you should only run full chip flat timing analysis.

## Encounter User Guide

### Partitioning the Design

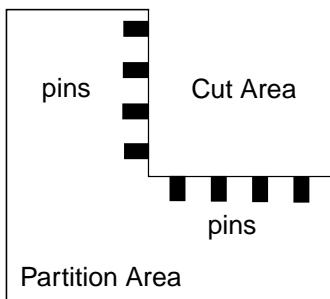
The following example shows a design that has R90, R180, and R270 orientation clones:



**Note:** The illustration above only shows the wire information inside the partition, and does not include the top-level connection.

## Specifying Rectilinear Partitions and Blackboxes

You can specify a rectilinear (non-rectangular) partition shape by adding a cut area. The partition's cut area will have no cell placement and no routing. Pins are assigned to the rectilinear partition edges, as shown in the following figure:



The rectilinear pin assignment recognizes the rectilinear edges when assigning pins, and supports any rectilinear shape. See [Assigning Pins on Rectilinear Edges](#) on page 300 for more information.

To add a cut area to the partition or blackbox, complete the following steps:

1. Click on the *Cut Rectilinear* widget from the *Tools* area.
2. Move the mouse to an edge or corner of the partition or blackbox.
3. Left click and drag over the area.
4. Left click again to complete the cut.

A macro definition file (LEF) will be created with blockage on the overlap layer covering the cut area. For the top-level partition, the cut area allows block or cell placements.

The equivalent text command is `setFPlanObjBoxList` with the *Module* object type. For backward compatibility, you can also use the `createPtnCut` text command. You should specify a module as a partition before using `createPtnCut`.

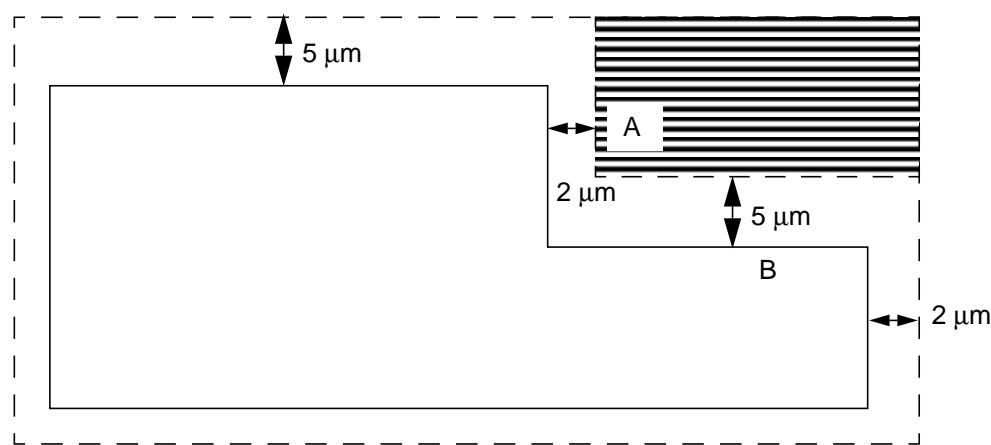
For repeated partitions or blackboxes, when you create a cut on one instance—either master or clone—the cut is applied to the other instances as well.

**Note:** If a cut is made on a blackbox that has pins assigned to it, the affected blackbox pins are automatically moved to the new edge boundary created by the cut.

## Specifying Core-to-I/O Distance for Partition Cuts

Core-to-I/O distance is specified in the Specify Partition form. If the partition has a partition cut, core-to-I/O distance is honored where the cut is specified. The specified top, bottom, left, and right core-to-I/O distances are automatically assigned for the cutting edges that face the north, south, west, and east side, respectively.

For example, if you specify a core-to-I/O distance of 5  $\mu\text{m}$  for the top and bottom, and 2  $\mu\text{m}$  for left and right sides:



The core to I/O distance for the edge A (facing east) should be 2  $\mu\text{m}$ . The core to I/O distance for the edge B (faced to north) should be 5  $\mu\text{m}$ , same as the top side.

## Specifying Nested Partitions

The Encounter software does not support a partition that is nested inside another partition. For nested partitions, you can work around this limitation by specifying the second-level partition at the partition-level design.

For example, consider a case where the module `mult_32` is a nested module inside the module `tdsp_core` and you want to define both `mult_32` and `tdsp_core` as partitions. For this, first define `tdsp_core` as a partition and then follow the normal partition flow to define `mult_32` as a partition. Here are the steps:

1. Import the design.
2. Specify `tdsp_core` as partition.
3. Perform placement and routing.
4. Commit the partition `tdsp_core`.
5. Save the partition.
6. Change to the `tdsp_core` partition directory.
7. Define `mult_32` as a partition.

## Assigning Pins

You can optimize partition and blackbox pins in the Encounter environment based on routing or placement information. You can assign the pins or ports to a location on a partition, and set various constraints as per your requirements on pin assignment, for example, you can create pin blockages on specified areas.

Run the Check Pin Assignment menu command (*Partition – Check Pin Assignment*) or the `checkPinAssignment` text command after pin assignment to make sure that all pins are assigned, are placed on routing grids, and are not overlapping.

Blackbox pins are assigned in the same way as partition pins.

Pin assignment supports the following:

- Rectilinear partitions and black boxes

## Encounter User Guide

### Partitioning the Design

---

- Repeated partitions and black boxes. Both master and clones are considered when assigning their pins.
- Non-uniform tracks.

**Note:** Pin assignment assigns only signal pins but it does honor power/ground stripes and followpins. Power and ground pins are created when the design is partitioned.



You cannot assign blackbox or partition pins when design is unplaced and unrouted. Make sure you place the design before partitioning; otherwise, all pins will be unplaced.

The following sections describe pin assignment in Encounter:

- [Assigning Partition and Blackbox Pins](#) on page 271
- [Assigning I/O Pins](#)
- [Performing Congestion-aware Pin Assignment for Channel-based Designs](#)
- [Assigning Pins on Rectilinear Edges](#)
- [Swapping Partition Pins](#)
- [Snapping Pins to the Grid](#)
- [Assigning Pins for Bus Guides](#) on page 303
- [Pin Assignment Limitations](#)

## Assigning Partition and Blackbox Pins

Assigning pins for partitions and blackboxes includes the following steps:

- [Setting Pin Constraints](#) on page 271
- [Assigning Pins](#) on page 281
- [Validating Pin Placement Results](#) on page 284
- [Refining Pin Assignment and Fixing Pin Violations](#) on page 288
- [ECO Pin Assignment](#) on page 291

### Setting Pin Constraints

The Encounter software provides a number of constraints to control or guide partition, blackbox, or I/O pin assignment:

- [Pin Group](#)
- [Net Group](#)
- [Pin Guides](#)
- [Pin Size \(Width and Height\)](#)
- [Pin Spacing](#)
- [Pin Layers](#)
- [Pin-to-corner distance](#)
- [Pin Blockage](#)

#### ***Pin Group***

While assigning bus pins or signal pins that you want to be placed together, you can specify a constraint for these pins by creating a cell pin group. You can create a cell pin group with the the [createPinGroup](#) text command or by using the [Edit Pin Group](#) GUI form (Edit – Object – Pin Group). You can add pins to a cell pin group with the [createPinGroup](#) text command or with the [addPinToPinGroup](#) text command.

Cell pin groups do not have to be associated with a partition pin guide because a pin group is not a constraint on any partition edge. In this case, the pin assignment program can freely place this group of pins on any edge of the partition. However, pins that belong to this pin group are still placed together in adjacent locations.

With a pin group you can:

- Optimize order of pins within a cell pin group to improve wire length using the `-optimizeOrder` option of the `createPinGroup` text command. If this option is not specified, the pin order is exactly as specified in the pin group.
- Place pin members of a pin group on alternate layers using the `-alternateLayer` parameter of the `createPinGroup` text command.
- Specify pin spacing. The default minimum pin spacing between pins of a cell pin group is two tracks.

The following commands create a pin group `pGroup1` that consists of 3 INT bus bit pins of the module ALU. These pins can be optimized within the pin group:

```
createPinGroup pGroup1 -cell ALU -pin {INT[0] INT[2] INT[3]} -optimizeOrder
```

Or

```
createPinGroup pGroup1 -cell ALU -optimizeOrder  
addPinToPinGroup -cell ALU -pinGroup pGroup1 -pin {INT[0] INT[2] INT[3]}
```

Use the `deletePinGroup` command to delete a pin group or all pin groups.

Use the `deletePinFromPinGroup` command to delete a pin from a pin group.

### **Net Group**

You can create a net group using the `createNetGroup` command or by using the Edit Net Group GUI form (Edit – Object – Net Group). You can specify net members when creating a net group or add them later using the `addNetToNetGroup` command. To be honored by pin assignment, net groups must be used in conjunction with a pin guide.

As for a pin group, you can optimize net pin order, alternate pin layers, and specify pin spacing for a net group.

The following commands create a net group `nGroup1` that has two nets `NET1` and `NET2` with minimum pin spacing of 2 tracks.

```
createNetGroup nGroup1 -net {NET1 NET2} -spacing 2
```

Or

```
createNetGroup nGroup1 -spacing 2  
addNetToNetGroup nGroup1 -net {NET1 NET2}
```

Use the `deleteNetGroup` command to delete a net group or all net groups.

**Note:** When you delete a net group, any bus guide associated with the net group also gets deleted.

Use the `deleteNetFromNetGroup` command to remove a net from a net group.

### **Pin Guides**

You can create a pin guide to constrain a bus, net, pin, net group, or pin group to be placed in specific areas. A pin guide is used for specifying a physical guided area where pins belonging to the pin guide will be placed.

**Note:** While creating a pin guide, you cannot optimize the order of pin members or specify minimum spacing. If you want to control the pin order and the pin spacing of the members of a pin guide, first create a net group or a pin group and associate this net group or pin group with a pin guide.

A pin guide can support multiple constraint pin layers. In addition, any bus, net, pin, net group, or pin group can be assigned to multiple pin guides.

You can create a pin guide using the `Add Partition Pin Guides` widget from the GUI or through the `createPinGuide` text command. A physical location constraint can be specified either as a rectangular area or as an edge constraint. If you specify a physical location constraint as an edge constraint, you will also need to specify the partition/black box cell name.

Here are a few examples of using the `createPinGuide` text command to create a pin guide.

**Example 1:** The following command creates a pin guide for a net group `nGroup1`. The pin order within this net group will be optimized. The pin members of this pin guide can be placed on the top edge of the cell `ALU`. Pins will be placed on `M2` or `M4` layers:

```
createNetGroup nGroup1 -net {NET1 NET2} -optimizeOrder  
createPinGuide -netGroup nGroup1 -edge 1 -cell ALU -layer {2 4}
```

**Example 2:** The following command creates a pin guide for a pin group `pGroup1` of cell/module `ALU`. Pins of this pin guide will have a minimum spacing of 2 tracks:

```
createPinGroup pGroup1 -cell ALU -pin {INT[0] INT[2] INT[3]} -spacing 2  
createPinGuide -area 678.52 371.25 778.53 787.33 -pinGroup pGroup1 -cell ALU
```

The pins will be assigned on the preferred layers.

**Example 3:** The following command creates a pin group `pGroup2`. This pin group can be placed on the top edge or the right edge of the cell `TDSP`. For top edge, pins can be assigned on the `M4` or `M6` layers. For right edge, pins can be assigned on the `M5` layer.

```
createPinGroup pGroup2 -cell TDSP -pin p_addr* -optimizeOrder  
createPinGuide -edge 1 -pinGroup pGroup2 -cell TDSP -layer {4 6}
```

## Encounter User Guide

### Partitioning the Design

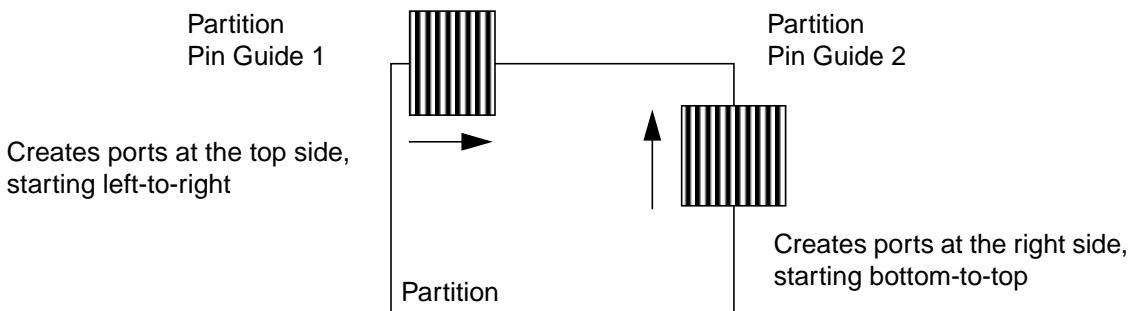
```
createPinGuide -edge 2 -pinGroup pGroup2 -cell TDSP -layer 5
```

You can also use the GUI to create a partition pin guide, as follows:

After you have determined a pin guide location in the design display area, you can create a partition port for a net or bus name and add a partition pin guide. To add a partition pin guide through the GUI, complete the following steps:

1. In the *Tools* area, click the Add Partition Pin Guides widget.
2. Press the F3 key to bring up the *Select Pin Guide Options* GUI form. Use this form to specify the pin guide name, cell name, mode (by area or by edge), and the applicable layers.
3. Click and drag over a partition fence overlap to specify the area or edge.

For vertical edges, the first pin generated starts from the bottom intersect point. For horizontal edges, the first pin generated starts from the left intersect point, as shown in the following figure:



The default pin spacing is 2, which places one pin for every two metal tracks. You can change the pin spacing with the *Minimum Pin Pitch* field in the Specify Partition form, or by changing spacing of the associated pin group or net group. You can use the Move/Resize/Reshape tool to modify the pin guide location.

**Note:** For a partition that has a rectangular cut, the partition pin guide must be placed on the edge of the cut. You can also use a pin guide to assign pins, net group, or a pin group to a specific side without specifying a pin guide area by using the createPinGuide command.

4. Change the partition pin guide object name to the net, bus, or net group name.

Use the partition pin guide attribute editor to change pin guide name to a net name, or the name of a predefined net group or pin group.

If the partition pin guide was assigned the net group name, all nets and buses added to this net group name will have partition pins generated for the partition. Pins are generated in the order the net or bus was entered by the [addNetToNetGroup](#) command. Pins for unconnected nets and buses are randomly assigned. You can also use the partition pin guide to assign floating pins.

Use the [deletePinGuide](#) command to delete a pin guide or all pin guides.

### ***Pin Size (Width and Height)***

By default, pin size will be created based on the minimum area rule. Use the [setLayerPinWidth](#) and [setLayerPinDepth](#) commands to set new pin width and depth of a routing layer for a specific partition/black box cell. When this constraint is defined, pin assignment will use these values for creating pin size.

You can also specify pin size for a specific pin or pin group using the [setPinWidth](#) and the [setPinDepth](#) commands.

Use the [getLayerPinWidth](#) and the [getLayerPinDepth](#) commands to retrieve pin width and depth for particular layer(s) of specific partition/black box cell.

Use the [getPinWidth](#) and the [getPinDepth](#) commands to retrieve width and depth of a specific pin or pin group.

Example 1: The following commands set the pin width and depth of layer M2 for partition cell ALU to 0.4 and 0.6 respectively.

```
setLayerPinWidth -cell ALU -layer 2 -width 0.4  
setLayerPinDepth -cell ALU -layer 2 -depth 0.6
```

Example 2: The following commands set the pin width of pin group pGroup1 to 0.3 and pin depth of pin pGroup1 to default.

```
setPinWidth -cell ALU -pinGroup pGroup1 -width 0.3  
setPinDepth -cell ALU -pinGroup pGroup1 -default
```

With this example, all the pins of pin group pGroup1 will have the width 0.3 and the default depth.

### ***Pin Spacing***

You can set minimum pin spacing in terms of track number using the [Specify Partition](#) form (*Partition – Specify Partition*). The default pin spacing is 2, which places a pin for every two metal tracks.

## Encounter User Guide

### Partitioning the Design

---

You can modify the pin spacing in the following ways:

- Global pin spacing at design level

Use the `setGlobalMinPinSpacing` and the `getGlobalMinPinSpacing` commands to set and retrieve global pin spacing. This spacing value will be applied to all partition/black box pins of the design.

- Partition/black box level

Use the `definePartition` command with `-minPitchTop`, `-minPitchBottom`, `-minPitchLeft`, and `-minPitchRight` parameters to specify minimum pin spacing for a partition.

- Specific partition/black box edge

Use the `setMinPinSpacingOnEdge` and the `getMinPinSpacingOnEdge` commands to set and get the minimum pin spacing for a particular edge or all edges of a partition/black box cell.

The `-edge` parameter of the `setMinPinSpacingOnEdge` and `getMinPinSpacingOnEdge` commands can take the following values:

- N, S, W, E (supports both upper and lower case)
- T, B, L, R (supports both upper and lower case)
- dbcN, dbcS, dbcE, dbcW

**Example1:** The following commands set the minimum pin spacing for top and bottom edge of partition cell ALU to 1 track.

```
setMinPinSpacingOnEdge -cell ALU -edge T -spacing 1  
setMinPinSpacingOnEdge -cell ALU -edge B -spacing 1
```

**Example 2:** The following command sets minimum pin spacing for all edges of partition cell TDSP to 3 tracks

```
setMinPinSpacingOnEdge -cell TDSP -all -spacing 3
```

- Pin group or net group

Use the `createPinGroup` or the `createNetGroup` commands to specify minimum pin spacing at the pin group or net group level. This specified minimum pin spacing will apply to all the pin members of the specified pin group or net group.

- Pin level

Use the `setPinConstraint` command to specify minimum pin spacing of a particular pin.

As spacing constraint can be specified at more than one level, pin assignment will honor spacing constraint in the following order:

- Pin spacing
- Net group or pin group spacing
- Partition/black box spacing on a particular edge
- Partition/black box spacing
- Global spacing

### ***Pin Layers***

Specify pin layers that will be used for placing pins on a specific partition side using the Specify Partition form (*Partition – Specify Partition* menu command). The equivalent text command is setAllowedPinLayersOnEdge.

You can specify layer constraints at partition level, pin guide level, or pin level.

- Partition level

Layer constraint per edge can be specified at partition level using either

- the Specify Partition form (*Partition – Specify Partition* menu command), or
  - the definePartition command with `-pinLayerTop`, `-pinLayerBottom`, `-pinLayerLeft`, and `-pinLayerRight` parameters. These layer constraints will be applied to all pins on a particular edge of the specified partition.
  - the setAllowedPinLayersOnEdge command with the `-layer` and `-edge` options. This command

- Pin guide level

Use the `-layer` parameter of the createPinGuide command to specify layer constraints for all pin members of a pin guide. Layer constraint at pin guide will override the layer constraint at partition level.

- Pin level

Use the `-layer` parameter of the setPinConstraint command to specify layer constraint for a specific partition/black box pin.

Layer constraint at pin level will have higher priority than layer constraint at partition level.

If a layer constraint is applied to a pin that also belongs to a pin guide, the pin guide layer constraint will have higher precedence.

If a layer constraint is being applied to a pin that already belongs to a pin group or net group, the layer constraint will not be applied. To apply layer constraint for this pin, first remove this pin from the pin group or net group, and then apply the pin layer constraint.

### ***Pin-to-corner distance***

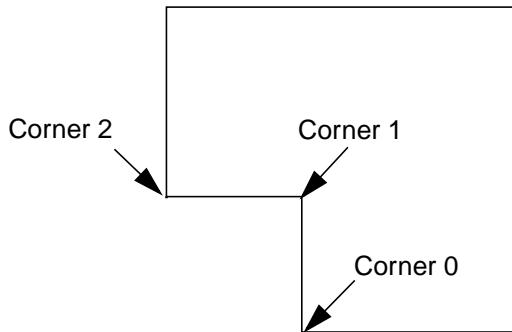
To keep pins away from partition/black box corners, you can set the pin-to-corner distance constraint.

Use the [setPinToCornerDistance](#) command to set pin to corner distance for a particular corner or all corners of a specific cell.

Use the [getPinToCornerDistance](#) command to retrieve the pin-to-corner value of a cell-specific corner or all corners.

Use `setPinToCornerDistance -cell *` to set global pin-to-corner distance that will be applied to all partition and blackboxes in the current design. The default value is 5 routing tracks.

The `-corner cornerNumber` parameter of the commands specifies the corner of the partition block. This is an integer value, where corner numbering starts at 0 from the lower-left corner of a partition clock-wise. Corner 0 is the corner that has the smallest y value.



Example: The following command sets pin-to-corner distance for corner 0 and corner 2 of the cell ALU to 8 routing tracks.

```
setPinToCornerDistance -cell ALU -corner 0 8  
setPinToCornerDistance -cell ALU -corner 2 8
```

### **Pin Blockage**

After determining the partition pin blockage point, you can block that area from assigning pins on specific metal layers. Pin assignment engines also honor regular routing blockages if they intersect with partition edges.

You can create pin blockages with the Add Partition Pin Blockage widget or by using the [createPinBlkg](#) command.

**Note:** Trial Route does not honor the partition pin blockage.

To create the partition pin blockage with the [Add Partition Pin Blockage](#) widget, complete the following steps:

1. Click the [Add Partition Pin Blockage](#) widget from the Tools area
2. Left click and drag over a partition fence overlap.
3. Use the Attribute Editor to specify the metal layers to block.

The following command creates a pin blockage for the entire left edge of cell TDSP on layer M5.

```
createPinBlkg -edge 0 -cell TDSP -layer 5
```

If the `-layer` option is not specified, the pin blockage will be created on all partition/black box reserved routing layers.

Use the [deletePinBlkg](#) command to delete a pin blockage or all pin blockages (`deletePinBlkg -all`).

### **Performing Pin Pre-Assignment**

You can pre-assign a pin before pin assignment using the [Pin Editor](#) or the [editPin](#) text command. These pre-assigned pins will have *fixed* placement status so pin optimizers will honor them. For more details, see the [Pin Editor](#) section in the “Edit Menu” chapter of the *Encounter Menu Reference*.

### **Setting Constraints on a Specific Pin**

Use the [setPinConstraint](#) command to specify the following constraints for a particular pin:

- Physical location

A pin can be constrained by specifying its coordinate (x, y) location and its preferred routing layer. If specified location is not on valid track, the pin will be snapped to the closest location. To keep the pin on non-preferred routing layer or to not snap the pin, use the [editPin](#) command instead.

Besides an actual physical location, a pin can also be constrained to a particular edge.

- Layer
- Spacing

For example, the following command specifies that the pin `reset` of partition cell `mult_32` should be placed on top edge with either `M5` or `M7` routing layer.

```
setPinConstraint -cell mult_32 -pin reset -edge 1 -layer {5 7}
```

For setting pin size constraint for a specific pin use the [setPinWidth](#) and [setPinDepth](#) commands.

The following salient points apply to setting the pin constraints for a specific pin:

- If constraints are applied to a pin that also belongs to a pin guide, the pin guide constraint will have higher precedence.
- If a location and/or layer constraint is being applied to a pin that already belongs to a pin group or a net group, the constraint will not be applied. To apply location and/or layer constraint for this pin, first remove this pin from the pin group or net group, and then apply the pin constraint(s).
- If a pin with layer constraints defined is added to a net group or pin group, the pin cannot be added to a pin group or a net group with the [createPinGroup](#), [createNetGroup](#), [addPinToPinGroup](#), or [addNetToNetGroup](#) commands because the pin has already been constrained. To add this pin to a pin group or net group remove the constraints first (using the [unsetPinConstraint](#) command).
- If the following constraints cannot be met during pin assignment, the Encounter software will issue a warning message and the constrained pins will be placed at the lower-left corner of the partition/black box with unplaced placement status:
  - Pin constraint
  - Pin group constraint
  - Net group constraint

Use the [unsetPinConstraint](#) command to remove constraint settings for a specific pin.

## Assigning Pins

There is no separate step required for assigning black box pins. To assign pins, use the Partition – Assign Pins GUI menu or the `assignPtnPin` text command.

Pin assignment supports the following:

- Rectilinear partitions and black boxes
- Repeated partitions and black boxes. For designs that have a master blackbox as well as its clones, the pin assignment for the blackbox clones is based on the master.
- Non-uniform tracks

Pin assignment assigns signal pins but it does honor power/ground stripes and followpins. Power and ground pins will be created during the partition step.

### ***Placement-based Pin Assignment***

Pin assignment is based on connectivity flightlines. Cell placement should be performed before running pin assignment.

### ***Route-based Pin Assignment***

For route-based pin assignment, routing should be performed prior to the `assignPtnPin` command. Routing cross points with partition/black box boundary will be used as guidance for pin assignment.

For a design that has blackboxes, if you want to have near-optimal locations for black box pins, the `-routeBasedBBPin` option should be used. The differences between Trial Route with and without `-routeBasedBBPin` options are as follows:

Default Trial Route performs the following:

- Assigns initial black box pins based on connectivity
- Creates temporary routing blockages over black boxes based on black box reserved routing layers
- Runs trialRoute to route to black box pins
- Removes temporary blockages

Trial Route with the `-routeBasedBBPin` parameter performs the following:

- ❑ Shrinks black box boundary and runs connectivity based pin assignment to get initial pin location
- ❑ Runs partition-aware routing (`ptnAwareRouteForPA`)
- ❑ Re-adjusts black box pins to actual black box boundary based on routing cross point
- ❑ Creates temporary routing blockages
- ❑ Runs trialRoute to route to black box pins
- ❑ Removes temporary blockages

For channel-based designs that are congested or has thin channels, it is recommended to run `trialRoute -fastRouteForPinAssign`.

However, if the design has black boxes then you can run Trial Route with –  
`routeBasedBBPin` and `-handlePartitionComplex` options.

### **Tips for Assigning Partition Pins**

For most of the designs, running the `assignPtnPin` command without any option should give a reasonable result. However specific options can provide better results in some cases. These options are described here:

- `-maxPinMovementForAlign` and `-skipPinRefine` parameters

If you have ran partition aware routing (`trialRoute -fastRouteForPinAssign` or `trialRoute -handlePartitionComplex`) for pin assignment, you should use these parameters to minimize pin movement from existing routing cross points because these routing cross points should give near-optimal pin locations.

Example:

```
trialRoute -fastRouteForPinAssign  
assignPtnPin -maxPinMovementForAlign 20 -skipPinRefine
```

- `-ptn ptnName -pin pinList` parameter

Use this parameter for running incremental pin assignment or assigning specific pins of specific partitions.

This parameter can be used in the following pin assignment scenarios:

- ❑ When you want to assign critical pins first and then assign the rest of partition and/or black box pins.

- First, run pin assignment to assign these critical or specific pins. Use the above option in conjunction with the `-markFixed` parameter so these pins will not be moved by second pin assignment run.
- Run pin assignment again to assign the rest of the pins.

Example:

```
assignPtnPin -ptn tdsp_core_glue -pin {p_address[0] p_address[3]} -ptn  
alu_32 -pin rom_data* -markFixed  
assignPtnPin
```

In the previous example, if you are running routed based pin assignment, you should run `trialRoute` between the first and the second pin assignment run so that the routing that will be used for the second pin assignment is based on pin locations of the first pin assignment step.

- Run incremental pin assignment

This scenario is in contrast to the first scenario where you would run pin assignment for all partition and/or black box pins, and then further re-optimize some specific pins.

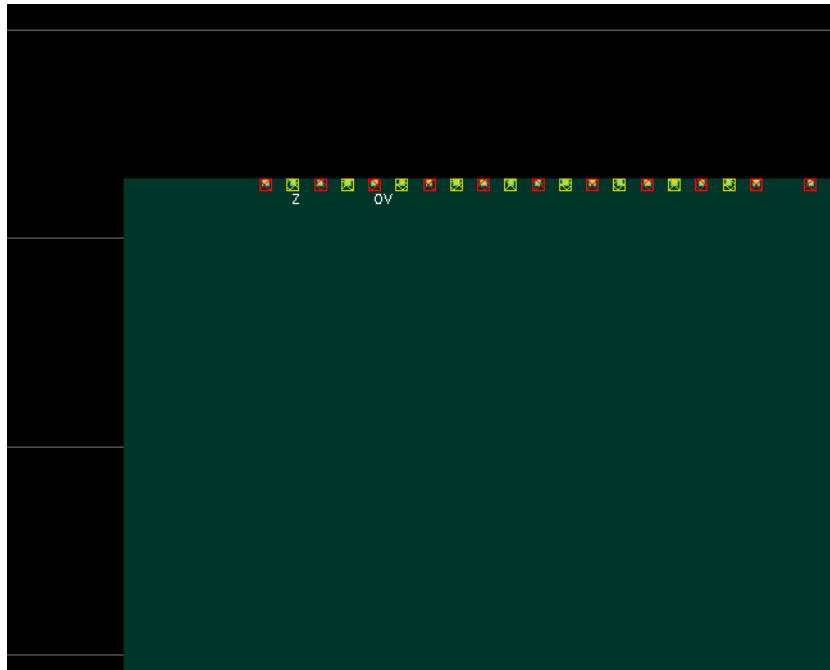
Example:

```
assignPtnPin  
assignPtnPin -ptn mult_32 -pin {reset addr*}
```

If `reset` and all `addr` pins of the partition `mult_32` have fixed placement status, you should also use `-moveFixedPin` option; otherwise pin optimizer will not move fixed pins.

■ `-noPinLayerOverlap` parameter

Use this parameter if you do not want the pin optimizer to place signal pins overlapping each other on different layers. Further, if the minimum pin spacing is 1, the adjacent pins will be placed on the alternate layers (in other words, adjacent pins will not be placed on the same layer). This option can be used to avoid DRC violations between adjacent pins and the routing connecting to these pins.



The previous figure shows that adjacent pins are placed on alternate layers M2 and M4.

- **-enforceRoute parameter**

With this parameter, pin assignment completely follows the routing information without honoring any user-specified pin constraints and pin locations may not be legal. This option should only be used for a rough pin assignment or for comparing pin locations based purely on routing result with pin locations that are legalized. If you want to use this pin placement result for your implementation stage, you need to run the `legalizePin` command after the `assignPtnPin` command to legalize them.

## Validating Pin Placement Results

You can perform the following steps to validate and correct pin placement results:

- [Checking the Pin Legality](#) on page 285
- [Reporting QoR of Pin Assignment](#) on page 285
- [Adjusting Pins](#) on page 288
- [Aligning Partition Pins](#) on page 289
- [Running incremental Pin Assignment](#) on page 290

- [Adjusting Floorplan or Floorplanning the Design Again](#) on page 290
- [Performing Pin Assignment Again](#) on page 290

### ***Checking the Pin Legality***

Use the [checkPinAssignment](#) command to make sure that pins are legalized (for example, the pins snap to routing grid, are on reserved routing layers, honor user-specified constraints, do not cause any DRC violations, and so on).

You can check

- All partition/black box pins

Example: The following command checks all partition/black box pins in the current design and saves the result into the output file `pinLegality.rpt`.

```
checkPinAssignment -outFile pinLegality.rpt
```

- All pins of a specific partition

Example: The following command checks all pins of the partition TDSP\_CORE

```
checkPinAssignment -ptn TDSP_CORE -pin *
```

- Specific partition pins

Example: The following command checks all bus pins `p_addrs` and `rom_data` of the partition TDSP\_CORE

```
checkPinAssignment -ptn TDSP_CORE -pin {p_addrs* rom_data*}
```

**Note:** You can use the `-verbose` parameter of the [checkPinAssignment](#) command to print detailed pin-specific information for each reported violation. You can also exclude certain checks, for example, checks related to pin spacing violation or pin layer violation. For more information, see the description of the [checkPinAssignment](#) command in the *Encounter Text Command Reference*.

### ***Reporting QoR of Pin Assignment***

You can use the [pinAnalysis](#) command to report certain Quality of Results (QoR) metrics for pin assignment. The [pinAnalysis](#) command deletes the existing routes, reroutes the design ensuring that the routes pass through partition pins, and reports pin assignment QoR metrics.



The `pinAnalysis` command creates new routes. The original routes are not retained.

**Note:** The `pinAnalysis` command reroutes the design using `trialRoute -honorPin`. This command thus takes at least as much time as running Trial Route. Also, because Trial Route is run, you can use the generated routing information for other applications such as time budgeting.

When you run the `pinAnalysis` command after assigning pins but before committing the partition, the following are reported:

- Pin-deviation from routing cross-points
- Estimated net-length and via-count
- Comparison between net lengths, via counts, and congestion indexes of the original and the new routes

**Note:** *Original route* refers to the routing that was performed before the pin assignment step. *New route* refers to the routing performed by the `pinAnalysis` command.

- Number of two-pin nets that have aligned pins and number of two-pin nets that have unaligned pins
- CPU time and memory usage

When you run the `pinAnalysis` command after committing the partition, the following are reported:

- Estimated total top-channel net-length and via-count
- Congestion report with overall congestion index values for top-level channel nets
- Number of two-pin nets that have aligned pins and number of two-pin nets that have unaligned pins
- Run time and memory usage

You can check the legality of pin assignment (similar to the functionality of `checkPinAssignment` command) by specifying the `-checkLegality` parameter with the `pinAnalysis` command.

You can also save the output of the command in a text file by specifying the `-outFile` parameter.

## Encounter User Guide

### Partitioning the Design

In addition to displaying the report on screen, this command also generates the report in an HTML file named `pinAnalysis.html`. The legality details for every partition are available through hyperlinks in the HTML report file.

As an example, the following command checks the legality of pin assignment, displays the QoR report for pin assignment on the screen, and also prints the QoR report to a file named `pinAnalysisMetricReport`.

```
pinAnalysis -checkLegality -outFile pinAnalysisMetricReport
```

The output of the previous command is similar to the following:

```
Analysing Pin Assignment.....
```

```
=====
```

Pin Legality Report:

	Partition Name	Total Pins	Internal Pins	Unplaced Pins	Legal Pins	Illegal Pins	
	results_conv	39	0	39	0	0	
	tdsp_core	114	0	114	0	0	
	dtsmf_chip	0	0	0	0	0	

```
=====
```

Pin QoR Report:

```
-----
```

Unaligned 2-pin Net: DTMF\_INST/t\_addrs[0].

Unaligned 2-pin Net: DTMF\_INST/m\_clk.

There are 2 unaligned 2-pin nets.

```
=====
```

```
-----| QoR Metric | Before Pin-Assignment | After Pin-Assignment | Percent Increase |-----
```

## Encounter User Guide

### Partitioning the Design

Horizontal Congestion	0.000%	0.032%	NA
Vertical Congestion	0.000%	0.697%	NA
Total Net-Length	6.089e+05um	6.150e+05um	1.011%
Total Via-Count	48300	54036	11.876%

Completed pinAnalysis (CPU=0:00:07.6 MEM=5.9)

**Note:** The internal pins (shown in the Legality Report) are not checked for legality. Internal pins are the pins that are not on the partition boundary.

In the previous table, the `Before Pin-Assignment` column shows the results of the routing before pin assignment and the `After Pin-Assignment` column shows the results of the routing after pin assignment.

### Refining Pin Assignment and Fixing Pin Violations

After assigning partition or blackbox pins, you can further refine the current pin assignment and fix any pin violations using one or more of the following methods:

- [Adjusting Pins](#)
- [Aligning Partition Pins](#)
- [Running incremental Pin Assignment](#)
- [Adjusting Floorplan or Floorplanning the Design Again](#)
- [Performing Pin Assignment Again](#)

These steps are explained in the following sections.

#### ***Adjusting Pins***

You can Adjust pins using the [Pin Editor](#) or the `editPin` text command. You can also use direct pin manipulation to manually move selected pins to different locations.

### **Aligning Partition Pins**

You can align partition pins with other block pins (using the Pin Editor or the pinAlignment text command).

The pinAlignment command can be used to align partition/black box pins with or without specified reference object(s). Reference objects can be hard macros, blackboxes, I/O pads, and standard cells.

You can use the pinAlignment command in different ways to align pins:

- Align specific pins with the specified referenced object

```
pinAlignment -refObj <refInstName> -ptnInst <instName> -pinNames <pinList>
```

- Align all pins of specified partition/blackbox instance that connect with the specified reference object

```
pinAlignment -refObj <refInstName> -ptnInst <instName>
```

- Align all pins of every partition/blackbox that connects with the specified reference object

```
pinAlignment -refObj <refInstName>
```

- Align specific pins of specified partition/blackbox instance

```
pinAlignment -ptnInst <instName> -pinNames <pinList>
```

- Align all pins of specified partition/blackbox

```
pinAlignment -ptnInst <instName>
```

- Align all possible partition/blackbox pins

```
pinAlignment
```

If the referenced object is not specified, the pinAlignment command will automatically derive referenced object based on connectivity information. If the aligned pin has multiple connections, the referenced object will be derived based on the following priority:

- Hard macro pin
- I/O pad pin or I/O pin
- Partition pin
- Standard cell pin

By default an aligned pin will:

- be snapped to routing grid. Use -noSnap option if you want that pins should not be snapped.

- have the same layer routing with the referenced pin. Use the `-keepLayer` option to keep existing aligned pin layer. Use the `-newLayer` option to assign new layer for aligned pin.
- not be legalized. Use the `-legalizePin` option to legalize aligned pin(s).
- have a fixed pin status. Use the `-markPlaced` option to assign placed status to aligned pin(s).

### ***Running incremental Pin Assignment***

Based on the current pin assignment result, re-run pin assignment. You can specify pin constraints to further guide new pin placement.

If you want to reoptimize only a few specific pins, use the `-ptn` and the `-pin` options of the `assignPtnPin` command to specify the list of pins that will be reassigned.

Example: The following command reoptimizes address bus bit pins, rom\_data bus bit pins of partition ALU, and reset pin of partition ARB:

```
assignPtnPin -ptn ALU -pin {address* rom_data[*]} -ptn ARB -pin reset
```

By default, `-ptn` and `-pin` options will not reassign specified pins if they have fixed status. Use the `-moveFixedPin` option with the `-ptn` and `-pin` options to force specified fixed pins to be reassigned.

However if you want to keep only a few existing pins and reoptimize the rest of the pins, instead of specifying many pins, you can mark these existing pins to fixed placement status using the `setPtnPinStatus` command and then re-run pin assignment (without using `-ptn` and `-pin` options):

```
setPtnPinStatus <partitionName> <pinName> fixed  
assignPtnPin
```

### ***Adjusting Floorplan or Floorplanning the Design Again***

Sometimes a sub-optimal floorplan can also lead to a bad pin placement result. If this is the case, re-adjust the floorplan and run pin assignment again.

### ***Performing Pin Assignment Again***

Perform pin assignment again. If the partitions or blackboxes have been committed, use the `flattenPartition` command to unassign the pins. If the partitions or blackboxes are not yet committed, use the `setPtnPinStatus` command to unplace the pins.

## **ECO Pin Assignment**

The Encounter software provides incremental or ECO pin assignment capability. This capability can be used in the ECO flow where partition/black box ports in the original netlist get modified (added/deleted). In this flow, you can preserve most of the existing partition/black box pin locations and let the software assign the newly added pins.

### ***General Flow***

1. Import design.
2. Floorplan design (specify partition/black box information in this step).
3. Run placement.
4. Route design.
5. Save full chip floorplan and/or save design for later use.
6. Assign pins ([assignPtnPin](#)).
7. Save partition/black box pin information into a partition file ([savePtnPin](#)).
8. Route design to connect to new partition/black box pins ([trialRoute -honorPin](#)).
9. Derive timing budget ([deriveTimingBudget](#)).
10. Commit partitions/black boxes ([partition](#)).
11. Save top and partition information into each directory ([savePartition](#)).

After having new modified netlist, ECO pin assignment can be run as follows:

12. Import design with new modified netlist.
13. Load full-chip floorplan that saved in the previous step 5.
14. Place and route the design. Placement and routing information that are saved in the step 5 can be restored if still applicable.
15. Use the [loadPtnPin](#) command to load the partition file that was saved in the previous step 7 or the partition file (or DEF file) of each partition block to preserve existing partition/blackbox pin locations. Make sure that partition/blackbox pins in partition file have fixed placement status so they will not be moved in the next step, pin assignment.
16. Run pin assignment to assign the newly added pins.

### **Saving the Partition Pins**

Use the savePtnPin command to save pin placement information for later use. The command provides options to save pin information of

- Specific partition/blackbox

Example: Save pin locations of partition execute\_i into file execute\_i.ptn

```
savePtnPin -ptn execute_i execute_i.ptn
```

- All partitions and/or blackboxes

Example: Save pin information of all partitions and/or black boxes in the current design

```
savePtnPin -all allPtnPin.ptn
```

- Current block-level design

Example: Save I/O pin locations of the current design

```
savePtnPin -design ioPin.ptn
```

### **Restore Partition Pin Information**

Use the loadPtnPin command to restore/load pin placement information of a particular partition/blackbox. The command restores the following:

- A partition file that is generated by the `savePtnPin` or the `saveFPlan` (`floorplan.fp.ptn`) commands

Example: Load pin locations of the partition ALU from partition file ALU.ptn

```
loadPtnPin -ptnName ALU -inFile ALU.ptn
```

- Block-level DEF file

Example: Load pin locations of partition ALU from ALU DEF file

```
loadPtnPin -ptnName ALU -def ALU.def
```

## Assigning I/O Pins

For a top-down hierarchical flow, I/O pins of a block-level design will normally be assigned during the full-chip pin assignment. This pin placement information is saved in a block-level floorplan partition file (`floorplan.fp.ptn`) or a DEF file that is generated by the `savePartition` command.

For a bottom-up hierarchical flow, I/O pin placement can be generated from an I/O constraint file or during the cell placement step.

You can also explicitly run I/O pin assignment with the `assignIOPins` command.

This section covers the following topics:

- [Setting Pin Constraints](#) on page 293
- [Performing Initial Pin Assignment](#) on page 293
- [Refining Pin Placement](#) on page 294
- [Validating Pin Placement](#) on page 295

### Setting Pin Constraints

The Encounter software provides a number of pin constraint commands to control or guide I/O pin assignment. The same set of pin constraint commands that are used for setting constraints for partition/blackbox pins can also be used for I/O pins. The only difference is that you do not need to specify the `-cell` option for I/O pins. For more information, see [Setting Pin Constraints](#) on page 271 in the [Assigning Partition and Blackbox Pins](#) section of this document.

### Performing Initial Pin Assignment

For a bottom-up flow, initial pin placement can be generated by any of the following methods:

- Using an I/O constraint file

An I/O constraint file can be read into the Encounter environment during the design import step. Or, you can use the `loadIOPin` command to load a constraint file after netlist had been read in.

An I/O constraint file can be created by manually editing a text file.

For more information about I/O constraint file, see the “Generating the I/O assignment File” section in the “Data Preparation” chapter of the *Encounter User Guide*.

- Randomly assigning I/O pins

You can create an I/O template file with random I/O pin assignment using the following steps. I/O placement is evenly distributed on design boundary:

- a. Import design
- b. Run the `saveIoFile` command with the `-template` option
- c. Use the `loadIoFile` command to load I/O file generated from the step 2

- Placing the design

After importing a design and floorplanning it, you should place the design. By default, the Encounter placer (`placeDesign`) internally invokes the I/O pin assignment to place I/O pins based on the current floorplan.

**Note:** Set the `-placeIoPins` option of the `setPlaceMode` command to False if you want to disable I/O pin assignment during the placement step.

## Refining Pin Placement

After I/O pins are assigned, you can further refine the current I/O pin assignment by one of the following methods:

- Manually adjust pins by direct pin manipulations or using pin editor.
- Use the `assignIoPins` command to further optimize I/O placement.

### ***Using the `assignIoPins` Command to Optimize I/O Placement***

The `assignIoPins` command assigns I/O pins based on placement information. The design must be placed before this command is run. The command supports:

- Rectilinear designs
- Non-uniform tracks
- User-specified constraints

By default, the `assignIoPins` command will honor fixed pins and only assign pins that have placed/unplaced placement status. If the initial I/O placement is generated by loading a constraint file (that is, the `loadIoFile` command automatically set I/O placement status to fixed) you should change I/O pins placement status to placed using `setPtnPinStatus` command before running I/O pin assignment.

To incrementally assign I/O pins, you can do one of the following:

- Specify pins that should be re-optimized using the `-pin` option.

Example: Re-assign all p\_address bus pins, int, and bio I/O pins of the design tdsp\_core. Optimize these specified pins even though they have fixed placement status.

```
assignIoPins -pin {p_address[*] int bio} -moveFixedPin
```

- Mark I/O pins that you want to keep with fixed status and run the `assignIoPins` command. This scenario can be used when you want to re-optimize most of I/O pins.

Example: Preserve `port_pad_data_in` and `port_pad_data_out` buses and clock pins, and re-optimize the rest.

```
setPtnPinStatus tdsp_core port_pad_data* fixed  
setPtnPinStatus tdsp_core clk fixed  
assignIoPins
```

## Validating Pin Placement

After assigning I/O pins, it is recommended that you check for I/O legalization.

Use the `checkPinAssignment` command to make sure that pins are legalized (such as they snap to routing grid, are on reserved routing layers, honor user-specified constraints, not cause any DRC violations, and so on).

You can check:

- All I/O pins

Example: Verify all I/O pins of the current design and output the result into the output file `pinLegality.rpt`.

```
checkPinAssignment -outFile pinLegality.rpt
```

- Specific I/O pins

Example: Verify all bus pins `BG_scan_in`, `BG_scan_out`, and the `write` pin of the design

```
checkPinAssignment -pin {BG_scan* write}
```

If any pin violation is detected, you can:

- Manually adjust pins by direct pin manipulation or using pin editor.
- Run the `legalizePin` command to automatically legalize pins. You can legalize all I/O pins or specific I/O pins of the design. Fixed pins will not be adjusted unless the `-moveFixedPin` option is specified.

## **Encounter User Guide**

### Partitioning the Design

---

#### **Example1: legalizePin**

With this example, the Encounter software will legalize all pins in the design. If the design is a block-level design that also has partition/blackbox -pins, it will also adjust the partition/ blackbox pins. If you want to legalize only the I/O pins but *not* the partition/black box pins, you should use `legalizePin -pin *` instead.

#### **Example2: legalizePin -pin \* -moveFixedPin**

With this example, the Encounter software will legalize all I/O pins. Fixed pins will also be adjusted because the option `-moveFixedPin` has been specified.

#### **Example3: legalizePin -pin {clock reset rom\_data\*}**

The Encounter software will legalize clock, reset, and all rom\_data bus bit pins of the design. Pins with fixed status will not be moved.

## Performing Congestion-aware Pin Assignment for Channel-based Designs

To perform route-based pin placement for channel-based designs, it is recommended that you run partition-aware routing instead of a routing that does not take partitions into consideration. Pin assignment decisions based on such partition-aware routing are more optimal with respect to top-channel congestion. However, Trial Route when run in partition-aware mode (`trialRoute -handlePartitionComplex`) is much slower compared to flat (partition-unaware) Trial Route.

To generate a partition-aware routing topology similar to `trialRoute -handlePartitionComplex`, but in much lesser time, you can use the [ptnAwareRouteForPA](#) command (or [trialRoute -fastRouteForPinAssign](#)).

This command generates a routing topology similar to the `handlePartitionComplex` topology for approximately 95% of the inter-partition nets, in about 3X-6X lesser time. For the remaining inter-partition nets, the topology is similar to that generated by flat Trial Route.

The syntax of the command is as follows:

```
ptnAwareRouteForPA trialRouteOptions -intraNets
```

where:

- `trialRouteOptions` are the parameters of the TrialRoute command
- `-intraNets` specifies that intra-partition nets should also be routed.

**Note:** If Trial Route is invoked with the `-handlePartition` option, the `-handlePartition` option is ignored and a warning is displayed.

The [ptnAwareRouteForPA](#) command should be called before pin assignment. The use flow is:

1. Import the design.
2. Floorplan the design.
3. Run the [ptnAwareRouteForPA](#) command.
4. Assign partition pins.
5. Run `trialRoute -honorPin`
6. Derive time budgeting.

## Encounter User Guide

### Partitioning the Design

---

The `ptnAwareRouteForPA` command generates a tabular output listing the nets that were routed in partition-aware manner and those that were not. An example of the output is as follows:

Inter Partition Net groups summary:

NetGrp	Normal/PtnAware	NetCount	ptnNames
1	PtnAware	87(223)	<code>tdsp_core(DTMF_INST/TDSP_CORE_INST)</code>
2	PtnAware	42(223)	<code>ram_256x16_test(DTMF_INST/RAM_256x16_TEST_INST)</code>
3	PtnAware	32(223)	<code>G1(DTMF_INST/G1_PH)</code>
4	PtnAware	20(223)	<code>results_conv(DTMF_INST/RESULTS_CONV_INST)</code> <code>tdsp_core(DTMF_INST/TDSP_CORE_INST)</code> <code>ram_128x16_test(DTMF_INST/RAM_128x16_TEST_INST)</code>
5	Normal	18(223)	<code>ram_128x16_test(DTMF_INST/RAM_128x16_TEST_INST)</code>
6	Normal	15(223)	<code>results_conv(DTMF_INST/RESULTS_CONV_INST)</code>
7	Normal	3(223)	<code>tdsp_core(DTMF_INST/TDSP_CORE_INST)</code> <code>ram_128x16_test(DTMF_INST/RAM_128x16_TEST_INST)</code>
8	Normal	2(223)	<code>G1(DTMF_INST/G1_PH)</code> <code>results_conv(DTMF_INST/RESULTS_CONV_INST)</code> <code>tdsp_core(DTMF_INST/TDSP_CORE_INST)</code>
9	Normal	1(223)	<code>G1(DTMF_INST/G1_PH)</code> <code>tdsp_core(DTMF_INST/TDSP_CORE_INST)</code>
10	Normal	1(223)	<code>G1(DTMF_INST/G1_PH)</code> <code>results_conv(DTMF_INST/RESULTS_CONV_INST)</code>
11	Normal	1(223)	<code>G1(DTMF_INST/G1_PH)</code> <code>tdsp_core(DTMF_INST/TDSP_CORE_INST)</code>
12	Normal	1(223)	<code>G1(DTMF_INST/G1_PH)</code> <code>results_conv(DTMF_INST/RESULTS_CONV_INST)</code> <code>ram_256x16_test(DTMF_INST/RAM_256x16_TEST_INST)</code> <code>ram_128x16_test(DTMF_INST/RAM_128x16_TEST_INST)</code>

- NetGrp: Indicates a group of nets. For example, NetGrp 7 indicates the set of nets that logically connect instances only in partition `tdsp_core` and in partition `ram_128X16_test`.

- **Normal/PtnAware:** Indicates whether the nets of this group are routed in partition-aware routing topology or flat routing topology.
- **NetCount:** Indicates the number of nets in the corresponding net group. For example, NetGrp 7 contains 3 nets out of a total of 223 inter-partition nets in this design.
- **ptnNames:** indicates the partitions to which the nets in this group of nets connect.

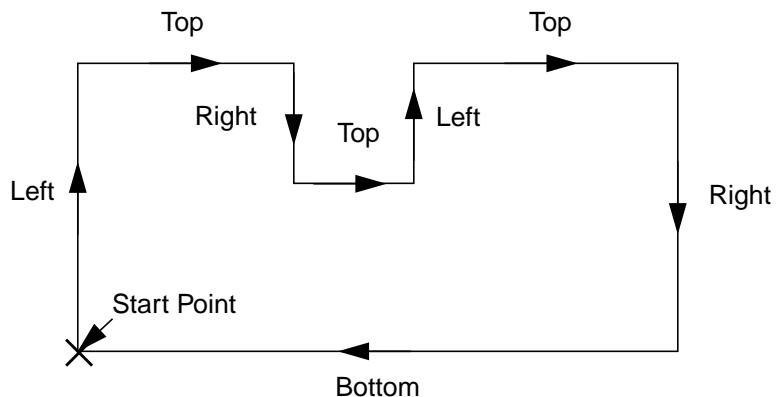
### **Salient Points About Congestion-aware Pin Assignment**

The following points apply to the behavior and usage of the congestion-aware pin assignment feature:

- The routing topology generated by the `ptnAwareRouteForPA` command should be used only for the pin assignment flow.
- The net groups are sorted in descending number of nets in them.
- The net groups that have a significant number of inter-partition nets are routed in a partition-aware manner. The remaining netgroups with fewer inter-partition nets are routed in a manner similar to flat `trialRoute`.
- There is a possibility of more DRC violations in the routing topology generated by the `ptnAwareRouteForPA` command as compared to `trialRoute -handlePartitionComplex`. However, for pin assignment purposes, it has little or no impact in deciding the location of partition pins.
- This command is suited *only for channel-based designs*. Also, the improvement in quality of results of pin assignment, with respect to top channel congestion, is more visible in case the design has thin channels.

## Assigning Pins on Rectilinear Edges

Rectilinear pin assignment can recognize rectilinear edges when assigning pins. It can support any rectilinear shape (such as L, T, and U shapes). For rectilinear boundaries created with partition cuts, the edges are identified by starting at the lower-left-most corner, moving clockwise to mark each edge with a direction flow, as shown in the following figure:



All the edges with the same direction flow are considered to be on the same side and have the same user-specified pin constraints.

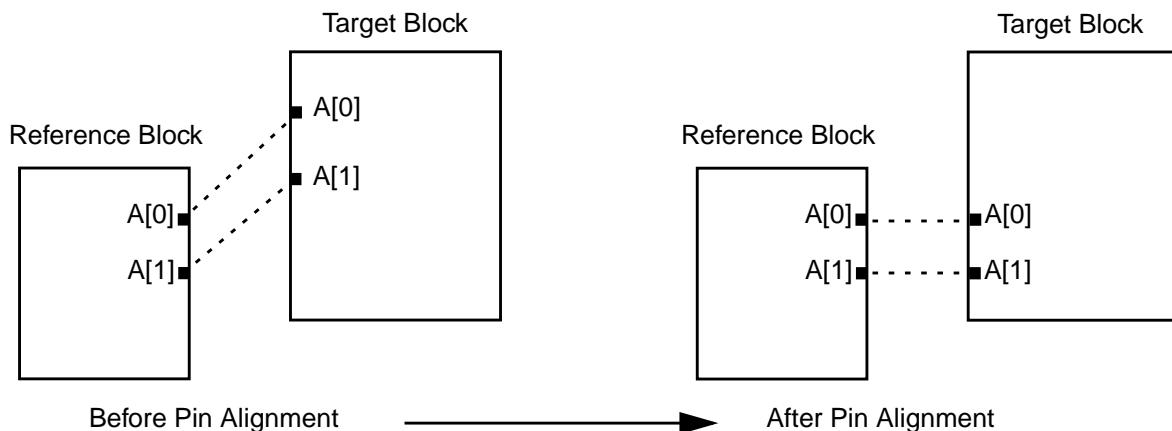
## Swapping Partition Pins

1. Select two pins of the same partition.
2. With the cursor over one of the selected pins, click and hold the middle mouse button to bring up the context pop-up menu.
3. Select Swap Pins (or use the swapPins command).

## Pin Alignment

Using pinAlignment, the following command aligns A0 and A1 pins of **blockB** to the reference pins of **blockA**:

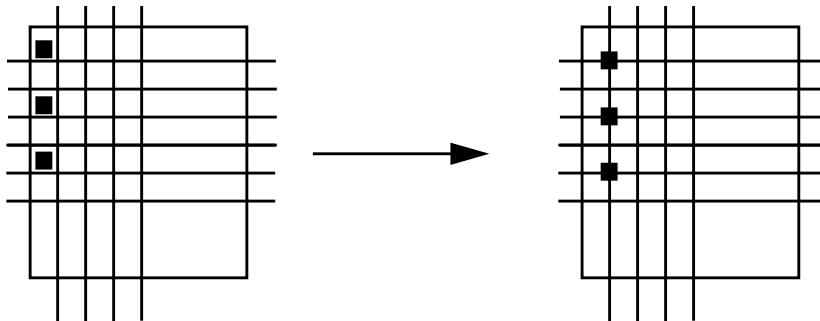
```
pinAlignment -block blockB -refBlock blockA {A0 A1}
```



## **Snapping Pins to the Grid**

To snap center of pins to nearest intersecting routing grid, where the horizontal and vertical routing tracks cross, use the `snapPtnPinsToTracks` text command. For example, the following command snaps center of partition ptn\_xy pins to the nearest intersecting routing grid:

```
snapPtnPinsToTracks ptn_yz
```



`ptn_yz before snapPtnPinsToTracks`

`ptn_yz after snapPtnPinsToTracks`

## Assigning Pins for Bus Guides

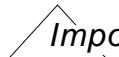
A bus guide helps ensure that buses are routed together over blocks and is typically used in early floorplanning stages. For more information on the Bus Guide feature, see Chapter 10, Bus Planning.

The use model of pin assignment for a bus guide is similar to that of a pin guide. The `assignPtnPin` command supports bus guides by treating a bus guide as a pin guide that is associated with a net group. When you assign pins for a design containing a bus guide, all pins of the corresponding net group are placed in the specified bus guide area.

If the specified bus guide area is not large enough to cover all the net group pins, the `assignPtnPin` command issues a warning message and places the maximum possible net group pins in bus guide area. The rest of pins are placed outside the pin guide area such that the pins stay together.

Bus guide pin assignment also supports all features of net group such as `-optimizeOrder`, `-alternateLayer`, and non-default rules.

The check pin assignment, pin legalization and pin refinement features also support bus guides.

 *Important*

The bus guide feature is intended to guide partition pins and blackbox pins and *not* I/O pins. The I/O pin assignment feature (`assignIoPins` command) does not, therefore, take bus guides into account.

## Pin Assignment Limitations

- Does not support non-R0 orientation black box (non-R0 master black box) pin assignment. For more information, see [Handling of Blackboxes with Non-R0 Orientation](#) on page 263.
- Does not assign or legalize pins on non-preferred routing layers
- Does not assign power/ground pins. For top-down hierarchical flow, power and ground pins will be created during the partition step. For bottom-up flow, power/ground pins should be created at design boundary during power planning stage.
- Partition/blackbox pin assignment may cause routing crossing. In such cases, run the [pinAlignment](#) command to improve pin QoR (Quality of Results).

## Inserting Feedthroughs

There are two types of feedthroughs you can use for partitions: feedthrough buffers and routing feedthroughs. Both types offer different approaches for inserting feedthroughs. Inserting feedthrough buffers allows a netlist change, whereas inserting routing feedthroughs does not.



Before inserting feedthroughs, you should determine what stage the design is in, such as prototyping, intermediate, tapeout, and set the appropriate global options by running the `setMode` commands, such as `setPlaceMode` and `setTrialRouteMode`. For example, when inserting feedthroughs during prototyping, you could set modes with the following commands:

```
setPlaceMode -fp  
setTrialRouteMode -floorplanMode true  
setExtractMode -default
```

You can use the `insertPtnFeedthrough` command (or the [Insert Feedthrough Buffer](#) form) to insert feedthrough buffers into the partitions, and the `createPtnFeedthrough` command (or the [Create Physical Feedthrough](#) form) to create a partition routing feedthrough object. The differences between how these two commands affect the design are as follows:

■ [insertPtnFeedthrough](#)

The `insertPtnFeedthrough` text command inserts feedthrough buffers into the partitions to change the partition netlists, and avoids routing nets over partition areas. This command affects the design in the following areas:

- ❑ Changes both the top-level and partition-level netlists.
- ❑ After inserting buffers, it automatically calls `ecoPlace` to place these buffers close to the partition boundary. However, `insertPtnFeedthrough` does not place the feedthrough pins, which should be assigned during partitioning.
- ❑ Inserted buffers will be part of the partition netlists and pushed down to the partition level during Partitioning.
- ❑ Wherever a net enters and exits a partition, two ports and a buffer (or two buffers with the `-doubleBuffer` option) are added to the partition netlist.
- ❑ For nets that enter or exit a partition several times, such as a “T” shaped connection, three ports will be created. For a cross shaped connection, four ports will be created.

## Encounter User Guide

### Partitioning the Design

---

- ❑ Use the Design Browser to view the newly added buffer instance and net names for each partition. The new port names have a `FE_FEEDX_... . . . net_name` prefix.
- ❑ For pure channelless designs, use the `-chanLess` option to insert feedthrough buffers for all nets that connect to partitions, except nets that can be connected directly between two adjacent partitions.
- ❑ For mixed designs, not all nets should become feedthrough nets. To exclude certain nets, such as clock nets or high fanout nets, use the `-excludeNet` option. This option is based on the topology of the partition neighborhood relationship, so trial routing is not required before inserting feedthrough buffers, although it could help improve the quality of results.
- ❑ To specify a file that contains net names for which to insert feedthrough buffers, use the `-selectNet` option. You can create this file manually, create a list of nets via a script, or use `showPtnWireX`.
- ❑ Whether you use the `-chanLess` or `-selectNet` options, the Encounter software does not necessarily insert a feedthrough.
- ❑ Feedthrough insertion is driven by connectivity when Trial Route is not run before `insertPtnFeedthrough`.
- ❑ You can save feedthrough insertion buffer topology tree information in a file by using the `-saveTopoFile` parameter. You can later use this topology tree file with another ECO netlist and replicate the feedthrough insertions. For more information, see [“Replicating Feedthrough Insertions Across ECO Netlists”](#) on page 314.
- ❑ The `insertPtnFeedthrough` command can detect if the design has power domains. This way, the appropriate buffer is selected from the libraries tied to the power domain. For placement-based feedthrough insertion, buffers are inserted only for those power domains that have the power sequence set to *Always On*. For route-based feedthrough insertion, buffers are inserted based on the routing.
- ❑ The `insertPtnFeedthrough` command removes nets that are inserted with feedthrough buffers from any net groups to which they belong. After running this command you should, therefore, update the net groups that contain feedthrough nets.

#### ■ createPtnFeedthrough

The `createPtnFeedthrough` text command inserts routing feedthroughs into the partitions without changing the design netlist. This command affects the design in the following areas:

- ❑ Manages only the physical aspect of a partition, not the logical aspect.

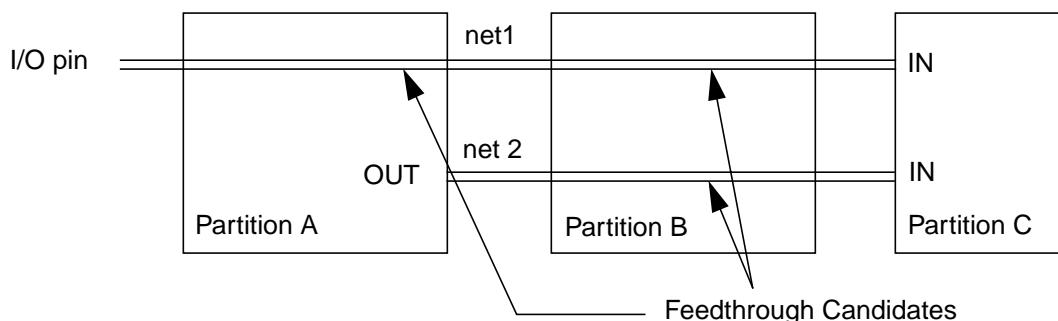
- ❑ No new ports are added to a partition and no buffers are added to the partition netlist.
- ❑ For channel feedthroughs, this creates channels for over-the-block routing on specified layers at the top-level design. These channels are pushed down as routing blockages on the correct routing layers at the partition level during Partitioning.
- ❑ For placement island feedthroughs, the Encounter software reserves these areas for inserting buffers at the top-level design after running the `insertRepeater` command. These island feedthroughs will be pushed down as placement blockages and routing blockages on all routing layers at the partition level during partitioning.

## Inserting Feedthrough Buffers

Partition feedthrough insertion manages partitioned designs that have nets that need to be pushed down to become a component of each partition design. That is, each feedthrough buffer must be added to the partitioned design, which changes the partition's netlist. This approach is typically used in channelless designs and in designs with limited channel resources.

A pure channelless design has no channel routing resource—connections among partitions are always done by means of module abutment and pin alignment. A mixed or partially channelless design has limited routing resource in the channels; therefore, abutment and pin alignment is only performed on selected nets.

The following example shows how nets are selected for feedthrough buffers:



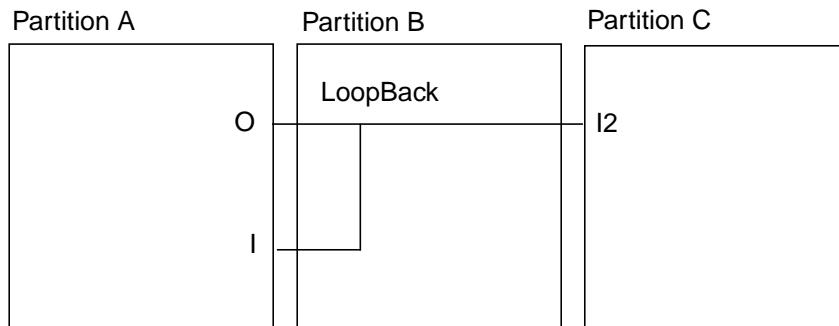
## Inserting Feedback Buffers

You can insert a feedthrough buffer to a net that loops back to an original partition to avoid the net routing over a partition area using the `insertPtnFeedBackBuffer` text command, which you should run after the feedthrough insertion step.

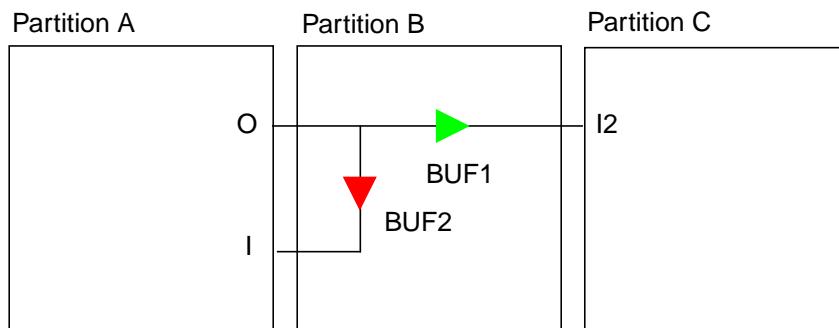
## Encounter User Guide

### Partitioning the Design

The following example shows a situation where net `LoopBack` connects to output pin `O` and input pin `I` of Partition A, and input `I2` of Partition C.



By inserting a feedthrough buffer (BUF1) with the `insertPtnFeedthrough` text command, and inserting a feedback buffer (BUF2) with the `insertPtnFeedBackBuffer` text command, `LoopBack` now connects to the input pins of BUF1 and BUF2, as shown in the following figure:



### Limitations

- Each partition must be intact. A non-child instance cannot be preplaced in another partition. This would present a top-level net connection problem.
- Partition pin guides cannot be used during feedthrough insertion.
- A partition design that has repeated partition modules is not supported. Exclude all nets that connect into a repeated partition module.
- The Unpartition program cannot remove the inserted buffers for the feedthrough nets.
- Does not handle blackboxes.
- It might not handle clock nets efficiently because the `insertPtnFeedthrough` text command does not take timing into account.

- It cannot handle nets that are connected to two or more glue logic standard cells. This type of net should be excluded from feedthrough insertion.
- It might not provide good quality of results for high fanout nets. You should exclude high fanout nets and clock nets from feedthrough insertion to avoid timing and routing problems.

### **Procedure**

1. Design the top-level floorplan for the partition design.
2. Run Placement.
3. (Optional) Run Trial Route.

 *Important*

Up to step 3, the flow is similar to a partition design flow. To control which nets get buffers inserted, complete step 4. If all nets require buffering, skip step 4 and use the `insertPtnFeedthrough` text command's `-chanLess` option.

4. Create a file to identify which nets get buffers.

You can manually edit the file, create a script, or generate a wire crossing file (see [Generating the Wire Crossing Report](#) on page 326).

5. Generate the feedthrough buffers and nets.

Use the `insetPtnFeedthrough -chanLess` command, or `insetPtnFeedthrough -selectNet` with the created net file.

**Note:** Step 6 returns to the normal partition design flow.

6. Run Trial Route to completely connect the design, including the inserted feedthrough buffers.
7. Run Partition to generate the partition pins and change the partition module status to hard block.
8. Run Save Partition.

This saves the design and generates a top-level directory and partition directories.

## Using a Topology File to Insert Feedthrough Buffers

You can guide the insertion of feedthrough buffers for specific nets by providing the feedthrough topology information for those nets in a topology file. You can manually create this file and subsequently edit it.

**Note:** If you are using topology files from releases prior to the 8.1 release, they will still work with this release.

**Note:** The syntax is case sensitive.

The syntax of the topology information in the file is as follows.

```
# Comment line
version version_string;
nametype netname
    fromtype-totype  from_name to_name route_data=(x,x,x,x,x,x);
    fromtype-totype  from_name to_name  route_data=(x,x,x,x,x,x);

.
.

end nametype
nametype netname
    fromtype-totype  from_name to_name route_data=(x,x,x,x,x,x);
    fromtype-totype  from_name to_name  route_data=(x,x,x,x,x,x);

.
.

end nametype
.
```

## Encounter User Guide

### Partitioning the Design

---

The description of the syntax is as follows

nametype

Can be net, bus, or netgroup. The value netgroup represents all nets in the net group. You should update the net group after feedthrough insertion step.

Here are some examples of nametype:

- bus myBus[0:1] specifies bus bits
- net myBus[0:1] specifies a scalar net.
- bus myBus[1] specifies a bus bit
- net mybus[1] specifies a scalar net or a bus bit. In case both exist in the design, use the Verilog escape name and use the `dbgIsBackSlashInNamesHiddenFlag` variable to resolve correctly.

## Encounter User Guide

### Partitioning the Design

---

netname      Can be a net name, bus name, or a net group name. Wildcards (\* or ?) can be used for net name, bus name, or net group name.

If more than one net group is matched with wildcard, the insertPtnFeedthrough command will issue a warning message and:

- use only the first matched net group
- ignore the other ones.

Wild cards can only be used for a bus name BUT not bus range. Example you cannot specify bus busname[1:\*].

*Specifying bus entries:* If a bus named databus has 32 bits (from 0 to 31), its r bus entries are specified as follows:

- bus databus specifies all 32 bits from 0 to 31
- bus databus[13 : 23] specifies databus[13] to databus[23]
- bus databus[13] specifies only the bit 13 of databus

You cannot provide any net-specific entries for multiple bus bits, net groups, or wildcard nets. Hence, bus topologies cannot be specified for bus nets connected to top-level instance pins or to I/O pins.

*Using escape mechanism for special characters:* The following escape mechanisms remove all restrictions on characters:

- \\ for the backslash character (\) itself
- \b for blank
- \t for tab
- \n for new line
- \0 for null
- \s for semicolon (semicolon (;) is the path statement terminator).

Any other character which follows a backslash (\) is taken literally. For example, \a is considered as a. If one wants to use \*,? literally then must use escaping as these are used for wildcards.

**Note:** If a net appears twice in any form, the first entry corresponding to the net is used. The subsequent entries generate an error.

**fromtype**

Can have one of the following values:

- `io` for I/O pins
- `hinst` for hierarchical instance name of a partition or partition clone
- `instterm` for top-level instance pins

**totype**

Can have one of the following values:

- `io` for I/O pins
- `hinst` for hierarchical instance name of a partition or partition clone
- `instterm` for top-level instance pins
- `hinstfb` for hierarchical instance name of a partition or partition clone. This can only be used as part of the combination `hinst-hinstfb`, which specifies a feedback buffer path.

**version**

Version is the format version. The format version for this release is 1.0.

If the topology file does not have the `version` statement then the `insertPtnFeedthrough` command will parse the file as per the format of the version prior to the 8.1 release.

**route\_data**

Optional field that specifies routing information.

This is not a user-specified field. This field is created when the `insertPtnFeedthrough` command is run with the `-saveTopoFile` parameter. This field is used only for ECO purposes.

The `route_data` parameter is not available if the `totype` is `hinstfb`.

All version- and topology-statements in the topology file end with a semicolon (;). Any extra spaces are ignored.

Here is an example of a topology file:

```
#####
version 1.0;
net net1
io-hinst net1 i_b;
hinst-instterm i_b inst_c/net1;
end net

net clk*
hinst-hinst i_a i_b;
hinst-hinst i_b i_c;
end net

netgroup group_a
hinst-hinst i_a i_b;
hinst-hinst i_b i_c;
end netgroup

bus databus[0:31]
hinst-hinst i_a i_b;
hinst-hinst i_b i_c;
end bus
#####
```

## Replicating Feedthrough Insertions Across ECO Netlists

While performing a feedthrough insertion through the `insertPtnFeedthrough` command, you can save the feedthrough buffer topology tree information in a file by specifying the `-saveTopoFile` parameter as follows:

```
insertPtnFeedthrough -saveTopoFile TopoFileName
```

where *TopoFileName* is the name of the file in which topology information is saved.

When you run the `insertPtnFeedthrough` command on another ECO netlist, you can use the saved file to replicate feedthrough buffer insertions by specifying the `-topoFile` parameter as follows:

```
insertPtnFeedthrough -topoFile SavedTopoFileName
```

where *SavedTopoFileName* is the name of the file that was saved earlier using the `-saveTopoFile` parameter.

This way, you can save a file with feedthrough buffer topology tree information and use it to create the same feedthrough buffer insertions across multiple netlists.

The flow can be summarized as follows:

1. Import a design.
2. Perform floorplanning on the design.
3. Perform feedthrough buffer insertion and save the feedthrough buffer topology tree information in a file (use the `-saveTopoFile` parameter of the `insertPtnFeedthrough` command).
4. Import design with a new ECO netlist.

**Note:** The ECO netlist should not contain the original inserted feedthrough buffers.

5. Perform feedthrough buffer insertion with the topology file saved from step 3 (use the `-saveTopoFile` parameter of the `insertPtnFeedthrough` command).

**Note:** If you use the `-topoFile` parameter, only those nets that are specified in the topology file are considered for feedthrough buffer insertion.

**Note:** If a net does not exist in the design, it should not be in the topology file. For example, if ECO changes remove a net, that net should be removed from the topology file.

6. Repeat steps 4 and 5 for more ECO netlists, if required.

## Reducing the Number of Buffers and Ports Added for Route-based Feedthrough Insertions

You can use the `-reduceAddedPort` parameter of the `insertPtnFeedthrough` command to specify that feedthrough insertion should follow the routing topology more closely. This can help reduce the number of added ports and buffers.

The ports are created at the route crossing points. The status of the added ports is set to *Fixed*. Subsequent use of Trial Route will make the routes pass through these pins. Therefore, there is no need to create partition pin guides for these pins.

**Note:** The `-reduceAddedPort` parameter is applicable only for route-based feedthrough insertions.

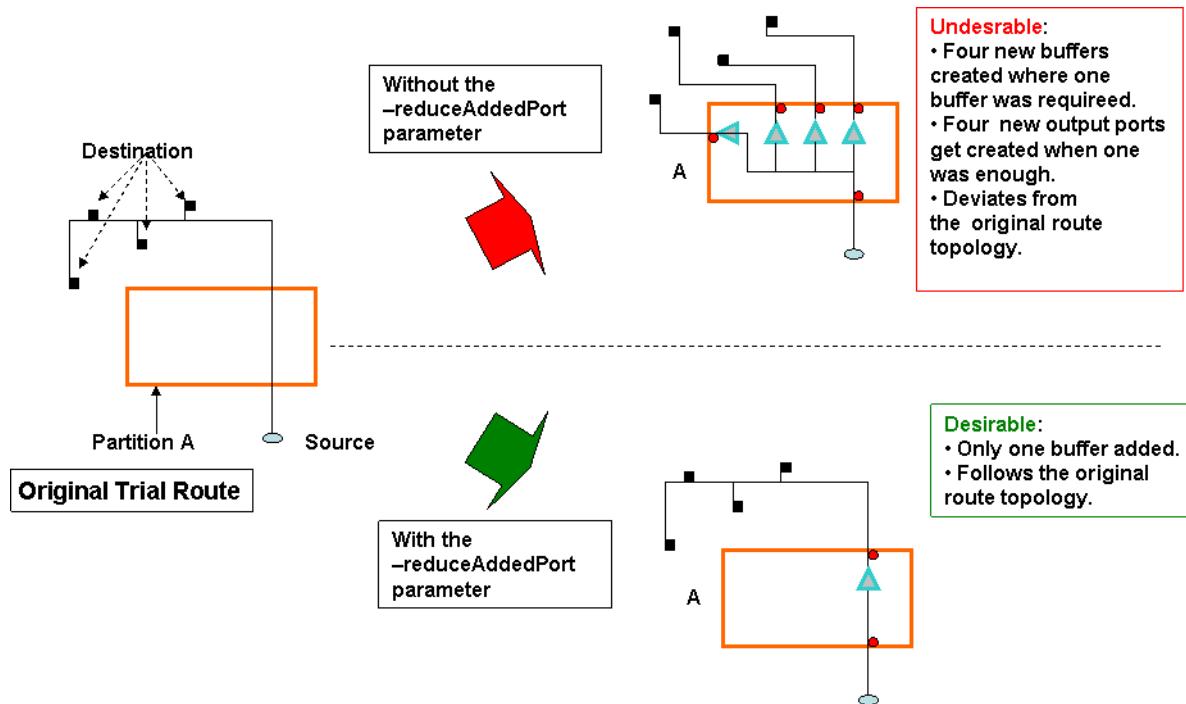
This behavior is illustrated through the following scenarios:

- Net Connecting to Non-partition Instance Terminals in the Top-level Routing Channels:

■ Net Connecting Through Adjoining Partition

***Net Connecting to Non-partition Instance Terminals in the Top-level Routing Channels***

The following diagram illustrates the improvement in feedthrough insertion where a net connects to a non-partition instance terminals in the top-level routing channels.



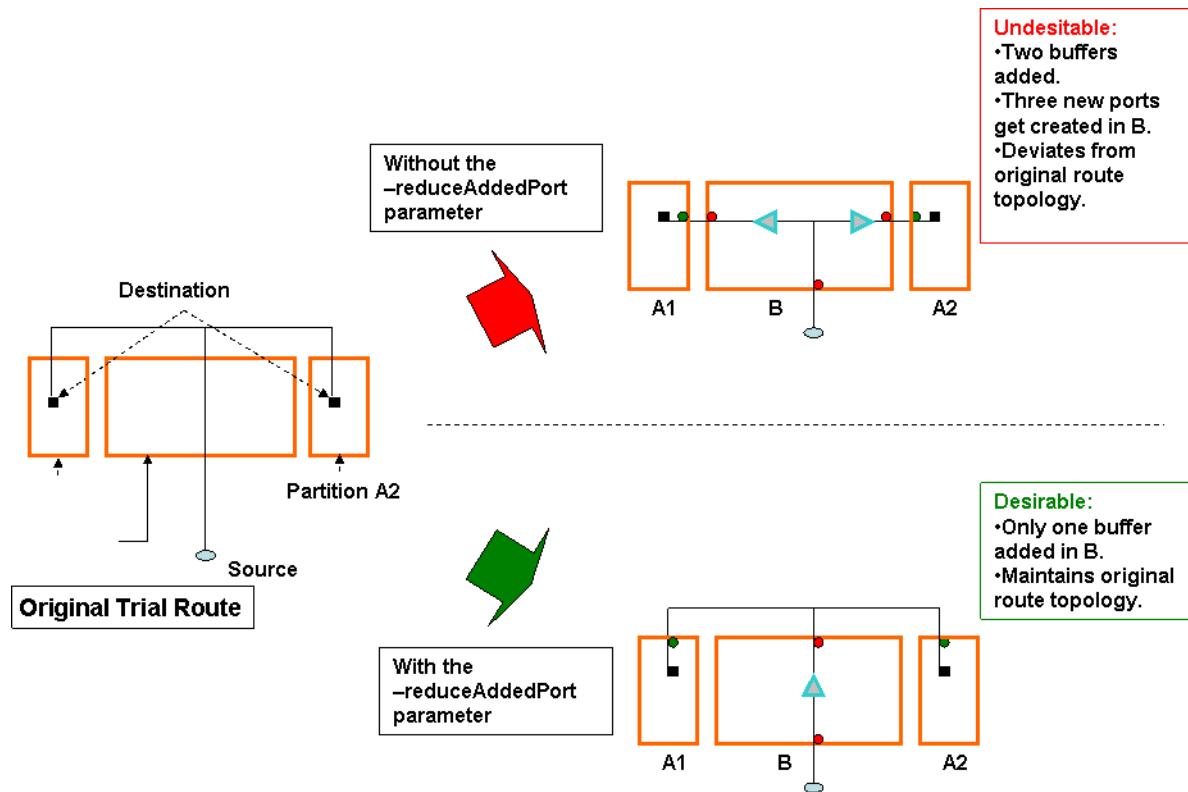
***Net Connecting Through Adjoining Partition***

The following diagram illustrates the improvement in feedthrough insertion between partitions where there is another partition in between.

## Encounter User Guide

### Partitioning the Design

---



### Abbreviating Lengthy Feedthrough Net Names

You can abbreviate inserted feedthrough net names so that the net names will not extend too long if you run the `insertPtnFeedthrough` or `insertPtnFeedBackBuffer` commands multiple times. With the `-useShortName` option, you can eliminate the use of the old net name and partition names, and instead use a running count for the new net names.

For example, if a feedthrough net reset connects two partitions `tt_chiplet` and `video_chiplet`, the feedthrough net name is:

```
FE_FEEDX_NET_C_tt_chiplet_video_chiplet_reset
```

The net name abbreviation convention for feedthrough buffer insertion when using the `insertPtnFeedthrough -useShortName` command are:

Net Names      `FE_FTN_n`, where *n* is an integer

Buffer Names    `FE_FTB_n`, where *n* is an integer

The net name abbreviation convention for feedback buffer insertion when using the `insertPtnFeedBackBuffer -useShortName` command are:

Net Names      `FE_FB_NET_n`, where *n* is an integer

Buffer Names    `FE_FB_BUF_n`, where *n* is an integer

## **Highlighting the Nets for which Feedthrough Buffers Have been Inserted**

Once you insert partition feedthrough buffers with the `insertPtnFeedthrough` command, you can highlight these nets with the `hiliteFeedthroughNets` command. The highlighted feedthrough path consists of the nets, the terms that the nets connect to, and the instances that contain those terms.

For the `hiliteFeedthroughNets` command to work, the `insertPtnFeedthrough` command must be run with the `-netMapping` parameter. The net mapping file generated with the `insertPtnFeedthrough -netMapping` parameter is used by the `hiliteFeedthroughNets` command to highlight the feedthrough nets.

To dehighlight the feedthrough nets, run the `dehighlight` command.

## **Utilizing Pre-defined Feedthrough Pins in Custom Macros**

Some designs contain hard macros, which could, for example, be IP blocks or analog blocks. chip-level routing might not be possible without passing over these blocks. Or, in other cases, routing might not meet timing requirements if it detours around these blocks. To facilitate routing these blocks might provide pre-defined feedthrough pins

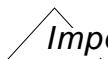
You can utilize these predefined feedthroughs using the `connectMacroFeedthrough` command. This command automatically connects the feedthrough pins to nets that have wires crossing over these blocks or macros.

## **Use Flow**

The `connectMacroFeedthrough` command uses the routing topology to connect the pre-defined feedthrough nets. Therefore, the design must be placed and routed before you run the `connectMacroFeedthrough` command. The use flow is as follows:

1. Import the design.
2. Floorplan the design.
3. Perform placement.

**4. Run Trial Route.**

 *Important*

At least one vertical and one horizontal routing layer must be available (that is, not blocked) on the macro(s). Otherwise, there will be no routing over the macro(s). In case the macro has all the layers blocked, manually remove the blockage over one horizontal and vertical layer.

**5. Connect the built-in feedthroughs through the `connectMacroFeedthrough` command.**

**Note:** Before running detailed routing, take care of the unused feedthrough input pins that are left floating. For example, you might want to assign them to tie-high or tie-low. You can save the list of the unused ports with the `connectMacroFeedthrough -floatingPortList` command.

### How the `connectMacroFeedthrough` Command Connects Feedthroughs

The following points illustrate the criteria for feedthrough selection and other important features of the `connectMacroFeedthrough` command:

- The `connectMacroFeedthrough` command considers all routing on all layers that cross the specified custom macro boundaries.
- The command searches for a feedthrough whose in and out pins lie *on the same sides* of the macro on which the wires enter and exit the macro.
- A feedthrough that has pins that are closer to the intersections has a higher probability of selection. Both input and output pins are considered. Layer information is ignored while evaluating the distance. To consider only pins within a specific distance from the wire crossing, use the `-maxSearchDistance` parameter.
- The command creates new nets and ports as required.
- If multiple feedthrough insertions are performed, the command keeps track of the feedthroughs already used, and does not assign such feedthroughs again.
- The new nets (the nets that connect to feedthrough output pins) have the following naming convention:

`FE_FTM_x_netName`

where `x` is a unique numeric identifier and `netName` is the name of the original net.

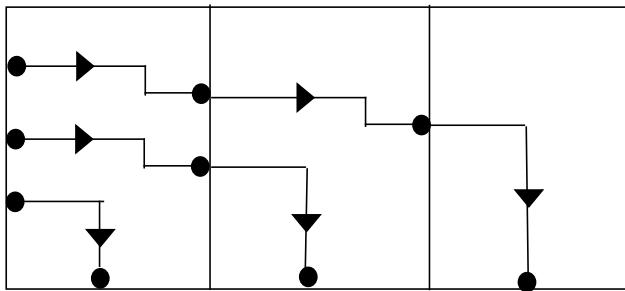
- You can select only specific nets for or exclude specific instances or nets. You can also specify the distance till which the command will search for a connected feedthrough. The

feedthrough connectivity is described through a mapping file, which is described in the section [Mapping File For Describing Feedthrough Connectivity](#) on page 322.

### ***Feedthrough Connection for Abutted Macros***

For abutted custom macros, the `connectMacroFeedthrough` command detects the paths formed by the abutted feedthrough pins. The Encounter software considers only the end points of the detected paths, and picks those feedthroughs that will give good results.

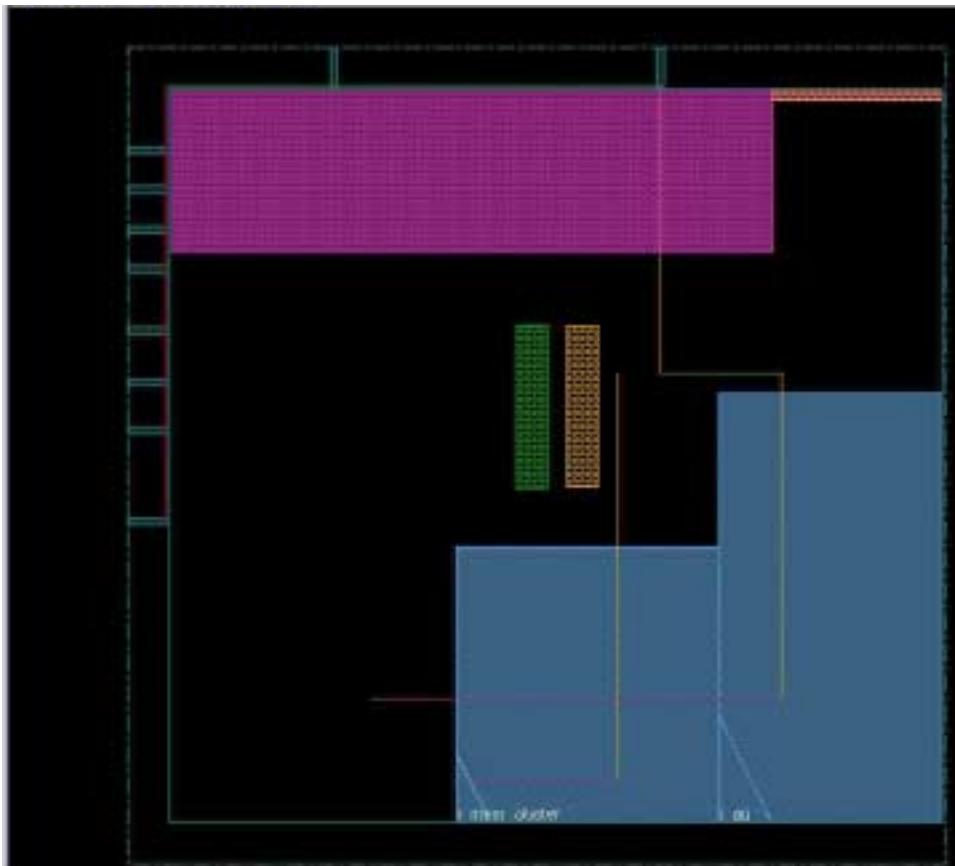
The following figures show how Encounter selects the feedthroughs for insertion in the abutted custom macros.



## Encounter User Guide

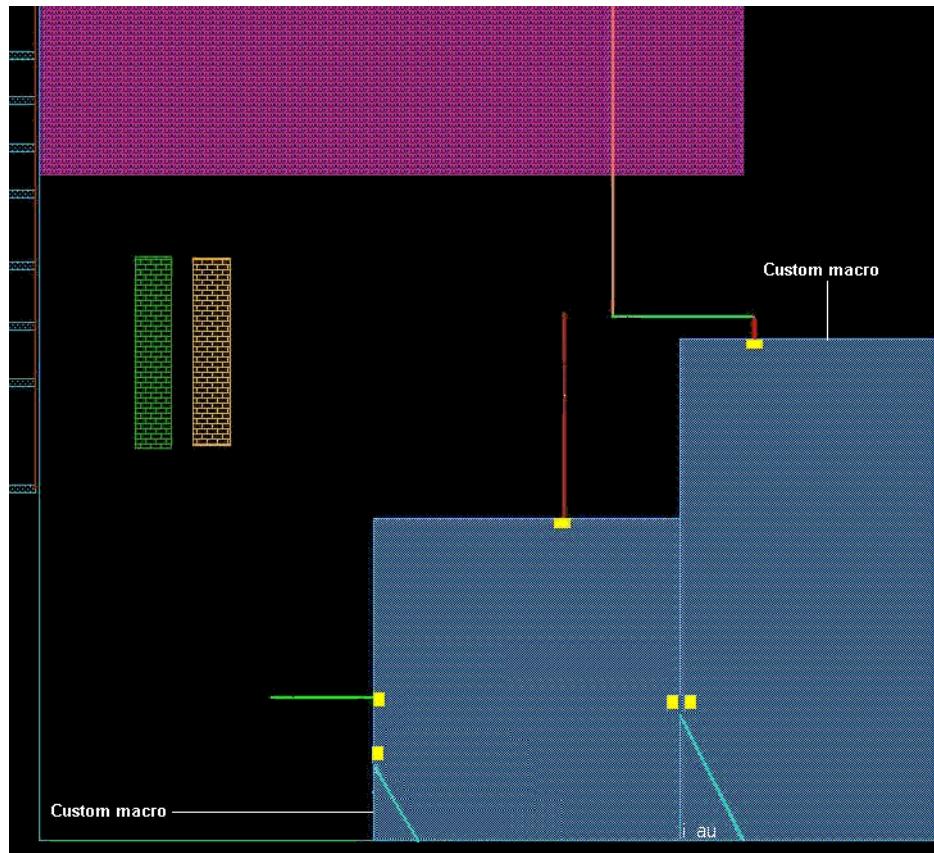
### Partitioning the Design

The following figure shows pre-defined custom feedthroughs in the design.



The following figure shows how these feedthroughs are utilized by the [connectMacroFeedthrough](#) command. Notice the feedthrough pins, represented by

yellow squares, that are added at the intersection of the macro boundary and the pre-defined nets.



### Mapping File For Describing Feedthrough Connectivity

The feedthrough connectivity is defined through a mapping file that is provided as a parameter to the. If a mapping file is not specified with the [connectMacroFeedthrough](#) command, Encounter assumes that a file with the name `portmap` in the current directory is used by default.

The syntax of the file is as follows:

```
MACRO MacroName  
Macro definition section  
END MACRO
```

The definition of the macro is provided in the *Macro definition* section, which can contain one or more feedthrough sections. The name of the feedthrough section is optional.

## Encounter User Guide

### Partitioning the Design

---

**Note:** The definitions for all custom macros to be used in the design should be in a single portmap file.

The syntax of the Feedthrough section is as follows. The name of the feedthrough is optional.

```
Feedthrough [FeedthroughName]
```

```
Pin Section
```

```
END FEEDTHROUGH
```

Each Feedthrough section contains one section for the input pin and one section for the output pin.

**Note:** Multi-fanout feedthrough sections are not supported.

The syntax of the pin section is as follows:

```
PIN PinName
```

```
END PIN
```

**Note:** All the predefined macro feedthrough pins should be floating pins.

Here is an example of a mapping file:

```
MACRO RAMXXX
  FEEDTHROUGH feedthrough1
    PIN feedthrough1_in
    END PIN
    PIN feedthrough1_out
    END PIN
  END FEEDTHROUGH
  FEEDTHROUGH feedthrough2
    PIN feedthrough2_in
    END PIN
    PIN feedthrough2_out;
    END PIN
  END FEEDTHROUGH
END MACRO
```

## Limitations

The `connectMacroFeedthrough` command has the following limitations:

- Multi-fanout feedthroughs are not supported.
- Routing blockage and congestion are not considered. However, because topology is derived from routing, this should not be a concern.
- Bidirectional pins (INOUT) are not supported.
- The topology is derived from the routing results. Therefore, you might need to specify certain Trial Route options (for example, options to block or unblock certain routing tracks) to get the desired routing results.
- Floating module ports connected to a net are not supported because there is no routing to the floating module ports.
- Rectilinear hard macros are not supported.

## Inserting Routing Feedthroughs

Routing feedthroughs and hole punch buffers reserve a portion of the partition area for top-level use. Because the partition's netlist does not change, no new ports are created for the partition. Buffers are inserted in top-level netlist but occupy space within the partition's fence. Partition feedthroughs are used to indicate the top-level partition's concession within the partition fence.

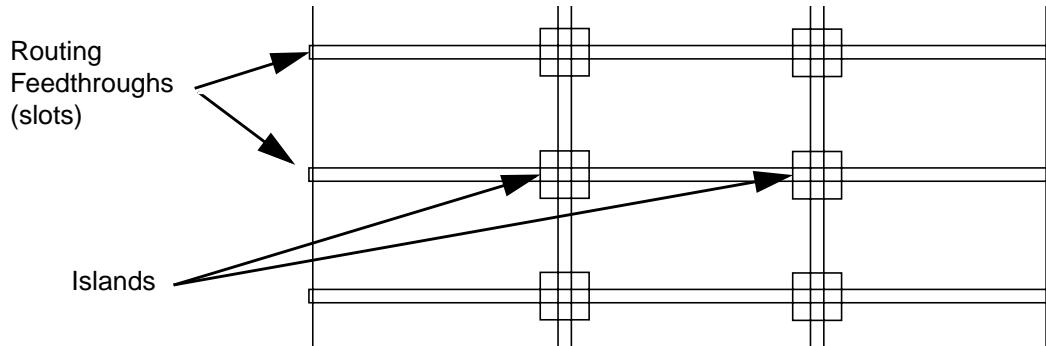
Partition feedthroughs should be defined before running the Partition program, which automatically generates appropriate placement and routing blockages within the partition and in top-level view to reflect the real estate ownership scheme. For example, a routing feedthrough with *Meta/6* will generate a *Meta/6* routing blockage for the partition, and an opening in the *Meta/6* blockage in the top level.

**Note:** The partition feedthrough discussed in this section is a floorplan object. It affects a partition only physically (not logically) and does not affect partition feedthrough buffer cells.

## Encounter User Guide

### Partitioning the Design

A *routing feedthrough* (slot) within the partition's fence is used by the top-level partition's routing, and an *island* within the partition's fence can be used by the top-level partition's placement, as shown in the following figure:



**Note:** Routing feedthroughs can be used without placement islands.

To create a channel-type feedthrough, use the *Partition Feedthrough* tool widget. After adding a partition feedthrough to the design, you can use the Attribute Editor to change its layers. The specified routing layers are reserved for top-level use, and the partition uses the other layers. You can create an island type partition feedthrough in a similar way, but all layers are deselected.

To insert routing feedthroughs and hole punch buffers, complete the following steps:

1. Create routing feedthroughs using one of the following methods:

**Method 1:** Use the *Add Partition Feedthrough* widget to create the feedthrough buffer on the partition. Select the buffer and open the Attribute Editor form, specify the metal layer, and click *OK*. This creates the channel for the routing on the specified layers at the top level, and pushes down appropriate routing blockages at the block level.

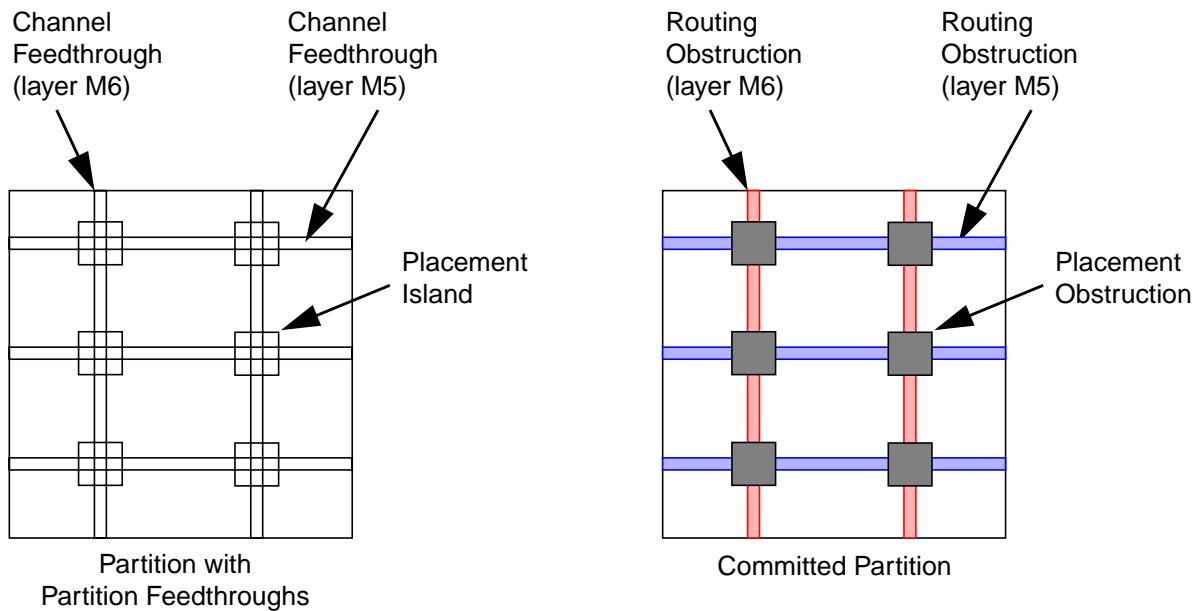
**Method 2:** If you want to specify narrow feedthroughs or several of them on a given partition, choose *Partition – Create Feedthroughs* to open the Create Feedthrough form. To specify which partition you want, click on the partition in the design display area, then click *get selected*. Complete the form and click *OK*.

2. (Optional) if you have hole punch buffers, create an island to specify where the holes are to be punched in the partition.

To do this, use the *Add Partition Feedthrough* widget to create a routing blockage and placement island, run IPO or buffer insertion to place buffers into the island, then deselect all layers after double-clicking on the island. This creates the island for buffer placement at the top level, and pushes down the appropriate routing and placement blockage at the block level.

### 3. Run Partition.

This automatically creates routing blockages for the channel feedthroughs, and placement blockages for the placement island, as shown in the following figure:



## Generating the Wire Crossing Report

You can display and write a file of wires that physically cross over partitions using the `showPtnWireX` text command or the *Partition – Show Wire Crossing* menu command.

The results are saved to a `designName.wirecrossing` file that reports nets that cross each partition in a design. For any net that crosses more than one partition, you can use it as a starting point for generating a list of nets for feedthrough insertion.



**Tip**  
Edit the `designName.wirecrossing` file to exclude high fanout nets, clock nets, and nets that are connected to two or more glue logic standard cells to avoid timing and routing problems on these nets. You can use the resulting file with the `insertPtnFeedthrough` text command's `-selectNet` option. Note that the Encounter software determines the buffer tree topology, so not all specified nets will receive inserted feedthroughs. For example, nets that connect directly between adjacent partitions are not candidates for feedthrough insertion.

## Interpreting the Wire Crossing Report

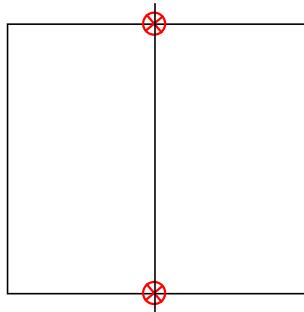
The wire crossing report section lists the nets, their wire lengths, in micrometers, and the shape of the wire in relation to the partition. For example, the following report segment is for a partition module named ptn01:

```
#####
# Nets that cross partition module ptn01
#   Box (335 335) (833 567)
#   Format: Net <netName> <wireLength> <shape>
#####

Net A 65 I
Net B 80 L
Net C 1050 T
Net D 132 X
...
```

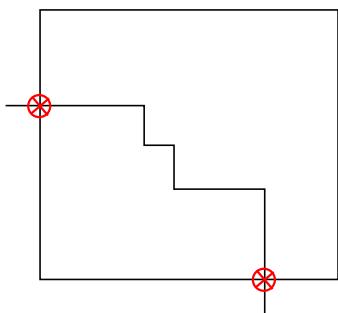
The first net in the report, A, has a wire length of 65 micrometers in an ‘I’ shape, which indicates that the net crosses the partition on opposite sides, as follows:

Net A 65 I



The second net in the report, B, has a wire length of 80 micrometers in an ‘L’ shape, which indicates that the net crosses the partition on adjacent sides, as follows:

Net B 80 L

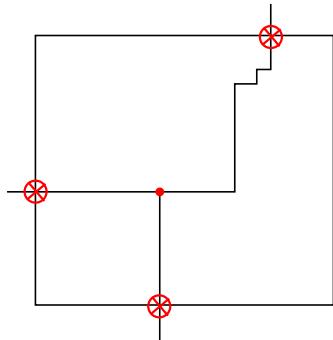


## Encounter User Guide

### Partitioning the Design

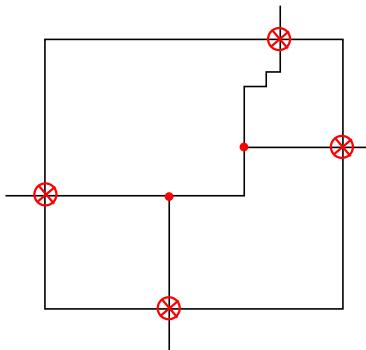
The third net in the report, C, has a wire length of 105 micrometers in an ‘T’ shape, which indicates that the net crosses the partition on three sides, as follows:

Net C 105 T



The fourth net in the report, D, has a wire length of 132 . 30 micrometers in an ‘x’ shape, which indicates that the net crosses the partition on all four sides, as follows:

Net D 132 X

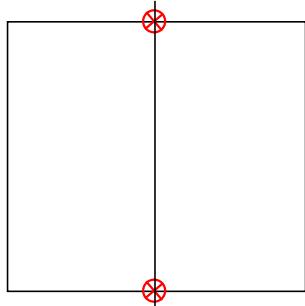


In the report, you can also include the total length of the wire crossing the block in the horizontal X direction and total length of the wire crossing the block in the vertical Y direction using the -delta option of the showPtnWireX command. For example, the following report segment is for the same partition module named ptn01 using the -delta option:

```
#####
# Nets that cross partition module ptn01
# Box (335 335) (833 567)
# Format: Net <netName> <wireLength> <shape> <deltaX> <deltaY>
#####
Net A 65 I 0 65
Net B 80 L 38 47
...
```

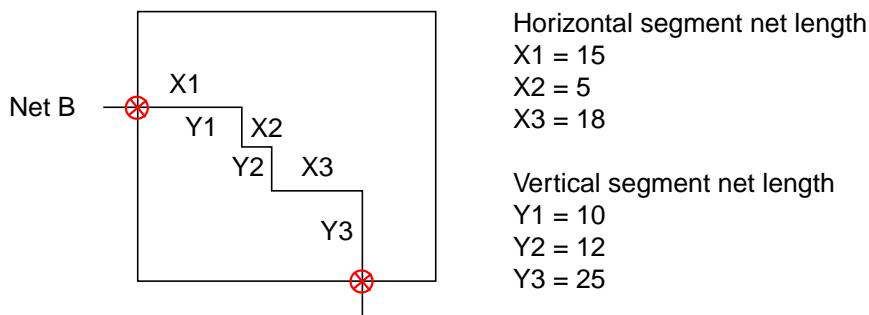
The first net in the report, A, has a wire length of 65 micrometers in an ‘I’ shape, with a total of 0 length in the horizontal X direction, and 65 in the vertical Y direction:

Net A 65 I 0 65



The second net in the report, B, has a wire length of 80 micrometers in an 'L' shape, with a total of 38 length in the horizontal X direction, and 47 in the vertical Y direction:

Net B 80 L 38 47



In the above example, the 38 length in the X direction is calculated for the X direction net segments ( $X_1 + X_2 + X_3$ ), and the 47 in the Y direction is calculated for the Y direction net segments ( $Y_1 + Y_2 + Y_3$ ).

## Estimating the Routing Channel Width

For committed partitions and blackboxes with assigned pins, channel width estimation uses the current pin assignment. If partition pins are not assigned, they are placed at the lower-left corner. In this case, the Encounter software issues a warning message because the estimator cannot produce a good result.

For uncommitted partitions, channel width estimation runs the Partition program, assigns pins, estimates the channel widths, and runs the Unpartition program. For blackboxes without assigned pins, it assigns pins and estimates the channel widths.

Channel width estimation also considers topology constraints to drive block placement. These constraints are block-to-block boundary, block-to-block distance, block order and

## Encounter User Guide

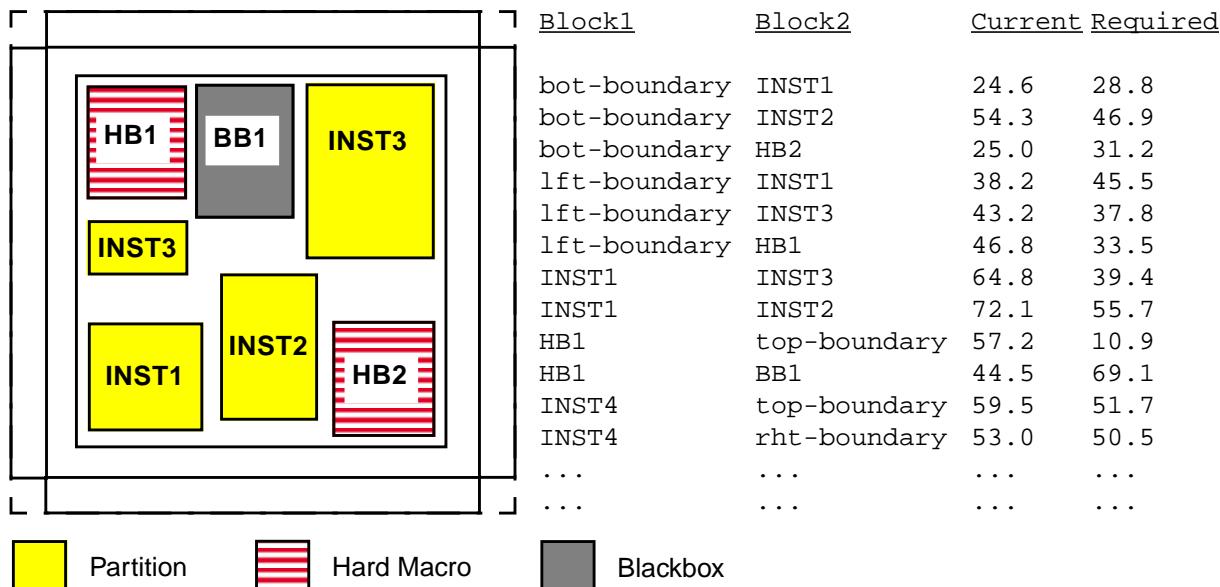
### Partitioning the Design

alignment, block aspect ratio, net weight (from global `trialRoute`), and block halo. The channel width estimator also respects these constraints so that their top-level block floorplans are not dramatically changed. If there is conflict between a specified constraint and the minimum required channel spacing, the Encounter software honors the minimum required channel spacing.

This feature produces a report containing the following information:

- Estimated required spacing, in micrometers, between partitions, blackboxes, and hard macros.
- Estimated required spacing surrounding each partition based on its pins (the relative distance between partition blocks required for top-level routing).
- Estimated distance between blocks and core boundaries (top, bottom, left, right).

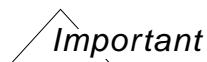
The following figure shows an example of how the channel estimation report relates to the design:



## Running the Partition Program

The Partition program creates the partitions in the top-level design. This changes the module's status from a fence to a block and generates pins if routing data exists from running Trial Route. When the Partition program is run, the Trial Route data is deleted because the current placement and route data are not suitable for further work at the top level. The partition pin guide (floorplan) object can be used to determine the location of the pins, and nets or buses will be assigned to the partition pin guide objects.

If the partitions are changed, then the placement and Trial Route programs must be rerun. To change the status of the partition from being a hard block, you must run Unpartition to flatten the partition.



After you run the Partition program and save the partition data, you should exit the session and start a new session for the top-level design and for each partition in their newly created UNIX directories.

**Note:** Running the Partition program creates a blockage on an OVERLAP layer even though the OVERLAP layer is not defined in the technology section of the LEF file. As a result, the partition LEF file cannot be loaded into either the Encounter software or any standalone tools. If your design has rectilinear partitions or feedthroughs, the OVERLAP layer must be defined in the technology section of the LEF file.



If a partitioned design is unpartitioned and then partitioned again, it will lose the original routing and timing information. The routing and timing information are not preserved during the unpartition-partition process.

To restore the timing information, Save your routing data before partitioning. If you unpartition later, run the `restoreRoute` text command to get the routing information, then run `extractRC`, and then `buildTimingGraph`, to restore timing information.

You can save the partition data in an OpenAccess database. For more information, see [Working with OpenAccess Database](#) on page 351.

### Creating a Top-Level Partition

1. Run the Partition program.
2. Run Trial Route on the top-level partition.

**3.** Check for routing congestion.

If there is no congestion, you are done. If there is congestion, continue to step 4.

**4.** Run the Unpartition program and add more routing resources to the congested area.

**5.** Rerun the Partition program.

Repeat steps 1 – 5 until there is no routing congestion.

### **Block-Level Partition**

To create a block-level partition, complete the following steps:

**1.** Run the Partition program.

**2.** Check to see if each partition size is suitable.

If it is, you are done. If it is not, continue to step 3:

**3.** Run the Unpartition program.

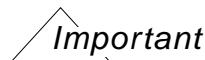
**4.** Increase the size of the block.

**5.** Rerun the Partition program.

Continue with the steps above until you have reached suitable partition sizes.

### **Pushing Down Signal Routes**

During partition program, you can use the `-pushRoute` parameter of the [partition](#) command to push down signal routes to the respective partitions.



Before running the `partition -pushRoute` command, you can check the hierarchy violations for nets on the partitions with the [checkHierRoute](#) command.

Here's the pushdown behavior with the `-pushRoute` parameter of the [partition](#) command:

- The following routes are pushed down:
  - ❑ Intra-partition nets routed completely within the routed boundary.
  - ❑ Inter-partition nets that cross the partition boundary only once *and* that pass through the partition pin location.

- Top nets that are routed completely in the top channels are retained at the top
- All other nets are deleted.

For nets that have a hierarchy violation, only the wire segments that have a hierarchy violation on the nets are discarded. The other wire segments are retained.

## How Top-level Stripes Are Pushed Down

This section explains how stripes on the top level are pushed down into the partition when you run the partitioning program. The following scenarios are discussed:

- [The Default Behavior](#) on page 333
- [Behavior with the -stripStayOnTop Option](#) on page 334

### The Default Behavior

The following table summarizes the default behavior.

Stripe Position	How Stripe Is Pushed Down
Stripe is completely inside partition boundary	<ul style="list-style-type: none"><li>■ Top-level: The stripe is removed from the top.</li><li>■ Block-level: The stripe is pushed down as two pins and one stripe.</li><li>■ Block Abstract:<ul style="list-style-type: none"><li>□ On layers reserved for partition, two pins are created on the boundary.</li><li>□ On layers not reserved for partition, one big LEF pin is created.</li></ul></li></ul>
Stripe is partially inside partition boundary.	<ul style="list-style-type: none"><li>■ Top-level: The stripe is retained at the top.</li><li>■ Block-level: The stripe is pushed down as two pins and one stripe.</li><li>■ Block Abstract: The stripe is pushed down as a big LEF pin.</li></ul>

Stripe Position	How Stripe Is Pushed Down
Stripe is outside but close to partition boundary	<ul style="list-style-type: none"><li>■ Top-level: The stripe is retained at the top.</li><li>■ Block-level: The stripe is retained at the top and is copied as a routing blockage (same size as stripe) with a +PUSHDOWN attribute.</li><li>■ Block-abstract: No effect.</li></ul>

### Behavior with the `-stripStayOnTop` Option

The `-stripStayOnTop` parameter in the [partition](#) command specifies that stripes that are not on a layer reserved by the partition are retained at the top level and are also copied into the partition. The following table explains how the stripes on the top level are pushed down to the partition when you run the partitioning program with the `-stripStayOnTop` parameter.

## Encounter User Guide

### Partitioning the Design

---

Stripe Position	How Stripes Are Pushed Down
Stripe completely inside partition boundary	<ul style="list-style-type: none"><li>■ Top-level:<ul style="list-style-type: none"><li>□ On layers not reserved for partition, the stripe is retained at the top and is copied to the block-level design.</li><li>□ On layers reserved for partition, the stripe is pushed down to the block-level design.</li></ul></li><li>■ Block-level:<ul style="list-style-type: none"><li>□ On layers not reserved for partition, the stripe is retained at the top and is copied as two pins and one stripe.</li><li>□ On layers reserved for partition, the stripe is pushed down as two pins and one stripe.</li></ul></li><li>■ Block Abstract:<ul style="list-style-type: none"><li>□ On layers not reserved for partition, two pins are created at the edges.</li><li>□ On layers reserved for partition except the topmost layer, two pins are created at the edges.</li><li>□ On the topmost layer reserved for partition, one big LEF pin is created.</li></ul></li></ul>
Stripe is partially inside Partition boundary.	<ul style="list-style-type: none"><li>■ Top-level: The stripe is retained on the top and is copied to the block-level design.</li><li>■ Block-level: The stripe is retained at the top and is copied as two pins and one stripe.</li><li>■ Block Abstract: The stripe is retained at the top and is copied as a big LEF pin.</li></ul>
Stripe is outside but close to boundary	<ul style="list-style-type: none"><li>■ Top-level: Stripe is retained at the top.</li><li>■ Block-level: Stripe is retained at the top and is copied as a routing blockage (same size as wire) with a +PUSHDOWN attribute.</li><li>■ Block-abstract: No effect.</li></ul>

## How Bumps, Routes, and Area I/O Cells Are Affected

This section illustrates how bumps and routes are handled when the design uses hierarchical partitioning with flip chip RDL routing and 45-degree routes. This information pertains to the [partition](#) command.

**Note:** In the releases of Encounter prior to 6.1, the area I/O cells had to be part of the top-level netlist; otherwise, DRC violations were reported during block implementation. From the 6.1 release onwards, the area I/O cells can be at the top level or be a part of the partition netlist. This section describes the behavior for both the cases.

After the partition, LEF obstruction is cut against the overlapping bumps at the top. This is done for all the bumps (power/gnd/signal/unused). Similarly the routing blockages inside the partition is cut against the pushed down bump.

The following scenarios are discussed:

- [Area I/O Cells are Part of the Top-level Netlist](#)
- [Area I/O Cells are Part of the Partition Netlist:](#)
  - [Bumps and Routing are on Top Routing Layer—Behavior with the stripStayOnTop parameter](#)
  - [Bumps and Routing are on Reserved Routing Layer—Behavior with the stripStayOnTop parameter](#)
  - [Bumps and Routing are on Top Routing Layer—Default Behavior](#)
  - [Bumps and Routing are on Reserved Routing Layer—Default Behavior](#)

### Area I/O Cells are Part of the Top-level Netlist

When area I/O cells are part of the top-level netlist, signal bumps and routes remain bumps and wires at the top level, but become routing blockages at the partition level. This allows routing at the block level while preserving the space for the signal bumps and routes.

Power and ground bumps and routes are copied and pasted (duplicated) from the top level to the partition. This allows power analysis at the block level. When the design is flattened, the duplicate power and ground bumps and routes are removed from the block level.

	<b>Top Level</b>	<b>Partition Level</b>
Area I/O cell	Area I/O cell	Placement and Routing Blockage

	<b>Top Level</b>	<b>Partition Level</b>
Signal bump	Signal bump	Placement and Routing Blockage
Signal route	Signal Route	Routing blockage
Power and ground bump	Bump	Bump (copied and pasted)
Power and ground route	Route	Route (copied and pasted)

### **Area I/O Cells are Part of the Partition Netlist**

When area I/O cells are part of the partition netlist, the pushdown behavior depends on:

- whether the `stripStayOnTop` parameter has been specified with the [partition](#) command.
- whether the bumps and routing are on the top routing layer or the reserved routing layer



In this case (that is, area I/O cells are part of the partition netlist), the behavior applicable to area I/O cells is also applicable to any other instance to which the bump is logically connected.

If the area I/O cell and the bump connection pass through a partition pin, the pin will not be assigned when you assign partition pins. These partition pins are assigned only when you run the [partition](#) command. If the bump overlaps the partition, a partition pin is created, with a geometry similar to that of the bump. If the bump does not overlap the partition, the pin is created during special route pushdown. The pin is created on the partition boundary where the routes between the bump and the area I/O cross the partition boundary.

For floating partition pins that are connected to a bump, the [assignPtnPin](#) command will check if the bump physically overlaps with the partition. If so, the command will not assign the pin and a partition pin is created, with a geometry similar to that of the bump, only when the [partition](#) command is run. Otherwise, the pin is assigned on the partition boundary by [assignPtnPin](#) command.

The following sections discuss the behavior for the following cases:

- [Bumps and Routing are on Top Routing Layer—Behavior with the stripStayOnTop parameter](#)
- [Bumps and Routing are on Reserved Routing Layer—Behavior with the stripStayOnTop parameter](#)

- Bumps and Routing are on Top Routing Layer—Default Behavior
- Bumps and Routing are on Reserved Routing Layer—Default Behavior

**Note:** For all the listed scenarios, the push down behavior for signal routes is similar to the behavior described in the [How Top-level Stripes Are Pushed Down](#) on page 333.

***Bumps and Routing are on Top Routing Layer—Behavior with the `-stripStayOnTop` parameter***

The following table summarizes the behavior when the bumps and the routing are on the top routing layer and you run the `partition` command *with* the `-stripStayOnTop` parameter.

Object Type	Top Level	Partition Level
Area I/O cell	An pin equivalent pin to the area I/O pin is created in the partition LEF file. This pin has the same size, location, and metal layer as the area I/O pin.	Area I/O cell is retained in the partition netlist
Signal bump	Signal bump stays on top and, additionally, an equivalent pin is created in the partition LEF file.	<ul style="list-style-type: none"><li>■ If the bump overlaps fully or partially with the partition, and connects to the partition:  An equivalent pin for the signal bump is created in the partition LEF file. This pin has the same size, location, and metal layer as the bump.</li><li>■ If the bump overlaps with the partition but is <i>not</i> connected to the partition:  The signal bump is pushed down as a routing blockage.</li></ul>

## Encounter User Guide

### Partitioning the Design

---

Object Type	Top Level	Partition Level
Signal Routes	Signal Routes routed on the top routing layers stays at top.	<ul style="list-style-type: none"><li>■ If the signal route overlaps the partition and is also connected to a area I/O cell inside the overlapping partition and a signal bump at the top, the signal route is copied and pasted to the partition. The pushed down net will be the internal net in the partition and will be named based on the partition port it is connected to inside the partition.</li><li>■ If the signal route overlaps the partition to which it is not connected (that is, it is not connected to any instance inside the partition but to a bump at top), these routes are copied and pasted as routing blockages inside the overlapping partition.</li></ul>

**Bumps and Routing are on Reserved Routing Layer—Behavior with the `-stripStayOnTop` parameter**

The following table summarizes the behavior when the bumps and the routing are on the reserved routing layer and you run the [partition](#) command with the `-stripStayOnTop` parameter.

Object Type	Top Level	Partition Level
Area I/O cell	Not applicable because area I/O cell is already part of the partition netlist.	Area I/O cell is retained in the partition netlist.
Signal bump	Signal bump stays on top and, additionally, an equivalent pin is created in the partition LEF file.	Bumps get pushed down to the partition as an equivalent pin in the partition DEF file.
Signal route	Signal routes are removed from the top.	Routing gets pushed down inside the partition block.

**Bumps and Routing are on Top Routing Layer—Default Behavior**

The following table summarizes the default behavior when the bumps and the routing are on the top routing layer. The default behavior in this context refers to the behavior that occurs when you run the [partition](#) command without the `-stripStayOnTop` parameter.

Object Type	Top Level	Partition Level
Area I/O cell	Not applicable because area I/O cell is already part of the partition netlist.	Area I/O cell is retained in the partition netlist.

## Encounter User Guide

### Partitioning the Design

---

Object Type	Top Level	Partition Level
Signal bump	Bump Stays at Top. An additional Bump pin is created in partition LEF file.	Bumps get pushed down inside the partition block as an equivalent pin in partition DEF file.
Signal route	Signal routes are removed from the top.	Routing gets pushed down inside the partition block The routes on top routing layer are cut from the top and pasted inside the partition. For details, please refer to <a href="#">How Top-level Stripes Are Pushed Down</a> on page 333.

### ***Bumps and Routing are on Reserved Routing Layer—Default Behavior***

The following table summarizes the default behavior when the bumps and the routing are on the reserved routing layer.

Object Type	Top Level	Partition Level
Area I/O cell	Not applicable.	Area I/O cell is retained in the partition netlist
Signal bump	Signal bump stays at top. An additional bump pin is created in the partition LEF file.	Bumps get copied down inside the partition block as a pin.
Signal route	Signal routes are removed from the top.	Routing gets pushed down inside the partition block.

## Limitations

- The pushdown of the signal bumps as an equivalent pin inside the partition is not supported for the non-rectangular shapes of the bump cell.
  - If the pushed down area I/O cell has pin shapes on the top routing layers, the blockages created on the top routing layers are not cut against these component pins.
  - If the signal routes are pushed down to the partition, any routes that do not overlap with the partition but lie close enough to the partition boundary and may thus result in spacing violations at chip assembly, will be pushed down as blockage inside the partition. This may result in some blockages being pushed down to the partition but outside the partition box.

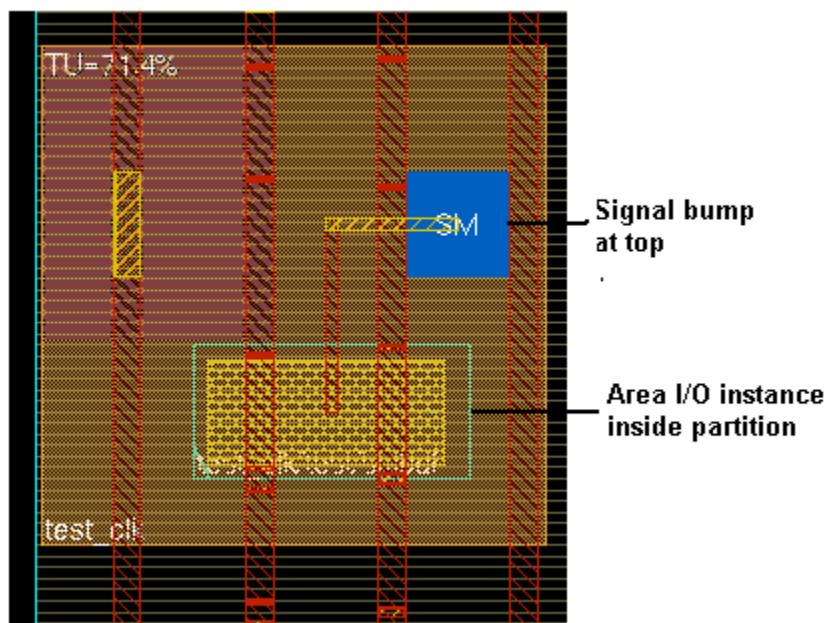
The following examples illustrate the behavior:

- Case 1: All Routing Layers Reserved for the Partition
  - Case 2: Top Layer Not Reserved for Routing

### **Case 1: All Routing Layers Reserved for the Partition**

The design has six routing layers. All the layers are reserved for the partition. Signal Bump SM is connected to area I/O cell inside the partition.

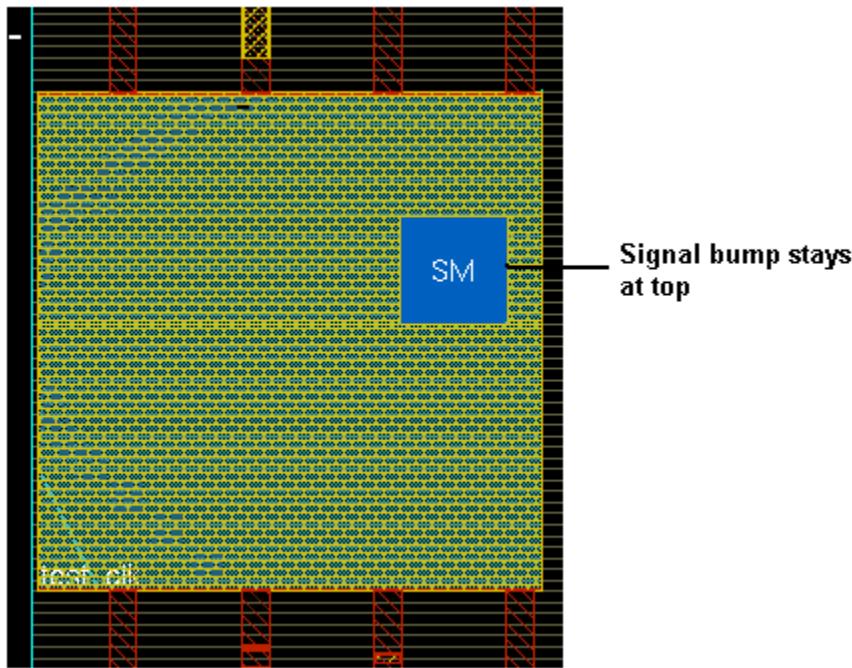
The following diagram shows the floorplan view before partitioning.



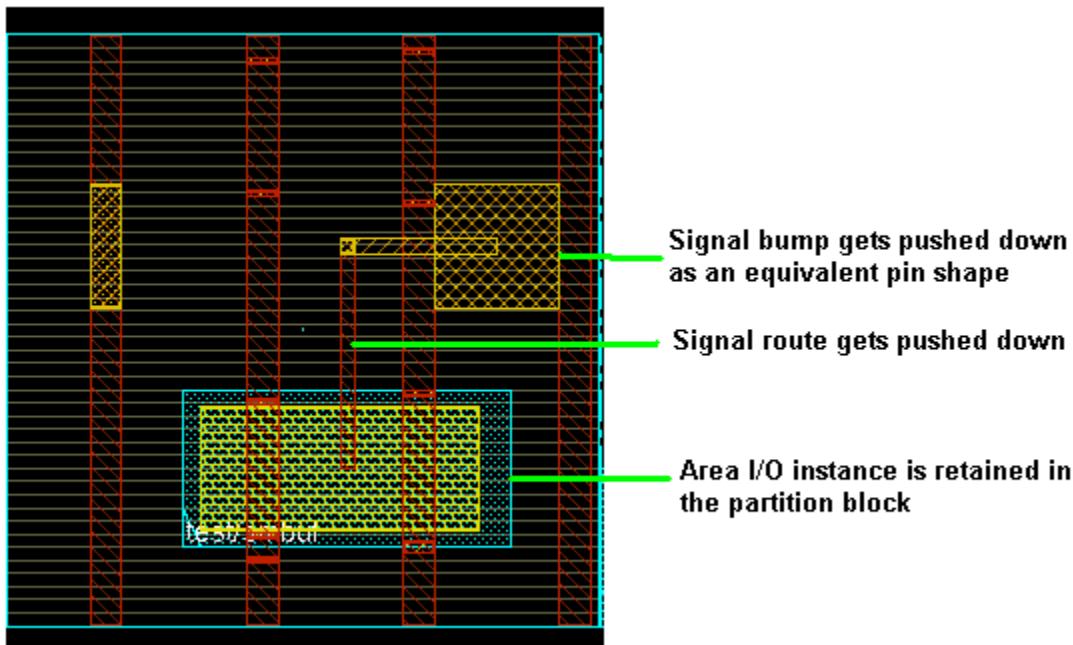
## Encounter User Guide

### Partitioning the Design

The following figure shows the view at top after partitioning.



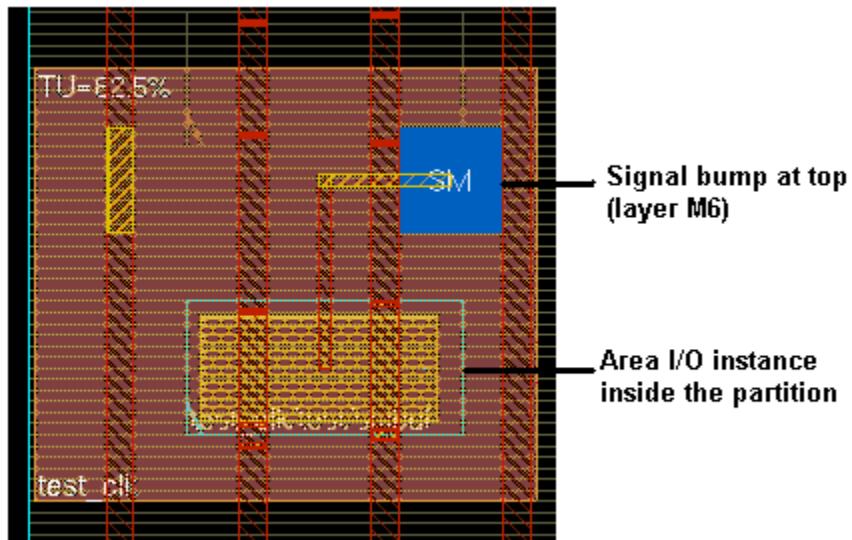
The following figure shows the view inside the partition



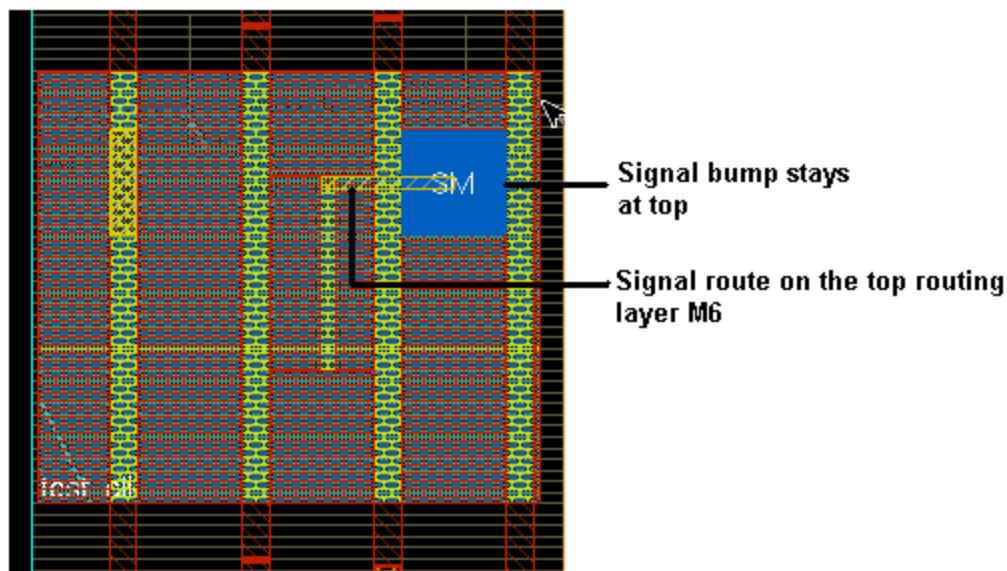
### **Case 2: Top Layer Not Reserved for Routing**

The design has six routing layers. Layers M1-M5 are reserved for the partition. M6 is the top routing layer.

The following diagram shows the floorplan view before partitioning.



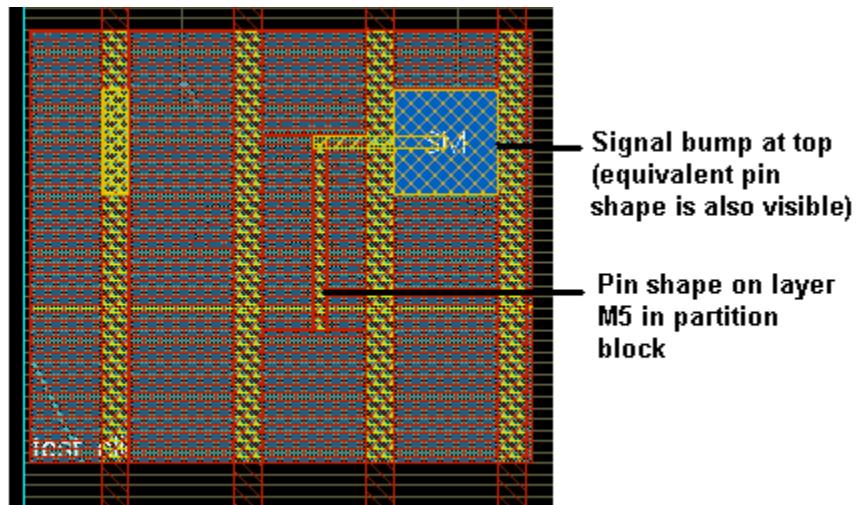
The following diagram shows the view at the top after partitioning with the `-stripStayOnTop` parameter specified.



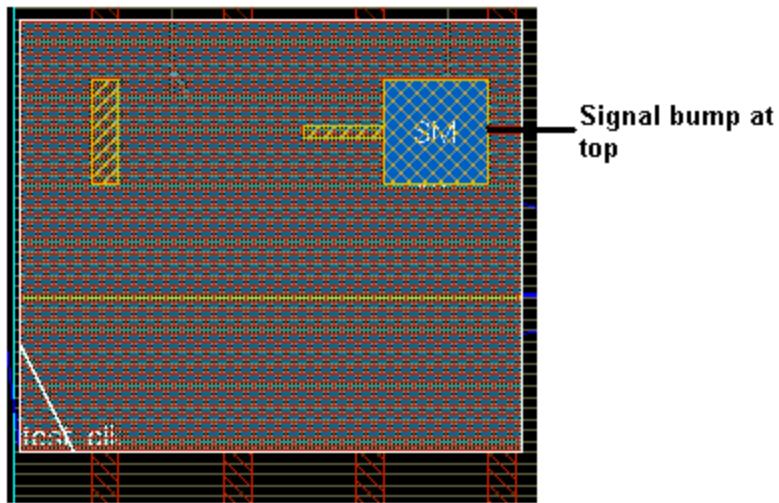
## Encounter User Guide

### Partitioning the Design

The following figure shows the view on the top after partitioning with the pins visible.



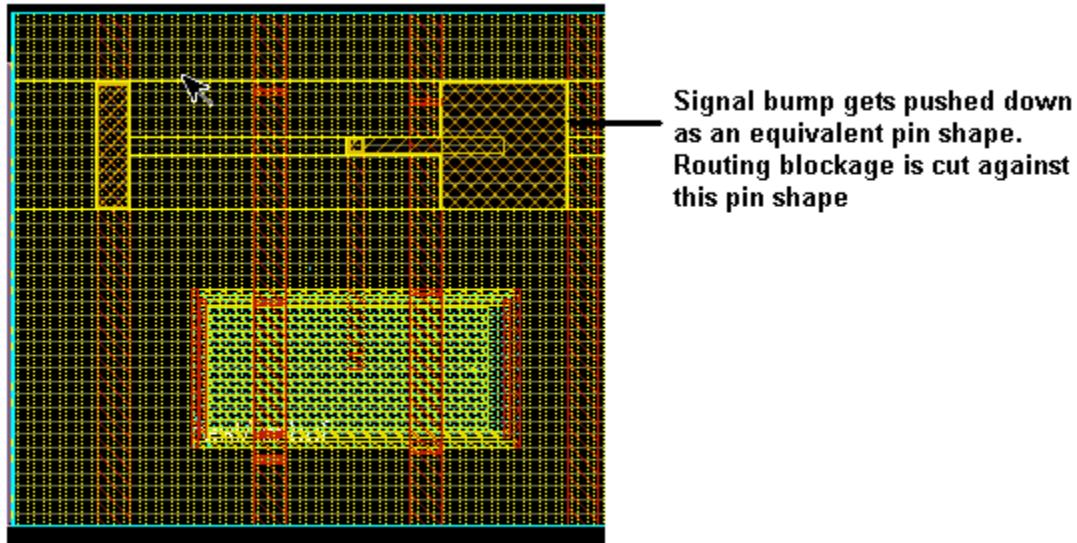
The following figure shows the view on the top after partitioning *without* the -stripStayOnTop parameter specified.



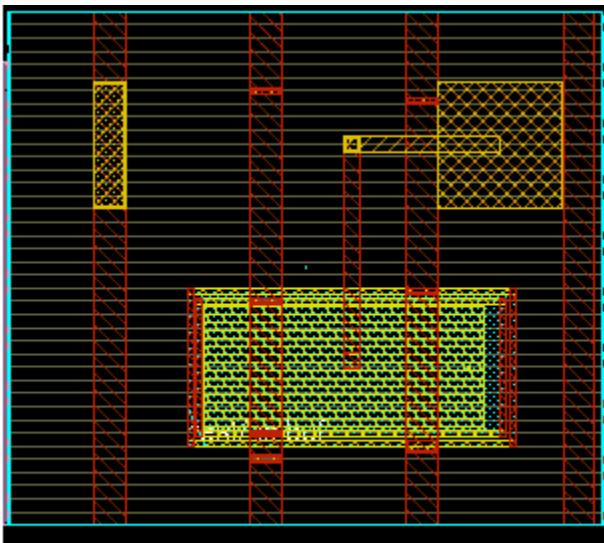
## Encounter User Guide

### Partitioning the Design

The following figure shows the view on the top after partitioning with visible routing blockages on layer M6.



The following figure shows the view inside partition with the display of the routing blockages turned off

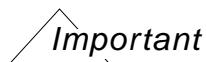


**Note:**

## Restoring the Top-Level Floorplan with Partition Data

1. Import the entire design from the top-level directory that was created or updated when you saved the partition.

If any portion of the design (top level or any partition) was changed by running scan optimization, CTS, or IPO, the changed netlist of the entire design is imported, not the original netlist. This changed netlist is usually created by concatenating each of the partition netlists to the top-level netlist. To do this, use a text editor to manually edit it, or use the Design Import form to create a single Verilog netlist of the entire design (see ["Concatenating Netlist Files of a Partitioned Design" on page 348](#)).



If a tool changes the partition netlist, you must update the full chip netlist. Some routers, such as NanoRoute™, might modify the partition netlist. The Encounter software requires that the full chip netlist, loaded during Design Import, be consistent with the routed partition netlist.

2. Load the top-level floorplan used to partition the entire design.
3. Set the Partition forms with *Perform Pin Assignment* deselected and partition the design.
4. Load the top-level placement data from the top-level directory.
5. Choose *Design – Load – Partition*.

This opens the Load Partition form.

6. In the Load Partition form, enter the directory name where the partition data was saved.
7. Click OK.



In place of steps 5, 6, and 7, you can use the `setTopCell` command to restore the partition and top-level data for the entire design. This is especially useful for restoring placement data from a DEF or TDF file.

**Note:** To perform a full chip analysis or a timing budget refinement analysis, use the Unpartition form to flatten the design.

## Concatenating Netlist Files of a Partitioned Design

To create a single Verilog netlist of the entire design, including the top level and all the partitions, complete the following steps:

1. Start a new Encounter session.
2. Choose *Design – Design Import* to open the Design Import form, and click the *Design* tab if it is not selected.
3. In the *Verilog Files* field, enter each netlist filename in the order from top-level netlist followed by the partition netlist files.  
**Note:** The partition netlist are read from each of the partition's work directories.
4. Click *OK*.
5. Choose *Design – Save – Netlist* to open the Save Netlist form.
6. Enter a Verilog file netlist name in the *Netlist File* field.
7. Click *OK*.
8. Use the saved Verilog file to restore the top-level floorplan with partition data (see [“Restoring the Top-Level Floorplan with Partition Data” on page 347](#)).

## Saving Partitions

You can save partition results, including the top-level partition, to their own subdirectories so that each partition can be worked on concurrently. Each partition directory contains all files necessary to run the Encounter software. Files necessary to run back-end tools in DEF, PDEF, and TDF formats can be selected when saving partitions.

To save a partition, use the [Save Partition](#) form or the `savePartition` text command.



***Do not use the Save Design form to save a partition.***

You can also save partitions in the OpenAccess database format. For more information, see [Working with OpenAccess Database](#) on page 351

## Loading Partitions

After completing the design work for each partition and the top level, you can restore a partitioned design to the top level, which includes loading all the partition design directories and its data.

To restore a saved partition design, use the [Load Partition](#) form.

## Unpartitioning with Routing Data

When loading a partition, it is important that the loaded routing results correctly correspond to the new netlist. To ensure that the netlist and routing file are consistent, you need to unpartition with the routing data using the following steps:

1. Load the original flat design.

This is the original design before running the partition steps.

2. Specify the partition and save to a file (to be loaded later in step 7).
3. Run partitioning with pin assignment.
4. Save the partitions and the top level.
5. For each partition and the top-level, run the block-level implementation with the following commands:
  - `encounter`

- `restoreDesign` (for the block or top level)
  - `trialRoute` or `nanoroute` or `wroute`
  - `saveRoute`
- 6.** Load the original design (the same design loaded in step 1).

If the netlist has been modified after step 1 (for example, in the case where a netlist is modified after in-place optimization or running NanoRoute) use the updated netlist instead.

To specify the updated netlist, you must first specify top-level netlist, then the block-level netlists in the *Verilog Files* field of the Design Import form's *Design* page. For example, `top.v block1.v block2.v ...`

 **Important**

The netlist and routing must be consistent when loading a partition with routing data, be sure you load the design with floorplanning, placement, and routing data that is consistent with the data saved in step 4.

- 7.** Load the partition file (specified in step 2).
- 8.** Run partitioning without pin assignment.
- 9.** Load the partition data.

For each partition, select the partition, then change the partition view (using the *Partition – Change Partition View* menu command) and load all the data for the viewed partition. You can use either the DEF file, or the .fp, .place and .route files.

- 10.** Reset the view back to the top level (using the *Partition – Change Partition View* menu command).

- 11.** Load the top-level data.

You can read in the top-level physical information by either using the DEF file or the placement (.place) and routing (.route) file. You must not read in the floorplan (.fp) file again because the floorplan information was already read in at the very beginning.

**Note:** Top-level physical information can only be loaded using DEF.

- 12.** Unpartition the design (`flattenPartition`).

## Working with OpenAccess Database

You can save and load designs using the OpenAccess database. The following commands and parameters are used for OpenAccess database designs.

- The savePartition command can save files in OpenAccess database format:
  - `-oaPtnLib`  
Specifies an OpenAccess directory library name where the top-level and the block-level designs will be saved.
  - `-oaPtnView`  
Specifies a view name for the top view and the partition view.
  - `-refLibs`  
Specifies a list of reference libraries.
- The assembleDesign command supports assembling the saved OpenAccess format files.
  - `-topDesign`  
Specifies the top-level name.
  - `-block`  
Specifies the block names.
- The updateBlock command can bring back block information from OpenAccess database files.
  - `-topDesign`  
Specifies the top-level name.
  - `-block`  
Specifies the block names.

The general flow for designs that use an OpenAccess database is the same as described throughout this chapter.

The following command saves the partition information/files in the OpenAccess database format. The information for the top and the block level designs (all blocks) will be written in the `libForOA` directory view with the view name `ptnView1`.

```
savePartition -oaPtnLib libForOA -oaPtnView ptnView1
```

The following command assembles the design after bringing back information from the top-level cell DTMF and block-level cells TDSP\_CORE and TDSP\_ARB.

```
assembleDesign -topDesign libForOA DTMF ptnView1 -block libForOA TDSP_CORE ptnView1  
-block libForOA TDSP_ARB ptnView1
```

For ECO flow, the updateBlock command can be used to bring back the information from the top- and block level-cells. Here is an example:

```
updateBlock -topDesign libForOA DTMF ptnView1 -block libForOA TDSP_CORE ptnView1  
-block libForOA TDSP_ARB ptnView1 -all
```

## Parallel Job Processing

With parallel processing, you can distribute jobs using a remote shell (rsh) or load sharing facility (LSF), specify host names for running jobs, and specify job information, such as block working directories and their run scripts.

The following procedure provides the most common steps for parallel job processing:

1. Import the design.
2. Floorplan the design.
3. Assign pins.
4. Run Timing Budgeting.
5. Partition the design.
6. Save the partition
7. Run parallel job processing to implement the blocks.

You can use the Run Parallel form for this.

For more information, see:

- the description of the Run Parallel form in the “Partition Menu” chapter of the *Encounter Menu Reference*
- Multiple CPU Processing in the “Design Menu” chapter of the *Encounter Menu Reference*.

---

## Floorplanning the Design

---

- [Overview](#) on page 354
- [Common Floorplanning Sequence](#) on page 355
- [Viewing the Floorplan](#) on page 357
- [Module Constraint Types](#) on page 360
- [Grouping Instances](#) on page 366
- [Creating and Editing Rows](#) on page 373
- [Using Vertical Rows](#) on page 373
- [Using Multiple-height Rows](#) on page 375
- [Performing I/O Row Based Pad Placement](#) on page 387
- [Resizing Rectilinear Blocks](#) on page 392
- [Using Blackblobs](#) on page 395
- [Editing Pins](#) on page 412
- [Running Relative Floorplanning](#) on page 423
- [Saving and Loading Floorplan Data](#) on page 426
- [Resizing the Floorplan](#) on page 427

## Overview

Floorplanning a chip or block is an important task of physical design in which the location, size, and shape of soft modules, and the placement of hard macros are decided. Depending on the design style or purpose, floorplanning can also include row creation, I/O pad or pin placement, bump assignment (flip chip), bus planning, power planning, and more. For example, floorplanning is very important when preparing the design for timing closure and detailed routing. Floorplanning, in conjunction with placement and trial routing, can be an iterative design process.

The Encounter software provides a rich set of commands and GUI functions to floorplan your design interactively. There are also commands for creating an initial floorplan automatically, or, resize a finished floorplan while keeping relative placement of objects.

- For information on floorplan commands, see the [Floorplan Commands](#) chapter, in the *Encounter Text Command Reference*.
- For information on floorplan GUI, see the [Floorplan Menu](#) chapter, in the *Encounter Menu Reference*.

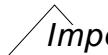
Encounter includes several keyboard shortcuts for use with the floorplanning feature. Make sure you type the bindkey while the main Encounter window is active and the cursor is in the design display area. The [Binding Key](#) form contains a complete list of bindkeys. To display this form, select *Design – Preferences* from the Encounter menu, then click the *Binding Key* button on the Preferences form, or use the default **b** binding key.

## Common Floorplanning Sequence

Floorplanning usually starts by replacing blocks, modules, and submodules according to the prepared floorplan. All other modules or blocks not in the prepared floorplan are left outside the chip area.

The following steps describe the most common sequence for floorplanning:

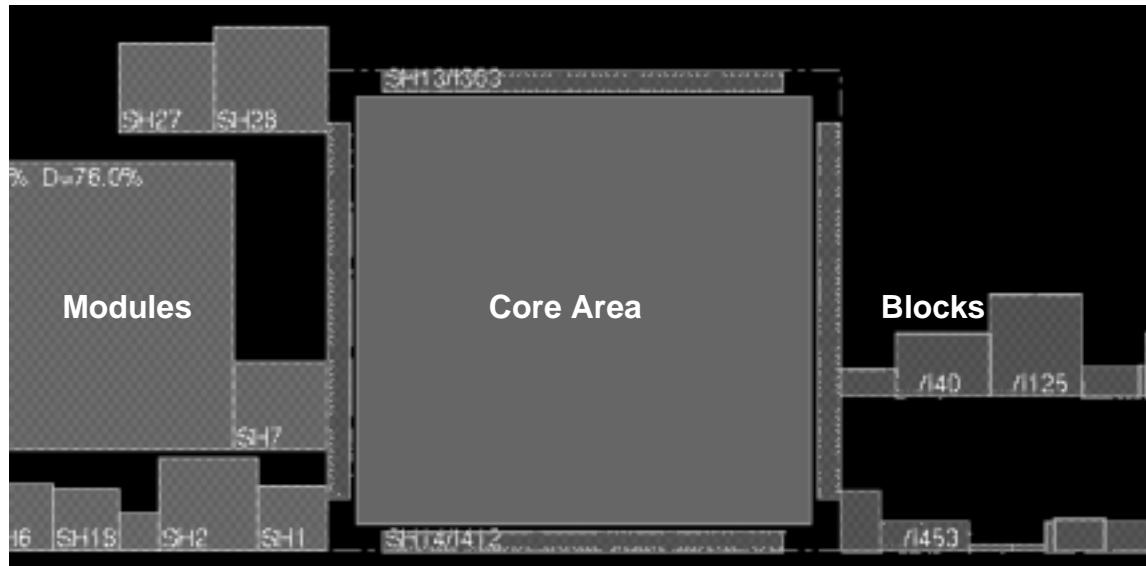
1. Importing the design.
2. Studying the design's connectivity.
3. Performing the minimum amount of floorplanning based on the chip design floorplan, or do no floorplanning at all.
4. In some cases, no floorplanning is required. For example, a front-end designer might want to predict the quality of the design's netlist by initially placing the entire design without any floorplanning. This iteration provides a good indication of how the blocks should be located and arranged together with the larger modules. After a few iterations, it should be clear how to position the blocks and modules in the floorplan.
5. Running placement and Trial Route to view placement and routing congestion.  
Optionally, running resize floorplan to enlarge or shrink the die after placement and routing. See [Resizing the Floorplan](#) on page 427.
6. In this case, floorplanning is done to detail the pre-placement of all blocks, most likely done by a back-end designer to gauge the feasibility of a prepared floorplan.
7. The placer places all remaining blocks that were not preplaced in the floorplan, and also recognizes the floorplan object, such as power and ground routes.
8. If you are at the design's top-level in the display area and want to generate a guide for a submodule, ungroup the top module until you have reached the submodule.
9. Using the full chip placement to refine block (hard macro and blackbox) locations.  
(Optional) Based on the full chip placement results—placement density and routing congestion, running resize floorplan to enlarge or shrink the die.
10. View the placements of blocks to determine if you need to change the alignment or orientations.
11. Looking for congestion in modules and change heavily congested modules' placement density to a lower percentage (using the `createDensityArea` text command).
12. (Optional) If you made any changes in step 5, or especially step 6, rerun placement.

 *Important*

To place macros that were not pre-placed during floorplanning, it is recommended that you run `planDesign` first and set the status 'fixed', before proceeding to `placeDesign`. This is because, `placeDesign` may not be able to place unfixed hard macros optimally.

## Viewing the Floorplan

In the design display area, the objects to the left of the core area are the top-level modules, which can be moved and reshaped. The objects to the right of the chip area are the blocks, which can be moved but not reshaped.

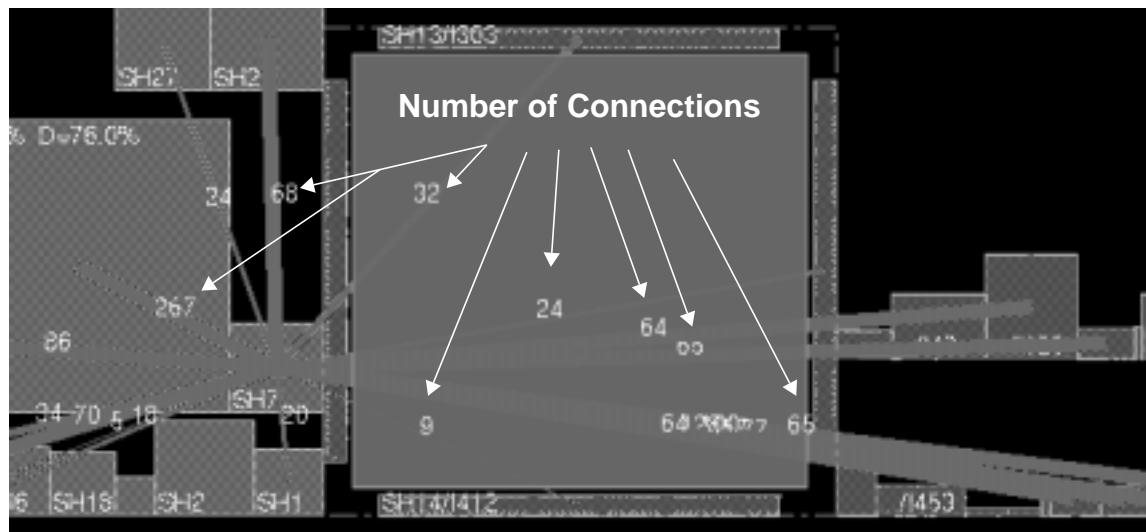


Use the **G** key (ungroup), or click the *Hierarchy Down* icon, to display the submodules for a selected module guide. Each time you use the **G** key, you move further down the hierarchy. Use the **g** key (group), or click the *Hierarchy Up* icon, to move up the hierarchy.

## Encounter User Guide

### Floorplanning the Design

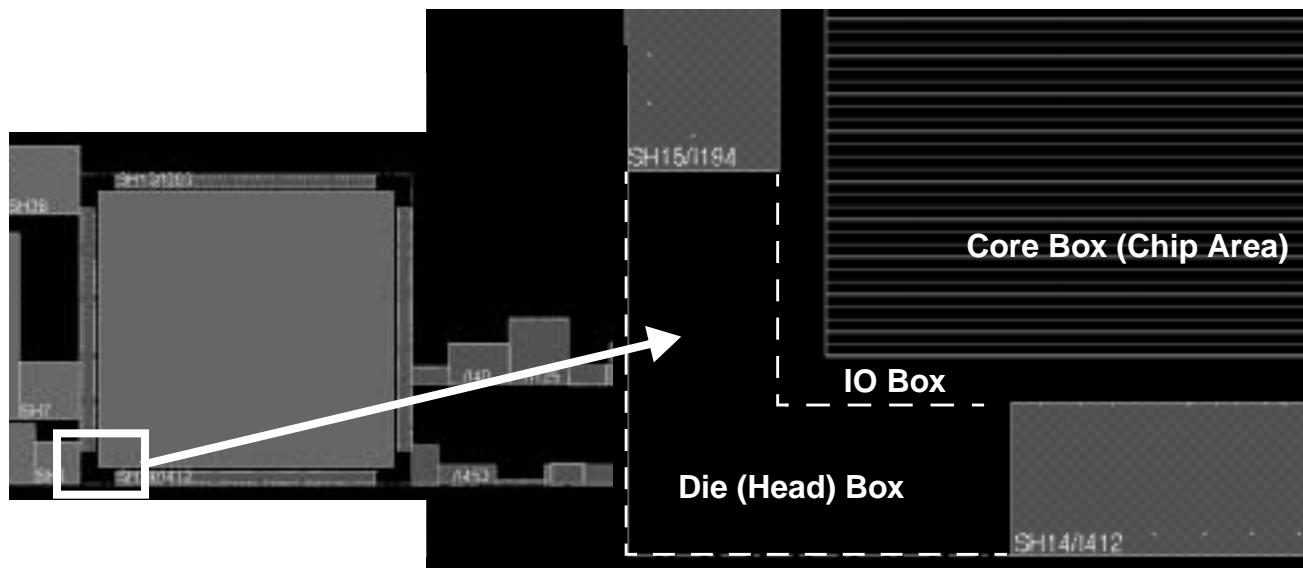
In Floorplan view, you can view the block pins and connection flight lines by clicking on a block or module. Flight lines show the connections and number of connections between the selected module or block to any other modules and blocks.



The pins for blocks are displayed where the flight lines terminate to help you orient the blocks so that the block pins face in the direction that best reduces routing congestion.

To set options for displaying flight lines in the design, use the *Flightline* page of the *Design – Preferences* form in the GUI.

You can change the die or core size; the margins between the core box and I/O pad instances; and the individual die (head), I/O, or core box sizes. These boxes are shown in the following figure.



You can move module or instance groups outside the core area.

## Module Constraint Types

The entire design size is initially calculated during design import, and each module size is calculated. The size of the modules are determined by either the core utilization or the core width and height specifications. The imported design modules can have one of the following constraint types:

- None—The module is not pre-placed in the core design area. The contents of the module are placed without any constraints.
- Guide—The module is preplaced in the core design area.

A module guide represents the logical module structure of the netlist. The purpose of a module guide is to guide placement to place the cells of the module in the vicinity of the guide's location. The preplaced guide is a *soft* constraint, which is discussed later in this section. After the design is imported, but before floorplanning, you can locate module guides on the left side of the core area, which appear as pink objects (by default) in the Floorplan view.

When a module is preplaced in the core design area, it snaps to a standard cell row in the vertical direction and to a *metal 2* pitch in the horizontal direction (the default). This default can be changed to snap to the manufacture grid (in the Preferences form's *Floorplan* page).

To create a guide for a module, or a group that contains hierarchical instances, instances (leaf instance), or other groups, use the [createGuide](#) command or select *Guide* from the Attribute Editor's *Constraint Type* pulldown menu.

- Fence—The module is a hard constraint in the core design area.

After specifying a hierarchical instance as a partition, the constraint type status of a module guide is automatically changed to a fence.

The physical outline of a fence module is rigid, and the design for the module is self-contained within the rigid outline. Only child instances must be contained within the partition physical outline; non-child blocks or modules that do not belong to the partition are excluded, and should not be pre-placed within another partition. This restriction is a hard restriction for third party back-end tools where the placement file for a partition does not match the partition netlist.

To create a fence for a module, or a group that contains hierarchical instances, instances (leaf instance), or other groups, use the [createFence](#) command or select *Fence* from the Attribute Editor's *Constraint Type* pulldown menu.

**Note:** Fence groups can potentially cause overlaps that cannot be corrected because the Encounter software cannot move the cells out of the group.

- Region—This constraint is the same as a fence constraint except that instances from other modules can be placed within its physical outline by placement. A module guide is changed to a status of Region when preplaced in the core design area.

To create a region for a module, or a group that contains hierarchical instances, instances (leaf instance), or other groups, use the [createRegion](#) command or select *Region* from the Attribute Editor's *Constraint Type* pulldown menu.

**Note:** Region groups can potentially cause overlaps that cannot be corrected because the Encounter software cannot move the cells out of the group.

- Soft Guide—This constraint is similar to a guide constraint except there are no fixed locations. This provides stronger grouping for the instances under the same soft guide. The soft guide constraint is not as restrictive as a fence or a region constraint, so some instances might be placed further away if they have connections to other modules.

To create a soft guide for a module, or a group that contains hierarchical instances, instances (leaf instance), or other groups, select *SoftGuide* from the Attribute Editor's *Constraint Type* pulldown menu.

## Target Utilization Display

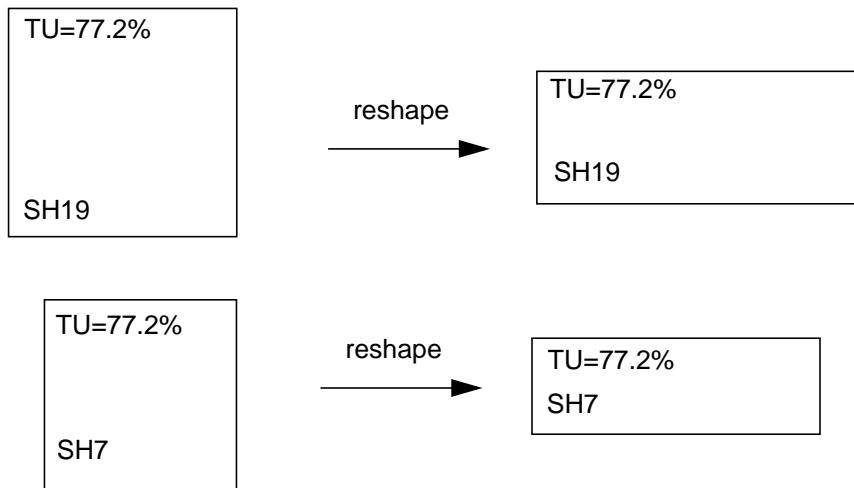
Module constraints display a target utilization (TU=%) value to represent their physical design size. This is an estimation of module utilization for the given size of the module where only standard cell and hard macro areas are considered; floorplan constraints, such as placement blockages, are not considered. This value is calculated by the standard cells area plus the hard macros area, divided by the module area. The initial TU values are calculated during design import.

The TU percentage helps judge the physical size of a module guide to customize the shape of the module in the floorplan. For example, modules SH19 and SH7 have a TU values of

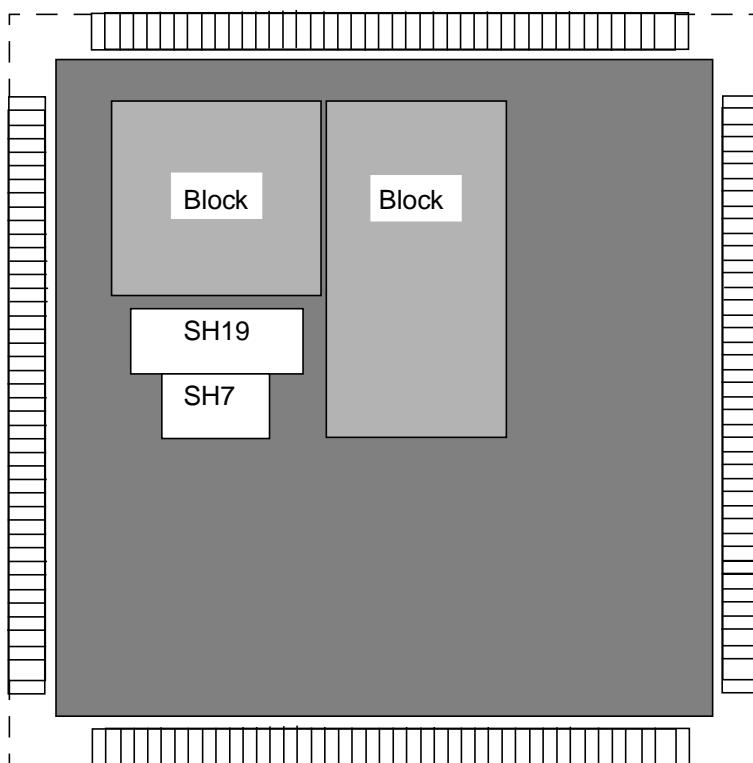
## Encounter User Guide

### Floorplanning the Design

77.2%. If the modules are reshaped with the same area, they retain their TU values, as shown in the following figure:



You can place them in the core area so they are preplaced close to one another, as shown in the following figure:



The position of the module guide is a placement constraint, and the final definition of the module is determined by several factors. The most important factor—the highest priority of

constraint—is the connectivity between itself and other modules. Other floorplan constraints, such as neighboring preplaced module guides, preplaced blocks, placement blockages, and routing blockages, are also considered, but at a lower priority than connectivity.

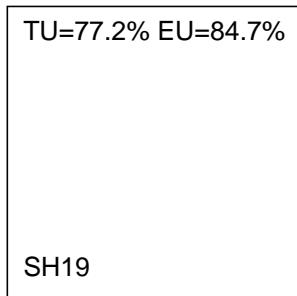
**Note:** You can use a stronger constraint for keeping modules SH19 and SH7 close together using the Group Instances form, and even a stronger constraint by saving the regrouped netlist.

Unlike module guides, the positions of fences and regions is a hard placement constraint and are not moved by the same factors.

## Effective Utilization Display

For fences and regions, you can display the effective utilization (EU=%) value. The EU value takes into account the actual cells and hard macros in the fence or region, placement or routing blockages, partition cuts, and other floorplan constraints. It is a good practice to update the EU value before running placement.

Click the *Display/Calculate Effective Utilization* toolbar widget (the % button above the design display area) to display the EU value for each fence and region, as shown in the following figure.



**Note:** The displayed EU values are not automatically updated. You must click the *Display/Calculate Effective Utilization* toolbar widget each time you want to display the updated EU value. This calculation could be time consuming, especially for larger designs.

**Note:** If the EU value is at or exceeds 100% for a fence or region, placement changes the fence or region to a guide. To avoid this, before you run placement, make sure to check and update the EU value, if necessary.

## Calculating Density

When specifying the floorplan, you can determine the core and module sizes by total density or standard cell density using the *Core Utilization* or *Std. Utilization* options, respectively, in the Specify Floorplan form.

Core Utilization determines the initial size of the core area and the initial size of the pink module guides off to the left of the die area. The total density is calculated as follows:

$$\text{Core Size} = (\text{standard cell area}/\text{core utilization}) + (\text{macro area} + \text{halo})$$

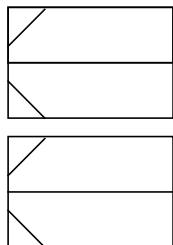
In determining the size of the core area and module guides, standard cells and hard macros are treated the same. However, you can determine how densely objects can be packed by weighing the standard cell density separately from the hard macro density. The standard cell density is calculated as follows:

$$\text{Core Size} = (\text{standard cell area} / \text{standard cell utilization}) + (\text{macro area} + \text{halo})$$

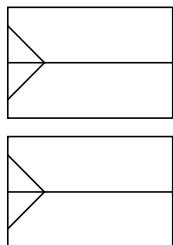
The size of the core is smaller once you specified your floorplan by using *Std. Utilization*.

## Standard Row Spacing

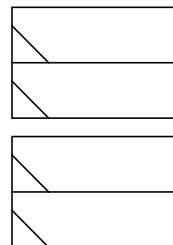
To configure the rows, use the `setFFPlanRowSpacingAndType` command, or the options from the *Standard Cells Rows* panel of the Specify Floorplan form. The following row configurations are supported:



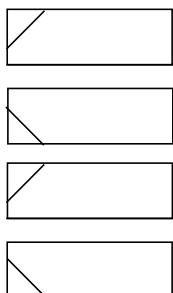
Bottom R0 and flip/abut



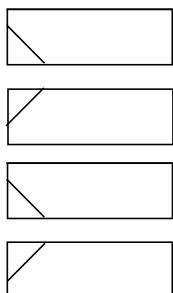
Bottom MX and flip/abut



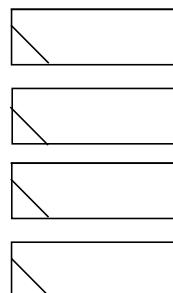
Bottom R0 and abut



Bottom R0 and flip



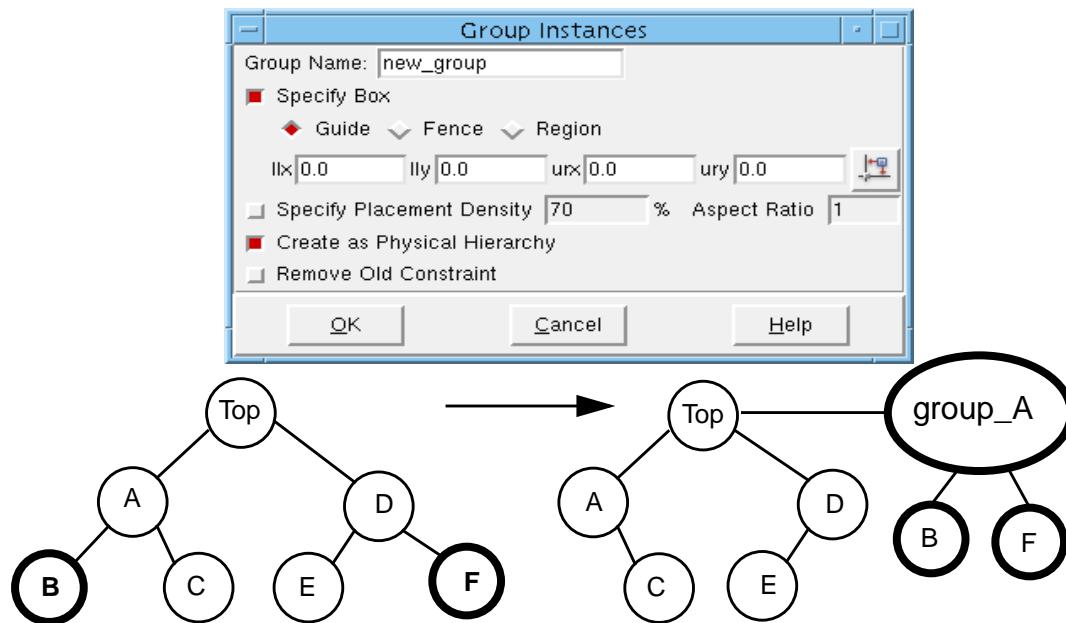
Bottom MX and flip



Bottom R0

## Grouping Instances

The hierarchy of the new instance group is formed at the common point of the modules and submodules. The following example shows how the hierarchy is changed from the common point if submodules B and F are added to a new group called *group\_A*.



To delete an instance from an instance group, complete the following steps:

1. Choose *Tools – Design Browser*.
2. In the Design Browser, click on and highlight the module or submodule guide(s) to be deleted from the instance group.
3. Click the *Delete Group/Group Member* icon.

To add an instance to an existing group name, complete the following steps:

1. Click on and highlight the module or submodule guide(s) to be added to an instance group.
2. Choose the *Floorplan – Create Physical Hierarchy* submenu to select the group name.

To save the instance group back to the netlist, use the Generate Regrouped Netlist form (*Floorplan – Generate Regrouped Netlist*).

## Defining the Bounding Box

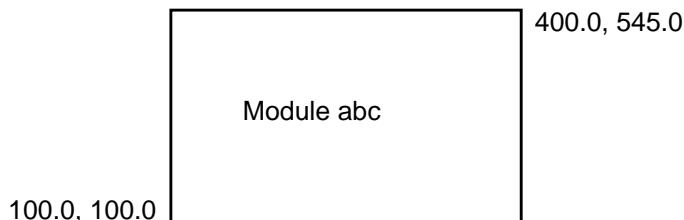
During floorplanning, you can use the `setObjFPlanBox` command to define a bounding box of a specified object, and the `setObjFPlanBoxList` command to define rectilinear shape of an object, which is comprised of two or more boxes.

This section provides graphical information to illustrate some of the command examples in the Floorplan Commands chapter of the *Encounter Text Command Reference*.

### **setObjFPlanBox**

The following command specifies a bounding box for Module abc at a lower left x coordinate of 100.0, a lower left y coordinate of 100.0, and upper right x coordinate of 400.0, and an upper right y coordinate of 545.0:

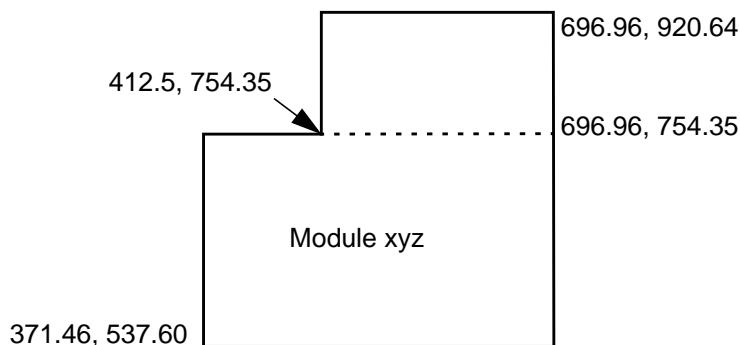
```
setObjFPlanBox Module abc 100.0 100.0 400.0 545.0
```



### **setObjFPlanBoxList**

The following command defines a rectilinear boundary for Module xyz. The rectilinear boundary is made up of two bounding boxes: (371.46, 537.60) (696.96, 754.35), and (412.5, 754.32) (696.96, 920.64):

```
setObjFPlanBoxList Module xyz 371.46 537.60 696.96 754.35 412.5 754.35 696.96  
920.64
```



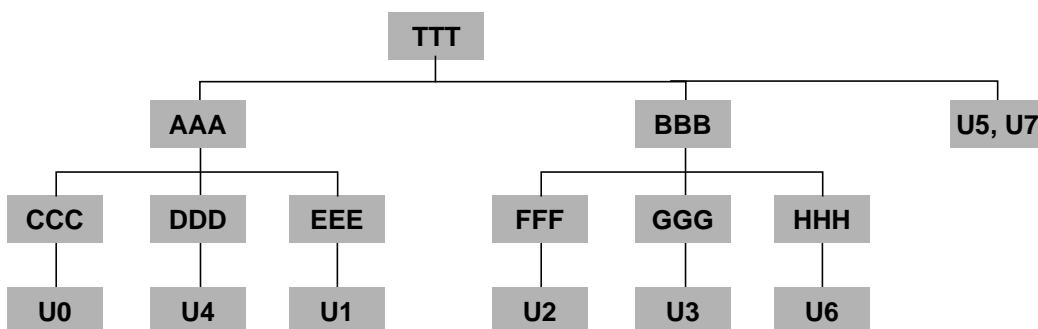
### **Adding Logical Hierarchy Without Creating Additional Hierarchy**

The Encounter software enables you to add logical hierarchy without creating additional hierarchy. For example:

```
createInstGroup /TTT -isPhyHier  
addInstToInstGroup /TTT U5  
addInstToInstGroup /TTT U7  
runRcNetlistRestruct
```

**Note:** The leading slash character (/) in /TTT is required for the software to create a temporary group named /TTT.

After restructuring, the result looks like this:



For more information, see the [runRcNetlistRestruct](#) command in the *Encounter Text Command Reference*.

## Logical Hierarchy Manipulation

In addition to [Adding Logical Hierarchy Without Creating Additional Hierarchy](#), you can also manipulate the logical hierarchy as follows:

- [Moving Instances to a New Top Module](#) on page 369
- [Moving Instances to an Existing Module](#) on page 370
- [Moving Instances to the Top Root Level](#) on page 371

For more information, see the [runRcNetlistRestruct](#) command in the *Encounter Text Command Reference*.

### Moving Instances to a New Top Module

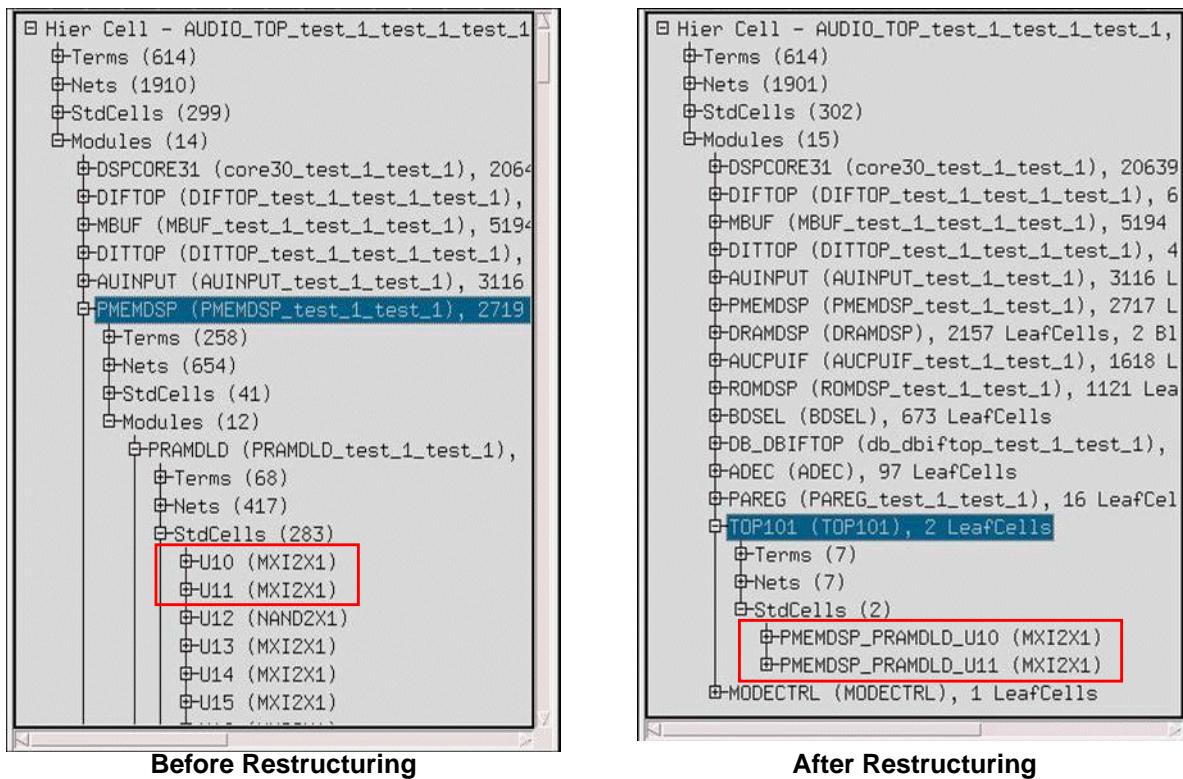
- To move an instance to a new top module named TOP101, you can do the following:

```
createInstGroup TOP101 -isPhyHier  
addInstToInstGroup TOP101 PMEMDSP/PRAMDLD/U10  
addInstToInstGroup TOP101 PMEMDSP/PRAMDLD/U11
```

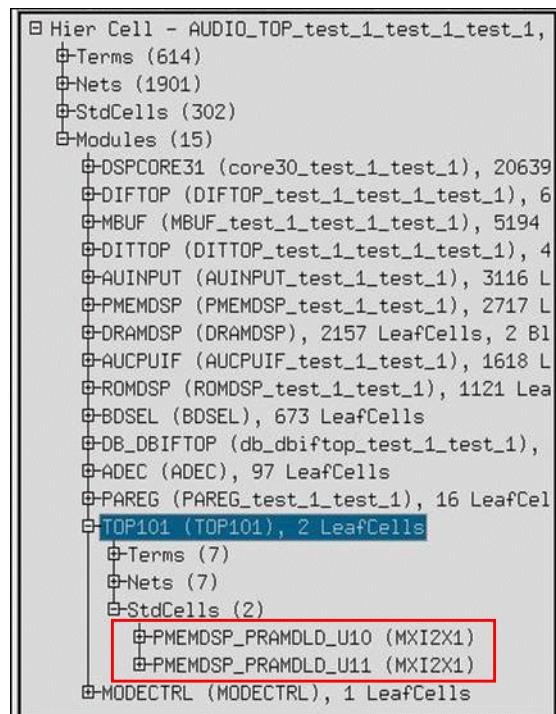
# Encounter User Guide

## Floorplanning the Design

runRcNetlistRestruct



Before Restructuring



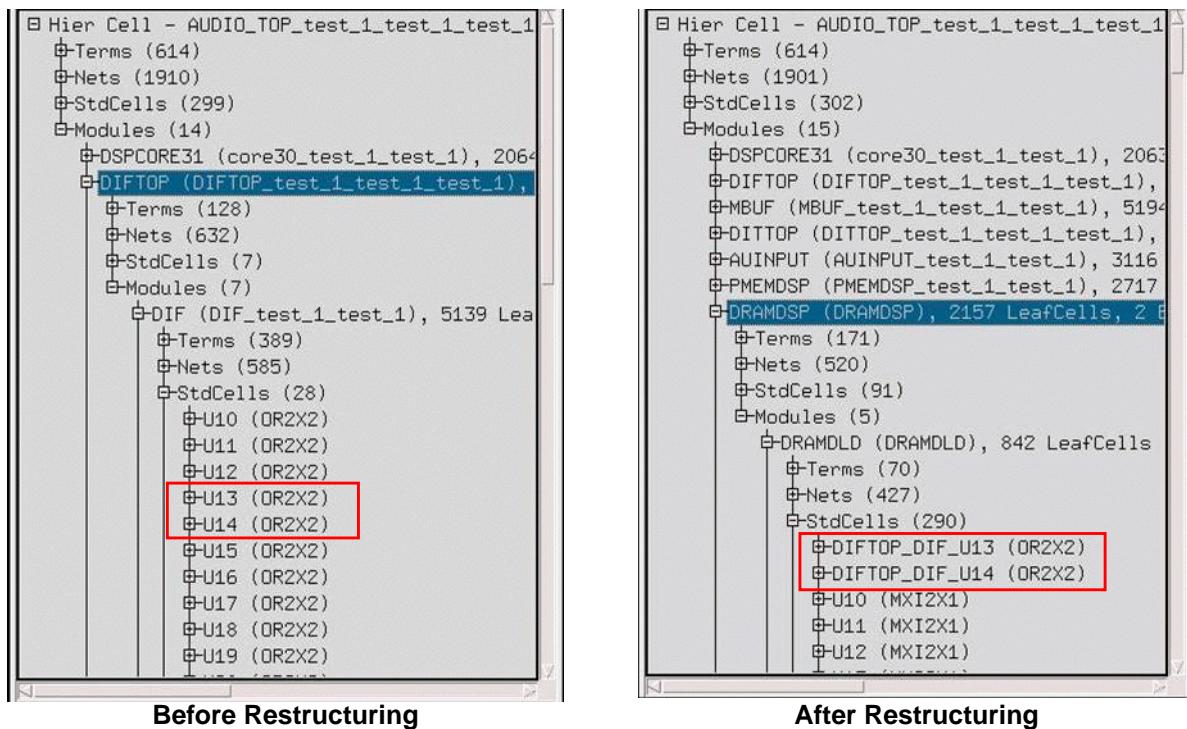
After Restructuring

### Moving Instances to an Existing Module

- To move an instance to an existing module named DRAMDSP / DRAMDLD, you can do the following:

```
createInstGroup /DRAMDSP/DRAMDLD -isPhyHier
addInstToInstGroup /DRAMDSP/DRAMDLD DIFTOP/DIF/U13
addInstToInstGroup /DRAMDSP/DRAMDLD DIFTOP/DIF/U14
runRcNetlistRestruct
```

**Note:** The leading / (slash) is required for an existing module.



## Moving Instances to the Top Root Level

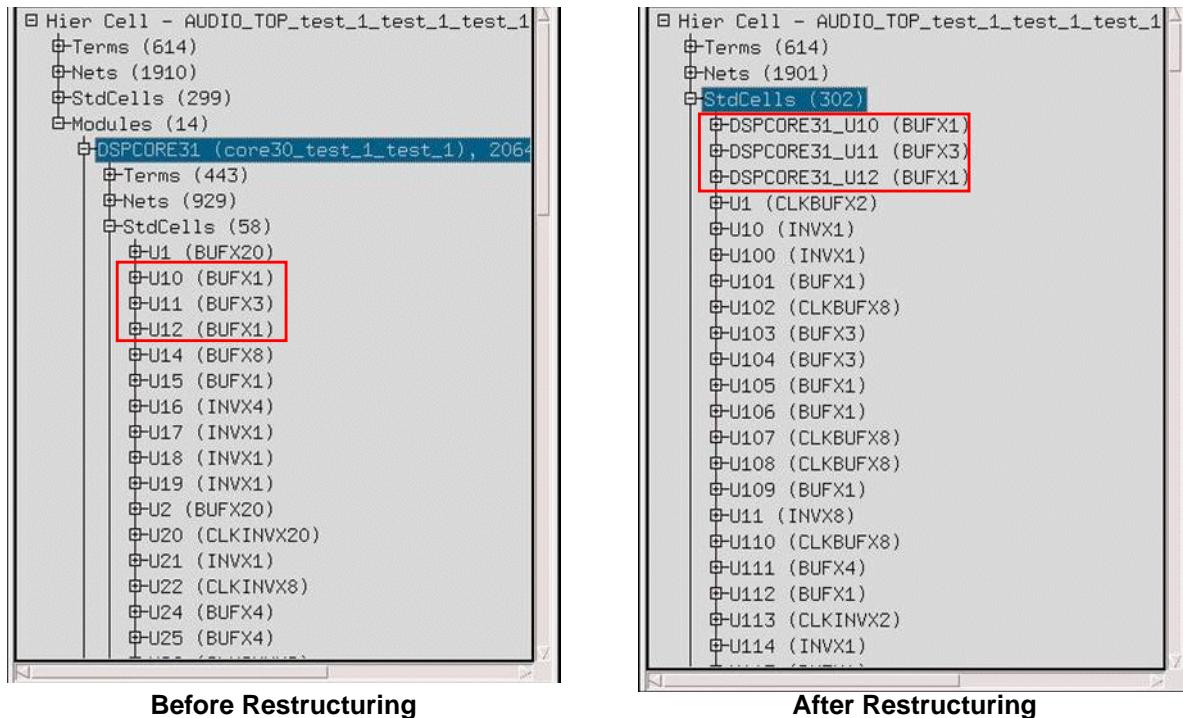
- To move an instance to the top root level, you can do the following:

```
createInstGroup /AUDIO_TOP_test_1_test_1_test_1 -isPhyHier  
addInstToInstGroup /AUDIO_TOP_test_1_test_1_test_1_DSPCORE31/U10  
addInstToInstGroup /AUDIO_TOP_test_1_test_1_test_1_DSPCORE31/U11  
addInstToInstGroup /AUDIO_TOP_test_1_test_1_test_1_DSPCORE31/U12  
runRcNetlistRestruct
```

# Encounter User Guide

## Floorplanning the Design

**Note:** The leading / (slash) is required for the top root level.



**After Restructuring**

## Creating and Editing Rows

You can create and edit rows in different regions. The following table lists the commands that you can use to create and cut rows.

<u>createRow</u>	Creates rows for the specified site. The row boundary can be defined by core area or the area that you specify. This command supports the creation of overlapping rows. This command can create only horizontal rows. By default, the rows are flipped and abutted.
<u>cutRow</u>	Cuts site rows that intersect with the specified area or object.
<u>deleteRow</u>	Deletes the specified row(s).
<u>stretchRows</u>	Stretches selected rows. For example, you can specify that the left edge of all selected rows should be aligned to the left-most edge among all selected rows.

For more information on using these commands, see the *Encounter Text Command Reference*.

You can also use the following menus available through the GUI:

- Create Core Rows, available through *Floorplan – Edit Floorplan – Core Row – Create*.
- Cut Core Rows, available through *Floorplan – Edit Floorplan – Core Row – Cut*.
- Stretch Core Rows, available through *Floorplan – Edit Floorplan – Core Row – Stretch*.

## Using Vertical Rows



***Support for vertical rows is a beta feature. Usage and support of this beta feature are subject to prior agreement with Cadence. Contact your Cadence representative if you have any questions.***

In addition to horizontal rows, Encounter also supports vertical rows. You can import designs with vertical rows and output design data containing the layout of vertical rows. Vertical rows appear vertically in the display. You can select and query vertical rows and delete them with

the delete key. The commands that snap rows to row grid also support vertical rows. These commands are [snapFPlan](#), [relativeFPlan](#), and the interactive move command.

Horizontal and vertical rows can co-exist, but at different layers of hierarchy. You cannot create horizontal and vertical rows together on the same level of hierarchy.

The configuration variable `ui_isVerticalRow` is used to specify whether the rows are horizontal or vertical. The variable can be set to 0 (for horizontal rows) or 1 (for vertical rows) as follows:

```
set rda_Input(ui_isVerticalRow) {0|1}
```

By default, Encounter imports designs with the rows horizontal, that is, the value of the variable is set to 0. To import designs with vertical rows, set this variable to 1.

While specifying a floorplan, you can specify that the rows are vertical by using the `-verticalRow` parameter of the [floorplan](#) command, or by specifying *Vertical Rows* in the *Row Direction* field of the [Specify Floorplan](#) form in the GUI.

The Encounter software generates vertical rows, provided the design data or the library meets the following prerequisites:

- The SITE width is more than the SITE height.
- Each SITE is stacked one above the other, when rows are created.
- MY and R0 row orientations are supported.
- The cells are stacked vertically, when they are placed and their power rails run vertically.
- The preferred direction of M1 is vertical.

## Limitations

The following limitations apply to vertical rows:

- Vertical rows cannot be created inside power domains.
- Non-integer and multiple integer vertical rows are not supported.
- Vertical rows cannot be created or edited directly. That is, the [createRow](#), [cutRow](#), [deleteRow](#), and [stretchRows](#) commands are not supported for vertical rows.

**Note:** You can, however, select rows to query, and delete rows with the delete key.

## Using Multiple-height Rows

In many cases, designs contain cells with different standard cell heights. For example, a design might utilize multiple standard cell libraries—possibly from different foundries or library vendors—which might have different standard cell heights.

Standard cell designers create multiple-height standard cells for improving performance. Also, in a design with multiple power domains, standard cells with different voltages will probably have different footprints and different heights.

The Encounter software supports multiple-height standard cells by supporting:

- a combination of integer multiple-height rows
- a combination of non-integer multiple-height rows

### Using Integer Multiple-height Rows

The Encounter software automatically generates integer multiple-height rows overlapping the single-height core rows provided the design data or the library meets the following prerequisites:

- The LEF file contains integer multiple-height SITE definitions and MACROS that use the SITE.
- The netlist includes at least one instantiation of such an integer multiple-height cell.

After you import the design or specify the floorplan, the core area is automatically populated with default rows and multiple-height rows are automatically generated.

Here is an extract from a sample LEF file that contains integer multiple-height SITE definitions and a MACRO that uses a SITE:

```
SITE coreSite
    SYMMETRY X Y ;
    CLASS CORE ;
    SIZE 0.660 BY 5.040 ;
END coreSite

SITE doubleHeightSite
    SYMMETRY X Y ;
    CLASS CORE ;
    SIZE 0.660 BY 10.080 ;
END doubleHeightSite
```

## Encounter User Guide

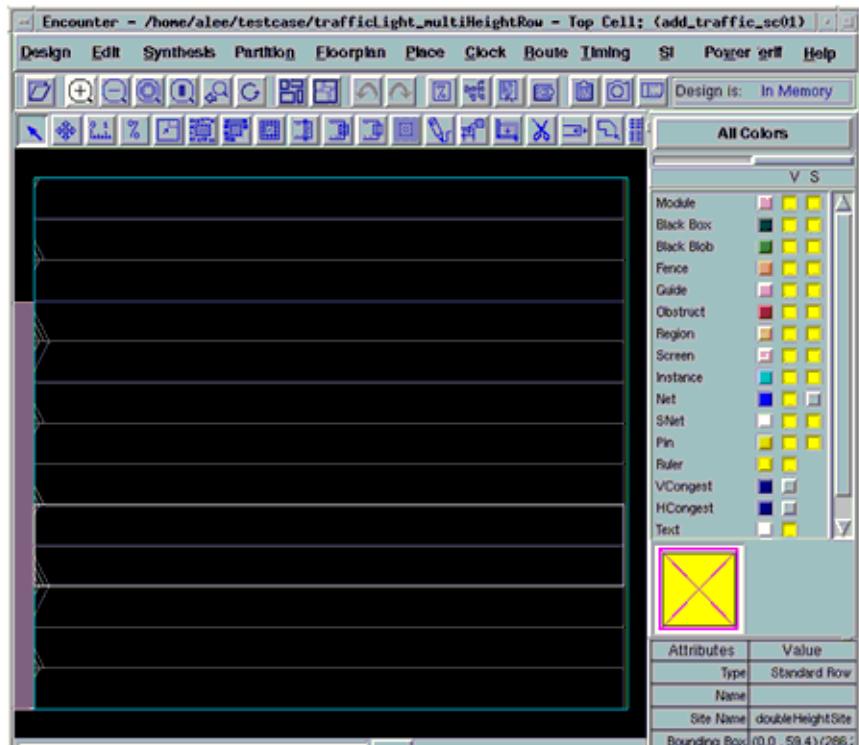
### Floorplanning the Design

```
MACRO DFFX64
  CLASS CORE ;
  FOREIGN DFFX64 0 0 ;
  ORIGIN 0 0 ;
  SIZE 21.12 BY 10.080 ;
  SYMMETRY X Y ;
  SITE doubleHeightSite ;
  ...
END DFFX64
```

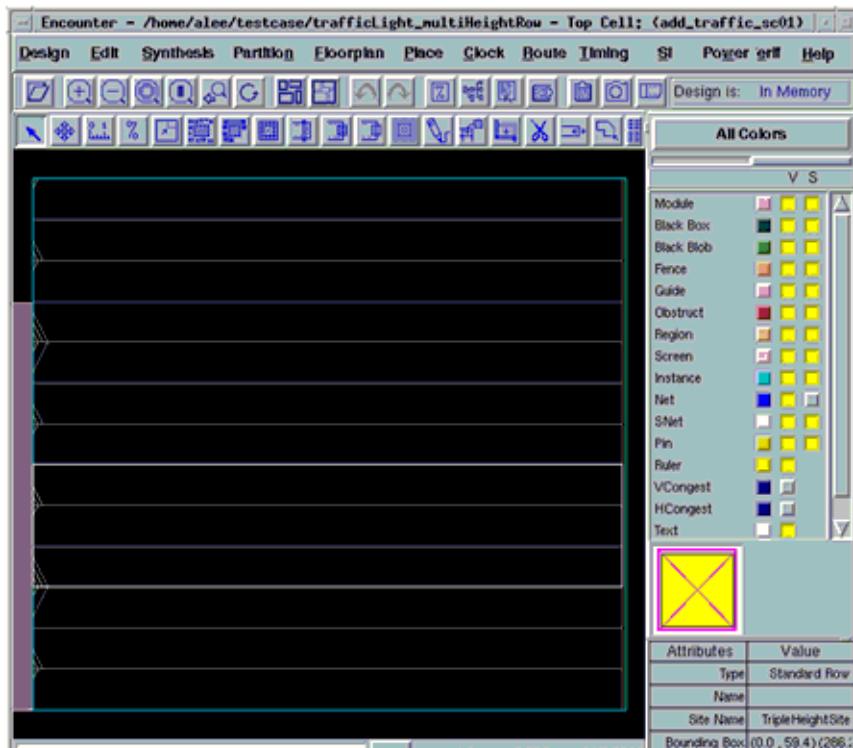
When you create integer multiple-height rows, the rows are automatically aligned with the single-height row. You cannot create unaligned integer multiple-height rows.

For information on creating and editing rows, see [Creating and Editing Rows](#) on page 373

The following figure illustrates a double-height row flipped to align with the orientation of the single row.



The following figure illustrates a triple-height row flipped to align with the orientation of the single row.



## Using Non-Integer Multiple-height Rows

You can also use non-integer multiple-height (NIMH) rows in your designs.

While creating NIMH rows, ensure the following:

- NIMH rows must be created *only* in those areas that have power domains associated with them.
- any newly created NIMH rows in an area must be an integer multiple of any existing rows in the area.

You can create a power domain using the `createPowerDomain` command or through the `Create Power Domain` form, available in the GUI through *Power – Multiple Supply Voltage – Create/Modify Power Domain*.

Each hierarchical instance to be declared as a power domain can only have one type of NIMH standard cells. In other words, NIMH rows inside any particular power domain must have the same height. Multiple types of NIMH standard cells require multiple power domains to be created. For example, if you want to use standard cells with heights that are respectively 2.5,

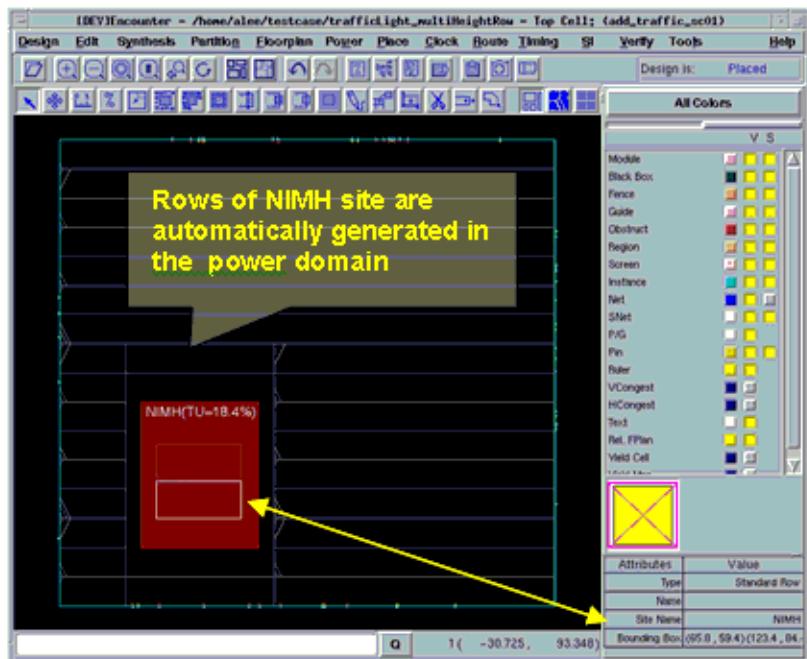
## Encounter User Guide

### Floorplanning the Design

3.25, and 4.1 times the height of a standard single-height cell, you should create three power domains, with each power domain containing one type of NIMH row.

When you create a power domain, Encounter automatically detects the SITE that is common to all the cells and creates the rows inside the power domain.

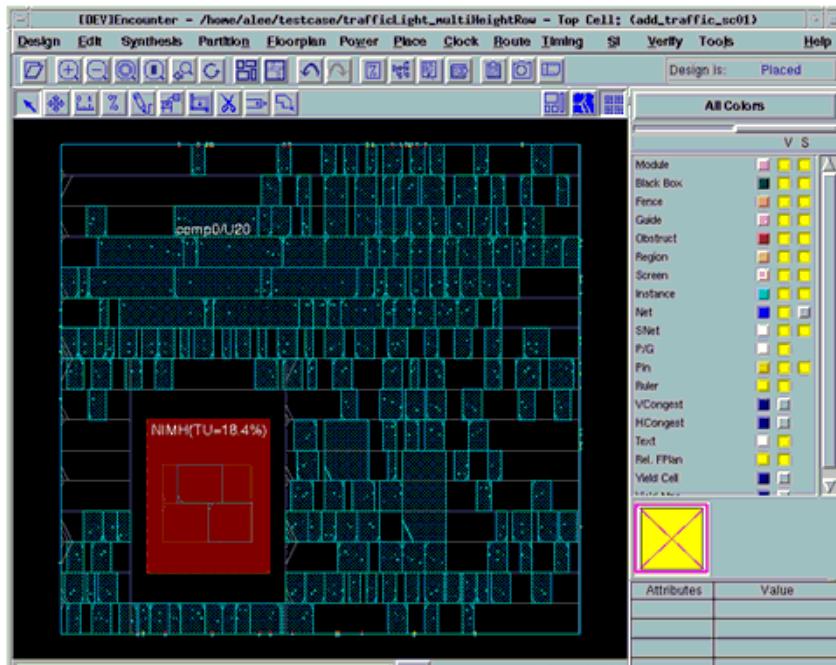
The following diagram illustrates how rows are automatically generated within the power domain for standard cells of the hierarchical instance.



## Encounter User Guide

### Floorplanning the Design

The following diagram illustrates how standard cells of the hierarchical instance are all placed on the row inside the power domain.



If NIMH standard cells of different height are included in one hierarchical instance, use the [createPowerDomain](#) and the [modifyPowerDomainMember](#) commands to ensure that the same types of NIMH cells are grouped within the same power domain.

For example, consider the case where the instances `inst1` and `inst3` can have NIMH rows of one height and the instances `inst2` and `inst4` can have NIMH rows of another height. The following commands create two power domains `NIMH1` and `NIMH2`, associate the instances `inst1` and `inst3` with the power domain `NIMH1`, and associate the instances `inst2` and `inst4` with the power domain `NIMH2`.

```
createPowerDomain NIMH1
createPowerDomain NIMH2
modifyPowerDomainMember NIMH1 -instances TOP/MixLevel/inst1
modifyPowerDomainMember NIMH2 -instances TOP/MixLevel/inst2
modifyPowerDomainMember NIMH1 -instances TOP/MixLevel/inst3
modifyPowerDomainMember NIMH2 -instances TOP/MixLevel/inst4
```

The modules and/or instance groups can be moved outside the core boundary but within the die. Since new rows are not created automatically in this area, you can use the [createRow](#) command to create new rows.

Manual editing of rows might not be preserved by [floorPlan](#), [resizeFP](#), and [initCoreRow](#) commands.

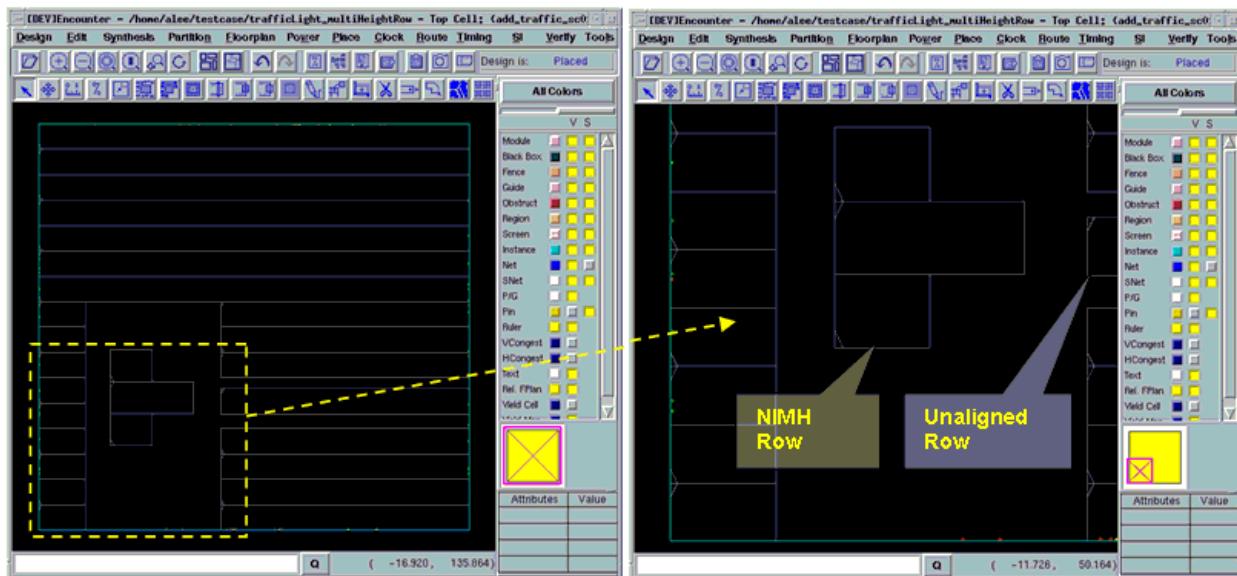
Encounter displays a warning message if you try to move a power domain outside the core boundary, and snaps the power domain inside the core.

## Working with User-defined DEF Files that Contain NIMH Rows or Unaligned Rows

In case of integer multiple-height rows, as long as the rows are overlapping the single-height rows, standard cells will by default be legally placed on their corresponding site or row definition.

However, if you import a user-defined plan through a DEF file that contains NIMH rows and unaligned rows, you need to define power domain(s) for each of these disjoint special row style. Otherwise, these rows might not be correctly placed.

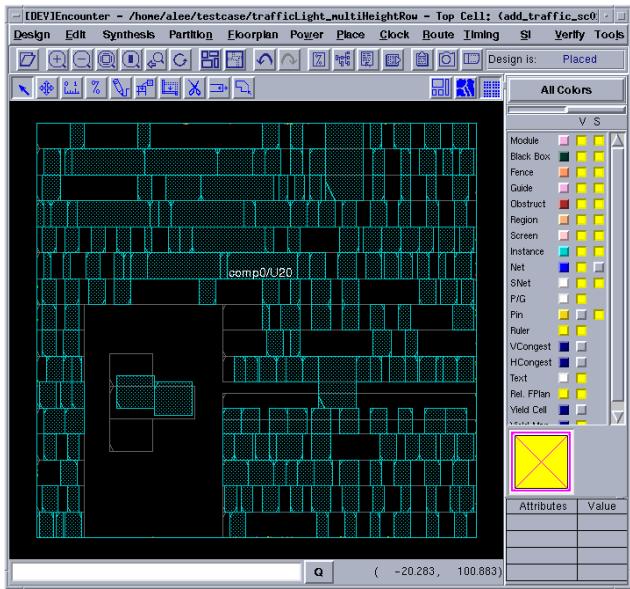
The following figure illustrates a user-defined floorplan brought in through a DEF file.



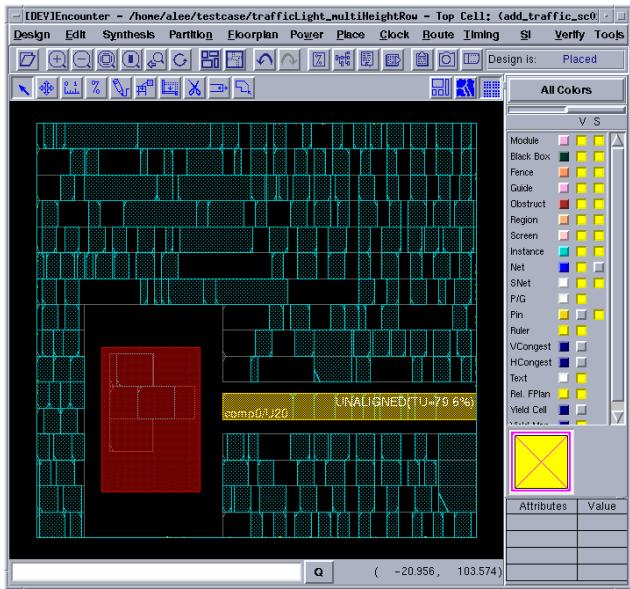
## Encounter User Guide

### Floorplanning the Design

The following figure illustrates how placement without power domain association results in illegal placement.



The following figure illustrates how placement with the correct power domain association results in legal placement.



By default, when a power domain is moved or (re)located in the main core row area, rows are initialized. To keep the rows brought in by the DEF file, you should pre-place and pre-size the power domains that cover the NIMH rows and the unaligned rows.

Here is an example flow:

```
createPowerDomain NIMH -timingLibs slow(common)
createPowerDomain UNALIGNEDROW -timingLibs slow(common)
modifyPowerDomainMember NIMH -instances NIMH_inst \
    power (VDD:VDD) -ground (VSS:VSS)
modifyPowerDomainMember UNALIGNED -instances comp0 \
    power (VDD:VDD) -ground (VSS:VSS)

modifyPowerDomainAttr NIMH -box 50 35 125 145
modifyPowerDomainAttr UNALIGNEDROW -box 142.2 90.2 336.6 110.0

#note. Box (50,35)(125,145) Covering NIMH rows
#note. Box (142.20,90.2) (336.6,110.0) Covering the unaligned rows

defIn userDefineRow.def
placeDesign
```

## Merging Hierarchical Floorplans from Partitions

While flattening partitions with the [flattenPartition](#) command, you can bring back row information, including NIMH rows and unaligned rows, from an existing floorplan. You can then run placement and routing to further improve the design performance.

Use the [flattenPartition](#) command with the `-bringBackRow` parameter to preserve the row information. The [flattenPartition](#) command also supports rotated partitions.

Power domains are automatically created and associated with each of the partition that have NIMH or unaligned rows. These automatically created power domain have the following characteristics:

- The `minGap` value is the same as the placement halo defined for the partition at the full-chip level.
- The timing library is the same as that specified at the full-chip level.
- The global net connection is the same as that for the full-chip level, which is the same as the partition floorplan global net connection.
- The value of the `RouteSearchExt` field is set to the default value of `0 .0`.
- The core-to-edge distances are *not* preserved as an attribute of the power domain, but are preserved in the merged floorplan.
- Row Parameters are not translated or detected.

- The power domain is set to `alwaysOn`.

The recommended use model for bringing back non-integer multiple-height rows or unaligned rows is as follows:

Top-down flow

1. Create power domains at full-chip level design.
2. Specify the same hierarchical instance for *power domain* as defined for *the partition*.

Bottom-up flow

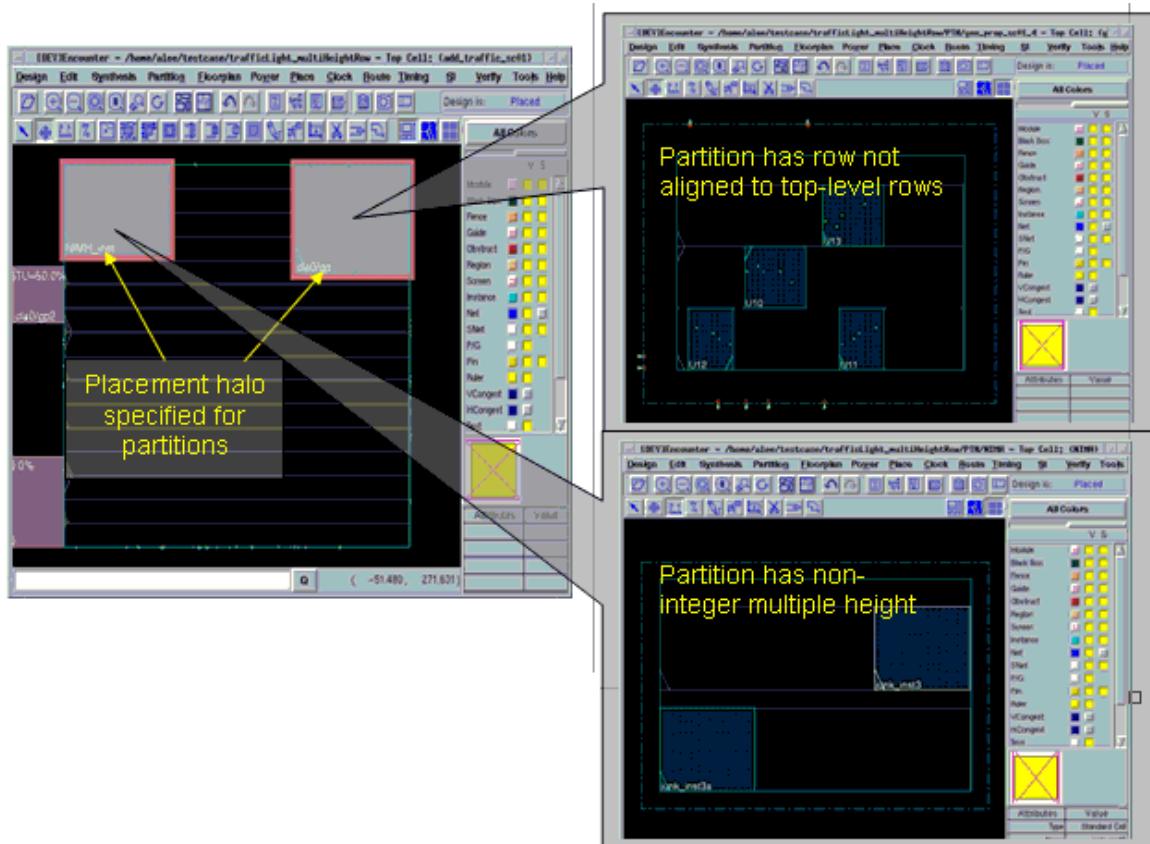
1. Create power domains at top-level design.
2. Create one PD for each of the partitions.
3. Assign instance blocks as a member of the created power domain.

For more information on the `flattenPartition` command, see the *Encounter Text Command Reference*.

# Encounter User Guide

## Floorplanning the Design

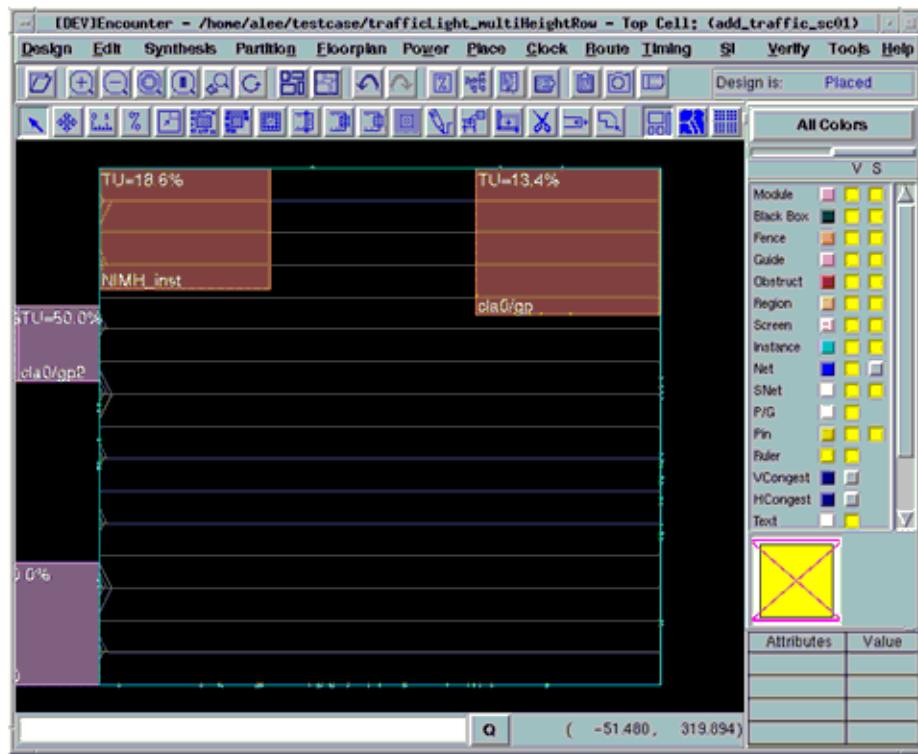
The following figure shows a full-chip view with the partition halos specified.



## Encounter User Guide

### Floorplanning the Design

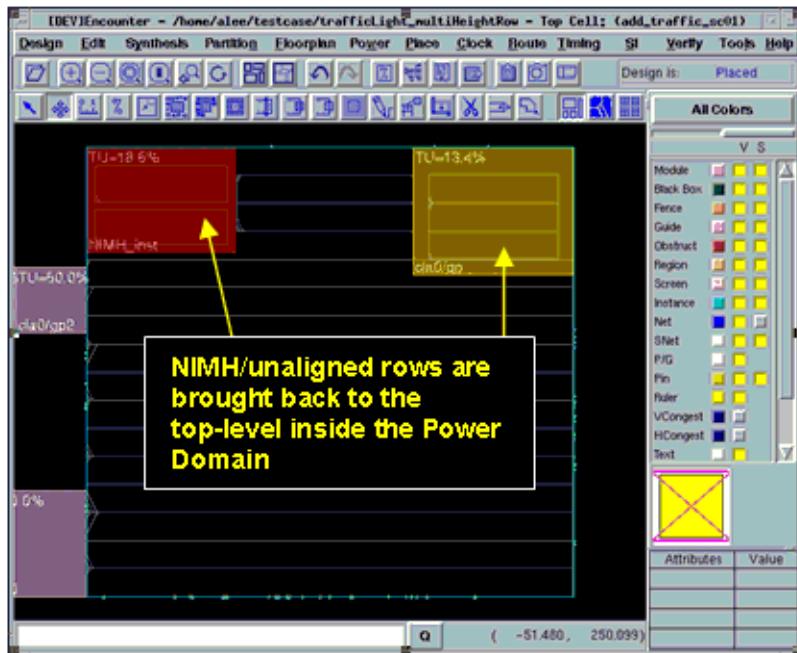
The following figure illustrates the result when you use the `flattenPartition` command *without* the `-bringbackRow` parameter



## Encounter User Guide

### Floorplanning the Design

The following figure illustrates the result with you use the `flattenPartition` command with the `-bringbackRow` parameter. In this case, the NIMH rows and the unaligned rows are brought back to the top-level inside the power domain.



## Performing I/O Row Based Pad Placement

In many cases, designs contain multiple-height I/O pads or asymmetric I/O rings, for example, a design might have a single I/O ring on one side and double rings or no rings on the other side, or no rings on part of a certain side. For such designs, the Encounter software enables you to create, edit, save, and restore I/O rows and perform pad placement based on the I/O rows.

You can create I/O rows anywhere in the die - within the core or in the periphery and use the I/O row flow for both, pad and area I/Os.

### Prerequisites

1. LEF technology file should contain I/O SITE definition.

Before you begin the I/O row flow in Encounter 8.1, you must first define I/O SITE for each type of I/O cell in the LEF I/O macro (LEF technology file).

2. Each I/O cell LEF must have correct CLASS and SITE type specified.

Consider the following examples that define LEF I/O SITE and CLASS PAD MACRO in the I/O assignment file, each I/O CLASS PAD macro is referenced with the I/O SITE:

### Example 1

The I/O SITE IOPFC is referenced from the I/O CLASS PAD MACRO pn1\_qdr\_vp:

```
SITE IOPFC
    SYMMETRY Y ;
    CLASS PAD ;
    SIZE 0.1 BY 321.94 ;
END IOPFC
```

```
MACRO pn1_qdr_vp
    CLASS PAD ;
    FOREIGN pn1_qdr_vp ;
    ORIGIN 0.000 0.000 ;
    SIZE 35.000 BY 321.940 ;
    SYMMETRY X Y R90 ;
    SITE IOPFC ;
    ...
END pn1_qdr_vp
```

### Example 2

The corner site, IOPFCCRNR, is referenced from the CLASS PAD MACRO pn1\_qdr\_iocrnr:

```
SITE IOPFCCRNR
    SYMMETRY y ;
    CLASS PAD ;
    SIZE 321.94 BY 321.94 ;
END IOPFCCRNR

MACRO pn1_qdr_iocrnr
    CLASS PAD ;
    FOREIGN pn1_qdr_iocrnr ;
    ORIGIN 0.000 0.000 ;
    SIZE 32.940 BY 321.940 ;
    SYMMETRY X Y R90 ;
    SITE IOPFCCRNR ;
    ...
END pn1_qdr_iocrnr
```

For more information, see “[Generating the I/O Assignment File](#)” in Data Preparation chapter of the *Encounter User Guide*.

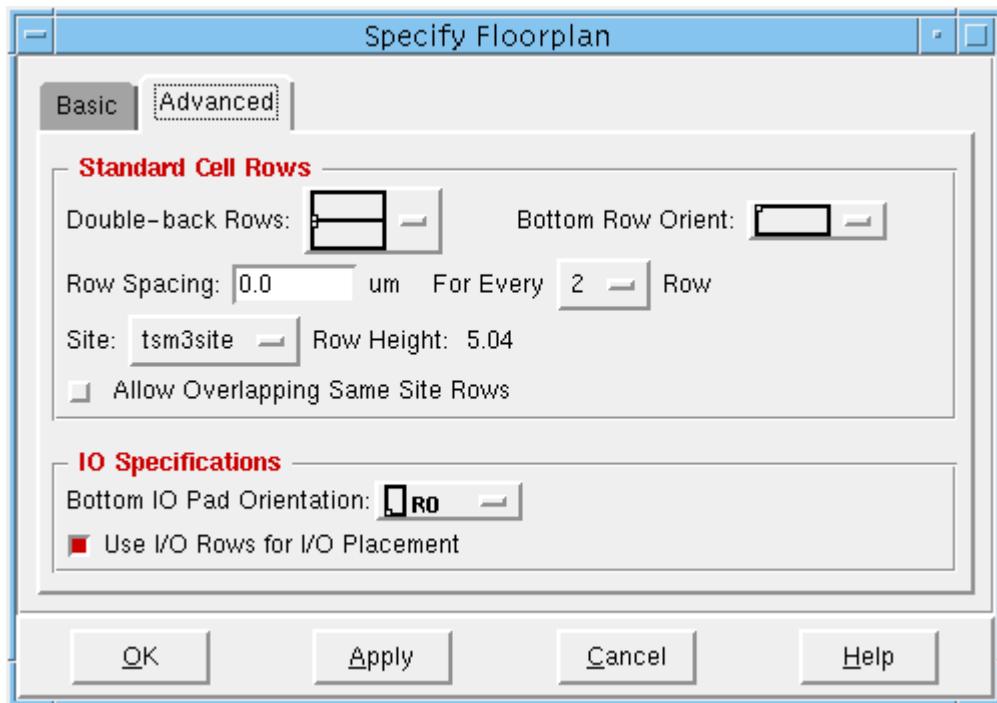
3. If the design contains multiple I/O SITES, the gap between the core boundary and the die boundary must be greater than the biggest I/O SITE; Otherwise, the Encounter software issues a warning and no I/O rows are created. Encounter does not automatically expand the core or die boundary to accommodate all the I/O pads.

## Enabling the I/O Row Flow in Encounter

To start using the I/O row flow in Encounter, you must enable the I/O row flow by doing one of the following:

- Specify set rda\_Input(use\_io\_row\_flow) {1} in the Encounter configuration file.

- Select *Use I/O Row for I/O Placement* check box in the *Floorplan - Specify Floorplan - Advanced* GUI form.



- Set the setIoFlowFlag command to 1.

**Note:** By default, the I/O row flow is *Off*.

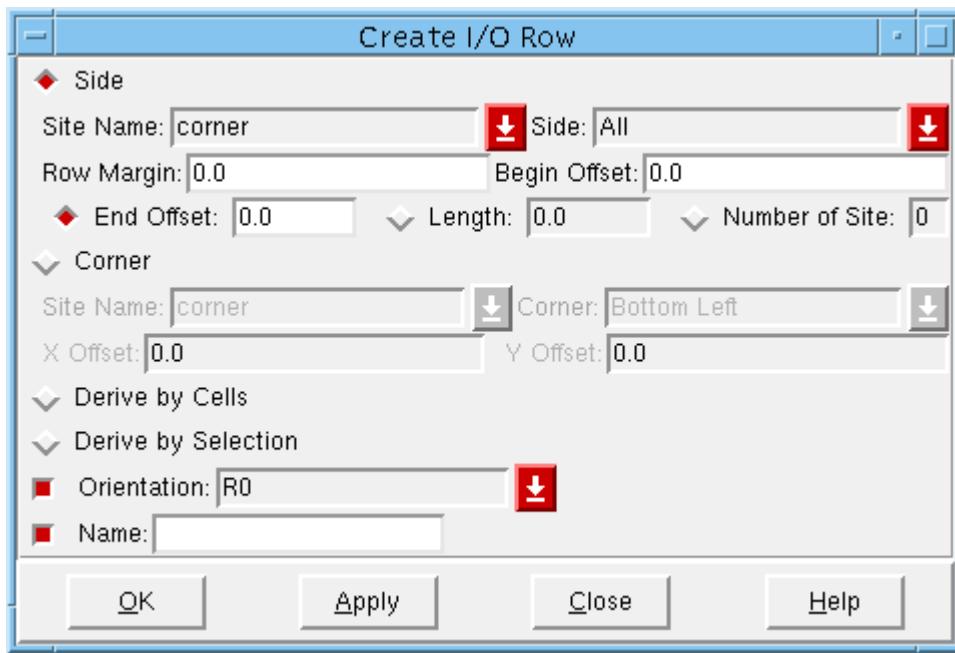
## Use Models

### Starting a new design in SoCE, v8.1

1. Import the design in Encounter
2. Set the I/O row flow by selecting the *Use I/O Row for I/O Placement* check box in the *Floorplan - Specify Floorplan - Advanced* GUI form.
3. By default, one I/O row is created on each side of the chip, between the core boundary and the die boundary. If the design has multiple I/O sites in the library, then rows are created for each side based on the number of I/O sites used in the design.

By default, the I/O pads are placed randomly on the I/O rows.

4. In the resulting design, you can create I/O rows using the *Floorplan - Edit Floorplan - Create I/O Row* form.



5. Edit (move/stretch/rotate/flip) the I/O rows using the *Floorplan - Edit Floorplan - Edit I/O Row* form or using the text commands.

The text commands that can be used for creating and editing I/O rows are described in the following table:

Commands	Usage
<u><a href="#">createIoRow</a></u>	Creates an I/O row.
<u><a href="#">flipInst</a></u>	Flips the selected I/O row through x or y axis.
<u><a href="#">fplanFlipOrRo</a></u>	Flips or rotates the selected instances.
<u><a href="#">tateInstance</a></u>	
<u><a href="#">stretchRows</a></u>	Stretches the selected I/O rows.
<u><a href="#">moveSelObj</a></u>	Moves the selected row to a specific location. Since the pads are already placed on the I/O rows, by default when you move the I/O rows, the pads also move with the rows.
<u><a href="#">deleteRow</a></u>	Deletes the selected I/O row. You can also delete the row by selecting the row and pressing the Del key on the keyboard.

Commands	Usage
<u>snapFPlanIO</u>	Snaps the I/O pads onto the correct side of the I/O rows if the pads are not already on the rows.
<u>spaceIoInst</u>	Spaces the selected I/O pads on the I/O rows, horizontally or vertically by a specified distance value.

Optionally, you can specify the I/O row constraints in the I/O assignment file. For more information, see “[Generating the I/O Assignment File](#)” in the [Data Preparation](#) chapter of the *Encounter User Guide*.

**Note:** To add I/O filler cells to the new I/O rows, use the [addIoRowFiller](#) command. You can delete the filler cells using [deleteIoRowFiller](#) command.

After designing the rows, save the design in a floorplan file (\*.fp) using the [saveDesign](#) or [saveFPlan](#) command.

### Reading an old design in SoCE v8.1

If you already have a design with placed pads, created using an earlier version of Encounter (v6.2 and above) and you want the design to be read into Encounter v8.1 to use the new I/O row flow:

1. Restore the design with placed pads, using [restoreDesign](#) command or load the floorplan information from the file, using the Load FPlan File form or the [loadFPlan](#) text command. Optionally, load the I/O constraint file using [loadIoFile](#) command.
2. Turn on the I/O row flow by setting the I/O flow flag ([setIoFlowFlag](#)) to 1.
3. Create the initial I/O rows using the [createIoRow](#) -deriveByCells command. This command creates I/O rows based on the existing pad placement.

Once you have rows and pads placed in the design, you can continue to create more rows or edit the rows.

Use the [changeIoConstraints](#) command to change the constraints of the I/O row read from the I/O constraints file.

You can also use the [Attribute Editor](#) to change the I/O pad location or pad orientation from the GUI.

4. After editing the I/O rows and the I/O row constraints, run the [snapFPlanIO](#) command to snap the I/O pads and area I/O's onto the legal sites/rows.

5. Save the design in a floorplan file (\*.fp) using the `saveDesign` or `saveFPlan` command.

## Resizing Rectilinear Blocks

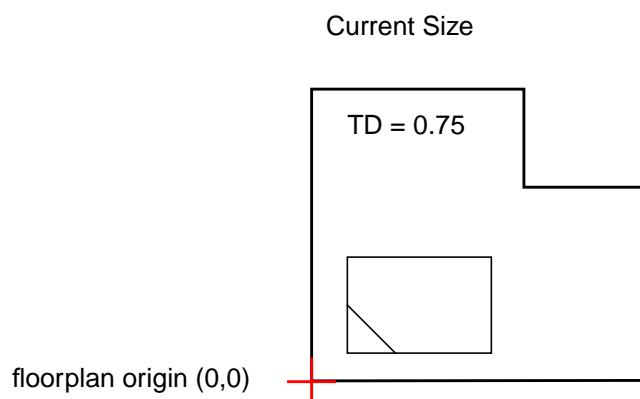
Given an initial rectilinear block in the floorplan, Encounter automatically resizes its bounding box by enlarging or shrinking the edges of the box proportionally in the x and y directions, ensuring that the specified target utilization is met.

During this process, to retain the original shape of the rectilinear block, you must specify the `-keepShape` parameter in the `floorPlan` command. Consider analog designs where you have digital blocks that need to be fit into the analog chip and the shape of the block is already pre-defined. In such mixed-signal designs, you can retain the block shape during resizing, and also meet the specified target utilization value by shrinking or expanding the floorplan using the `floorplan -keepShape util` value, where `util` is the target utilization value.

### Use Models

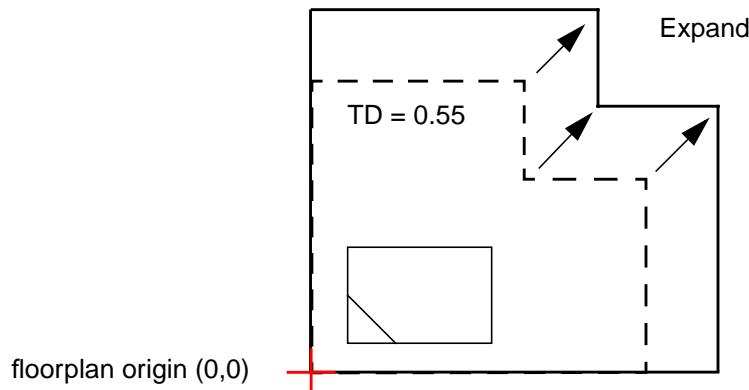
1. Import a DEF file or a floorplan file (\*.fp) that contains a rectilinear block boundary or you cut one corner of a rectangular block. This results in the core utilization missing the target value.
2. Run the `floorplan -keepShape util` command to automatically shrink or expand the block boundary, proportionally, in the x and y directions, trying to meet the specified target utilization.

Consider the following example of a rectilinear block whose current target utilization value is 0.75.



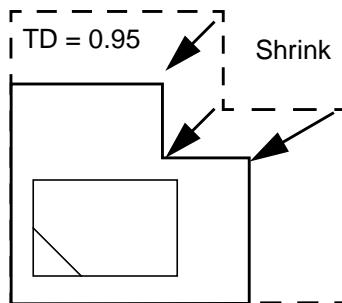
To meet a target utilization of 0.55, expand the rectilinear block.

```
floorplan -keepShape 0.55
```



To meet a target utilization of 0.95, shrink the rectilinear block.

```
floorplan -keepShape 0.95
```



In both the cases, the shape of the block is retained, and the required target utilization is met.

For I/O pins, prior to resize, Encounter saves the I/O file sequence internally and loads the file back after resize. The side and sequence of the I/O pin remains the same as in the old block, but the pins get distributed evenly. To redistribute the I/O pins, you must edit the pins manually in the resize block.

## Assumptions

The automatic resizing of rectilinear blocks is based on the following assumptions:

- The rectilinear blocks are L-shaped.

- The floorplan origin, (0,0) remains unchanged during the resize.
- The instances inside the block move proportionally or stay fixed during the resize.

## Results

The results of automatic resizing of rectilinear blocks are as follows:

- The shape of the block is preserved during the resize.
- Pre-placed macros are adjusted to the new size as much as possible.
- Pre-routed wires are removed after the resize.
- The core rows and block pins are automatically adjusted after the resize.

## Using Blackblobs

- [Defining Blackblobs](#) on page 395
- [Specifying Blackblobs](#) on page 396
- [Blackblob Useflow](#) on page 400
- [Blackblob Display](#) on page 403
- [Blackblob Overlap](#) on page 408
- [Saving and Restoring Blackblobs](#) on page 411

### Defining Blackblobs

The Encounter software provides support for blackboxes, which are used for modeling entities that will ultimately be implemented as standalone partitions. Blackboxes have a fixed shape, have full routing blockages along one or more layers, and require specific pin assignment along the edges.

You can use blackboxes for performing what-if timing analysis. For more information, see [Performing Blackbox What-If Timing Analysis](#) in the *Timing Analysis* chapter of the *Encounter User Guide*.

However, in many cases, the design might contain entities for which a netlist is not initially available but eventually, the entity will have a flat implementation, that is, it will not be implemented as a hard macro. Such entities ultimately comprise standard cells, and possibly some macros and are called *blackblobs*.

Because of their characteristics, blackblobs have the following advantages over blackboxes:

- Blackblobs need not necessarily have a square or rectangular shape. Therefore, the partition area is better utilized.
- The pins need not necessarily be along the edges of the blackblob—they can be anywhere in the blackblob because the blackblob is not implemented as a macro. You do not need to assign pins for the blackblob; the connections and pins are represented by super cell pins.
- Blackblobs can overlap during placement—their placement is more typical of a placement guide, instead of a fence.
- In general, iterative closure on a floorplan is faster for blackblobs.

The shape of a blackblob is determined by the placer. The display of a blackblob is similar to that of a regular instance. You can select a blackblob and query it in the floorplan view. You can move a blackblob in the floorplan view, but not in other views. You should not adjust or stretch a blackblob.

**Note:** If a design contains blackblobs, there might be a difference between the wire lengths reported by Trial Route and by placement. This is because while reporting the wire length, Trial Route does not include the intra-net routing inside blackblobs whereas placement does.

## Specifying Blackblobs

You can use the specifyBlackBlob command to create a blackblob. You can also use the Floorplan – Specify Blackblob form in the GUI to create a blackblob.

Any module that does not have to be implemented as a partition can be specified as a blackblob.

The following points apply to the specification of blackblobs:

- Blackblob module netlist can be undefined, partially defined, or have instantiated hard macros and/or standard cells. These macros and cells may or may not have connections.
- You should specify the overall blackblob area.
- The shape of the blackblob is automatically determined during blackblob placement.
- Pins need not be assigned for a blackblob because the external connections or pins are represented by super cell pins.
- Timing for a blackblob is specified using What-if Timing Analysis capability. You can create a timing model for a blackblob right after specifying it (not requiring you to floorplan a design first).
- You can use empty Verilog modules while creating blackblobs—the empty modules are displayed and are available for selection when you create blackblobs with the Specify Blackblob form, accessed from the Floorplan – Specify Blackblob menu.



To have the list of empty modules available in the GUI, add the following line to the configuration file:

```
set rda_Input(import_mode) {-treatUndefinedCellAsBbox 0 -keepEmptyModule 1  
-useLefDef56 1}
```

- You can create blackblobs inside power domains. For this, the blackblob module must be inside the power domain module.

The following options are used for specifying a blackblob:

■ **Area**

Initial blackblob area can be specified in any of the following manner:

- Total blackblob area. This includes the area of any hard macros that you have specified.
- Gate area. This area may or may not include any hard macros that you have specified.
- Gate count and area per gate values. This area may or may not include any hard macros that you have specified.

If the blackblob netlist contains instantiated hard macros and/or standard cells, the initial blackblob area includes these macro and/or cell areas.

■ **Target utilization**

Similar to a module guide target utilization. This utilization is mainly used for deriving the initial blackblob area.

■ **Placement porosity**

The placement porosity specifies the percentage of cell area that will be added to the blackblob area during placement. This is used by the Encounter placer for cell padding during the placement step. The cell padding reserves some area for the IPO cells to be placed over the blob area.

■ **Routing porosity**

Specifies the routing resources, per layer, that will be reserved for the blackblob (this porosity is for standard cell area only). A routing porosity of 50% on M1 suggests that 50% of M1 can be routed over the blob area. A higher routing porosity implies more routing resources.

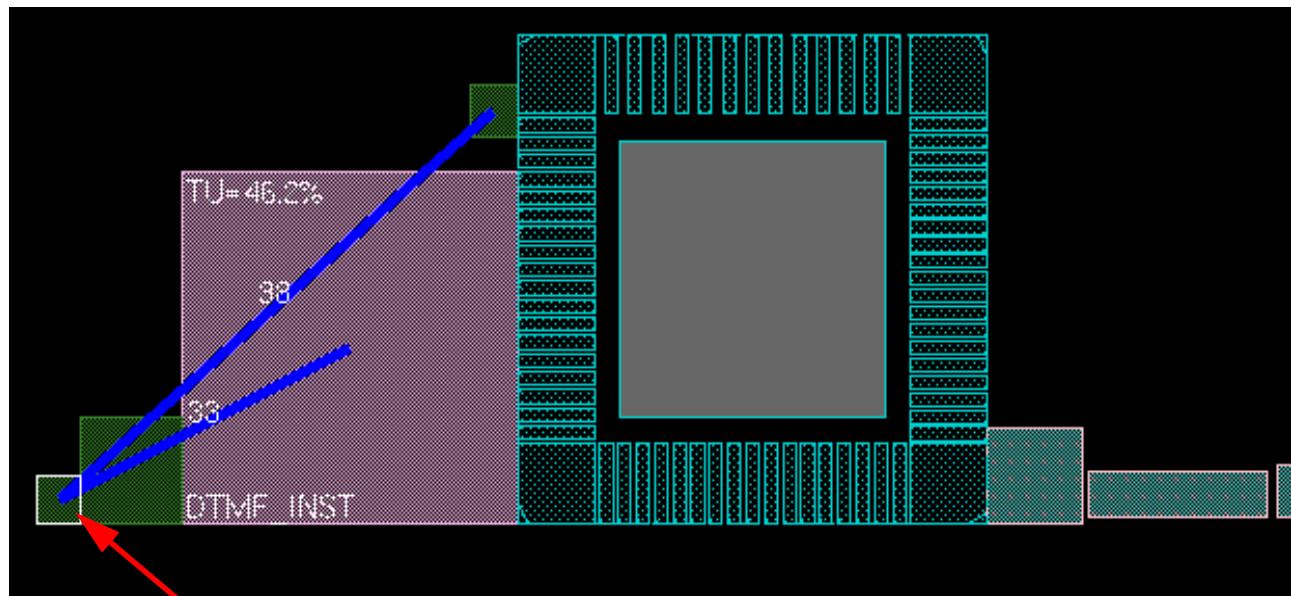
The following `specifyBlackBlob` command specifies a blackblob `alu_32` with area 20000 square microns and a target utilization value of 20%.

```
specifyBlackBlob - cell alu_32 -area 20000 -targetUtil 20
```

After specifying the blackblob using the `specifyBlackBlob` command, the blob is represented by the blob guide in the floorplan view. The display of a blackblob is similar to that of a regular instance. The shape of a blackblob is determined by the placer.

**Note:** You can move a blackblob in the floorplan view, but not in other views. You cannot, however, stretch a blackblob.

The figure below displays the results of this command. By default, blackblob objects are displayed in green as shown here:



Blackblob cell `alu_32`

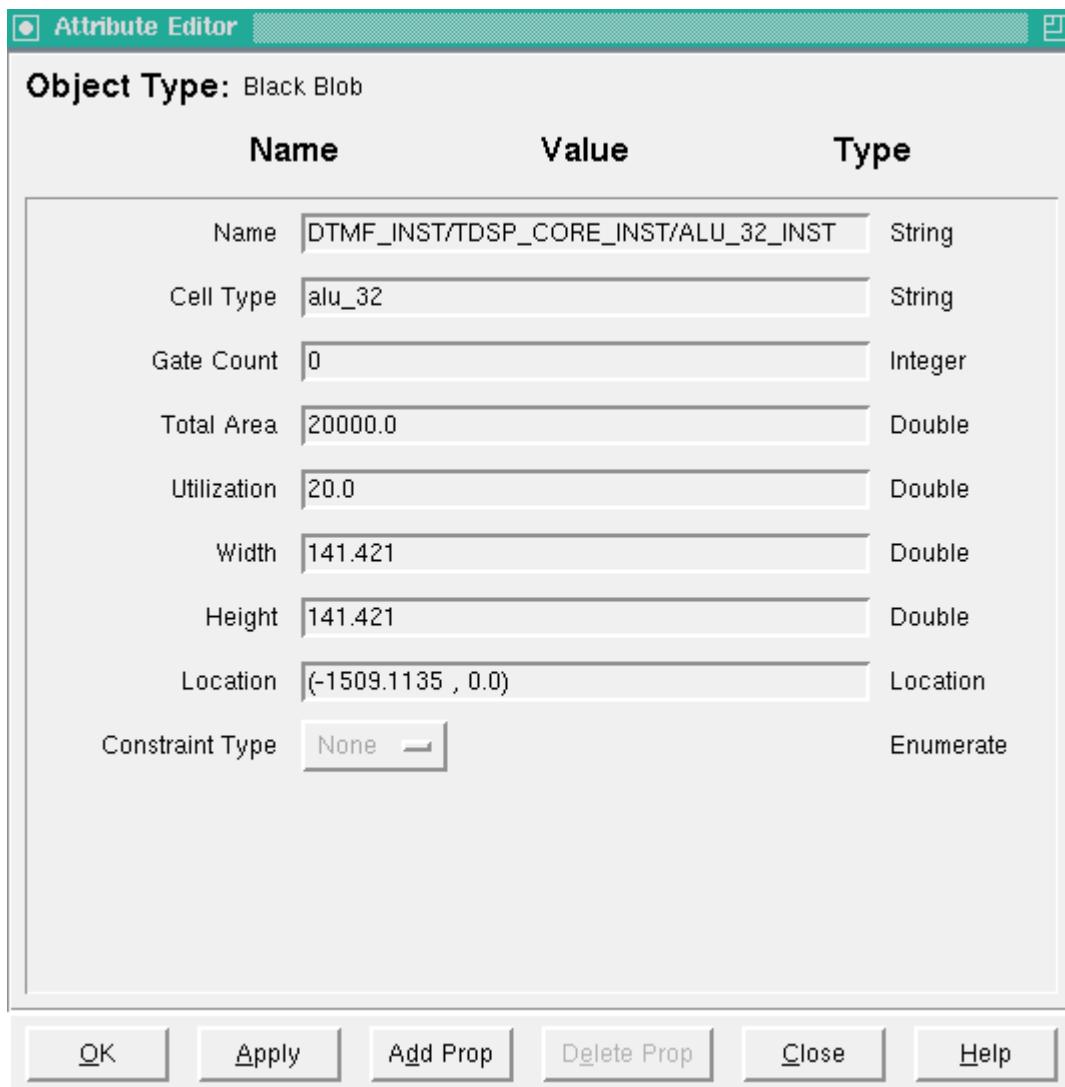
<code>totalArea = 20000</code>	<code>Height = 141.421</code>
<code>targetUtil = 20%</code>	<code>Width = 141.421</code>

$$\begin{aligned} \text{gateArea} &= (\text{totalArea} * \text{targetUtil}) / 100 \\ &= (20000 * 20) / 100 \\ &= 4000 \end{aligned}$$

## Encounter User Guide

### Floorplanning the Design

To view the attributes of a blackblob that you created, double-click the blackblob to display the Attribute Editor as shown in the following example.



Blackblobs nested inside a partition are supported. However such blackblob(s) will be automatically unspecified before partitions are committed. If you want to have a blackblob inside a partition block-level design, you will need to drill down to the partition block level design and specify the blackblob again.

Once you have specified blackblobs, run the `elaborateBlackBlob` command. This command instantiates the hard macros inside blackblobs and completes the virtual connections for these hard macros. The blackblobs are also filled with dummy instances as per the size/area specified.

Typically, you would run the `elaborateBlackBlob` command once after you have specified the blackblobs in the design with the `specifyBlackBlob` command. For example, you might call the `specifyBlackBlob` command several times in a script, and when all blackblobs have been specified, you would run the `elaborateBlackBlob` command once to instantiate the hard macros and/or complete the virtual connections for macros and blob cells.

The `elaborateBlackBlob` command is run automatically when you restore a design that contains blackblobs.

If you specify blackblobs through the **Specify Blackblob** GUI form, the `elaborateBlackBlob` command will automatically be invoked when you click the Apply or the OK button.

You can unspecify a blackblob through the `unspecifyBlackBlob` command.

## Limitations

The following limitations apply to blackblobs:

- Non-uniquified netlists are not supported. Make sure all netlists are uniquified before you specify a blackblob.
- The gate area of the blackblob remains the same even after you resize the blackblob in the floorplan view. In order to adjust the size of a blackblob, you should re-specify the blackblob using the `specifyBlackBlob` command.

## Blackblob Useflow

This section describes the blackblob useflow for the following scenarios:

- [General Blackblob Flow](#) on page 400
- [ECO Blackblob Flow](#) on page 401
- [Blackblob Flow for Placing Macros Using Masterplan](#) on page 402

### General Blackblob Flow

The general blackblob flow, that is, when you do not perform an ECO netlist update for the blackblob, is as follows:

1. Import the design.

2. Specify blackblobs.
3. Run the `elaborateBlackBlob` command if you specified the blackblobs with the `specifyBlackBlob` command and not through the GUI.
4. Optionally use What-If Timing Analysis capability to create timing model for the blackblobs and to perform a quick timing analysis.
5. Optionally pre-place blackblobs in the floorplan view to guide blackblob placement.
6. Run placement (`placeDesign`):
  - Encounter placer treats blackblobs as soft module guides. It places all the instantiated hard macros and/or cells, and user-specified hard macros (if any) to derive blackblob shapes.
  - If blackblob placement porosity information is specified, placer adds cell pads during blob placement to honor user-specified placement utilization.
  - Blackblobs placement can be overlapping.
  - You run either non-timing-driven or timing-driven placement.
7. If you re-specify a blackblob specification or manually change location of a blackblob module guide in the floorplan, run the placement step again to get correct blackblob placement.
8. Optionally run the `generateGuide` command to generate blackblob guide in the design for later use.
9. Run Trial Route. Trial Route take into account user-specified blackblob routing porosity information to derive blob routing resources on the fly. Maximum blackblob porosity value will be used for calculating overlapping blackblob area(s).
10. Save the design.
11. Optionally run What-If timing analysis.
12. Continue with the normal flow.

## ECO Blackblob Flow

The flow when you have updated blackblob netlists is as follows:

1. Restore a blackblob design. Or, perform the following steps:
  - a. Import the design.
  - b. Specify blackblob(s).

- c. Run the `elaborateBlackBlob` command if you specified the blackblobs with the `specifyBlackBlob` command and not through the GUI
    - d. Run placement
  2. Optionally run the `generateGuide` command to generate blackblob module guide(s) in the core area based on the current cell placement. This step is recommended.
  3. Use the `loadBlackBlobNetlist` to incrementally load updated blackblob netlist(s).
  4. After loading new update blackblob netlist,
  5. If you want to retain the blackblobs as blackblob objects:
    - a. Re-specify the blackblob area if the existing block area is smaller than the area generated by the updated netlist.
    - a. Run placement once again.
- Otherwise
- a. Run the `unspecifyBlackBlob` command to convert blackblob object back to the module with the new updated netlist.
  - a. Run placement.
6. Proceed with the normal flow.

### **Blackblob Flow for Placing Macros Using Masterplan**

You can use Masterplan to place macros inside and outside blackblobs because it generates a quick, initial floorplan that can be used as a starting point for making the final floorplan. The blackblob flow for running the Masterplan to place macros inside and outside blackblobs is as follows:

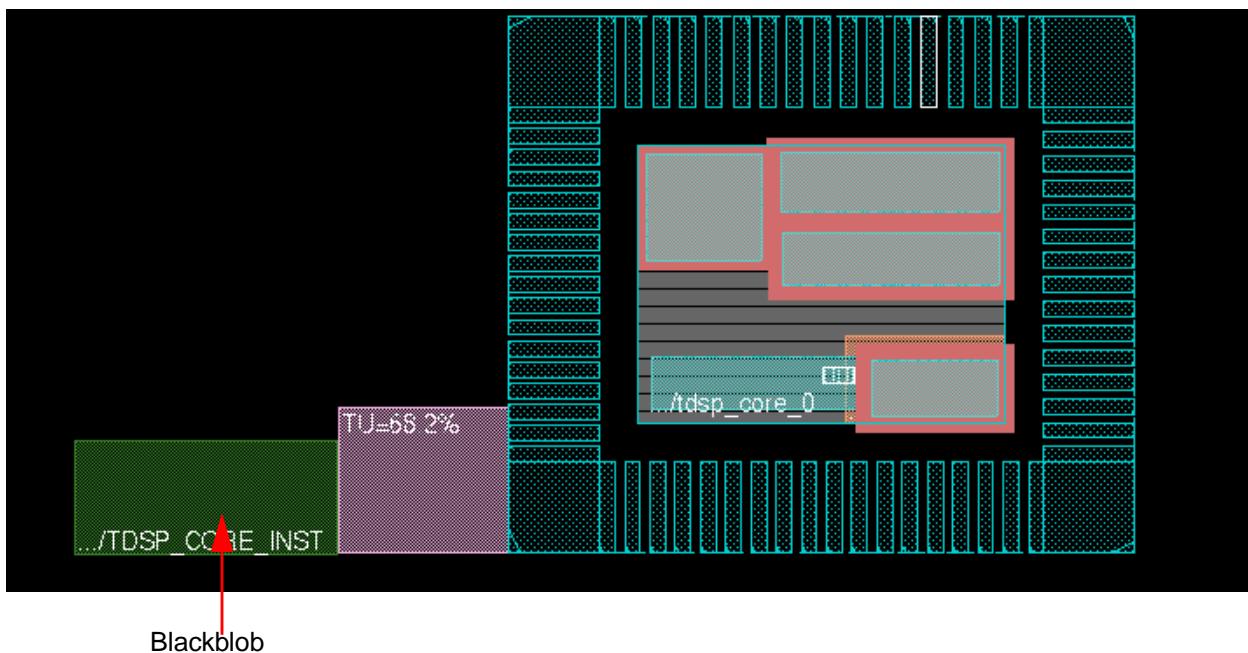
1. Import the design.
2. Specify blackblob(s).
3. Run the `elaborateBlackBlob` command if you specified the blackblobs with the `specifyBlackBlob` command and not through the GUI.
4. Generate an initial floorplan with the `planDesign` command to place all macros in the design including embedded blob macros.
5. Manually adjust the macros if needed and mark them as fixed.
6. Optionally run What-If timing analysis.

7. Optionally pre-place blackblobs in floorplan view to guide placement.
8. Run [placeDesign](#) to place blackblobs to derive the blackblob shape.

## Blackblob Display

After you specify a module as a blackblob, the module becomes a block instance. However, in the floorplan view, a blackblob object is represented as a blackblob module guide instead of an instance block. This way, you can guide the placement of the blackblob.

After blackblob specification, the blackblob is displayed on the left side of the design in the floorplan view. By default, blackblob objects are displayed in green as shown here:



You can perform the following actions on the blackblob:

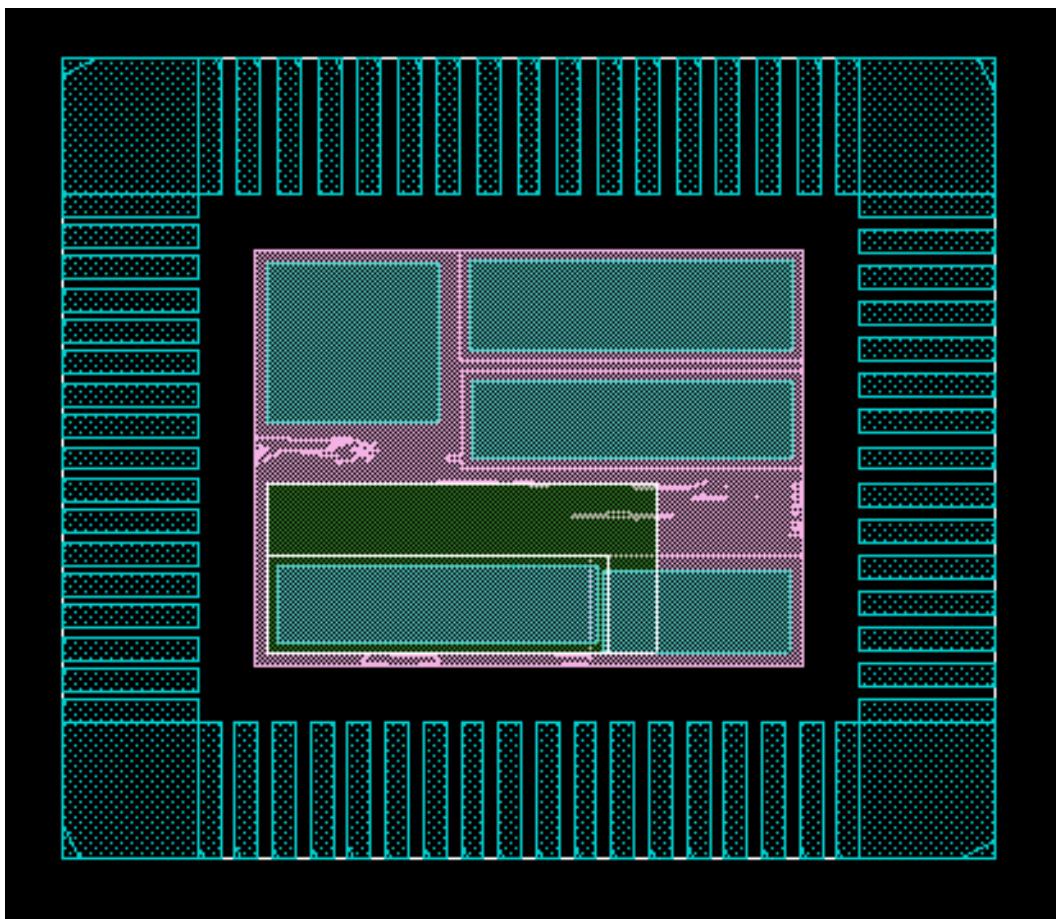
- Select the blackblob.
- Move the blackblob to pre-place it in the floorplan view to guide the placement.
- Query a blackblob. In the floorplan view, a blackblob has the type **Module**. In the physical view, the type is **Block**.

**Note:** Although you can resize the blackblob in the floorplan view, the blackblob gate area remains the same. If you want to adjust the size of a blackblob, you should re-specify the blackblob using the [specifyBlackBlob](#) command with the same target utilization value.

After specifying a blackblob, you need to run placement to place the design and obtain the blackblob shape in the amoeba and physical views. Blackblob instances will have non-contiguous shapes (islands) if the macros and/or cells of the blackblob are not placed close together. After placement step, when you select a blackblob:

- In the floorplan view, the instantiated hard macros and the user-specified hard macros will be highlighted.
- In the amoeba and physical view, blackblob instance will be highlighted.

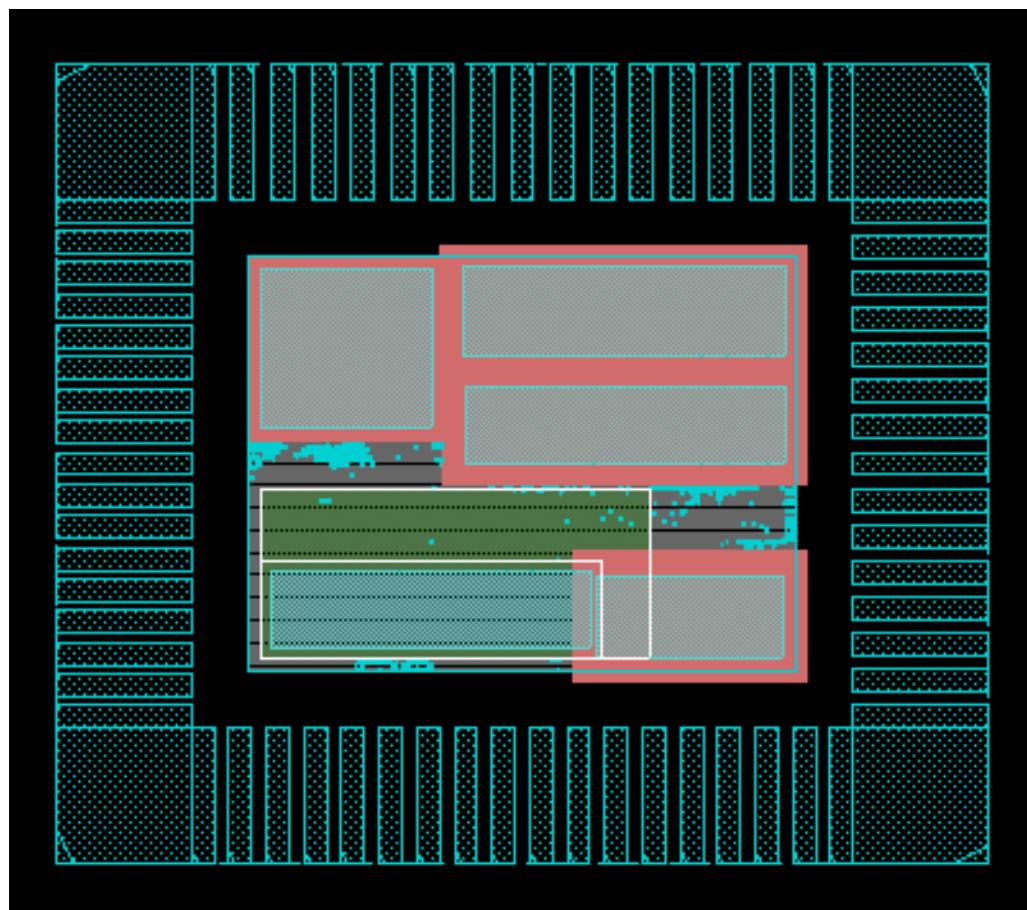
The following figure shows the amoeba view after placement.



## Encounter User Guide

### Floorplanning the Design

The following figure shows the physical view after placement.



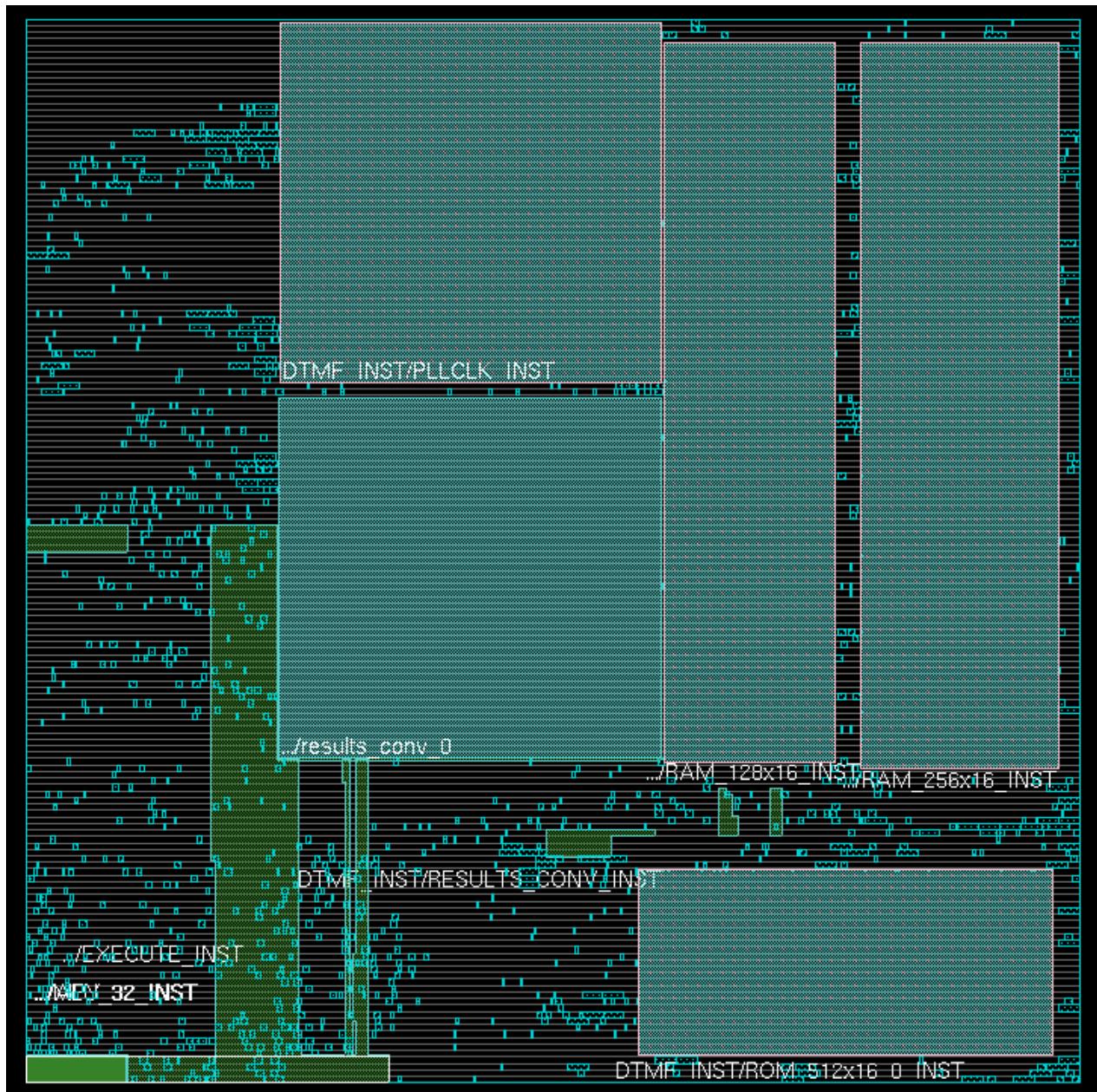
The physical view display of the blackblob is based on the union of all the bloblets (gate area), standard cells and hard macros that belong to the blackblob.

The blackblob in the physical view after placement step will correspond to the total blackblob area specified during the blackblob specification.

## Encounter User Guide

### Floorplanning the Design

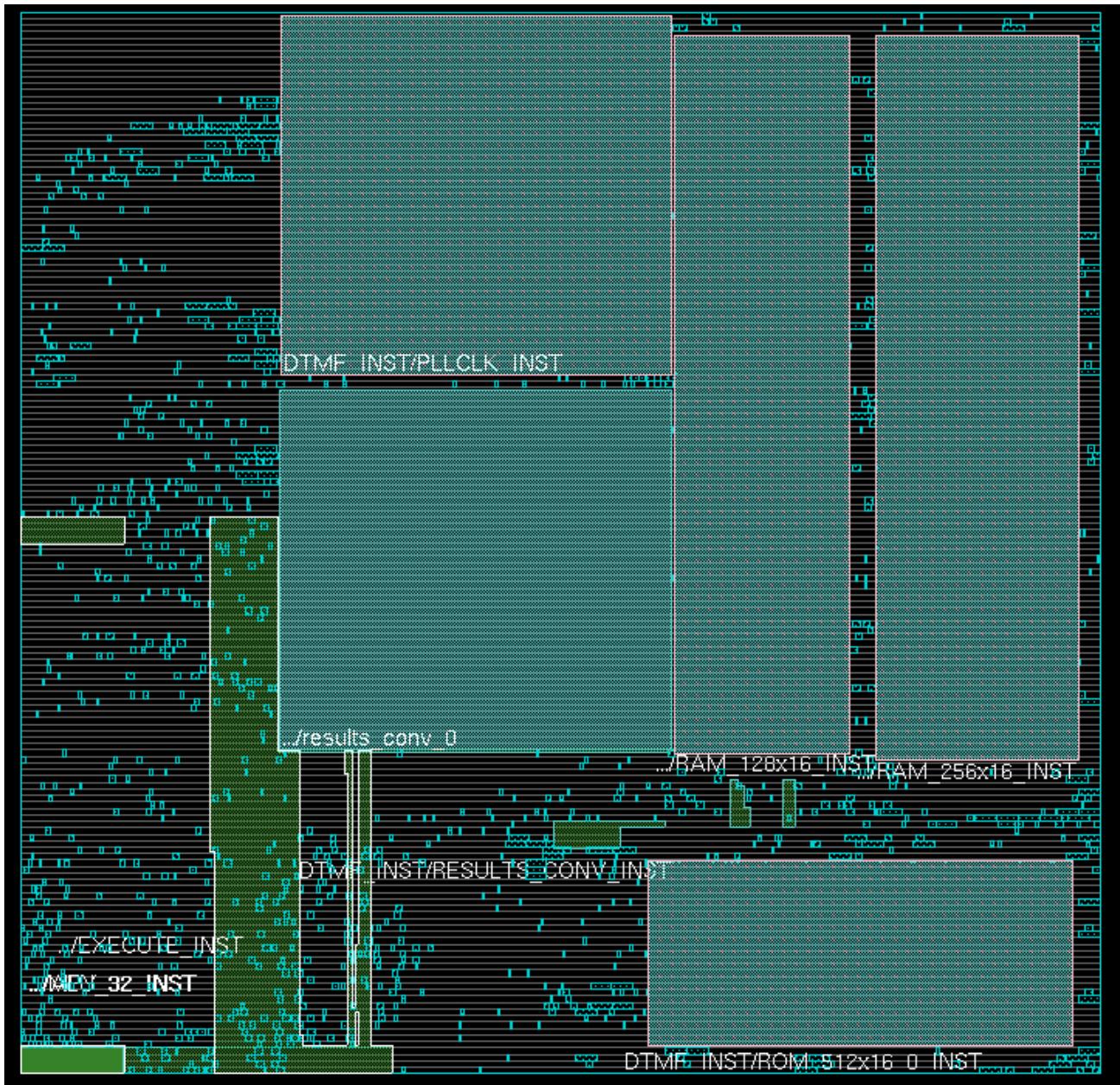
The following figures show the different types of blackblob enclosures after placement.



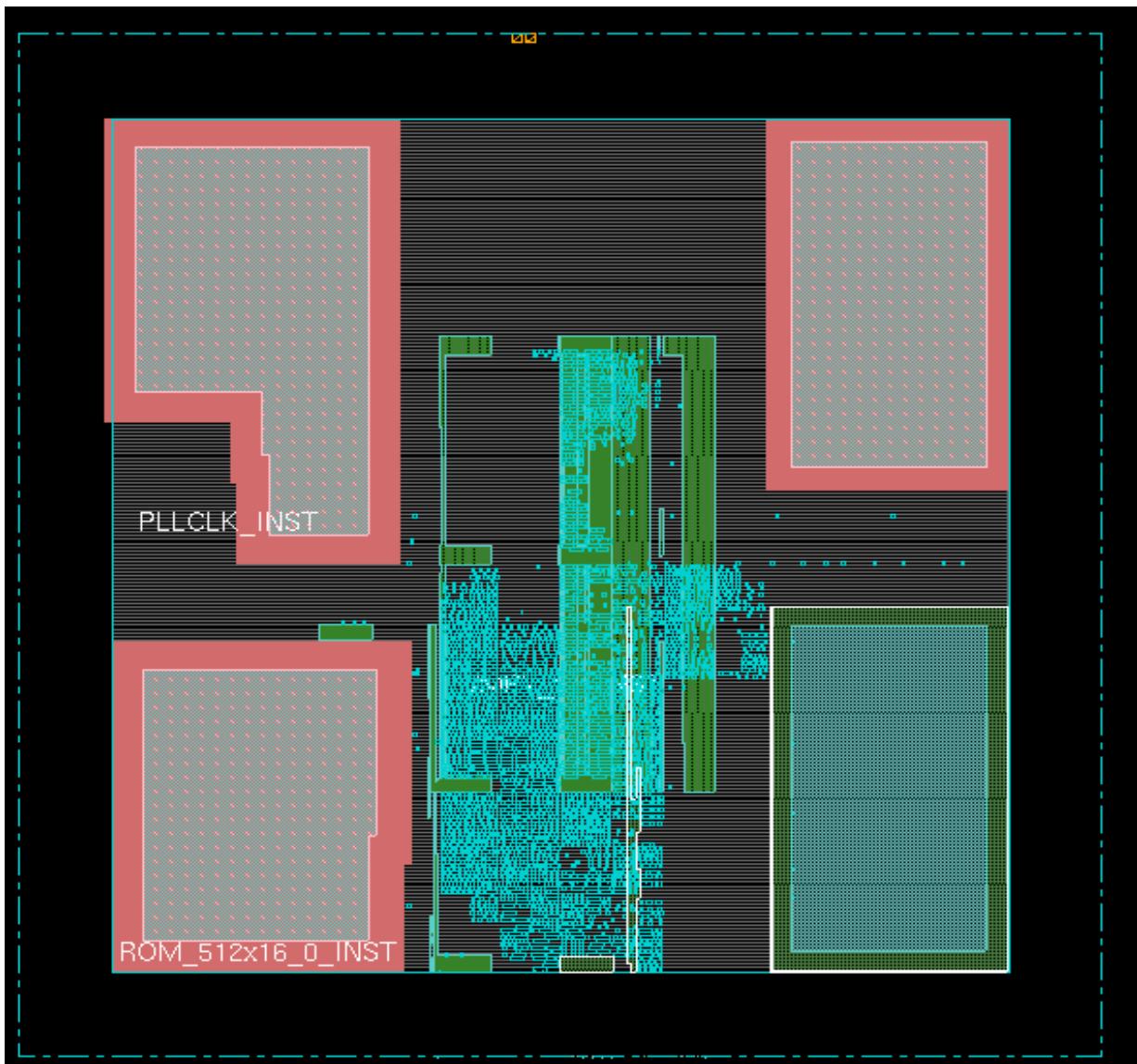
Blackblob with a singe shape

## Encounter User Guide

### Floorplanning the Design



Blackblob with multiple shapes



Blackblob with a macro inside

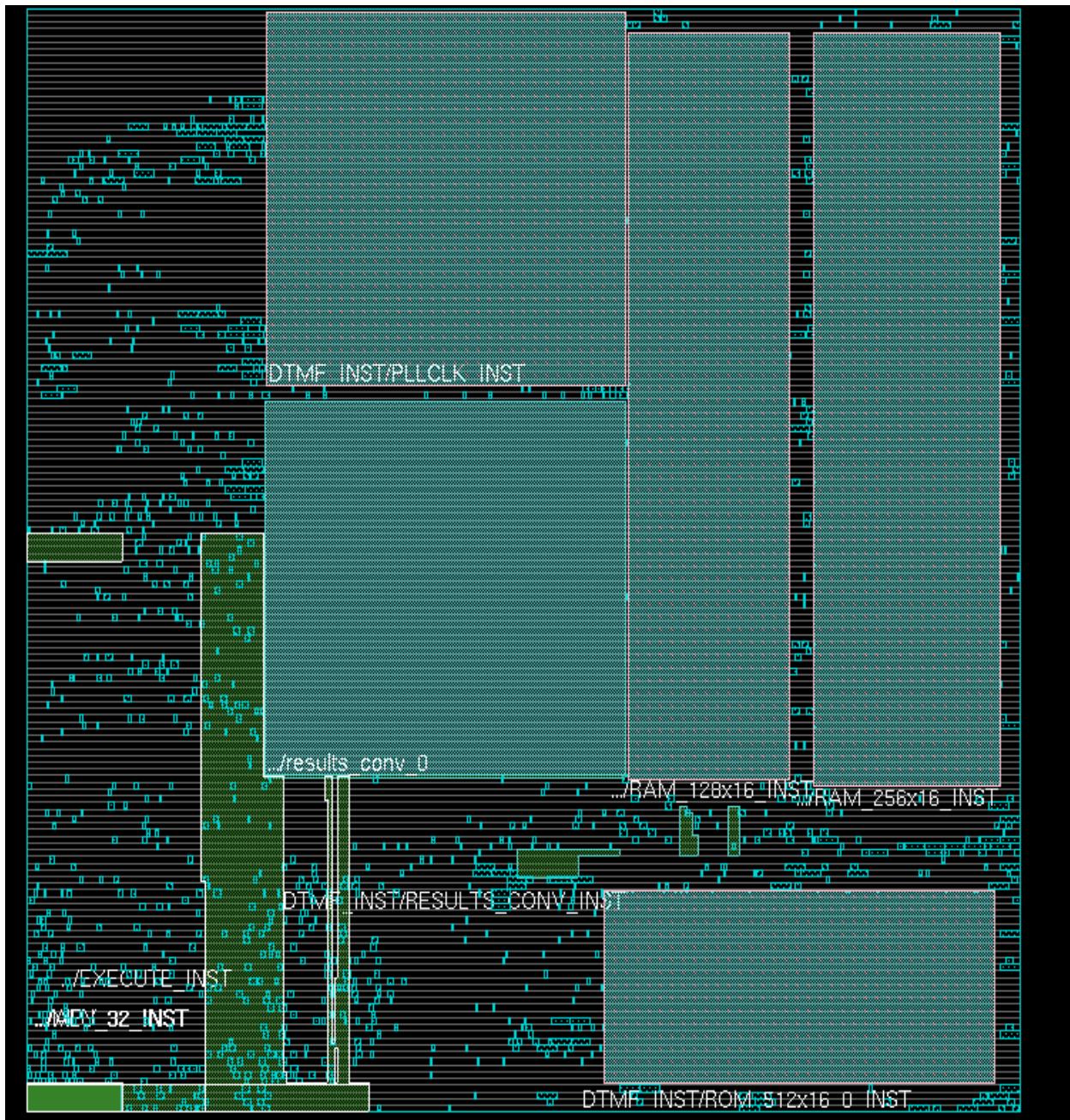
## Blackblob Overlap

The blackblob overlap can happen in any of the following ways:

- [Overlap between Blackblobs](#) on page 409
- [Overlap between Standard Cells and Blackblobs](#) on page 410

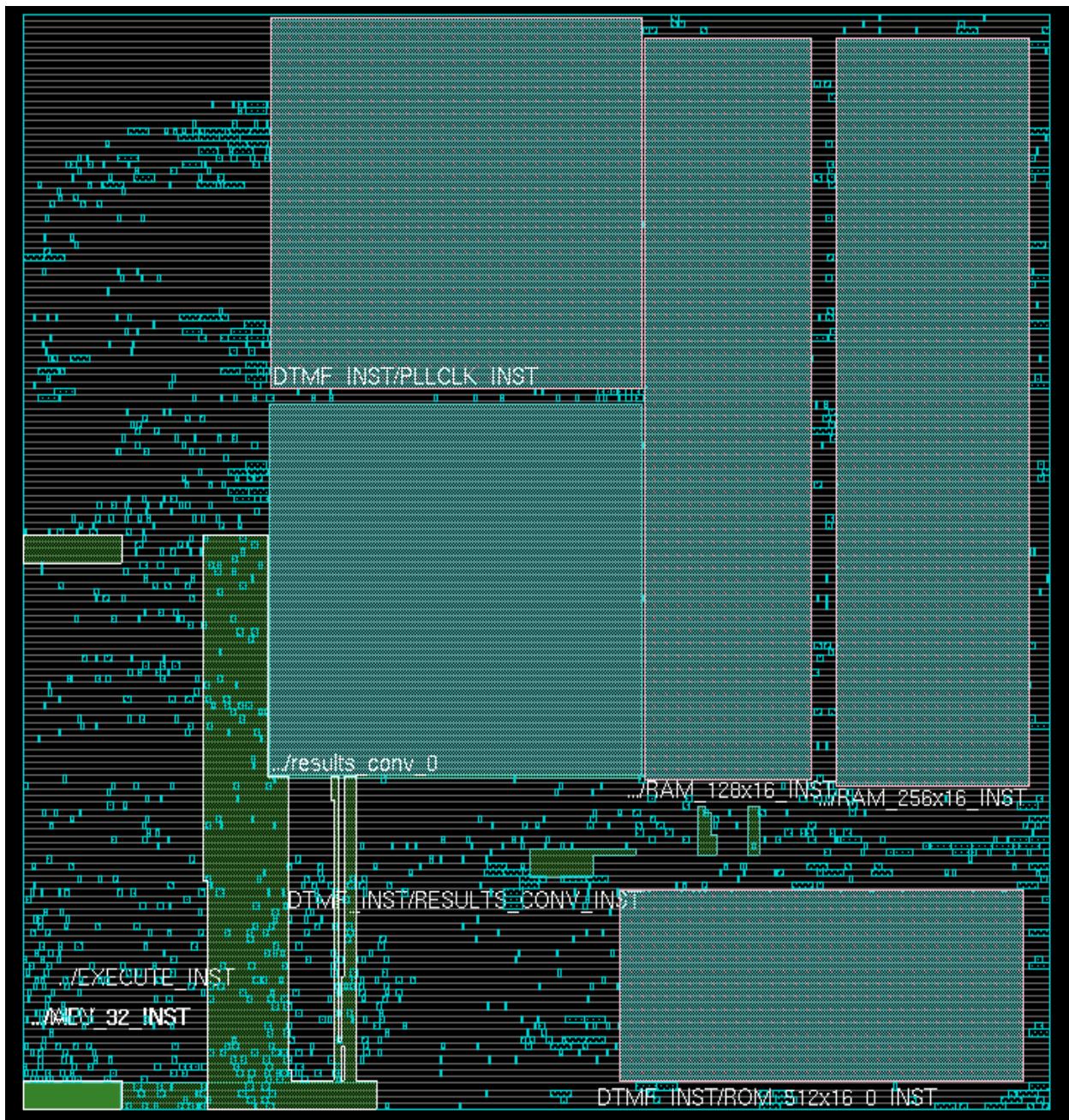
## Overlap between Blackblobs

Blackblobs can be overlapped because they resemble module guides whose cells can be placed outside the guides. The following figure shows the overlap between two blackblobs, multi\_32 and execute\_i.



## Overlap between Standard Cells and Blackblobs

Standard cells can be placed over blackblobs because the guide allows other cells to be placed in their guide area. The following figure shows the overlap between standard cells and a blackblob.



## Saving and Restoring Blackblobs

The blackblob placement information and the netlist information are saved in different files. It is, therefore, recommended that you save a design with blackblob information using the `saveDesign` command so that the design floorplan can be restored correctly. It is *not* sufficient to save only the floorplan file, except when blackblobs have not yet been placed.

Blackblob specifications such as area-related information, placement porosity, routing porosity, target utilization, and user-specified macros are saved in a floorplan file.

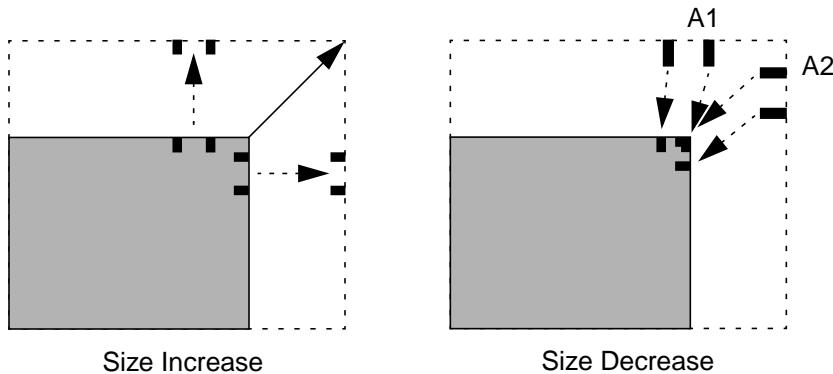
The blackblob placement information is saved in a design placement file. Blackblob boundary, location, and pin information are automatically derived when the design is restored. The `elaborateBlackBlob` command is also run automatically.

## Editing Pins

This section describes how you can move and manipulate pins in your design. For information on blackbox and partition pins, see the [Assigning Pins](#) section in the “Partitioning the Design” chapter of the *Encounter User Guide*.

### Pin Snapping on Resized Boundaries

As the boundary size increases, the pins maintain their exact horizontal and vertical coordinates, depending on the modified edge. As the boundary size decreases, the pin snap retains its relative position on the modified edge. This following figure illustrates this capability. For the size decreasing example, pins A1 and A2 are both snapped to the upper right corner.



**Note:** This feature is limited to rectangular edges.

### Moving Pins

To move a pins or a group pins, they should be at the same block and same side of the block. By default, all pins will move together relatively and the layer will be changed to the appropriate layer if the side was changed. For example, layer M2 is changed to M1 when moving pins from top to left. Moving pins from top to bottom does not change the layer.

To move a selected pin or group of pins in the design display area from one edge to another edge (including rectilinear edges) on a module, complete the following steps:

**Note:** For pin groups, this will preserve the relative position between pins.

1. Click the *Move/Resize/Reshape* widget.
2. Select (left-click) the pin in the design display area.

For a group of pins, press the `Shift` key to highlight each pin.

3. Left click on the pin(s) and move them to the new location.



*Tip*

To zoom out on the design display area while dragging the pins, press the Shift-Z key combination.

During the move, or when the selected pin(s) are in move mode, you can press the F3 key to open the Group Pin(s) Move form to change the pin layer, the pin size, pin status, and resolve pin overlap. You can use the moveGroupPins command to perform the same actions as the form, with the addition of moving the pin(s) to the new location.

## Swapping Pins

You can swap pins using the swapPins command or the Swap Pins option in the design display area by completing the following steps:

1. Select two pins of the same block.
2. With the cursor over one of the selected pins, click and hold the middle mouse button to bring up the context pop-up menu.
3. Select Swap Pins.

## Using the Pin Editor

You can use the Pin Editor to display and edit pins and pin groups. To open the Pin Editor, choose *Edit – Pin Editor*. For information in the fields and options, see Pin Editor in the Edit Menu chapter of the *Encounter Menu Reference*.

Here are the main features of the Pin Editor:

- Works for all type of pins such as partition pins, blackbox pins, and I/O pins
- When moving pins, associated pin geometry is moved or updated accordingly
- Can be used to move a single pin and/or a group of pins
- Supports pin editing by various criteria such as location, layer, side/edge, and so on
- Provides pin spreading capabilities. For more information, see Using the Pin-Spreading Feature on page 414
- Provides pin snapping capabilities such as to manufacturing grid, user grid, and layer track.

- Supports non-preferred routing layers for all supported snapping grids.
- Honors pin constraints at partition-level and pin-level, as well as constraints defined through the GUI.
- Supports pre-assigned pins.
- Supports rectilinear edges (multiple edges per side).

The [editPin](#) command provides the equivalent functionality of the Pin Editor.

The following sections describes some of the features that you can use with the Pin Editor.

## Using the Pin-Spreading Feature

The Pin Editor includes a utility to spread pins along the edges of a block. There are four different methods of spreading pins:

- Use a pin as the starting point (anchor) and provide a pin spacing distance.
- Use the center of a side or edge as the starting point and provide a pin spacing distance.
- Space the pins evenly along the side or edge, using the ends of the side or edge as the starting and ending points. The Pin Editor calculates the pin spacing distance.
- Space the pins evenly using explicit starting and ending points on the side or edge. The Pin Editor calculates the pin spacing distance.

## Basic Concepts for Pin Spreading

Two basic concepts underlie the pin-spreading functionality of the Pin Editor:

- Pin ordering affects the starting point for pin spreading.

Use the Pin Editor's *Group Bus*, *Reverse Order*, or *Reorder Pin List* functions to specify the first pin in a group. The coordinates of the first pin in a group provide the starting point from which to spread pins.

- Pin spacing distances can be expressed in either positive or negative values:

*Positive* spacing values spread pins to the right along a horizontal block edge, or up along a vertical block edge.

*Negative* spacing values spread pins to the left along a horizontal block edge, or down along a vertical block edge.

**Note:** You cannot specify pin spacing distances with spacing methods that rely on the

Pin Editor to determine the spacing.

The following sections provide details on the four pin-spreading methods supported by the Pin Editor.

### Using a Pin as the Starting Point for Spreading Pins

For this method, you select a group of pins and sort them in the desired order. The *first pin in the list* serves as the starting point (anchor) for spreading the other pins in the group.

You must also provide the pin spacing distance if you are spreading more than one pin.

Assume that your design contains four pins (*A1*, *A2*, *A3*, and *A4*) that are currently spaced 2.0  $\mu\text{m}$  apart. You want to spread the pins to the right with 3.0  $\mu\text{m}$  spacing, using *A1* as the starting point. To do this you must

1. Sort the pins so that *A1* is the first pin in the list.

The coordinates of *A1* appear in *Starting X/Y*.

2. Select *Spread – From Starting Point*.
3. Specify a *positive* spacing value: 3 . 0.

The following figure illustrates this situation:

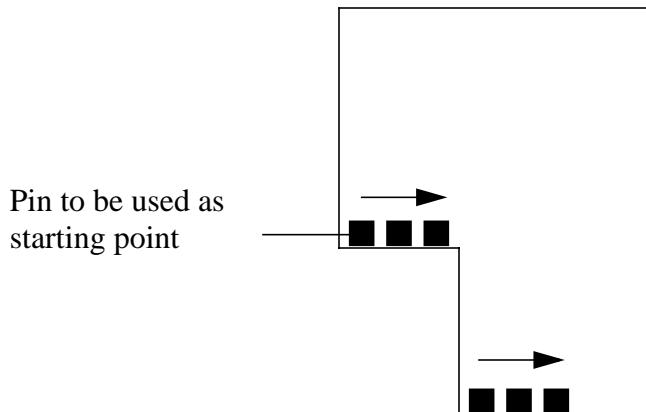
Original spacing: 2.0  $\mu\text{m}$



Desired spacing: 3.0  $\mu\text{m}$ , with *A1* as starting point and the pins spreading to right



The following figure shows how pins are spread from a pin as starting point in case of rectilinear partitions for a side with multiple edges. In this case, the specified side is bottom, and the pins are therefore spread along the edges of the bottom side.



Now assume that you want to spread the pins to the left with  $4.0 \mu\text{m}$  spacing, using  $A4$  as the starting point. To do this you must

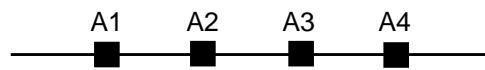
1. Sort the pins so that  $A4$  is the first pin in the list.

The coordinates of  $A4$  appear in *Starting*.

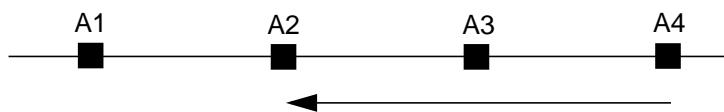
2. Specify a *negative* spacing value:  $-4.0$ .

The following figure illustrates this situation:

Original spacing:  $2.0 \mu\text{m}$



Desired spacing:  $4.0 \mu\text{m}$ , with  $A4$  as starting point and the pins spreading to the left



### Using the Center of a Side or Edge as the Starting Point for Spreading Pins

For this method, you select a group of pins and sort them in the desired order.

You must also provide the pin spacing distance.

Assume that your design contains four pins (*A1*, *A2*, *A3*, and *A4*). You want to define new spacing and then group the pins so that the group is centered on the midpoint of the block edge. To do this you must

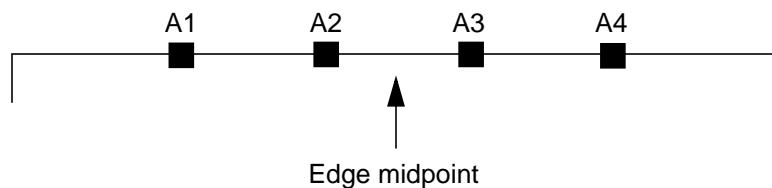
1. Sort the pins in the desired order (optional).
2. Select *Spread – From Center*.
3. Specify a *positive* spacing value: 3 . 0.

The following figure illustrates this situation:

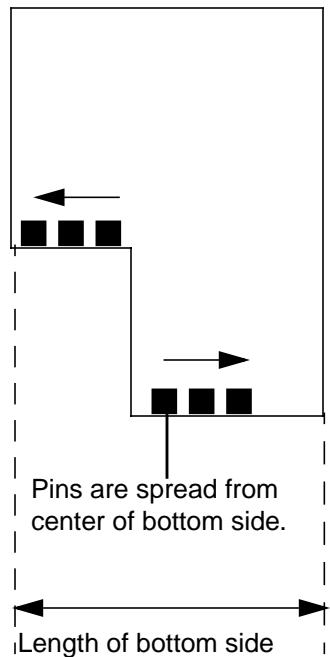
Original pin spacing



New spacing, with the pin group centered on the midpoint of the block edge



The following figure shows how pins are spread from a center point in case of rectilinear partitions where a side with multiple edges has been specified. In this case, the specified side is bottom, and the pins are therefore spread from the center of the bottom side.



### Spacing Pins Evenly Along an Edge or Side

For this method, you select a group of pins and sort them in the desired order.

You do not specify a pin spacing distance because the Pin Editor calculates the appropriate distance, based on the length of the block edge or side, and spaces the pins evenly along the block edge or side.

Assume that one edge of your design contains four pins ( $A_1$ ,  $A_2$ ,  $A_3$ , and  $A_4$ ). You want to spread the pins evenly along the block edge. To do this you must

1. Sort the pins in the desired order (optional).

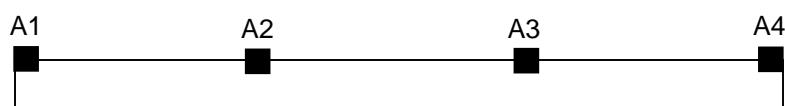
**2. Select Spread – Along Entire Edge.**

The following figure illustrates this situation:

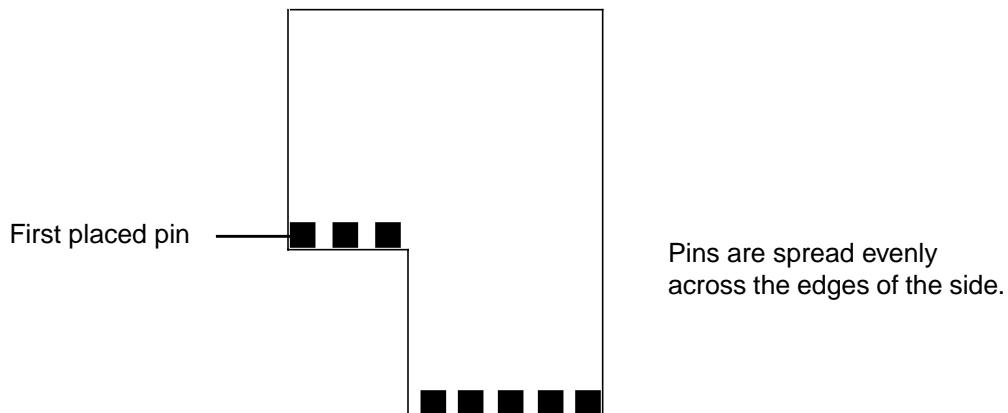
Original pin spacing



New spacing, with pins spread evenly between the ends of the block edge



The following figure shows how pins are spread along a side in case of rectilinear partitions where a side with multiple edges has been specified. In this case, the specified side is bottom, and the pins are, therefore, spread along the edges of the bottom side.



### **Spacing Pins Evenly Using Explicit Starting and Ending Points**

For this method, you select a group of pins and sort them in the desired order.

You do not specify a pin spacing distance because the Pin Editor calculates the appropriate distance, based on the specified starting and ending points, and spaces the pins evenly along the edge or side.

## Encounter User Guide

### Floorplanning the Design

Assume that one edge of your design contains four pins (*A1*, *A2*, *A3*, and *A4*). You want to spread the pins evenly along the block edge between two sets of coordinates. To do this you must

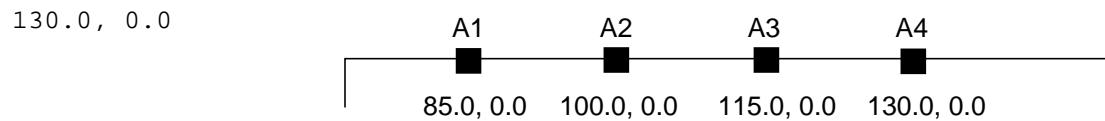
1. Sort the pins in the desired order (optional).
2. Select *Spread – Between Points*.
3. Revise the starting and ending coordinates as desired.

The following figure illustrates this situation:

Original pin spacing



New spacing, with pins spread evenly between the two sets of coordinates 85.0, 0.0 and 130.0, 0.0

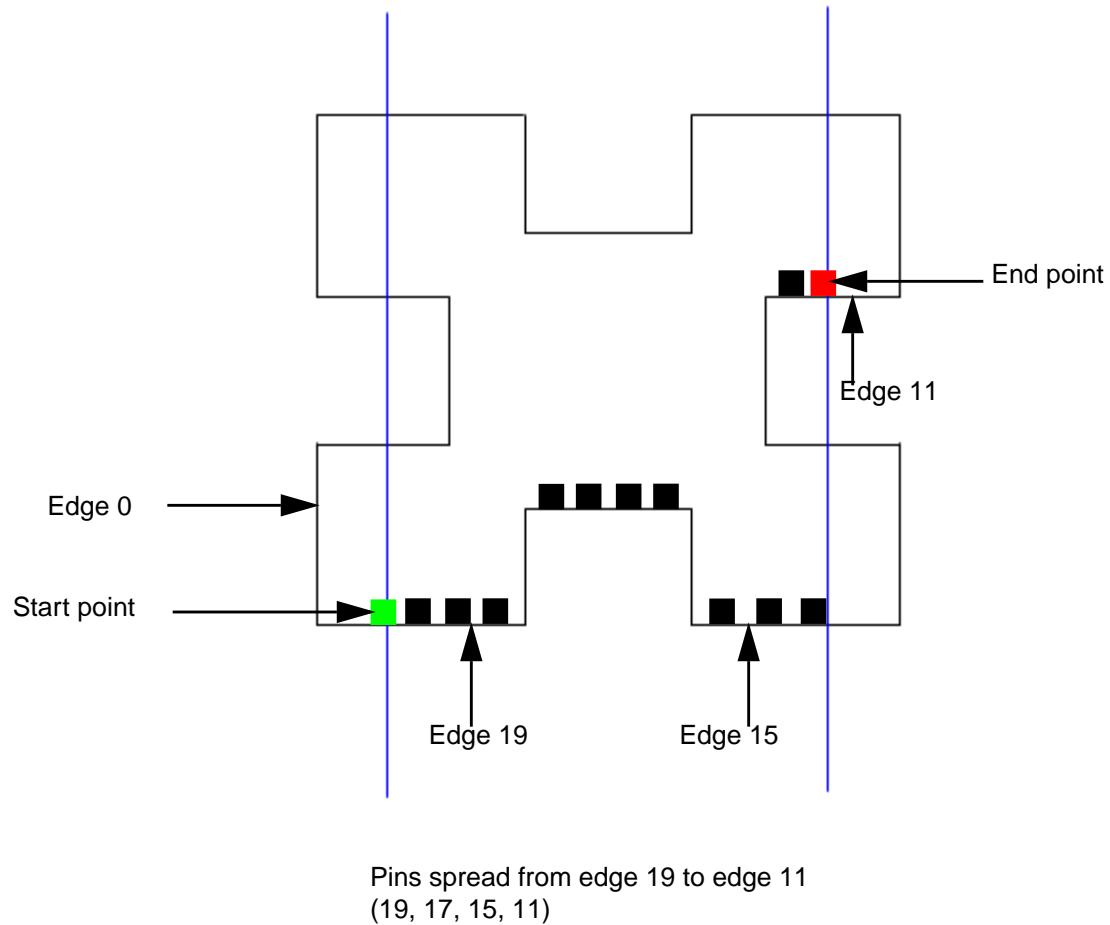


The following two figures shows how pins are spread along a side in case of rectilinear partitions where a side with multiple edges has been specified.

## Encounter User Guide

### Floorplanning the Design

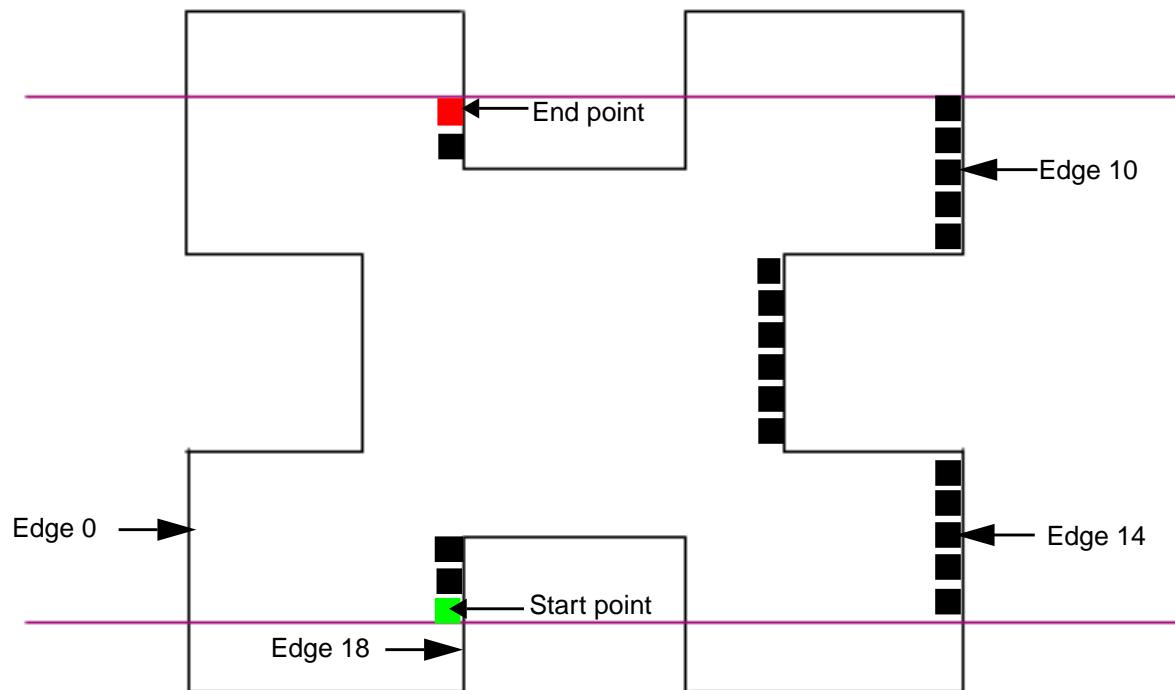
In the following case, the specified side is bottom, and the pins are, therefore, spread along the edges of the bottom side.



## Encounter User Guide

### Floorplanning the Design

In the following case, the specified side is right, and the pins are, therefore, spread along the edges of the right side.



Pins spread from edge 18 to edge 6  
(18, 14, 12, 10, 6)

## Running Relative Floorplanning

This section describes how to use the *Floorplan* menu's Relative Floorplan form to capture and define the placement relationship of floorplan objects independently from the actual coordinates in a floorplan.

The Relative Floorplan program provides a more flexible way to place objects, such as modules, blocks, groups, blockages, pin guides, pre-routed wires, and power domains. Block I/O pins can be used as reference objects but they cannot be relative objects. You can capture and define the placement relationship of floorplan objects independently from the actual coordinates in a floorplan. You can also resize a module or blackbox based on other floorplan objects, while maintaining its current area based on a specified width and height, a given dimension (width or height), and a target utilization value. You can also specify a wire's start or end point relative to the reference object's reference corner, or specify a wire's start or end point directly.

Before relative floorplanning, the design must be loaded into the current Encounter session.

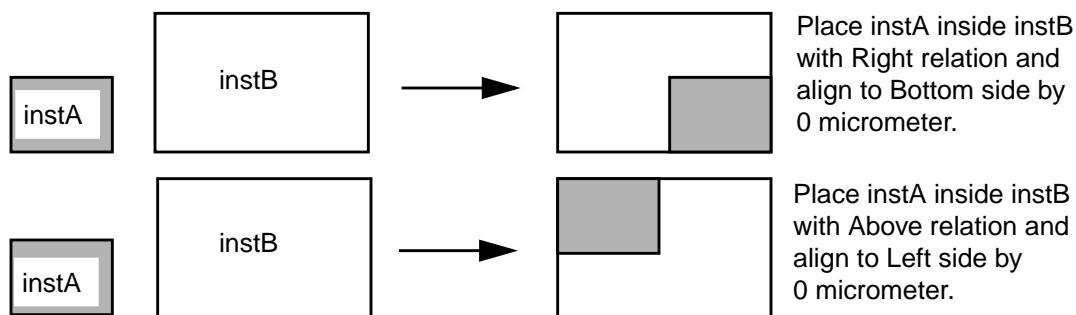
### Orientation Key

The following table is a key to the orientation of placed objects:

<b>Value</b>	<b>Definition</b>
R0	 No rotation
MX	 Mirror through X axis
MY	 Mirror through Y axis
R180	 Rotate counter-clockwise 180 degrees
MX90	 Mirror through X axis and rotate counter-clockwise 90 degrees
R90	 Rotate counter-clockwise 90 degrees
R270	 Rotate counter-clockwise 270 degrees
MY90	 Mirror through Y axis and rotate counter-clockwise 90 degrees

## Instance Place Example

The following figure shows an example of instance placement.

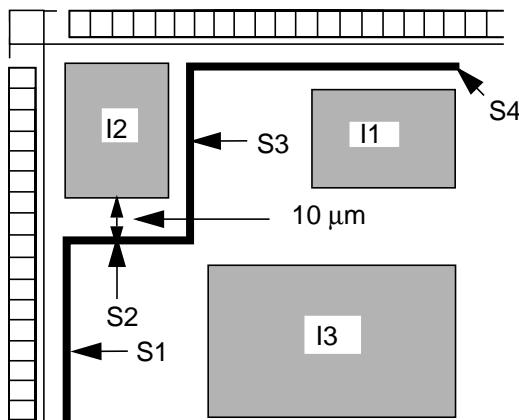


## Pre-Route Examples

You can generate pre-routed relative floorplan information automatically after you have pre-routed a relative floorplan. The following example and illustrations show how the relative floorplan pre-route feature operates.

The following commands specify that S3, S2, and S1 are relative to object I2 and the core:

```
relativeFPlan --preRoute VDD 1 2 0.44 0.44 I2 BR X2 Y2 I2 TR X3 Y3
relativeFPlan --preRoute VDD 1 2 0.44 0.44 I2 BL X1 Y1 I2 BR X2 Y2
relativeFPlan --preRoute VDD 1 2 0.44 0.44 core BL X0 Y0 I2 BL X1 Y1
```

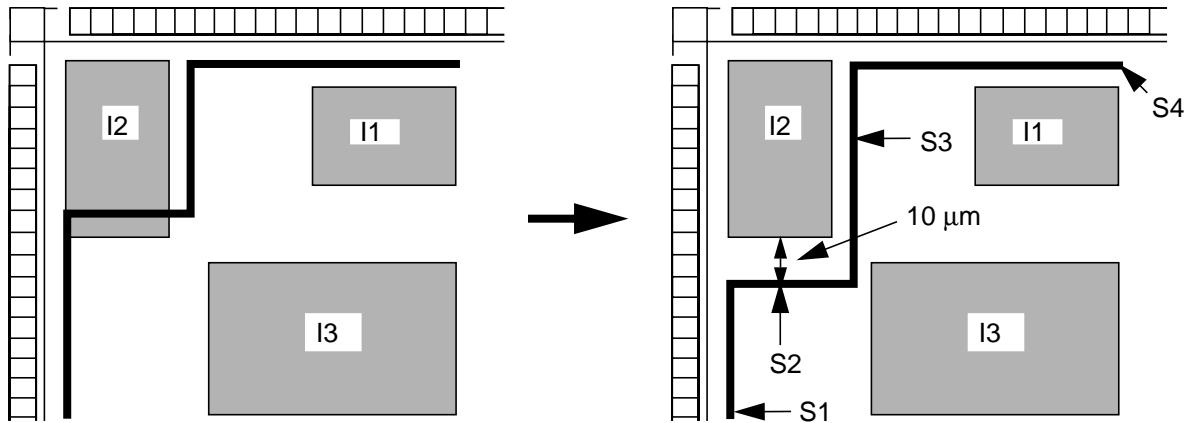


## Encounter User Guide

### Floorplanning the Design

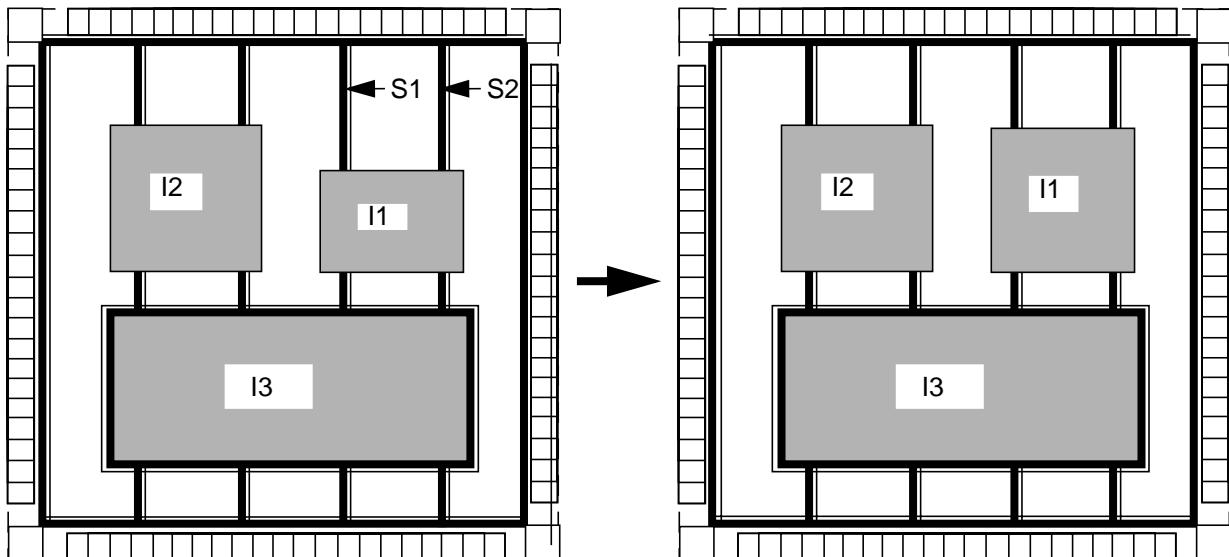
---

When I2 is stretched southward, the original distance between its south side and the S2 side of the wire is readjusted, but maintains its distance from the south side at 10 µm.



The following commands specify that S1 and S2 are relative to the object I2 and the Core\_Boundary:

```
relativeFPlan --preRoute VSS 1 2 0.44 0.44 I1 TL x1 y1 Core_Boundary TR x2 y2
relativeFPlan --preRoute VDD 1 2 0.44 0.44 I1 TR x3 y3 Core_Boundary TR x4 y4
relativeFPlan --preRoute VSS 1 2 0.44 0.44 I1 TL x5 y5 Core_Boundary TR y6 y6
relativeFPlan --preRoute VDD 1 2 0.44 0.44 I1 TR x7 y7 Core_Boundary TR x8 y8
```



In the above example, when I1 is stretched northward, the S1 and S2 wires are shortened.

## Saving and Restoring Relative Floorplan

The Encounter software automatically saves all the executed menu and text commands for the relative floorplanning actions in the `encounter.cmd` file.

To save all the relative floorplan commands that were executed during a session, click the Save button on the Relative Floorplan form. This saves a script that can be used later for updating or adjusting an existing floorplan based on the new blocks' size and position. You can also save the constraints associated with an object through the `saveRelativeFPlan` command.

To restore the relative floorplan information to the design display area, use the `restoreRelativeFPlan` text command.

For more information, see “Floorplan Commands” in the *Encounter Text Command Reference*.

## Saving and Loading Floorplan Data

You can save and load floorplan data at any time during a session.

- To save the floorplan information to a file, use the Save FPlan File form or the `saveFPlan` text command.
- To load the floorplan information from a file, use the Load FPlan File form or the `loadFPlan` text command.

### Related Topics

To see where this step fits in the design flow, see [Load and Check Data](#) in the *Encounter Flat Implementation Flow Guide*.

## Resizing the Floorplan

The resize floorplan feature enables you to resize a floorplan while maintaining the relative locations of the floorplan objects.

Normally, floorplan resizing is done

- to reduce the die size on a finished floorplan.
- to expand or shrink the die during floorplan creation, based on the full chip placement results.

You can use the [Floorplan - Resize Floorplan](#) form in the Encounter GUI to perform the resize functions in the design. Alternatively, you can also run the [`resizeFP`](#) command to resize the floorplan.

### Use Case 1

To reduce the die size on a finished floorplan, i.e floorplan of the previous tapeout.

1. Import the design.
2. Load the floorplan file or DEF file.
3. Run resize floorplan using the [`resizeFP`](#) command.
4. Run placement, trial route, and power planning. For more details, see [Common Floorplanning Sequence](#) on page 355.

### Use Case 2

To expand or shrink the die during floorplan creation, based on the full chip placement results.

1. Import the design.
2. Perform floorplanning based on the chip design floorplan.
3. Run placement and trial route to view placement and routing congestion.
4. Run resize floorplan to enlarge or shrink the die after placement and routing, using the [`resizeFP`](#) command.
5. Manually refine floorplan—include macro/IO placement, module constraints, blockages, power plan.

6. Rerun placement, if required. For more details, see [Common Floorplanning Sequence](#) on page 355.

## Resize Floorplan Options

The space among floorplan objects can be resized in two ways:

### Proportional Spacing

Distributes the space among floorplan objects proportionally (`resizeFP -proportional`). It can shrink or expand the space in both, X and/or Y directions. However, you cannot adjust pre-routed wires using proportional spacing.

See [Resize Floorplan Proportional mode](#) in the “Floorplan Menu” chapter of the *Encounter Menu Reference* for more information.

### Shift-based Spacing

Shifts the selected floorplan objects at appropriate location(s) without changing the location of all the existing floorplan objects. (`resizeFP -shiftBased`)

You can perform automatic resizing or resize the floorplan based on resize lines defined using the `setResizeLine` command. The shift-based resize maintains the existing pre-routed wire topology and automatically adjusts bus guides during resizing.

See [Resize Floorplan Shift Based mode](#) in the “Floorplan Menu” chapter of the *Encounter Menu Reference* for more information.

## Setting Resize Lines

For performing shift-based resizing, you must specify one or multiple resize lines. These resize lines must be orthogonal since they can overlap if specified in the same direction. For example,

```
setResizeLine -direction H (x1 y1) (x2 y1) (x2 y3) (x4 y3) (x4 y5) (x6 y5) -width 20
```

When specifying a resize line, if you do not specify a coordinate for the resize line, the `setResizeLine` command automatically derives the missing coordinate. For example, the following command automatically derives the missing coordinate for setting the resize line:

```
setResizeLine -direction H (x1 y1) (x2 *) -width 20
```

## Specifying Resize Directions

Resizing can be done in X and Y directions. Positive values mean expanding the space and negative values indicate shrinking. However, Encounter does not support a scenario where resizing line by expanding and shrinking, both occur on the same direction. For example, the following command specifies a resize line in vertical direction with a resize width of 100 microns:

```
setResizeLine -direction V (x1 y1) (* y2) -width 100
```

The following command again resizes the floorplan in the same X direction with a negative value of -200 microns:

```
resizeFP -xSize -200
```

Encounter displays a warning message in such a situation.

## Snapping Resize Values

The resize values (shrink/expand) can be snapped to a multiple integer of the metal layer pitch.

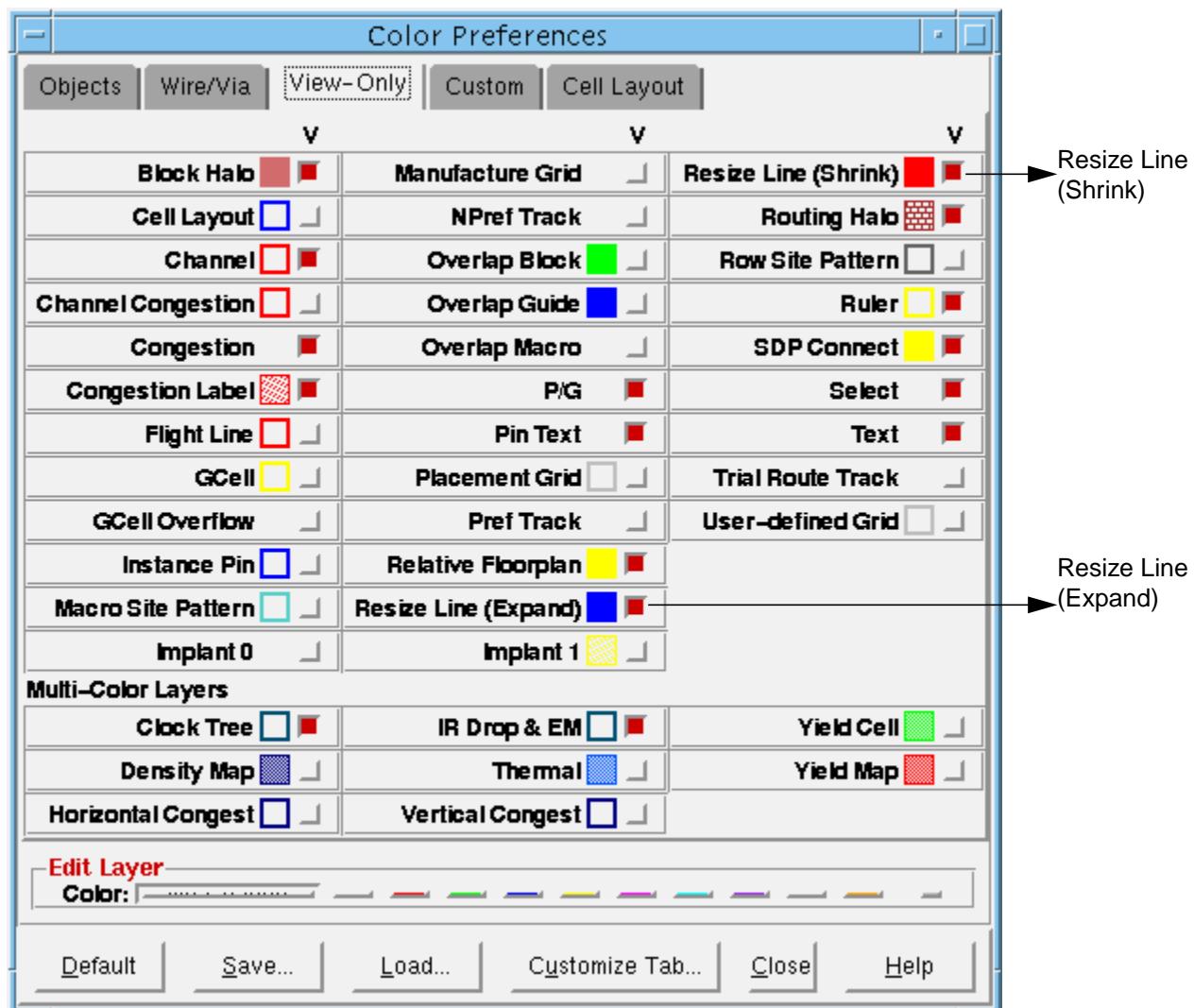
**Note:** Specify the `resizeFP -snapToTrack` option to snap resize values.

For example, if the horizontal metal pitch is 1.5 microns and you want to shrink the floorplan by 8 microns in y direction, the actual shrink value is 7.5 microns, the nearest multiple integer of the metal pitch.

See [Resize Floorplan](#) in the “Floorplan Menu” chapter of the *Encounter Menu Reference* for more information.

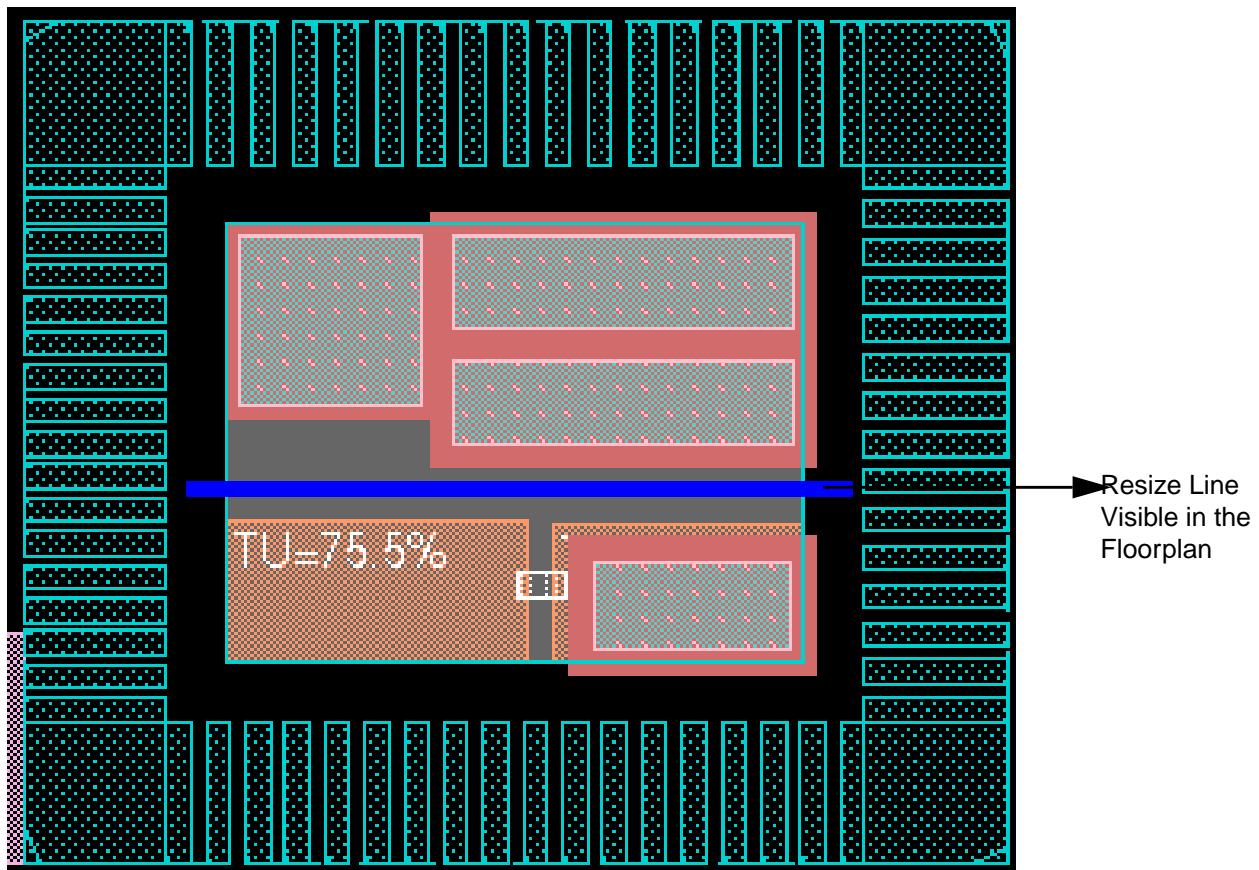
## Viewing Resize Lines using Color Preferences

Once you specify the resize lines to perform shift-based resizing, you can display the resize lines in Encounter by setting the *Resize Line* option in the *Color Preferences - View-Only* form.



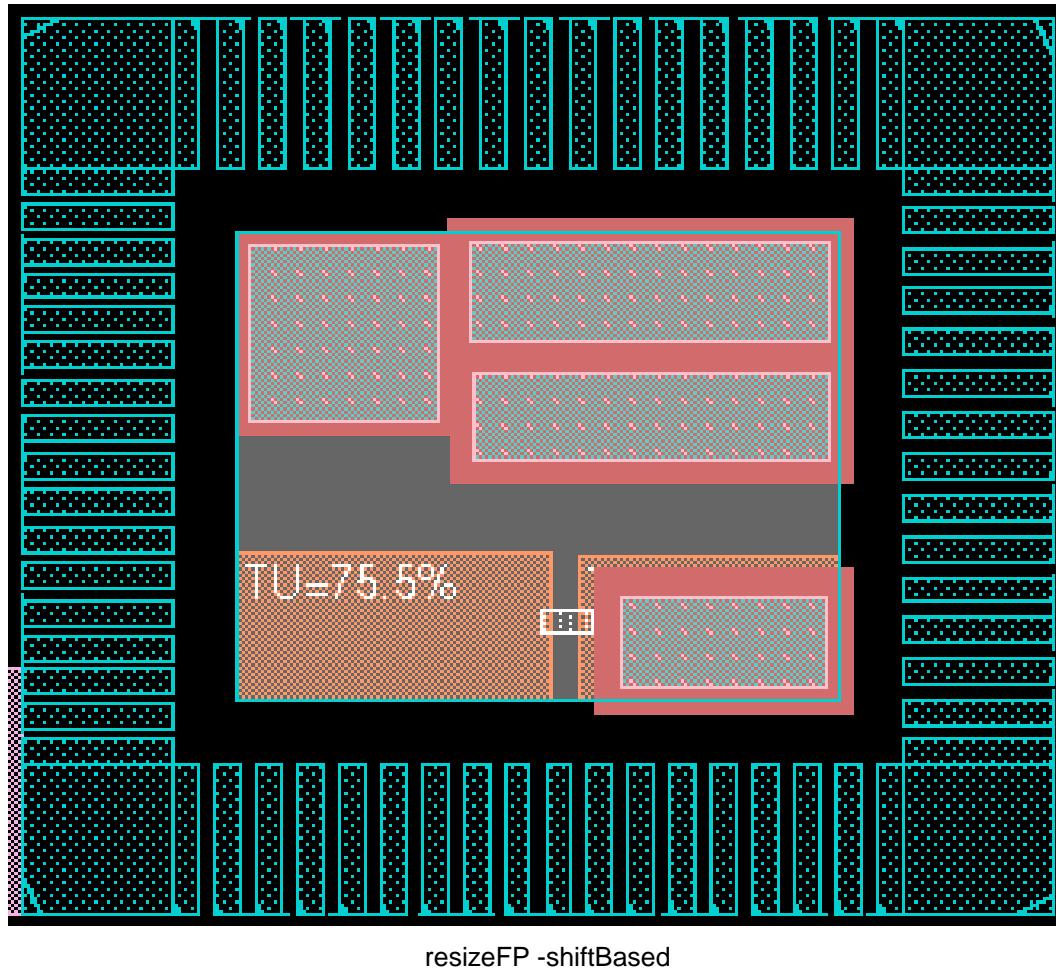
However, the resize lines disappear once you run the `resizeFP` command.

**Example 12-1 Setting a Resize Line Before running resizeFP**



```
setResizeLine -direction H -width -20.0000 (269.00 621.00) (1367.00 621.00)
```

**Example 12-2 Resize Line Disappears After Running `resizeFP -shiftBased` Command**



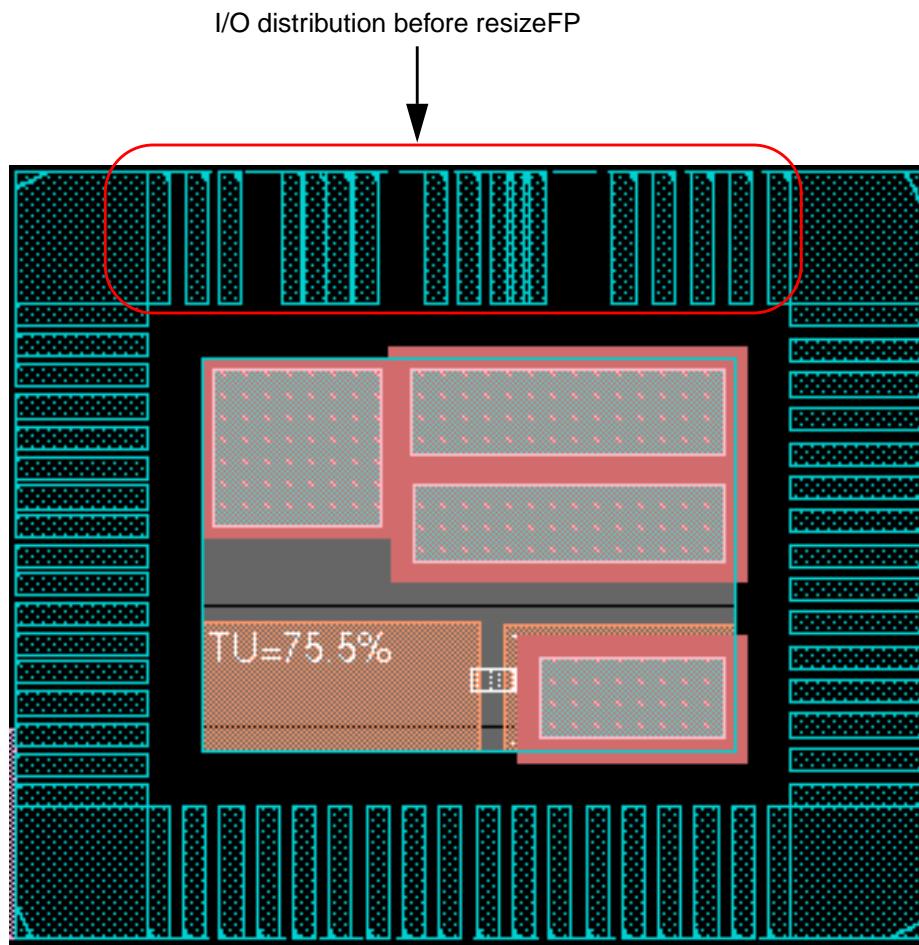
**Note:** During resizing, the target size may not be achievable. You have to force resize to meet the target size as much as possible, using the `resizeFP -forceResize` option.

### Distributing I/O's using Resize Floorplan

When distributing the I/O's using `resizeFP`,

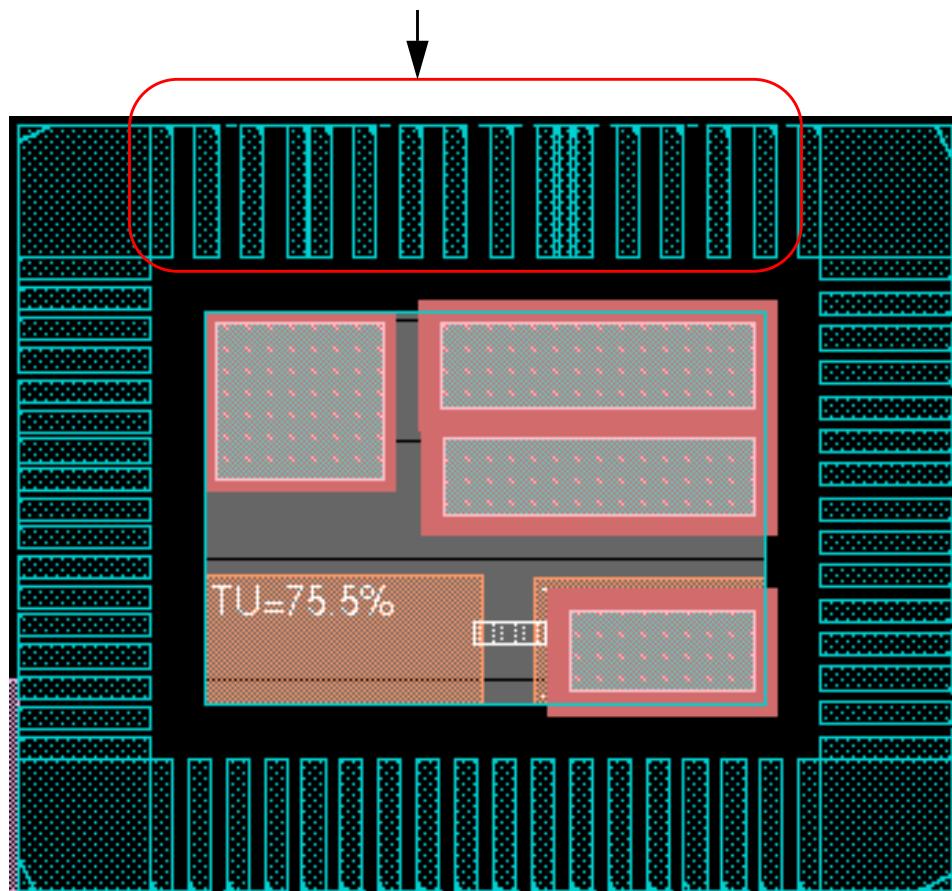
- The space between I/O pads can be adjusted evenly or proportionally. By default, the I/O's are distributed evenly.
- The I/O side constraints and order constraints are honored.
- The offset that exists between I/O's and the design boundary is preserved.

**Example 12-3 Distribution of I/Os Before resizeFP**



**Example 12-4 Distribution of I/O's after resizeFP -ioProportional**

Proportional I/O distribution after resizeFP -xSize 50 -ioProportional



---

## Power Planning and Routing

---

- [Overview](#) on page 436
- [Before You Begin](#) on page 437
- [Results](#) on page 438
- [Loading, Saving, and Updating Special Route](#) on page 438
- [Creating a Ring with User Defined Coordinates](#) on page 438
- [Global Net Connections](#) on page 439
- [Fixing LEF MINIMUMCUT Violations](#) on page 441
- [Fixing LEF Minimum Spacing Violations](#) on page 441
- [Adding Stripes to Power Domains](#) on page 441
- [Automatic Power Planning \(APP\)](#) on page 443
- [Creating a Template](#) on page 445
- [Specifying Template Parameters](#) on page 447
- [Instantiating a Template](#) on page 448
- [Using the Synthesize Power Plan Functionality](#) on page 449
- [Creating Differential Routing to Signal Bumps](#) on page 451

## Overview

Power planning and routing is composed of the following components:

- Adding a core ring
- Adding block rings
- Adding stripes to the core area
- Adding stripes over blocks within the design
- Adding ring pins
- Creating a pad ring
- Connecting pad pins
- Routing standard cell pins
- Connecting block pins
- Connecting unconnected stripe
- Routing to power bumps

Use the Encounter power planning features to create power structures such as rings, stripes, and ring pins for the design. To create power structures, complete the following steps:

1. Load a LEF file that contains technology information before you add power rings and power stripes. If the LEF file is not loaded, you will not be able to select metal layers on the power planning forms.
2. Specify the floorplan.
3. Establish logical power connectivity. You can issue the globalNetConnect command or use the Global Net Connections form, which is accessed from the Floorplan menu.
4. Cut standard cell rows around macros. Whenever a floorplan change is made, the rows must be cut. Issue the cutCoreRow command to cut the rows based on placement blockages.
  - To highlight the cut rows, issue the displayCutRow command.
  - To remove the display of cut rows, issue the clearCutRow command.
5. Add power rings around the core of the design, and block rings around blocks, row clusters, and power domains.

6. Add power stripes within the overall design, within a specific area, or over specified blocks or power domains.
7. Save special route data.

After your design is placed, you can use the Encounter power routing features to make the final power connections. The SRoute software creates pad rings and routes power and ground nets to the following power structures:

- Block pins
- Pad pins
- Standard cell pins
- Unconnected stripes

Use the SRoute form to specify routing to any or all of these structures. In addition, you can specify whether or not the software should make the connections by changing layers or allowing jogs.

## Before You Begin

Before you can begin power planning, the following conditions must be met:

- The design must be loaded into the current Encounter session.

The following input file is required:

- The design file

Before you can begin power routing, the following conditions must be met:

- The design must have power rings and stripes.

The following input file is required:

- A LEF file that contains technology and macro information.

## Results

After using the power planning software, your design has preliminary power structures that provide the foundation for hooking up each cell to a power source.

After using the power routing software, your design has power connections between pins of specified nets on the blocks and pads to nearby rings or stripes. Your design is ready for combined power and rail analysis to determine whether power structures and connections provide sufficient power to the design.

## Loading, Saving, and Updating Special Route

To load special route data into a design, use the [Load Special Route](#) form. When loading the floorplan, the `.fp` and `.fp.spr` files are included. The filename extension entered in the Load FPlan form is `.fp`, not `.fp.spr`.

When creating special routes, use the [Save Special Route](#) form or the [Save Design](#) form to save the special route data plus vias.

Floorplan files are saved with the following extensions:

- `.fp` — Contains the general floorplan information.
- `.fp.spr` — Contains the special route data.

You can also use the [Save DEF](#) form to save special route information in the DEF file.

## Creating a Ring with User Defined Coordinates

To create a block ring or a core ring in a specific location, complete the following steps:

1. Choose *Power – Power Planning – Add Rings*.  
This opens the *Basic* page of the Add Rings form.
2. Specify the net names for power rings to be created.
3. Select *User defined coordinates*.
4. Select either *Core ring* or *Block ring*.

If you select *Core ring*, the shape of the wires is `ring`. If you select *Block ring*, the shape of the wires is `blockring`.

**5. Specify a set of coordinates.**

You must specify at least four pairs of coordinates. Specify the x coordinate followed by the y coordinate for each corner of the ring. Separate each coordinate with a space.

For example, to define a 100 x 100 square ring at the bottom left corner of the design, specify the coordinates as follows:

```
0 0 0 100 100 100 100 0
```

You can create a rectilinear ring by specifying an even number of coordinate pairs. For example, specify the following set of coordinates to create an L-shaped ring:

```
0 0 0 100 50 100 50 50 100 50 100 0
```

You must specify the coordinates in a linear sequence. For example, if you specify the coordinates in the following sequence, the ring is not created because the sequence of the coordinates defines the bottom left, top right, top left, and top right corners:

```
0 0 100 100 0 100 100 0
```

You must also specify coordinates that create perpendicular wires. For example, if you specify the following coordinates, the ring is not created because the coordinates define an edge that slants:

```
0 0 0 100 100 100 50 0
```

**6. Click *Apply* or *OK*.**

The ring is created in the exact location specified by the coordinates.

## Global Net Connections

Global net connections connect terminals and nets to the appropriate power and ground nets so that power planning, power routing, detail routing, and power analysis functions operate correctly for the entire design. Some of these terminals and nets are contained in the Verilog® netlist, and others are contained in the LEF file.

From the Verilog netlist, you can connect the following type of nets to power and ground nets:

- Power and ground nets

Connect between the power and ground nets to the appropriate power and ground nets. These power and ground nets are `wire` keywords in the Verilog netlist.

- Tie-hi and tie-lo nets

Connect between the tie-hi and tie-lo nets to the appropriate power and ground nets. These are keywords in the Verilog netlist, such as `1'b0`, `1'b1`, `supply0`, and `supply1`.

- Local nets

Connect between the local nets to the global nets. These local nets are `wire` keywords in the Verilog netlist.

From the LEF file, you can connect the following type of terminals and nets to power and ground nets:

- Power and ground terminals

Connect between the power pins to the appropriate power and ground nets. `vdd!` and `gnd!` are examples of these power and ground pin and net names in the LEF file.

- Filler cell nets

Connect between the power pins to the appropriate power and ground nets. You can specify these connections before or after adding filler cells.

To assign pins or nets to a global net, use the Global Net Connections form (*Floorplan – Global Net Connections*). For more information, see [Global Net Connections](#) in the “Floorplan Menu” chapter of the *Encounter Menu Reference*.

 *Important*

The order of global net connections are important, especially when the *Apply All* or the *Override prior connection* options are selected in the Global Net Connections form. *Apply All* connects all pins or nets in the design to the specified global net. *Override prior connection* first disconnects pins and local nets that are already connected to a global net, then reconnects them to the specified global net specified in the form.

You can also use the [globalNetConnect](#) text command to assign global net connections. For more information, see “Power Planning Commands” in the *Encounter Text Command Reference*.

## **globalNetConnect Command and Connections for Signal Pins and Power/Ground Pins**

The `globalNetConnect -type net` command automatically connects *only* signal pins to the global net—not power or ground pins.

To reconnect power or ground pins to the global net after using the `commitConfig` command, use the command `globalNetConnect -type pgpin -pin pin_name -all -override`. All global net connection rules and any rules specified in the configuration file are saved in the floorplan file.

## Fixing LEF MINIMUMCUT Violations

If your design contains violations to the LEF MINIMUMCUT rule, use the [fixMinCutVia](#) command as a post-processing step. This command replaces power vias flagged by a violation marker because of a violation of the LEF MINIMUMCUT rule. The via is replaced with a via that contains the minimum number of cuts based on the LEF rule, and the violation marker is removed. If the via cannot be replaced, the violation marker is not removed.

See the “Power Route Commands” in the *Encounter Text Command Reference* for more information.

## Fixing LEF Minimum Spacing Violations

If your design contains violations of the LEF minimum spacing rule, use the [fillNotch](#) command as a post-processing step. The command fill gaps, which removes same net violations between generated vias and wires or pins where these violations are flagged by the Verify Geometry software. Using this command, you do not have to sacrifice via size by trimming vias in order to fix same net spacing violations.

See “Verify Commands” in the *Encounter Text Command Reference* for more information.

## Adding Stripes to Power Domains

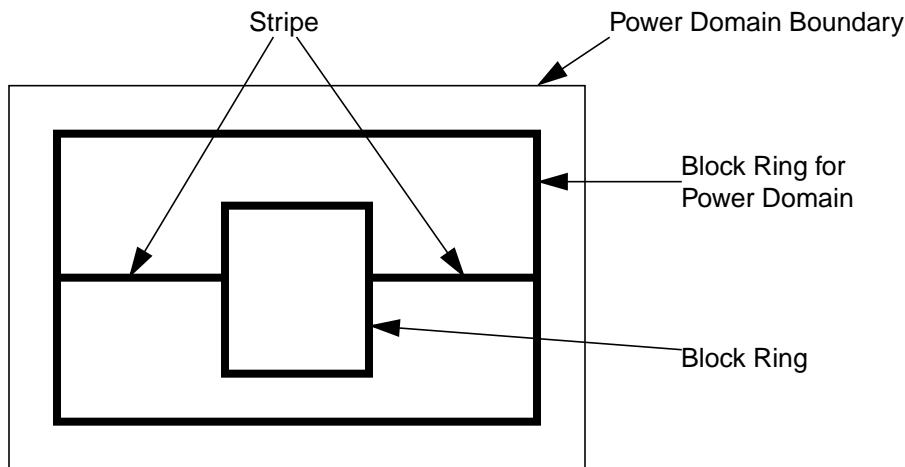
When you use the *Each selected block/domain/fence* option on the [Basic](#) page of the Add Stripes form, stripes are placed according to the location of the power domain ring. Sometimes the location of the power domain ring was not specified at the time the power domain was created. In this case, you must indicate the location of power domain rings to the power planning software by issuing the following command:

```
modifyPowerDomainAttr -rsExts top bottom left right
```

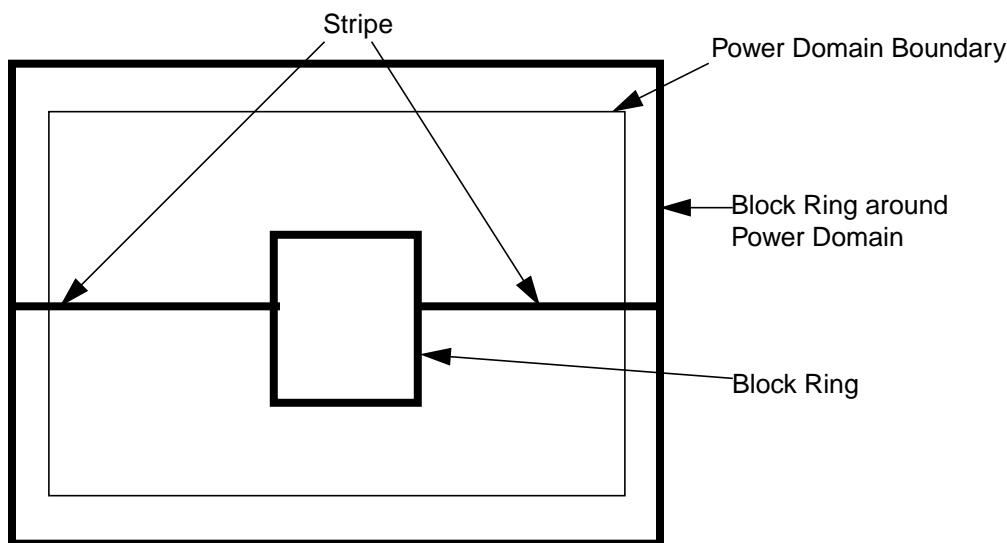
The *top*, *bottom*, *left*, and *right* values specify a distance from the edge of the power domain boundary. Rings between the power domain boundary and the specified distance are considered power domain rings.

- Specify negative values if the power domain ring is inside the power domain boundary. The power planning software considers any ring from the power domain boundary to the specified distance *inside* the boundary to be a power domain ring. When you add stripes, the software trims the stripes at the ring. The stripe also correctly recognizes block rings within the power domain and breaks at those rings if you specify the *Omit*

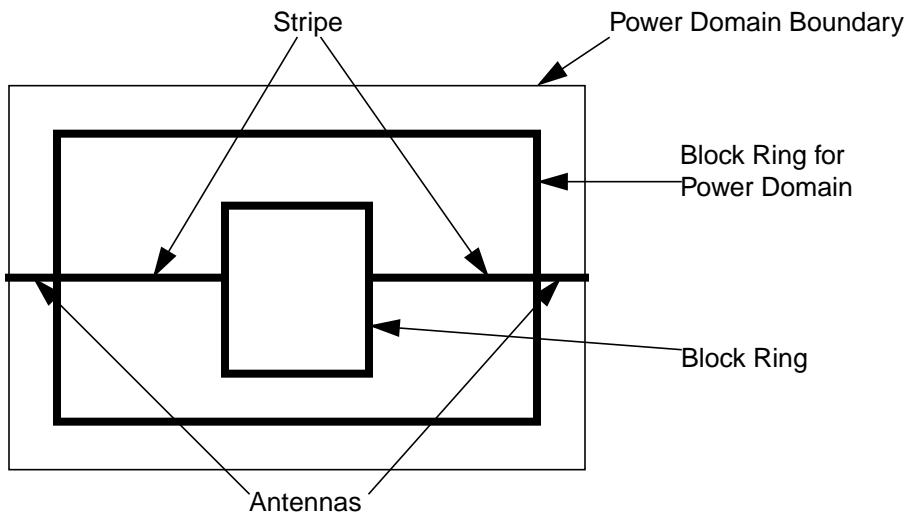
*stripes inside block rings* option on the [Advanced](#) page of the Add Stripes form. The following illustration shows how the stripe is created when you specify negative values.



- Specify positive values if the power domain ring is outside the power domain boundary. The power planning software considers any ring from the power domain boundary to the specified distance *outside* the boundary to be a power domain ring, and extends the stripes to that ring. The stripe also correctly recognizes block rings within the power domain and breaks at those rings if you specify the *Omit stripes inside block rings* option on the [Advanced](#) page of the Add Stripes form. The following illustration shows how the stripe is created when you specify positive values.



If the location of the power domain ring is not specified, the stripe begins and ends at the power domain boundary. If the power domain ring is inside the power domain, the stripe recognizes it as a block ring. This can cause the stripe to break in the wrong locations if you specify the *Omit stripes inside block rings* option on the *Advanced* page of the Add Stripes form, and can create antennas, as shown in the following illustration.



## Automatic Power Planning (APP)

With traditional power planning software, you must add core rings, pad rings, block rings, horizontal stripes, and vertical stripes separately. You could obtain this type of power plan by either issuing separate text commands or by filling out the Add Ring, Add Stripe, and SRoute forms multiple times.

Using the APP software, you can create a power plan template that includes all of these features. The template can be applied to specific designs, providing a simpler method for creating a cohesive power plan. You can also use the same template when the floorplan is modified to easily regenerate power structures.

Once you have created a template, you can instantiate it with design-specific information, such as the width and pitch of stripes. The Specify Template Parameter form also provides you with tools that help you determine the power needs of the design so that you can use these templates within a design to come close to a prototype power plan.

When creating a power plan using the synthesize power plan (SPP) functionality, the APP software uses an estimated power value that is based on the power constructs in the timing library files (.libs) and the capacitance values in the LEF file, an optional IR-drop threshold

value, an optional user template to analyze the design data, the floorplan data, and the cell library data to create a base system template. The APP software then uses this base system template, user template (if specified), and the power value to estimate the ring width, stripe width, and pitch to create the power structure.

The power grid synthesis (PGS) feature of the APP software automatically creates and optimizes the power structure by properly scaling wire width and stripe pitch to meet IR-drop design constraints when such a threshold is specified.

You can also use the APP software to optimize an existing power structure.

To access the APP software, select *Power – Power Planning – Synthesize Power Plan* or *Power – Power Planning – Optimize Power Plan*.

## Creating a Template

The Edit Power Plan Template form allows you to set up a generic power plan for any design, as well as specific power plans for IP blocks within a particular design.

To create a template, complete the following steps.

1. Choose *Power – Power Planning – Synthesize Power Plan*.  
This opens the Synthesize Power Plan form.
2. Click *Use template to create power plan*.
3. Click *Design or IP*.
4. Click
  - The *New* icon to create a new template
  - The *Open* icon to view or edit an existing template after you choose a template from the drop-down list.

## Using the IP Block Page

1. Click the *IP* radio button on the Synthesize Power Plan form, then the *New* or *Open* icons.

This opens the *IP Block* page of the Edit Power Plan Template form. This page contains options that specify ring and stripe properties for each IP block. This page is divided into three areas:

- The upper-left quadrant contains a list of all IP blocks in the design. This list is obtained from the LEF file.
- The lower-left quadrant shows a preview of how the selected block or blocks will look in the design.
- The right side contains options for specifying rings around blocks and stripes over blocks. It also contains buttons that let you set or unset the options for the blocks selected in the upper right quadrant.

This part of the form has two subtabs:

- Use the *Block Ring* tab to specify whether a block ring is required for a particular IP block whether a block ring is to be created, and if so, whether the block ring can be shared by other blocks.

- Use the *Stripe* tab to specify whether stripes should be created over a particular block, and if so, which layer, width, and pitch values are to be used.
2. Select one or more IP blocks from the list in the upper left quadrant.
  3. Click the *Block Ring* tab, then specify block ring options for the selected IP blocks.
  4. Click the *Stripes* tab, then select *Stripe over the block*.
  5. Select the *R0* option, then specify stripe configuration options for the selected IP blocks when they are not rotated from their original positions.
  6. Select the *R90* option, then specify stripe configuration options for the selected IP blocks when they are rotated 90 degrees from their original positions.
  7. When the display in the upper right quadrant looks correct, click the *Set* button.
  8. Repeat steps 3-7 for additional IP blocks.
  9. When all blocks that require block rings or stripes have been set:
    - Click *Save* to save the configuration to the existing template.
    - Click *Save as*, which opens the *Specify Template Name* form; specify a name in *Template Name*, then click *OK*.

## Using the Design Page

1. Click the *Design* radio button on the *Synthesize Power Plan* form, then the *New* or *Open* icons.

This opens the *Design* page of the *Edit Power Plan Template* form. This page contains ring and stripe configuration options for the core area of the design. This page is divided into four quadrants:

- The upper left quadrant, *Display Template Hierarchy*, displays the name(s) of regions, IP library template, and option set.
- The upper right quadrant allows you to define ring and stripe characteristics for a region in the design, and allows you to add or modify regions in the design.
- The lower left quadrant shows a preview of how the rings and stripes will be created in the actual design.
- The lower right quadrant allows you to define region, net template, IP library template, and power planning option set names.

2. Specify the *Ring* and *Stripe* configurations in the upper right quadrant.

The configuration is shown in the lower left quadrant.

3. Specify a region name to indicate the configuration to be used at the top level of the design or by a particular power domain.
4. Specify a name in the *Net Template Name* field in the lower right quadrant of the form if you want the configuration to be used by a particular power or ground net.

The software uses the region name as the net template name to indicate a configuration for use by both the power net and ground net when the *Net Template Name* field is empty.

5. Select the *IP Library Template* you want to use from the dropdown list in the lower right quadrant of the form.
6. Choose a power planning option set from the dropdown list if you want to use a set of special stripe or ring options.

Use the create/edit option set icon to edit or view the option set you have chosen.

7. Click the *IP Block* tab to view (but not edit) the IP library template used after you add and double-click the region template in *Design Template Hierarchy*.
8. When you have finished configuring all region templates:

- Click *Save* to save the configuration to the existing template.
- Click *Save as*, which opens the *Specify Template Name* form; specify a name in *Template Name*, then click *OK*.

## Specifying Template Parameters

When specifying the template parameters, a system template is not required. The software creates a system template if one is not provided. The template parameters are based on the system template and the total average power value that is specified. You can specify the template parameters by clicking on the *Specify Template Parameter* icon, which allows you to add specific information for the chip-level power structures, such as the width, offset, and stripe count or pitch for each layer.

The *Design* and *Template* windows of this form allow you to associate templates with design components.

The appearance of the top portion of this form (*Template Parameter*) depends on the power structures you have selected. The bottom half of the form (*Configure Template Parameter*) is always the same. The top half of the form contains a set of default values based on the total power value specified on the Synthesize Power Plan form.

The *Configure Template Parameter* area contains options for using power analysis data to provide better estimates for the width, offset, and count/pitch values in the top portion of the form. You can click the *Update Template Parameter* button to update them. You can also modify these parameter values manually.

## Instantiating a Template

In general, to create a power structure with the synthesize power plan software, you need three sets of data:

- A power value for the entire design (estimated by the APP software)
- The IR-drop threshold for the entire design (optional)
- A power plan template—either a design template or IP library template (optional)

If you do not provide a template, the APP software creates a base system template. You can also start with an IP library template and the APP software will create the top-level design template.

To use power grid synthesis (PGS), specify an IR-drop threshold for the design whether you provide a power plan template or not.

## Template Naming Conventions

APP creates output templates according to the following naming conventions.

Input template for APP		Output template from APP	
Design template	IP template	Design template	IP template
<i>Without template</i>	—	appD	appD_appR

## Encounter User Guide

### Power Planning and Routing

---

Input template for APP		Output template from APP	
Design template	IP template	Design template	IP template
<i>With user's template</i>			
uD	—	appD_uD	appD_uD_appR_regionN
—	uIP	appL_uIP	appL_uIP_appR
<i>With system template</i>			
appD	—	appD	appD_appR
appD_uD	—	appD_uD	appD_uD_appR_regionN
appL_uIP	—	appL_uIP	appL_uIP_appR
—	appD_appR	appD	appD_appR
—	appD_uD_appR_region1	appD_uD	appD_uD_appR
—	appL_uIP_appR	appL_uIP	appL_uIP_appR

## Using the Synthesize Power Plan Functionality

The synthesize power plan functionality (SPP) can quickly create a power plan using only the total average power value (estimated by the APP software) and the layer electromigration (EM) limits. Such power plan creation is useful for both prototyping and regular flows.

- In a prototyping flow, you only need to provide the layer EM limits to generate a power plan.
- In a regular flow, you need to provide the layer EM limits and IR-drop threshold values to generate a non-pessimistic power plan. (Set the EM limits by choosing *Power – Power Planning – Specify Electromigration* to open the Specify Electromigration Limits form.)
  - To compute the parameters for the template, SPP takes the EM limits and IR-drop threshold (optional) into consideration.
  - Cadence strongly recommends that you manually define the EM limits in the Specify Electromigration Limits form if a `DCCURRENTDENSITY` specification is not available in the LEF file.

When you use SPP, IR-drop threshold is an optional value. If you do not provide an IR-drop threshold value, then the resulting power plan should be considered a prototype power plan.

The SPP process consists of the following general steps.

**1. Creating a base system template.**

SPP analyzes the design data, floorplan data, and cell library data to define the base system template.

SPP takes power domains into account and creates separate base system templates for each domain.

The base system template contains the following information:

- Core ring layer information (required); core ring information (optional). SPP analyzes the floorplan and block data to determine the core ring layers.
- Pad ring information (optional). SPP analyzes the pad cells and determines the pad ring requirements.
- Block ring layers and width per cell (required); block rings (optional). SPP analyzes block data to determine the block ring layers and width.
- Stripe layers (required); stripes (optional). SPP uses the information gathered for core rings and block rings to determine the stripe layers.

Use the [Edit Power Plan Template](#) form to define custom templates for the designs. If you have a custom template, then SPP overwrites the base system template with the custom template.

**2. Estimating the template parameters.**

SPP determines the width, spacing, and offset for core rings, and the width, spacing, offset, and pitch for stripes.

- SPP tries to create a finer mesh so that every block has stripes nearby.
- Template parameters are estimated for a prototype-based flow, hence the parameters are slightly pessimistic.

**3. Creating a power plan.**

SPP creates the power plan with the template and para meters that it calculated in [step 1](#) and [step 2](#).

When creating the power plan, SPP considers the following:

- Blocks to be excluded while creating core rings.
- The clustering of blocks to create shared block rings.

- ❑ The topmost stripe layers go over the blocks and bottommost layers break at the block rings.
- ❑ Stripes are extended based on the presence of outermost targets.

#### 4. Refining the power plan.

Once the power plan is created, SPP runs fast rail analysis to determine whether the power plan generated in [step 3](#) meets the IR-drop threshold constraint or not. If the power plan does not meet the IR-drop threshold constraint, SPP adjusts the power plan size virtually and determines the type of changes required in the prototype power plan to meet the IR-drop threshold constraint.

#### 5. Recreating the power plan.

Based on the feedback from [step 4](#), SPP updates the power plan.

## Creating Differential Routing to Signal Bumps

Differential routing creates wires of the same length or configuration between a set of sources and targets. Use the [Route Flip Chip Signal](#) form to specify differential routing parameters.

You can create a constraint file to specify differential pair definitions, as well as shield net definitions and differential group definitions. The following example shows how to set up a constraint file:

```
##### Differential pair definition
###balanced routing with shielding
out[10] PAIR out[11] SHIELDNET VDD
###balanced routing without shielding
out[8] PAIR out[9]

#####
Shield net defintion
out[5] SHIELDNET VDD
out[3] SHIELDNET VSS

#####
Differential group definition
out[15] GROUP out[16] out[17] out[18] resetn
clk GROUP out[12] out[13]
```

**Encounter User Guide**  
Power Planning and Routing

---

---

## Low Power Design

---

- [Overview](#) on page 454
- [Power Domain Shutdown and Scaling](#) on page 454
- [Support for the Common Power Format \(CPF\)](#) on page 456
- [Multiple Supply Voltage Flat Flow](#) on page 459
- [Multiple Supply Voltage Top-Down Hierarchical Flow](#) on page 479
- [Multiple Supply Voltage Bottom-Up Hierarchical Flow](#) on page 494
- [Leakage Power Optimization Techniques](#) on page 498
- [Power Shutdown Techniques](#) on page 503
- [Power Switch Optimization](#) on page 533

## Overview

This chapter describes how the multiple supply voltage (MSV) feature can help you save power in your design.

There are two types of MSV designs:

- **Multiple Supply Single Voltage (MSSV)**

Core logic runs at a single voltage, but some portions of the logic are isolated on their own power supply.

- **Multiple Supply Multiple Voltage (MSMV)**

Supplies of different voltages are used for core logic.

A power domain (also known as voltage island) is a floorplan object in the Encounter software. A power domain has a fence constraint and a specific library (.lib, .lef) associated with it. Cells that belong to a power domain can be placed only within that power domain. The exception to this rule is level shifter cells, which are used to communicate between power domains of different voltages. By constraining the design this way, a complete place and route flow can be used on an MSV design. You can automatically place level shifters, perform timing optimization, run clock tree synthesis (CTS) across domain boundaries, and obtain DRC-clean power routing.

## Power Domain Shutdown and Scaling

You can reduce power consumption either by shutting down a power domain or operating it at a reduced voltage (voltage scaling).

Power domain shutdown is a technique in which an entire power domain is shut down during a specific mode of operation. This results in both leakage power and dynamic power savings because the transistors are isolated from the supply and ground lines. You must use isolation cells when shutting down domains in order to drive the interface signals to predetermined known states. In many cases, a design in the shutdown mode operates at a single voltage throughout the design (an MSSV design); however, the portion of the design that is shut off must be in a different power domain. This is necessary because this portion must be isolated from the rest of the system so that it can be shut off independently from the rest of the core logic. For more information on power shutdown, see [“Power Shutdown Techniques”](#) on page 503

In power domain scaling (also known as voltage scaling), one or more domains operate at a lower voltage than that of the other core logic. Power domain scaling provides dynamic power

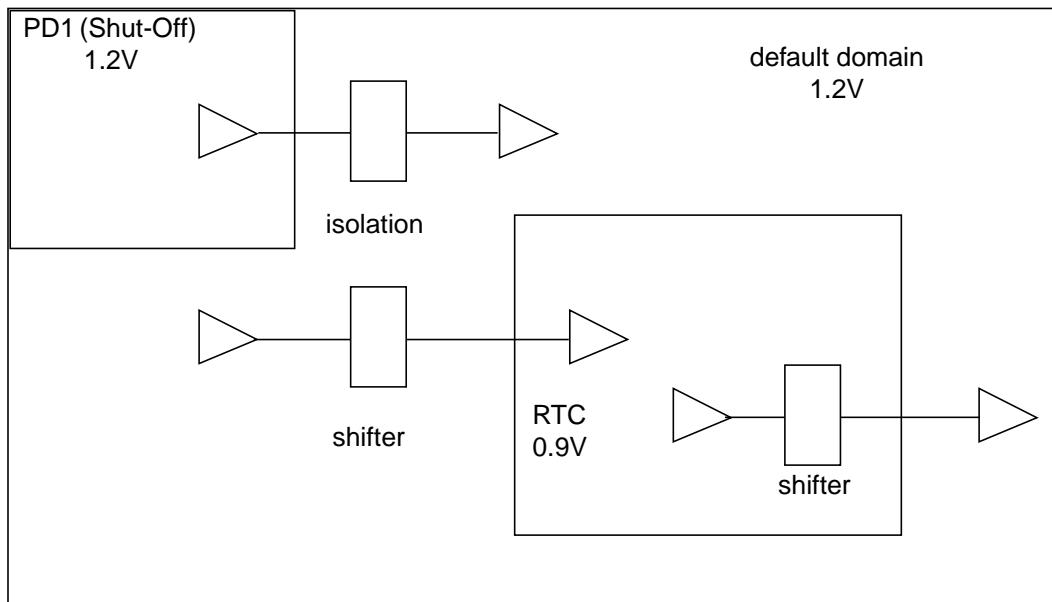
## Encounter User Guide

### Low Power Design

savings, and can provide leakage power savings, depending on the threshold characteristics of the library for the scaled domain.

**Note:** These techniques can be used separately or together in the same design.

The following figure shows three power domains: RTC, PD1, and the default power domain, which contains PD1 and RTC.



- The default power domain contains the PD1 and RTC power domains. PD1 and the default domain can share libraries.
- Power domain PD1 and the default power domain operate at the same voltage. Power switches enable PD1 to shut down.
- Power domain RTC operates at a different voltage than PD1 and the default domain.
- RTC can always remain on.
- You must insert voltage level shifters between the default domain and RTC, and between PD1 and RTC.
- Isolation cells (clamps) drive outputs of a power domain to known states when that power domain is shut down.

## Support for the Common Power Format (CPF)

Cadence provides a Common Power Format (CPF) that enables you to freely exchange data between Cadence tools supporting the low-power design flows, and most importantly, capture low power design intent early in the design process rather than late in the back-end cycle. A CPF file captures all design and technology-related power constraints, which can be used throughout the design flow.

You can do one or both of the following:

- Save a native Encounter design as CPF
- Create a Common Power Format (CPF) file, read the CPF file into Encounter, run the commands, and create a CPF database

CPF commands perform functions such as the following:

- Creating power domains and specifying their power/ground connections
- Specifying timing libraries
- Creating analysis view and defining library set for each power domain
- Defining operating conditions
- Defining low power cells
- Creating low power rules: isolation rules, level shifter rules, SRPG rules, power switch rules)

### CPF Version Support

The Encounter software supports the following versions of CPF:

- CPF 1.0 (default)
- CPF 1.0e
- CPF 1.1

### Encounter Commands Supporting CPF

- [loadCPF](#)
- [commitCPF](#)

- saveCPF

## Loading and Committing a CPF File

A GUI enables you to load and commit a CPF file:

- Power – Multiple Supply Voltage – Load/Commit CPF

This GUI corresponds to the following text commands:

- loadCPF

Reads a CPF file into Encounter for error checking

- commitCPF

Executes (commits) the CPF commands within the Encounter environment

## Saving a CPF Database

The saveDesign command prevents you from saving an Encounter database from obsolete Encounter MSV commands.

By default, if the design was created with MSV commands only and no CPF file, then the design cannot be saved to the Encounter database with saveDesign.

Complete the following steps to save a CPF database:

1. Save the CPF file from an Encounter database.

saveCPF

2. Exit the current session

3. Start a new session

4. Load the CPF file into Encounter.

loadCPF

5. Execute the commands in the CPF file.

commitCPF

6. Save the design.

saveDesign

## CPF Documentation

For more information about CPF, see the following documents:

■ *[Encounter Text Command Reference](#)*

Documents the following Encounter commands supporting the CPF flow. Descriptions on obsolete native Encounter commands are provided.

■ *[Encounter Menu Reference](#)*

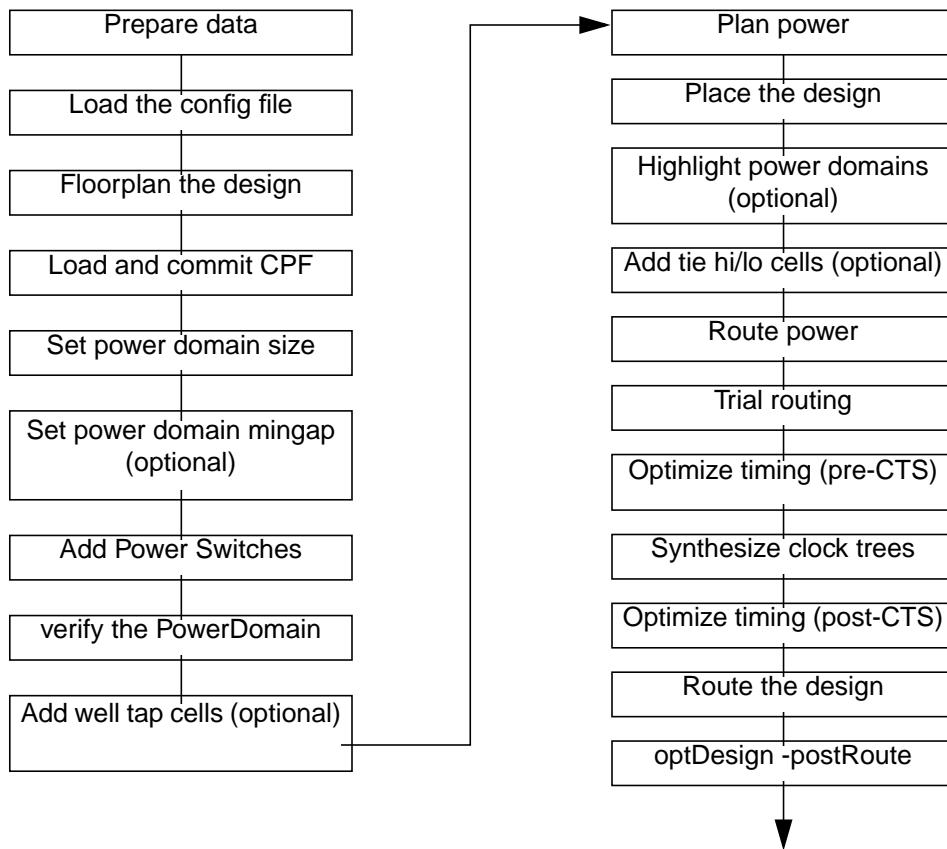
Documents the loadCPF/saveCPF GUI.

■ *[Encounter User Guide](#)*

- Provides a sample CPF 1.0 script in the “[CPF 1.0 Script Example](#)” chapter.
- Provides a sample CPF 1.0e script in the “[CPF 1.0e Script Example](#)” chapter
- Provides a sample CPF 1.1 script in the “[CPF 1.1 Script Example](#)” chapter
- Provides a list of supported CPF 1.0 commands in the “[Supported CPF 1.0 Commands](#)” chapter
- Provides a list of supported CPF 1.0e commands in the “[Supported CPF 1.0e Commands](#)” chapter.
- Provides a list of supported CPF 1.1 commands in the “[Supported CPF 1.1 Commands](#)” chapter.

## Multiple Supply Voltage Flat Flow

The MSV flat flow is based on CPF. The flow includes the following steps:



This section includes the following topics:

- [Preparing Data on page 461](#)
- [Loading the Configuration File on page 464](#)
- [Floorplanning the Design on page 464](#)
- [Loading and Committing the CPF File on page 465](#)
- [Setting the Power Domain Size on page 465](#)
- [Setting the Power Domain mingap on page 465](#)
- [Adding Power Switches on page 466](#)
- [Verify Power Domains on page 466](#)

## **Encounter User Guide**

### Low Power Design

---

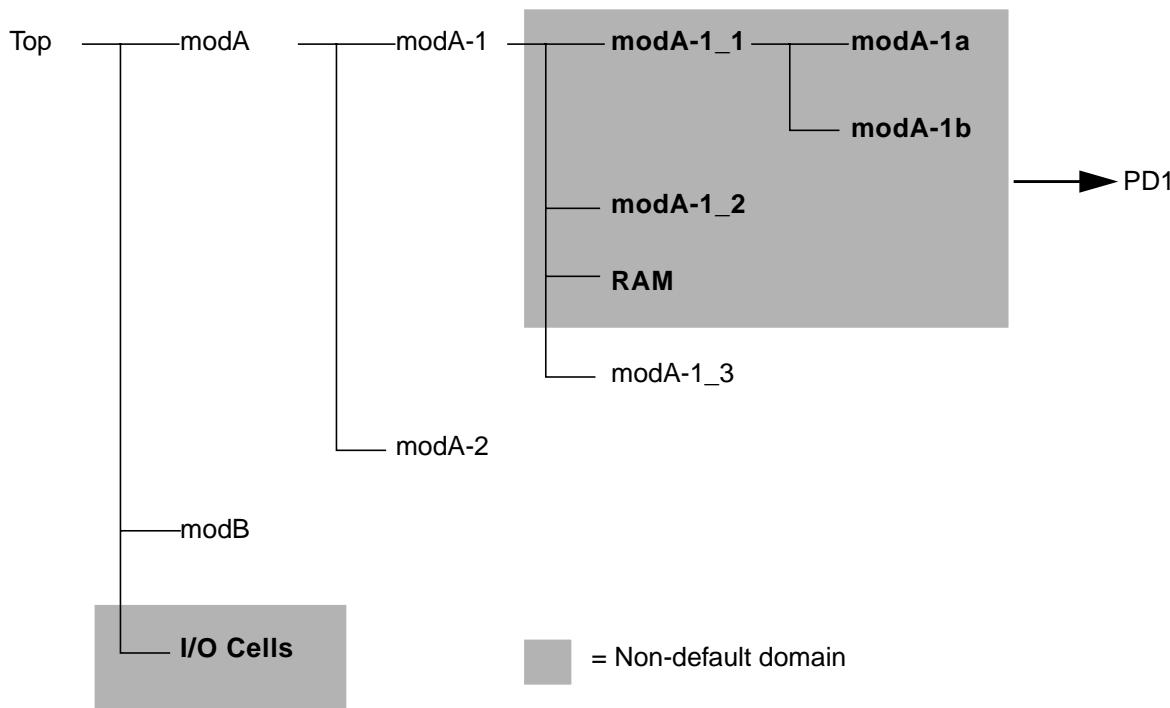
- [Adding Well Tap Cells](#) on page 466
- [Planning Power](#) on page 466
- [Placing Standard Cells and Macros](#) on page 467
- [Highlight Power Domains \(Optional\)](#) on page 469
- [Adding Tie High/Low cells](#) on page 470
- [Routing Power](#) on page 470
- [Trial Routing](#) on page 471
- [Optimizing Timing](#) on page 473
- [Synthesizing Clock Trees](#) on page 476
- [Optimizing Timing \(Post CTS\)](#) on page 477
- [Routing the Design](#) on page 477
- [Analyzing Timing](#) on page 477
- [Analyzing Power](#) on page 477
- [Optimizing Timing \(Post-Route\)](#) on page 478

## Preparing Data

You must prepare data as follows before you begin an MSV design:

- [Preparing the Netlist](#) on page 461
- [Preparing Libraries](#) on page 461
- [Defining Level Shifter Cells](#) on page 462

### Preparing the Netlist



- Power domains can contain one or more modules as members.
- A power domain can also be a partition, provided that the partition is the sole member of the domain.

## Preparing Libraries

In an MSV flow, the same cell may have instances in different regions operating at different voltages. For such a cell, characterize a different NLDM (or table look-up) for each operating voltage.

## Encounter User Guide

### Low Power Design

---

Your design must satisfy the following requirements:

- All library files loaded into the system to represent different voltage characterizations must have different library names.
- Any given library must have one and only one voltage associated with it.
- Min and max libraries may share the same library name.

Two or more power domains may have the same set of min/max libraries bound to them, as is usually the case in shutdown methodologies. You can then associate each library with a unique operating condition (and hence voltage) associated with it; for example:

Library File	Library Name	OpCondName
stdcell-1V.lib	Stdcell_1V	1V_min
		1V_max
stdcell-2V.lib	Stdcell_2V	2V_min
		2V_max
stdcell-3V.lib	Stdcell_3V	3V_min
		3V_max

If you specify min and max timing libraries when you import the design, the timing library information appears in the configuration file as follows:

```
set rda_Input(ui_timelib,max)  "stdcell_1V.lib stdcell_2V.lib stdcell_3V.lib"  
set rda_Input(ui_timelib,min)  "stdcell_1V.lib stdcell_2V.lib stdcell_3V.lib"
```

If you specify common timing libraries when you import the design, the timing library information appears in the configuration file as follows:

```
set rda_Input(ui_timelib)  "stdcell_1V.lib stdcell_2V.lib stdcell_3V.lib"
```

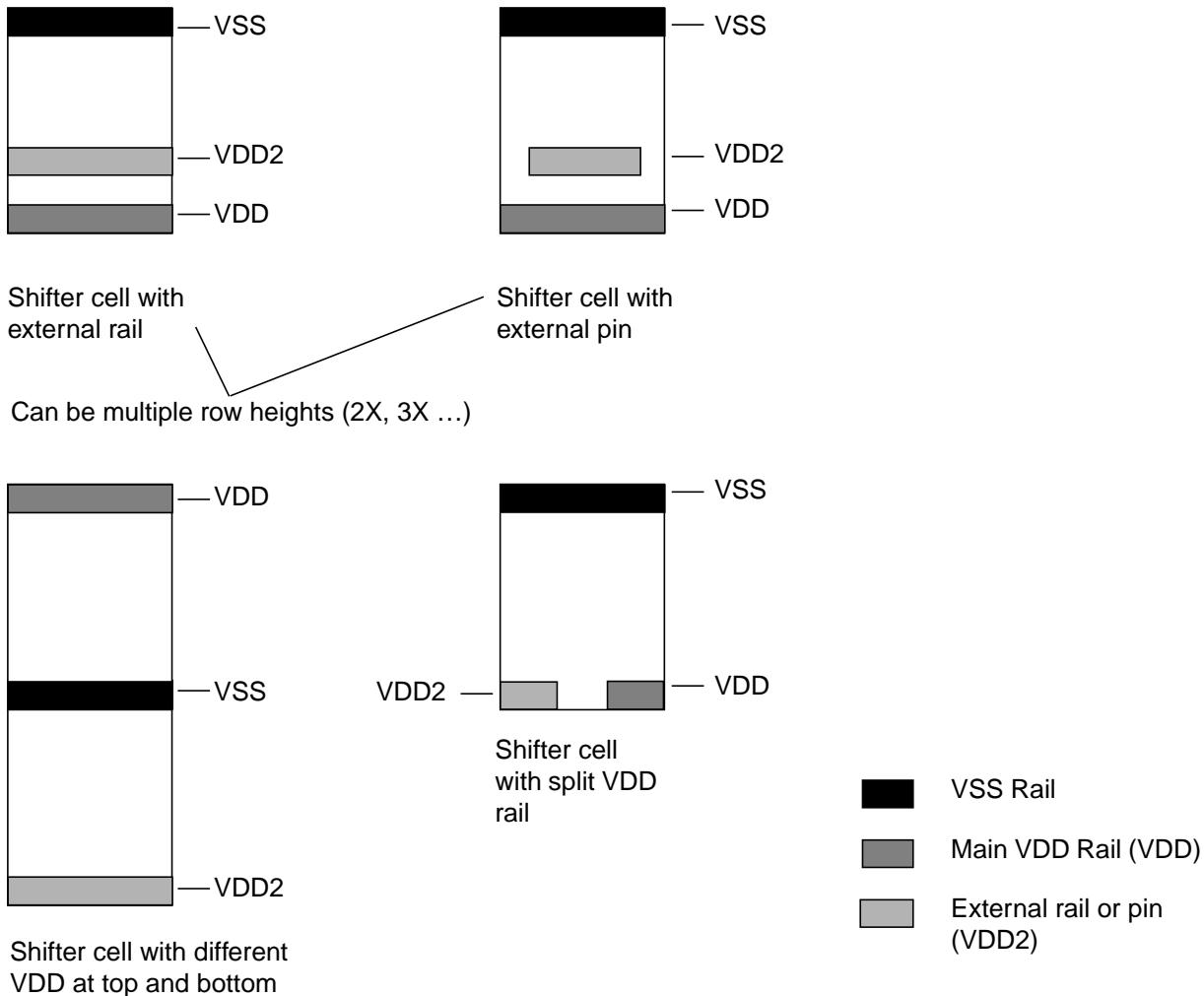
### Defining Level Shifter Cells

Level shifter cells are used to communicate between power domains of different voltages. The main characteristic of the level shifter cell is the presence of a secondary power rail or pin that operates at a different voltage than the primary power pins.

## Encounter User Guide

### Low Power Design

The Encounter software supports the following level shifter types:



The secondary power pin is modeled as a rail or pin. You must describe the secondary rail or pin as a power pin with shape ABUTMENT in LEF. The following example shows the LEF description of a pin named VDDL:

```
PIN VDDL
  DIRECTION INOUT ;
  USE power ;
  SHAPE ABUTMENT ;
  PORT ;
  LAYER METAL1 ;
  RECT 5.860 1.700 8.110 1.990 ;
```

**Note:** The shifter cell can be an integral multiple of the standard cell height.

## Loading the Configuration File

- To load the configuration file, use the following command:

```
loadConfig
```

## Floorplanning the Design

Create a floorplan for the design. For more information about creating floorplans, see “[Floorplanning the Design](#)” in the *Encounter User Guide*.

1. Place I/Os.
2. Use the Move/Shape/Resize icon to move control logic into the power domain.
3. Manually move the power domains into the core area.
4. (Optional) Change the shape of a power domain to a rectilinear shape by adding power domain cuts. The *Cut Rectilinear* icon allows you to interactively create power domain cuts that define rectilinear power domains. The power domain cuts mask out portions of the power domain to enable you to see the shape of the power domain. The equivalent text command is [createPowerDomainCut](#).
5. (Optional) Preplace hard macros.
6. Save the floorplan file (.fp).

Use the `saveFPlan` command or select *Design – Save – Floorplan* from the GUI.

The following example shows the kind of information saved:

- Power domain information: `design.fp`

```
POWERDOMAIN: NAME=TDSPCore
  VOLT=0.9680 LAYER=0 ALWAYSON=0
  NRLIB=4
  TIMELIB=tcbn45lpbwp_c060907bc0d88(common)
  TIMELIB=tcbn45lpbwp_c070208bc1d10d88(common)
  TIMELIB=tcbn45lpbwp_phvt_c070208bc0d88(common)
  TIMELIB=tcbn45lpbwp_c070208bc0d88(common)
  MINGAPTOP=16.0000 MINGAPBOT=16.0000 MINGAPLEFT=16.0000
  MINGAPRIGHT=16.0000
  RSEXTTOP=50.0000 RSEXTBOT=50.0000 RSEXTLEFT=50.0000 RSEXTRIGHT=50.0000
  ROWFLIP=3 SITE=core ROWSPACETYPE=2 ROWSPACING=0.0000
  MODULE=HDRDID1BWPVHT POWER=(VDD_TDSPCore:VDD)
  POWER=(VDD_TDSPCore_R:TVDD) GND=(VSS:VSS)
```

## Encounter User Guide

### Low Power Design

---

```
MODULE=LVLHLD2BWP POWER=(VDD_TDSPCore:VDD) GND=(VSS:VSS)
MODULE=PTLVLHLD2BWP POWER=(VDD_TDSPCore_R:TVDD) GND=(VSS:VSS)
MODULE=TDSP_CORE_INST POWER=(VDD_TDSPCore_R:TVDD)
POWER=(VDD_TDSPCore:VDD) GND=(VSS:VSS)
END_PD
POWERDOMAIN: NAME=PLL
...
END_PD
```

## Loading and Committing the CPF File

1. To read-in the CPF that contains CPF commands, use the following command:

```
loadCPF fileName
```

This command reads the file and performs lint, parsing, and semantics checking. This command does not execute any of the command within the CPF file.

2. To commit the CPF file, use the following command:

```
commitCPF
```

This command executes the CPF commands loaded by `loadCPF`. Running this command does the following:

- Creates power domains
- Creates logical power/ground net connections
- Specifies timing libraries for power domains
- Inserts shifter cells and isolation cells
- Replaces regular registers with state-retention (SRPG) registers

## Setting the Power Domain Size

- To do specify the power domain physical size, use the following command:

```
setObjFPlanBox
```

## Setting the Power Domain mingap

- (Optional) To set the power domain mingap, use the following command:

```
addPowerDomainAttr -rsExt
```

## Adding Power Switches

- To add power switches, use the following command:

```
addPowerSwitch
```

The tool supports two types of power switch insertion: ring or column style. For more information, see [“Power Shutdown Techniques”](#) on page 503.

## Verify Power Domains

1. Use the `verifyPowerDomain` command to verify that the following conditions were met:

```
verifyPowerDomain -gconn -place -xNetPD
```

- ❑ Power and ground pins of instances within a power domain are connected to the power and ground nets of that power domain.
- ❑ Only those cells assigned to a power domain are placed within the boundary of the power domain.
- ❑ The correct level shifters are inserted on all nets crossing power domains.

**Note:** The `verifyPowerDomain` command checks that libraries specified for each power domain of each `dc_corner` are complete for each power domain member list.

2. Correct all errors before continuing.

## Adding Well Tap Cells

- To add well tap cells, use the following command:

```
addWellTap
```

For more information, see [“Leakage Power Optimization Techniques”](#) on page 498

## Planning Power

Each domain must have complete power and ground rings. To create the power plan, use one of the following approaches:

- Use the automatic power planner (APP) to assign a template to each power domain. For more information, see [“Power Planning and Routing”](#) on page 435 in the *Encounter User Guide*.
- Use GUI or text commands

To create rings, complete the following steps:

1. Create core rings (no changes for MSV).
2. Create block rings for power domains and hard macros.

To use the GUI to create rings around a power domain, choose *Power – Power Planning – Add Rings*. Select the *Block ring(s) around: Selected domain/fences/reefs* option.

For more information, see “Power Menu” in the *Encounter Menu Reference*.

The equivalent text command is `addRing -type {block_rings -around power_domain}`.

3. Create block rings for hard macros (no changes for MSV).

To create stripes for a selected power domain or for all power domains, complete the following steps:

1. Create stripes in either of the following ways:

- Select a power domain, then use the *Each selected block/power domain/fence* option on the Basic tab of the Add Stripes form to create power stripes for a selected power domain.

The equivalent text command is `addStripe` with the `-over_power_domain` parameter set to 1.

- Use the *All domains* option on the Basic tab of the Add Stripes form to create power stripes for all power domains.

The equivalent text command is `addStripe` with the `-all_domains` parameter set to 1.

**Note:** To create a mesh, you must use the `addStripe` command twice: once for vertical stripes and once for horizontal stripes.

**Note:** If the power and ground nets for the stripes do not match the power domain power and ground nets, a warning message displays.

2. Create stripes to connect power domain rings to core rings.

## Placing Standard Cells and Macros

1. Place standard cells that have not already been placed.

```
setPlaceMode -timingDriven  
placeDesign
```

## Encounter User Guide

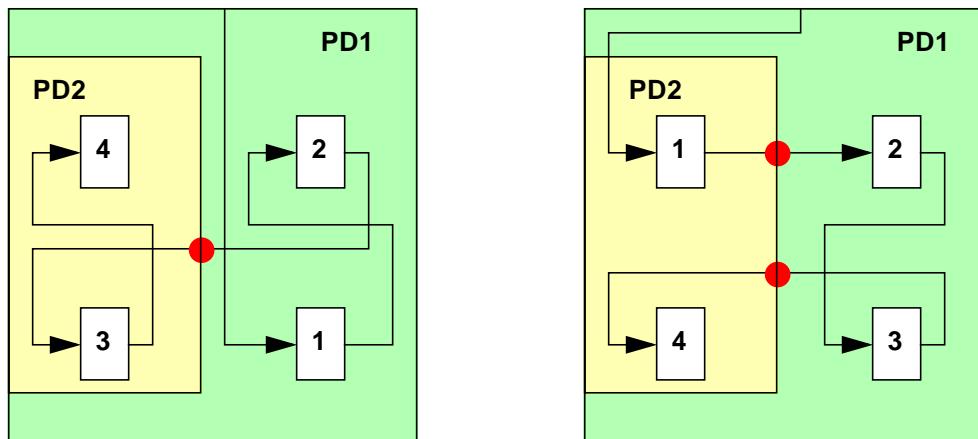
### Low Power Design

This software recognizes power domains automatically and ensures that the fences are respected.

In scan reordering, to maintain the hierarchical ports so that shifters and isolation cells will not be changed, use the following command:

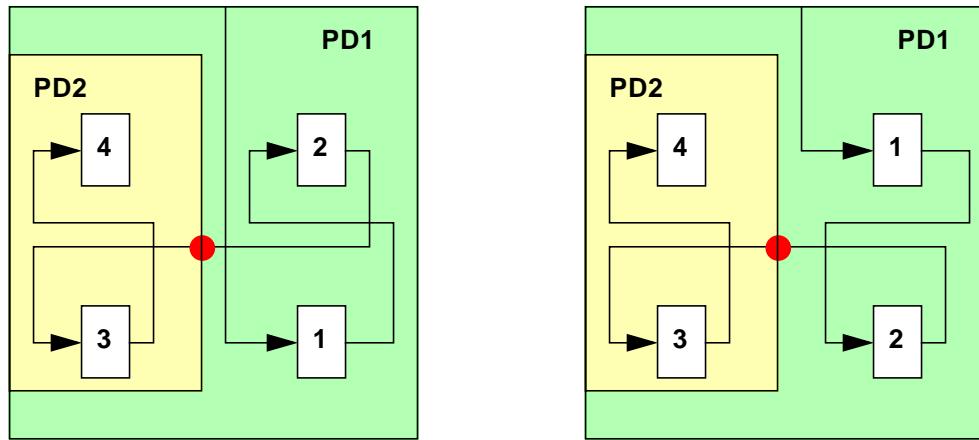
```
setScanReorderMode -keepHierPort
```

The following figure shows scan reordering without -keepHierPort:



As shown in this figure, the software adds two hierarchical ports.

The following figure shows scan reordering with -keepHierPort:



A single hierarchical port is maintained.

The software places all standard cells and hard macros within the boundary of the power domain to which they are assigned. In addition, voltage level shifters are placed around the power domain boundary.

Voltage level shifters are placed along the inside edge of the power domain boundary, on the row ends. For top and bottom rows, multiple shifter cells are placed at row ends, if needed.

## 2. Highlight shifters (Optional)

To highlight shifters, use the following command:

```
reportShifter -highlight
```

As an alternative, from the main menu, you can select *Power – Multiple Supply Voltage – Highlight Power Domains* to highlight the shifters.

**Note:** Shifter information is not saved in the database.

## 3. Verify power domains again (Optional)

To verify power domains (optional) and ensure that the design has no violations, use the following command:

```
verifyPowerDomain -xNetPD fileName -isoNetPD fileName
```

## Highlight Power Domains (Optional)

After the design is placed, you can highlight the power domain in the following ways:

- Using the Floorplan View

Select a power domain from the Floorplan view. When you switch to the Physical view, all power domain members are highlighted, except for I/O pads.

- Using the Design Browser

The software displays power domains as groups in the Design Browser. Select the Group object to view all power domains listed in the PDGroups category. Select one of the power domains and click on the *Highlight* button. All power domain members (including the I/O pads) that were assigned to that domain are highlighted in the Physical view.

- Using the Text Command

Use the `hilitePowerDomain` command to highlight all power domain members, including I/O pads, assigned to the power domain; for example:

```
hilitePowerDomain PD2
```

- Using the Highlight Power Domains GUIs:

[Power – Multiple Supply Voltage – Highlight Power Domains](#)

This form has three tabs:

❑ Power Domain

Highlights power domains, P/G nets, level shifters, and/or power switches in a power domains. You can highlight non-default power domains only.

❑ Signal Net/HLS Cell

Enables you to select categories of signal nets or Hvt, Lvt, or Svt cells.

❑ Highlight Set

Enables you to customize the highlight colors.

## Adding Tie High/Low cells

- Add tie high/low cells; for example, for PD1:

```
addTieHiLo -cell TIEONE_PD1 -tiHiPin Z -powerDomain PD1
```

## Routing Power

The Encounter software uses the `sroute` command to route the followpin connections and the additional power pins contained in level shifters, just as it routes regular followpins.

- To use the power router, use the following command:

sroute

The `sroute` command does the following:

- ❑ Connects power/ground nets from end to end, terminating at the power rings.
- ❑ Connects voltage level shifters pins to the closest segment of the ring surrounding the power domain.

You can also use the `sroute` command to route level shifter pins as well as primary nets within a domain. For example:

```
sroute -powerDomains { TDSP } -secondaryPinNet { vdd } -allowJogging 1  
-allowLayerChange 1 -nets { vdd_lp vss }  
sroute -powerDomains { DEFAULT } -secondaryPinNet { vdd_lp } -allowJogging 1  
-allowLayerChange 1 -nets { vss vdd }  
clearDrc
```

- To use the *Sroute* GUI, complete the following step.

Select *Route – Special Route* to display the Sroute – Basic form.

- ❑ If you select the *Standard cell pins* option on the *Basic* tab of the SRoute form, SRoute connects power and ground nets from end-to-end, terminating at the power rings.
- ❑ Standard cell pins automatically recognize voltage level shifters and connect the standard cells to the correct power nets.
- ❑ If you select the *Level shifter pins* option, sroute connects voltage level shifter pins to the closest segment of the ring around the power domain.

For more information, see Sroute – Basic in the “Route Menu” chapter of the *Encounter Menu Reference*.

- To route always-on pins in SRPG cells, specify the following command before you run sroute:

setPGPinUseSignalRoute

For example, NanoRoute connects cell RSDF to pin TVDD, and cell PTBUFF to pin TVDD:

```
setPGPinUseSignalRoute RSDF:TVDD PTBUFF:TVDD
routePGPinUseSignalRoute -nets VDD_TDSPCore_R
```

## Trial Routing

By default, the trialRoute command is not MSV-aware, so you must use the following options to setTrialRouteMode:

- `-handleEachPD true`
- `-handlePDComplex true`

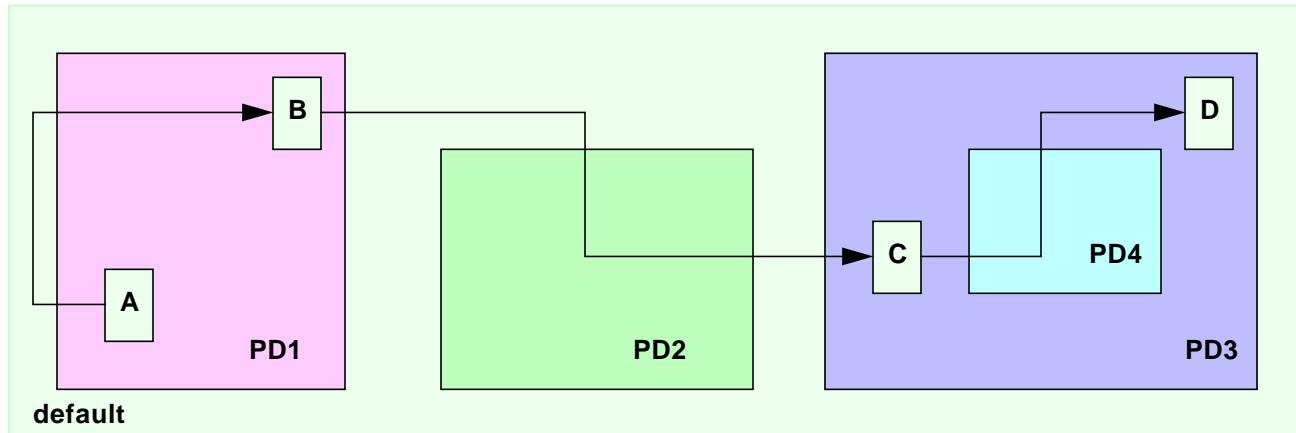
If you have nested power domains, use the following `setTrialRoute` parameter:

- `-handlePDComplex true`

## Encounter User Guide

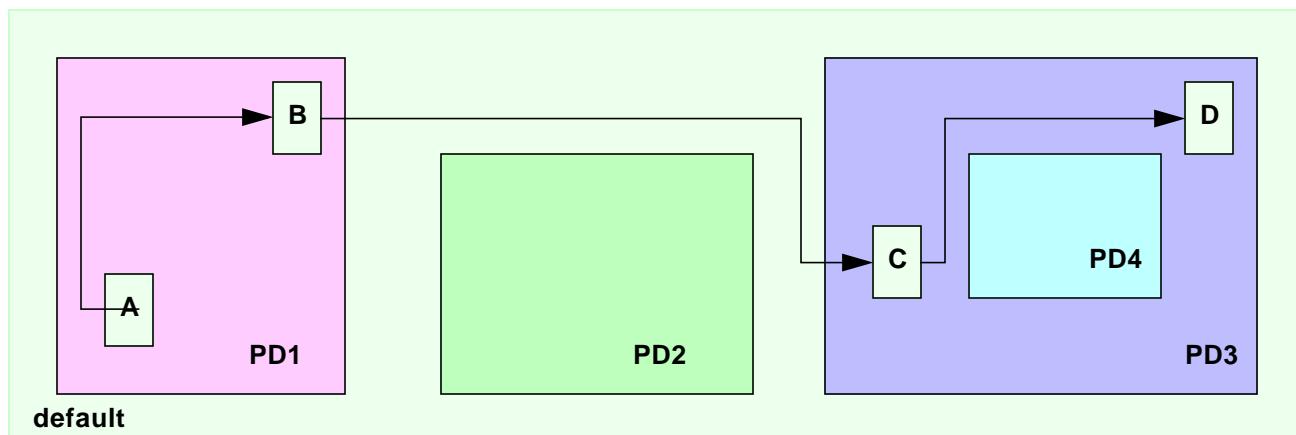
### Low Power Design

The following figure shows `trialRoute` behavior without MSV-aware settings:



- Net from A to B is routed outside of PD1
- Net from B to C is routed through PD2
- Net from C to D routed through PD4

The following figure shows trial routing when you specify `setTrialRouteMode -handlePDComplex true` and `-handleEachPD true`:



- Net from A to B is routed only inside PD1
- Net from B to C is routed only through the Default power domain
- Net from C to D is routed only inside PD3

## Optimizing Timing

- To optimize timing at this phase of the design, use the following command:

```
optDesign -preCTS
```

**Note:** In pre-CTS mode, the tool automatically synthesizes the buffer tree, but you must first specify always-on buffers in the CPF file.

Instance cells that belong to different power domains are bound to different timing libraries. Instances are bound to TLF or .lib libraries when you create power domains. Each power domain has a set of associated timing libraries. During timing optimization, new cells (added or changed) are associated only with a timing library that is part of the power domain.

You do not need to set any special options to optimize timing for an MSV design.

Depending on the stage at which you want to optimize timing, use one of the following commands:

- optDesign -preCTS
- optDesign -postCTS
- optDesign -postRoute

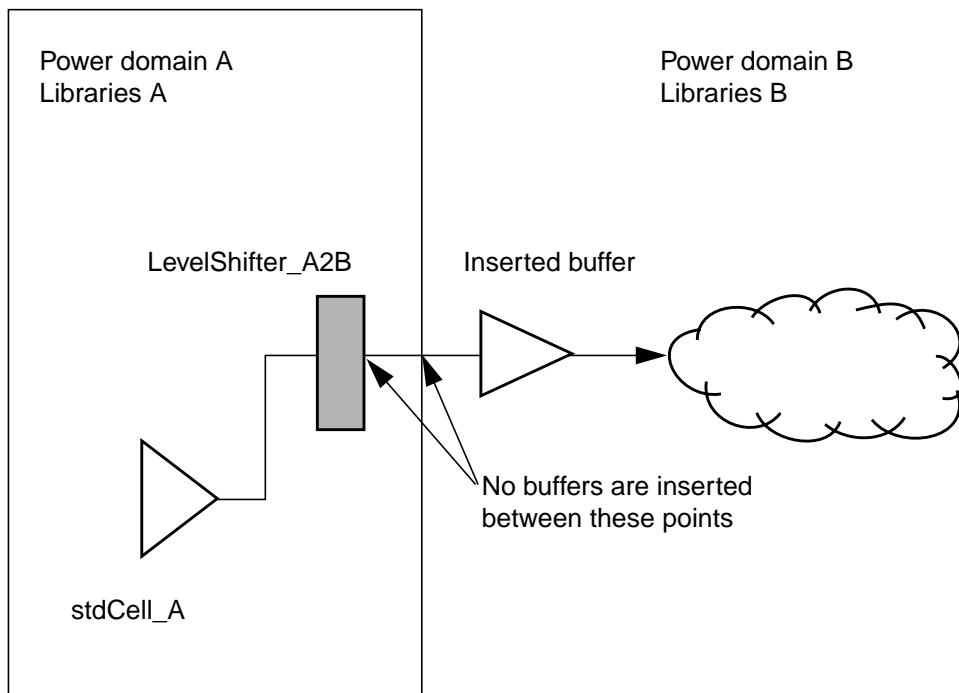
Inserted cells are assigned to the appropriate power domain, and power and ground pins are connected to the correct power and ground nets. The software places the inserted cells within the appropriate domain boundary.

**Note:** The same scenarios apply to isolation cells.

## Encounter User Guide

### Low Power Design

The following scenario shows how the software optimizes timing across a net that crosses power domains of different voltages.

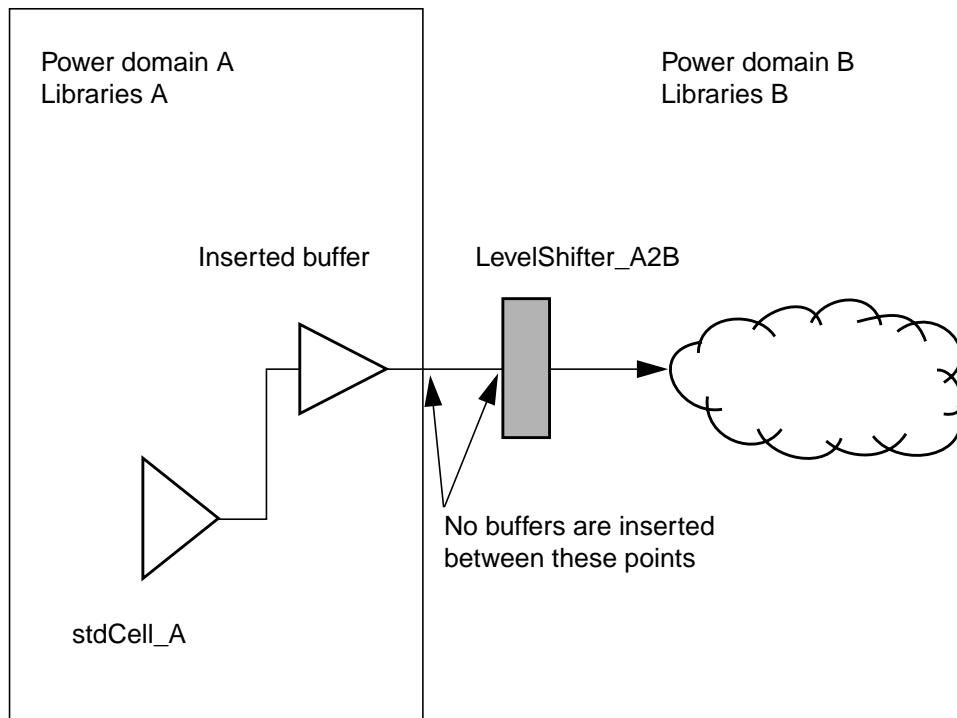


- Logically, the buffer instance name is in B HInst, and the master cell is from library B.
- Physically, the buffer is placed as close as possible to the level shifter on the opposite domain location determined by the need to fix timing.

## Encounter User Guide

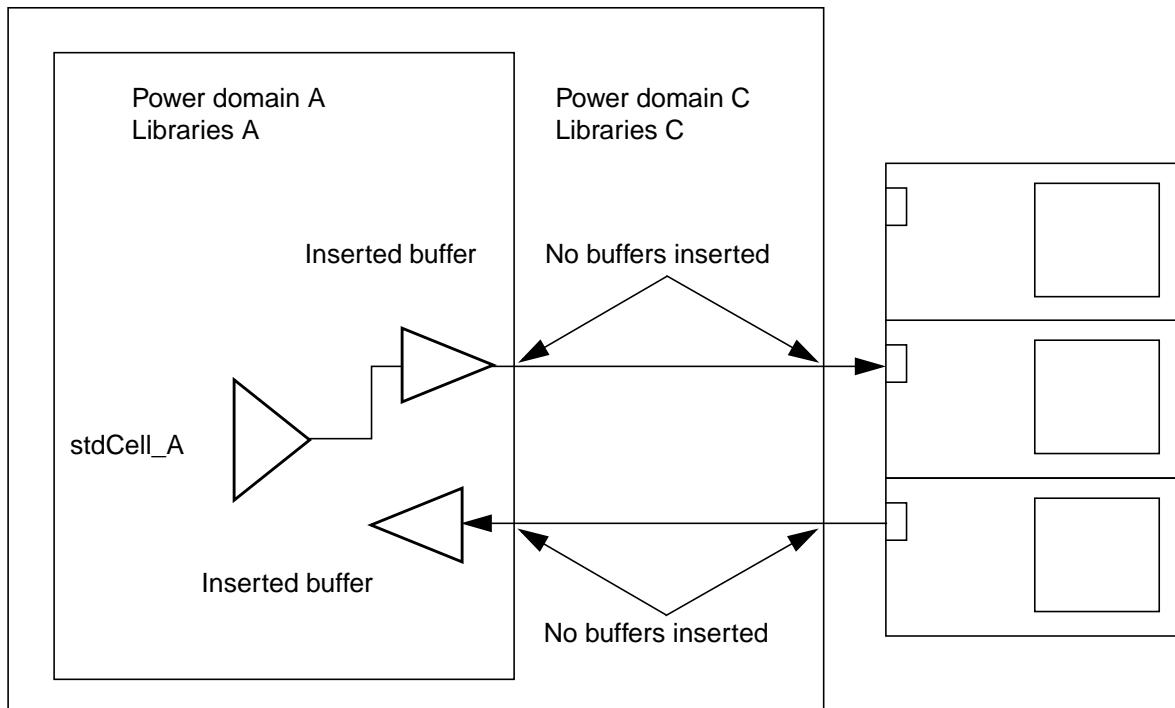
### Low Power Design

The following scenario shows how the software optimizes timing inside a power domain that does not contain a level shifter.



- Logically, the buffer instance name is in A HInst, and the master cell is from library A.
- Physically, the buffer is placed as close as possible to the level shifter on the opposite domain location determined by the need to fix timing.

The following scenario shows timing optimization behavior for nets connecting for I/Os.



- Logically, the buffer instance names are in a HInst, and the master cells are from library A.
- Physically, buffers are placed within power domain A only.
- The same applies to isolation cells.

## Synthesizing Clock Trees

1. Before running `clockDesign`, use the following command to make `clockDesign` power-aware:  
`setCTSMode -powerAware true`
2. To synthesize clock trees, run `clockDesign`.

For all clocks within a domain, CTS selects buffers from the domain's libraries and places buffers within the domain's boundary.

For all clocks crossing power domains, CTS assumes that level shifters are present. CTS does the following:

- Establishes a single entry/exit point for each domain
- Selects buffers from appropriate libraries
- Places the buffers within domain boundaries
- Performs cloning and decloning

## Optimizing Timing (Post CTS)

- To optimize timing in post-CTS mode, use the following command:

```
optDesign -postCTS
```

## Routing the Design

1. Before running NanoRoute, use the following command to make nanoRoute MSV-aware:  
`setNanoRouteMode -routeHonorPowerDomain true`
2. To route the design, run `globalDetailRoute`.

## Analyzing Timing

Each power domain has a set of associated timing libraries. Instance cells that belong to different power domains are bound to corresponding timing libraries, which allows you to perform timing analysis with no further changes for MSV.

- To analyze timing, run `timeDesign` as you would ordinarily.

## Analyzing Power

Power analysis software recognizes power domains and calculates power values accordingly.

To perform power analysis for a design with MSV, choose *Power – Rail Analysis* and use the Edit Pad Location form to create a power pad location file for each net. For more information, see [Edit Pad Location](#) in the “Power Menu” chapter of the *Encounter Menu Reference*.

The power analysis software can use an instance power file to calculate instance IR drop instead of a global net voltage.

You can specify that power analysis reports be organized by net. If you specify a report name, the report for all nets is placed in that one file. If you do not specify a report name, a separate

file contains each report. For example, if the net names are VDD1 and VDD2, the report names are VDD1.report and VDD2.report.

For more information about how power analysis supports MSV, see “Running Power Analysis with Encounter,” in the Power Analysis chapter.

## **Optimizing Timing (Post-Route)**

- After routing, use the following command to optimize timing:

```
optDesign -postRoute
```

## Multiple Supply Voltage Top-Down Hierarchical Flow

This section discusses the following topics:

- [Overview](#) on page 479
- [Always-On Feedthrough Handling](#) on page 480
- [Chip Partitioning](#) on page 482
- [Block-level CPF Generation](#) on page 482
- [Top-Level CPF Generation](#) on page 484
- [Block-Level Implementation](#) on page 485
- [Top-Level Implementation](#) on page 485
- [Chip Assembly](#) on page 485

### Overview

The Encounter tool supports the low power top-down CPF-based hierarchical flow built on the regular Encounter hierarchical flow. The difference is that, in the CPF-based hierarchical flow, you partition the chip-level CPF file into block-level CPF files and a top-level hierarchical CPF file. You then use those CPF files to implement the block level and top level designs.

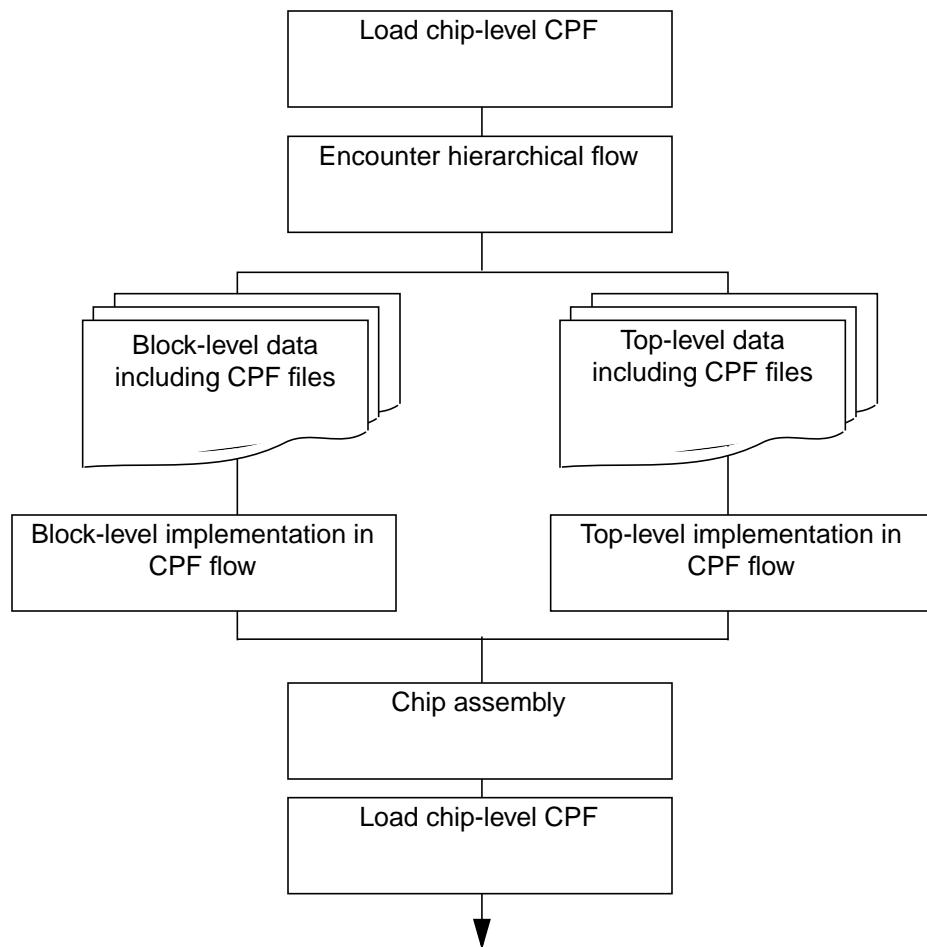
The CPF-based hierarchical flow supports the following scenarios:

- The partition is physically the same as the power domain. The hierarchical instance is logically the same for power domain and partition.
- The power domain is physically inside partition. The power domain hierarchical instance is logically a sub hierarchical instance of partition.

## Encounter User Guide

### Low Power Design

- Partition is physically inside power domain. The partition hierarchical instance is logically one (and only one) of the hierarchical instances of power domain.



## Always-On Feedthrough Handling

In the hierarchical flow, when you commit CPF, the tool replaces assign statements with always-on buffers for the net connecting to always-on logic or a net whose driver is always-on. The tool identifies these nets automatically as follows:

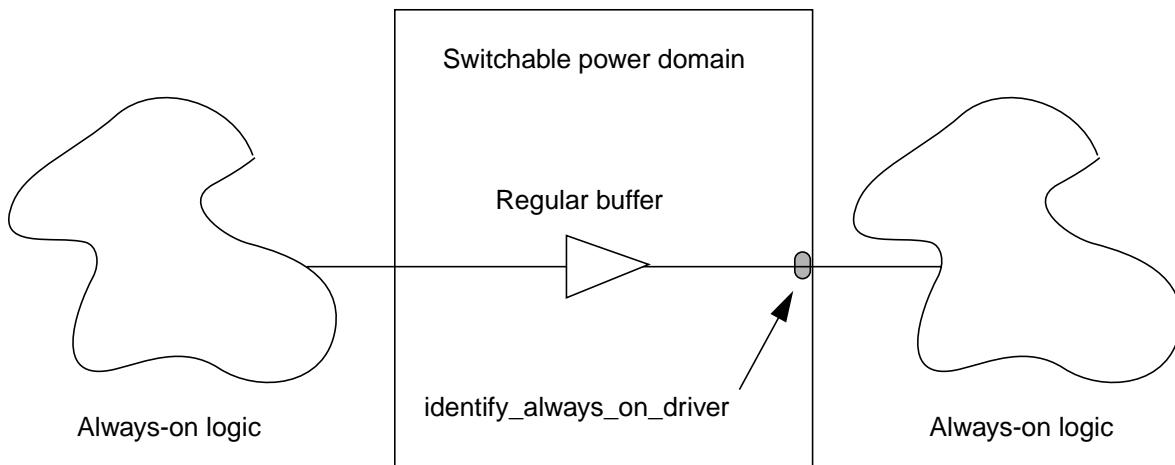
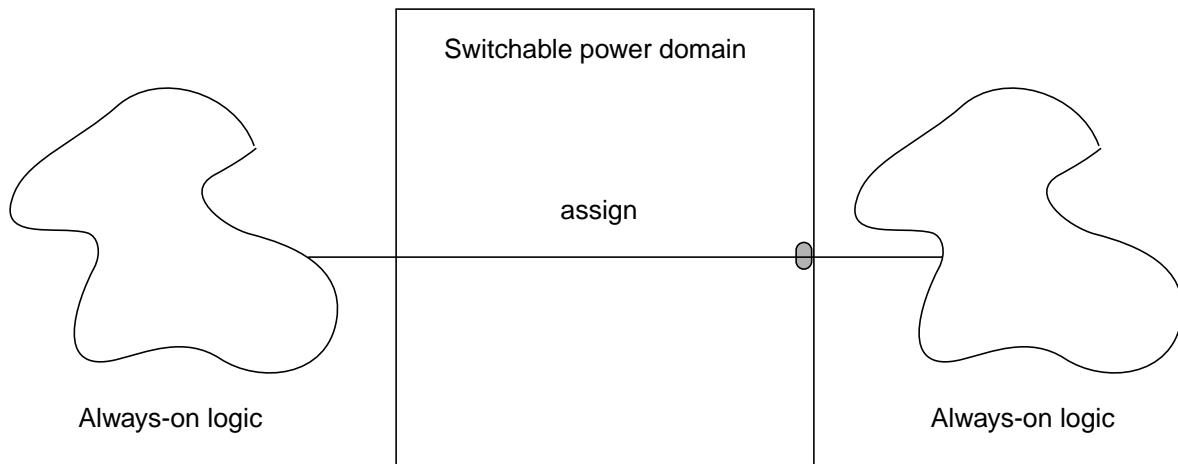
- The driver of the feedthrough and the feedthrough path are always on
- The output of the feedthrough is defined by `identify_always_on_driver` in the chip-level CPF file

The tool can also insert a feedthrough buffer in a partition that resides in a shut-off power domain.

## Encounter User Guide

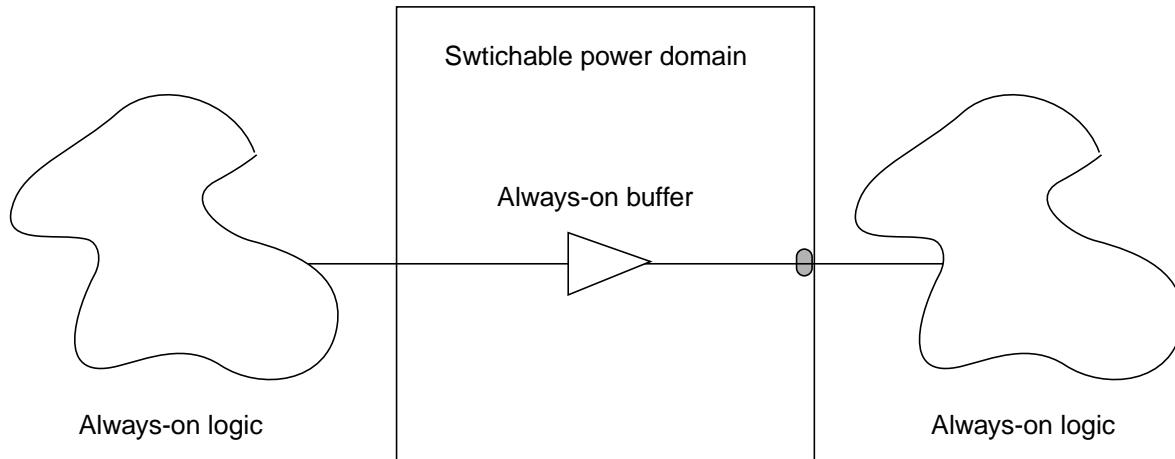
### Low Power Design

If a feedthrough already exists in an input netlist (for example, a netlist for some reused blocks), the feedthrough may use the assign statements and regular buffers. The tool identifies the always-on feedthroughs and replaces the regular buffers or assign statements with always-on buffers defined in the chip-level CPF when you commit the CPF file.



If you use `insertPtnFeedthrough` to insert a feedthrough in the hierarchical flow, use the `-buffer` parameter to add an always-on buffer on the feedthrough. The tool can trace

through always-on buffers and propagate the always-on attribute as shown in the following figure:



## Chip Partitioning

Low power hierarchical partitioning with CPF is a combination of the Multi-Mode Multi-Corner (MMMC) and CPF flows. Do the following:

1. Derive the MMMC timing budget by `deriveTimingBudget` and `saveTimingBudget`. For more information, see the “[Timing Budgeting](#)” chapter of the *Encounter User Guide*.
2. Use `savePartition` to generates CPF files for each block-level design (partition), and a hierarchical CPF file for top-level design.
3. (Optional) If the power domain is inside the partition, save the floorplan file for chip assembly.

## Block-level CPF Generation

Block-level CPF is used to implement block-level design and determine the power domain attribute for the partition boundary pin at top-level implementation. When you commit CPF, the tool generates block-level CPF from the chip-level CPF file as follows:

- Low power information in CPF

## Encounter User Guide

### Low Power Design

---

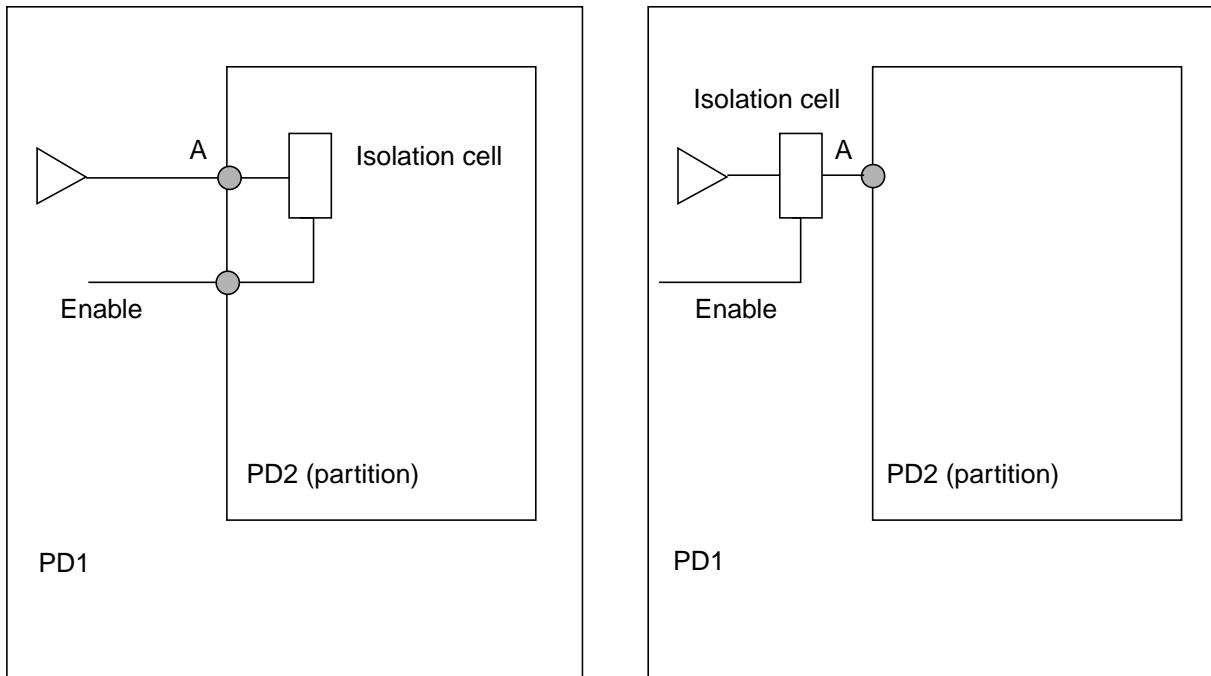
- ❑ Pushes down naming style, hierarchy separator, CPF version, and library set definitions
  - ❑ Pushes down low power cell definitions
  - ❑ Creates the power domains referenced by the block-level CPF files
  - ❑ Creates the power domains' power/ground nets and connections
  - ❑ Pushes down the scope-related rules or commands such as state retention rules, power switch rules, and `identify_always_on_driver` rules
  - ❑ For level shifter and isolation rules:
    - If the rules specify the shifter and isolation are added into a block, the tool pushes down those rules into block-level CPF
    - If the rules specify the shifter and isolation are added outside a block, the tool does not push down those rules into block-level CPF.
  - ❑ Assigns the power domain attribute to each partition boundary pin
  - ❑ Pushes down all the nominal conditions and power modes
  - ❑ Creates virtual ports to control low power logic by using `set_design -ports`
- Note:** Virtual ports do not exist in the netlist, but are needed to enable power switch, isolation, state-retention logic, and so on, in the block level

#### ■ MMMC information in CPF

The analysis views, operating conditions and power domain library binding are written into the `viewdefinition.tcl` file by timing budget commands as part of MMMC setup. Block-level CPF does not include this information.

 *Important*

The tool marks the partition boundary pin power domain attribute that will determine whether there is a domain-crossing for the net connecting to the pin. The following example shows how the tool marks the power domain attribute for the partition boundary pin and determines whether to push down the isolation rule:



1. Port A is marked as PD1 in the block-level CPF
2. The isolation rule is pushed down

1. Port A is marked as PD2 in the block-level CPF
2. The isolation rule is kept at the top level

For always-on feedthroughs, the tool automatically traces through the feedthrough and assigns both the input and output pins of the feedthrough as always on.

For the net connecting to the partition boundary pin without an isolation or level shifter cell, the tool assigns the pin to the power domain of its driver domain.

## Top-Level CPF Generation

The tool generates top-level CPF from the chip-level CPF file hierarchically, as follows:

- Sources the block-level CPF to find block-level boundary power domain information
- Maps all block power domains to top-level global power domains through the `set_instance -domain_mapping` command
- Maps virtual port in block CPF to the top-level signal by `set_instance -port_mapping`
- Creates the same power domains and their global connections as those at chip-level

The tool retains other CPF information at top-level CPF as follows:

- Retains isolation or level shifter rules when the isolation or level shifter is inserted at top level.
- Discards rules in block-level scope such as state retention and power switch rules
- Retains library sets, cell definitions, nominal condition, power mode and MMMC views at the chip level

## **Block-Level Implementation**

1. Implement the block-level design with an MMMC flow, using the block-level CPF file generated at the partitioning step and the `viewdefinition.tcl` file created by `deriveTimingBudget`.
2. After completing block-level implementation, use `saveDesign -def` to save the block-level design in `def` format for chip assembly.

## **Top-Level Implementation**

1. Implement the top-level design with an MMMC flow, using the hierarchical top-level CPF file generated at the partition step and the `viewdefinition.tcl` file created by `deriveTimingBudget`.
2. After completing top-level implementation, use `saveDesign -def` to save the top-level design in `def` format for chip assembly.

## **Chip Assembly**

Chip assembly assembles the physical data you generated at the block level and block level.

1. If there is a power domain inside partition, specify the chip-level floorplan with `assembleDesign -chipFP`.

## **Encounter User Guide**

### Low Power Design

---

- 2.** Load the chip-level CPF after the assembly to restore the low power settings and continue to chip-level verification and chip finishing.

## Example of Block-Level CPF Generated by Encounter

**Note:** A special construct is used to avoid duplicate definition for timing-related CPF files when sourced by top-level CPF. If the `[set_instance]==[set_hierarchy_separator]` condition is True, then the tool recognizes the implementation is at the block level, and loads the related timing information. If the condition is False, then the tools sources the block-level CPF file at top level, and does not load the timing-related information.

```
set_design tdsp_core \
-ports {n_41}
set_hierarchy_separator "/"
create_power_domain -name TDSPCore -default \
-shutoff_condition {n_41}
create_power_nets -nets VDD_TDSPCore \
-voltage 0.792 \
-internal
update_power_domain -name TDSPCore \
-internal_power_net {VDD_TDSPCore}
create_power_nets -nets VDD_TDSPCore_R \
-voltage 0.792
create_global_connection -net VDD_TDSPCore_R \
-domain TDSPCore \
-pins TVDD
create_global_connection -net VDD_TDSPCore \
-domain TDSPCore \
-pins VDD
create_ground_nets -nets VSS
create_global_connection -net VSS \
-domain TDSPCore \
-pins VSS
create_power_domain -name AO
-boundary_ports { clk reset SRPG_PG_in SRPG_PG_in_1 DFT_sen n_41,...}
create_power_nets -nets VDD -voltage 0.792
update_power_domain -name AO -internal_power_net {VDD}
if {[set_instance]==[set_hierarchy_separator]} {
define_library_set -name ao_wc_0v99
-libraries { ../../LIBS/N45/timing/wc.lib}
define_library_set -name ao_wc_0v792 \
-libraries { ../../LIBS/N45/timing/AOwc0d72.lib}
define_library_set -name tdsp_wc_0v792 \
-libraries { ../../LIBS/N45/timing/wc0d72.lib}
create_operating_corner -name WC08COM_AO \
-voltage 0.792 \
-process 1 \
-temperature 125 \
-library_set ao_wc_0v792
create_operating_corner -name WC08COM_TDSP \
-voltage 0.792 \
```

## Encounter User Guide

### Low Power Design

---

```
-process 1 \
-temperature 125 \
-library_set tdsp_wc_0v792
create_nominal_condition -name low_ao -voltage 0.792
update_nominal_condition -name low_ao -library_set ao_wc_0v792
create_nominal_condition -name off -voltage 0
create_power_mode -name PM_LO_FUNC \
-domain_conditions { AO@low_ao TDSPCore@off } \
-default
update_power_mode -name PM_LO_FUNC \
-sdc_files {../../RELEASE/mmmc/dtmf_reccvr_core_dull.sdc}
create_analysis_view -name AV_LO_FUNC_MAX_RC1 \
-mode PM_LO_FUNC \
-domain_corners { AO@WC08COM_AO TDSPCore@WC08COM_TDSP }
}
define_isolation_cell -cells {LVLLH} \
-ground {VSS} \
-enable {NSLEEP} \
-valid_location {to} \
-power {VDD}
define_level_shifter_cell -cells {LVLH} -valid_location {to} \
-output_power_pin {VDD} \
-input_voltage_range {0.792:0.99:0.099} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} -direction {down}
define_level_shifter_cell -cells {PTLVLH} \
-valid_location {to} \
-direction {down} \
-output_power_pin {TVDD} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} -input_voltage_range {0.792:0.99:0.099}
define_level_shifter_cell -cells {LVLLH} \
-valid_location {to} \
-input_power_pin {VDDL} \
-output_power_pin {VDD} \
-input_voltage_range {0.792:0.99:0.099} \
-output_voltage_range {0.792:0.99:0.099} \
-output_voltage_input_pin {NSLEEP} \
-ground {VSS} \
-direction {up}
define_level_shifter_cell -cells {LVLLHD} \
-input_voltage_range {0.792:0.99:0.099} \
-valid_location {to} \
-direction {up} \
-output_power_pin {VDD} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} \
-input_power_pin {VDDL}
define_state_retention_cell -cells {RSDF} \
-ground {VSS} \
-save_function {SAVE} -power {TVDD} \
-restore_function {!NRESTORE} \
-clock_pin {CP} \
-power_switchable {VDD}
```

## Encounter User Guide

### Low Power Design

---

```
define_power_switch_cell -cells {HDRDID HDRDIA}
-stage_2_enable {!NSLEEPIN2} \
-stage_1_output {NSLEEPOUT1} \
-power {TVDD} \
-stage_2_output {NSLEEPOUT2} \
-power_switchable {VDD} \
-stage_1_enable {!NSLEEPIN1} \
-type {header}

define_always_on_cell -cells {PTBUFF PTLVLH} \
-ground {VSS} \
-power {TVDD} \
-power_switchable {VDD}

create_level_shifter_rule -name LSRULE_H2L \
-to {TDSPCore} \
-from {AO} \
-exclude {n_41 SRPG_PG_in SRPG_PG_in_1}

update_level_shifter_rules -names LSRULE_H2L \
-cells {LVLH} \
-location {to}

create_level_shifter_rule -name LSRULE_H2L_AO
-from {AO} \
-to {TDSPCore} \
-pins {n_41 SRPG_PG_in SRPG_PG_in_1}

update_level_shifter_rules -names LSRULE_H2L_AO
-location {to} \
-cells {PTLVLH}

create_state_retention_rule -name SRPG_TDSP \
-save_edge {SRPG_PG_in} \
-domain {TDSPCore} \
-restore_edge {!SRPG_PG_in_1}

update_state_retention_rules -names SRPG_TDSP \
-cell {RSDF} \
-library_set {tdsp_wc_0v792}

create_power_switch_rule -name TDSPCore_SW \
-domain {TDSPCore} \
-external_power_net {VDD_TDSPCore_R}

update_power_switch_rule -name TDSPCore_SW \
-prefix {CDN_SW_} \
-cells {HDRDID} \
-acknowledge_receiver {switch_en_out}

end_design
```

## Example of Top-Level CPF Generated by Encounter

The top-level CPF file is a hierarchical. Since the block-level design is a blackbox during top-level implementation, the block-level CPF sourced by top-level CPF is only used to assign the blackbox boundary pin power domain.

```
set_cpf_version 1.0
set_design dtmf_recv_core
set_hierarchy_separator "/"
create_ground_nets -nets Avss
create_ground_nets -nets VSS
create_power_nets -nets VDD \
-voltage 0.792
create_power_nets -nets Avdd \
-voltage 0.990
create_power_nets -nets VDD_TDSPCore_R \
-voltage 0.792
create_power_nets -nets VDD_TDSPCore \
-voltage 0.792 \
-internal
define_library_set -name ao_wc_0v99 \
-libraries { ../LIBS/N45/timing/tcbn45lpbwp_c060907wc.lib}
define_library_set -name ao_wc_0v792 \
-libraries { ../LIBS/N45/timing/tcbn45lpbwp_c060907wc0d72.lib}
define_library_set -name tdsp_wc_0v792 \
-libraries { ../LIBS/N45/timing/tcbn45lpbwp_c060907wc0d72.lib}
source cpf_1.0_hierarchical_PD.tcl
set_instance TDSP_CORE_INST \
-port_mapping { {n_41 PM_INST/power_switch_enable} } \
-domain_mapping { {TDSPCore TDSPCore} {AO AO} }
source TDSP_CORE_INST.cpf
create_power_domain -name TDSPCore \
-shutoff_condition {PM_INST/power_switch_enable}
create_global_connection -net VDD_TDSPCore_R \
-domain TDSPCore \
-pins TVDD
create_global_connection -net VDD_TDSPCore \
-domain TDSPCore \
-pins VDD
create_global_connection -net VSS \
-domain TDSPCore \
-pins VSS
create_power_domain -name AO \
-default
update_power_domain -name AO \
-internal_power_net {VDD}
```

## Encounter User Guide

### Low Power Design

---

```
create_global_connection -net VDD_TDSPCore \
-domain AO \
-pins VDDL
create_global_connection -net VDD \
-domain AO \
-pins VDD
create_global_connection -net VSS \
-domain AO \
-pins VSS
create_power_domain -name PLL \
-instances { PLLCLK_INST}
update_power_domain -name PLL \
-internal_power_net {Avdd}
create_global_connection -net VDD \
-domain PLL \
-pins VDDL
create_global_connection -net Avdd \
-domain PLL \
-pins avdd!
create_global_connection -net Avdd \
-domain PLL \
-pins VDD
create_global_connection -net Avss \
-domain PLL \
-pins VSS
create_global_connection -net Avss \
-domain PLL \
-pins agnd!
create_operating_corner -name WC08COM_AO \
-voltage 0.792 \
-process 1 \
-temperature 125 \
-library_set ao_wc_0v792
create_operating_corner -name WC08COM_TDSP \
-voltage 0.792 \
-process 1 \
-temperature 125 \
-library_set tdsp_wc_0v792
create_nominal_condition -name low_ao \
-voltage 0.792
update_nominal_condition -name low_ao \
-library_set ao_wc_0v792
create_nominal_condition -name off -voltage 0
create_power_mode -name PM_LO_FUNC \
-domain_conditions { AO@low_ao PLL@high_ao TDSPCore@off}
update_power_mode -name PM_LO_FUNC \
-sdc_files {../RELEASE/mmmc/dtmf_recv_core_dull.sdc
create_analysis_view -name AV_LO_FUNC_MAX_RC1 \
-mode PM_LO_FUNC \
-domain_corners { AO@WC08COM_AO PLL@WCCOM_AO TDSPCore@WC08COM_TDSP}
define_isolation_cell -cells {LVLLH} \
-ground {VSS} \
```

## Encounter User Guide

### Low Power Design

---

```
-enable {NSLEEP} \
-valid_location {to} \
-power {VDD}

define_level_shifter_cell -cells {LVLH} \
-valid_location {to} \
-output_power_pin {VDD} \
-input_voltage_range {0.792:0.99:0.099} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} \
-direction {down}

define_level_shifter_cell -cells {PTLVLH} \
-valid_location {to} \
-direction {down} \
-output_power_pin {TVDD} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} \
-input_voltage_range {0.792:0.99:0.099}

define_level_shifter_cell -cells {LVLLH} \
-valid_location {to} \
-input_power_pin {VDDL} \
-output_power_pin {VDD} \
-input_voltage_range {0.792:0.99:0.099} \
-output_voltage_range {0.792:0.99:0.099} \
-output_voltage_input_pin {NSLEEP} \
-ground {VSS} -direction {up}

define_level_shifter_cell -cells {LVLLHD} \
-input_voltage_range {0.792:0.99:0.099} \
-valid_location {to} \
-direction {up} \
-output_power_pin {VDD} \
-output_voltage_range {0.792:0.99:0.099} \
-ground {VSS} \
-input_power_pin {VDDL}

define_state_retention_cell -cells {RSDF} \
-ground {VSS} \
-save_function {SAVE} \
-power {TVDD} \
-restore_function {!NRESTORE} \
-clock_pin {CP} -power_switchable {

define_power_switch_cell \
-cells {HDRDID HDRDIAO} \
-stage_2_enable {!NSLEEPIN2} \
-stage_1_output {NSLEEPOUT1} \
-power {TVDD} \
-stage_2_output {NSLEEPOUT2} \
-power_switchable {VDD} \
-stage_1_enable {!NSLEEPIN1} \
-type {header}

define_always_on_cell -cells {PTBUFF PTLVLH} \
-ground {VSS} \
-power {TVDD} \
-power_switchable {VDD}

create_isolation_rule -name ISORULE \
-from {TDSPCore} \
-isolation_condition {!PM_INST/isolation_enable} \
-isolation_output {high}
```

## **Encounter User Guide**

### Low Power Design

---

```
update_isolation_rules -names ISORULE \
-location {to} \
-cells {LVLLH}

create_level_shifter_rule \
-name LSRULE_H2L_PLL \
-from {PLL} \
-to {AO}

update_level_shifter_rules -names LSRULE_H2L_PLL \
-location {to} \
-cells {LVLHLD}

end_design
```

## Multiple Supply Voltage Bottom-Up Hierarchical Flow

This section discusses the following topics:

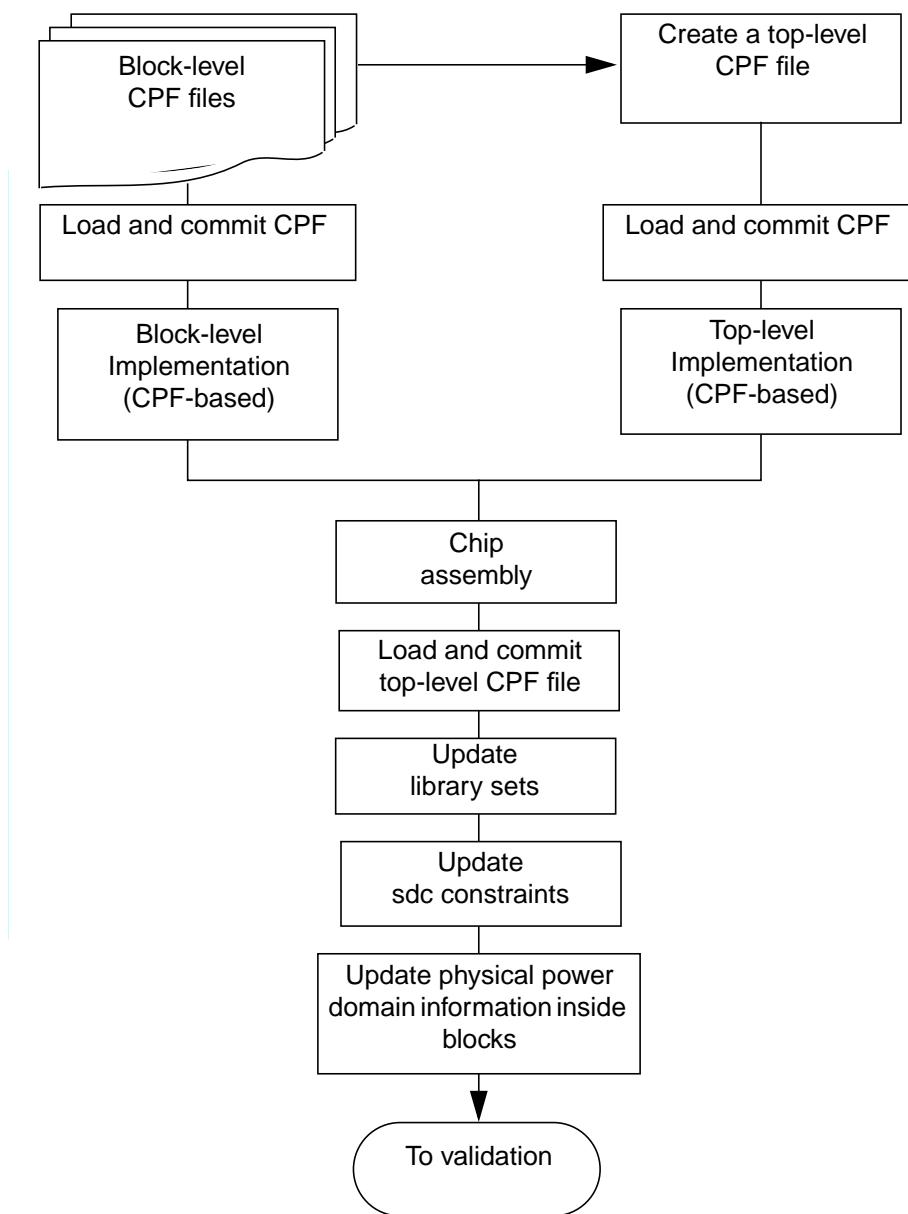
- [Block-Level Implementation](#) on page 495
- [Top-Level Implementation](#) on page 496
- [Chip Assembly](#) on page 496

The Encounter tool supports the low power bottom-up CPF-based hierarchical flow built on the regular Encounter bottom-up hierarchical flow. The difference is that you use the existing

## Encounter User Guide

### Low Power Design

block-level CPF files to construct the top-level hierarchical CPF file, and implement the design using the CPF flow.



## Block-Level Implementation

You can use any combination of hard and soft blocks.

For the hard blocks (that are already implemented),

- Place the blocks in top-level floorplan.

For the soft blocks,

1. Load and commit the block-level CPF files.
2. Implement the blocks using the block-level CPF implementation flow.

After completing block-level implementation,

1. Save the block-level design in def format for chip assembly.

```
saveDesign -def
```

2. If a power domain exists inside a block, use the following command to obtain the physical information about the power domain.

```
saveCPF tmp.cpf
```

The tool restores the physical information for the power domain inside block (partition) after chip assembly.

## Top-Level Implementation

Before top-level implementation,

- Manually build the top-level CPF file by reusing the block-level CPF files as follows:

```
Set_instance HinstOfBlock -domain_mapping { {..} } -port_mapping { {..} }  
Source block.cpf
```

The CPF file you create contains the same type of information as in the previous example file: [“Example of Top-Level CPF Generated by Encounter”](#) on page 490.

To implement the design,

- Use the CPF implementation flow using the hierarchical top-level CPF file.

After completing top-level implementation,

- Use the following command to save the top-level design in DEF format:

```
saveDesign -def
```

## Chip Assembly

- To assemble the design's physical data, use the following command:

```
assembleDesign
```

After chip assembly,

1. To restore the lower power setup for chip-level verification and chip finishing, load and commit the top-level hierarchical CPF file.
2. (Optional) Update power domain physical information inside block.

- a. To update power domain shape, use the following command:

```
setObjFPlanBox
```

- b. To update the power domain attribute, use the following command:

```
modifyPowerDomainAttr
```

**Note:** These steps are necessary only if you have a power domain inside a partition.

3. To update the library set, use the following command:

```
update_library_set -name -timing {..}
```

4. To apply chip-level timing constraints (sdc files), use the following command:

```
update_constraint_mode -name -sdc_files
```

5. Proceed to design verification.

## Leakage Power Optimization Techniques

- [Multi-Vth Optimization](#) on page 498
- [Substrate Biasing](#) on page 499

### Multi-Vth Optimization

You can optimize non-critical path logic for leakage, while preserving the critical timing slack (WNS).

**Note:** Run multi-Vth optimization only after your design meets timing.

1. Report the total leakage power in the design.

```
reportPower -leakage
```

If you want to obtain a report file, run `reportPower -leakage` with the `-outfile fileName` option.

The following example is a leakage report showing the total leakage power in microwatts, along with cell usage statistics. For each library, the number of cells used in the design and the total leakage power dissipated by the cells are listed.

```
Total leakage power = 785.079708uW
Cell usage statistics:
Library normalVt, 49265 cells (64.855650%), 733.007529uW (93.367269%)
Library highVt, 26696 cells (35.144350%), 52.072179uW (6.632725%)
```

2. Optimize leakage power.

```
optLeakagePower
```

This command resizes low voltage threshold gates in the design to gates with a higher voltage threshold, while maintaining timing. This command only resizes cells that have positive slack. Cells that belong to any library are candidates for swapping. The `-highEffort` parameter overrides effort levels set by `setOptMode`.

**Note:** The `optLeakagePower` is a standalone command that can be used when a netlist is already optimized in timing and on which you want to reclaim as much leakage power as possible. It is recommended that you use `optLeakagePower` without any options.

3. After running the `optLeakagePower` command, you can create a new leakage power report to view results.

```
reportPower -leakage -outfile fileName
```

## **Optimizing Leakage Power While Running optDesign (Recommended)**

You can optimize non-critical path logic for leakage power by using the `setOptMode` command in the following ways:

- If leakage power optimization is a second priority after timing convergence, then use `setOptMode -leakagePowerEffort low` after `placeDesign`. The leakage power optimization is automatically done by `optDesign -postRoute`.
- If leakage power optimization is as critical as timing convergence, then use `setOptMode -leakagePowerEffort high` right after `placeDesign`. The leakage power optimization is done by each `optDesign` step.

**Note:** When `setOptMode -leakagePowerEffort` is set to `low` or `high`, the hold fixing steps ensure to not insert any low-voltage threshold gates. Only high-voltage threshold gates are used to fix the hold-time violations.

## **Substrate Biasing**

Substrate biasing is another technique for reducing leakage power. Changing the body voltage of the field effect transistor (FET) affects both the threshold voltage and the static leakage current.

To bias the substrate, insert biasing cells into a region of the design. In Encounter, you can do this in either of two ways:

- Use the `addWellTap` command to add bias cells at regular intervals.

```
addWellTap -maxGap
```

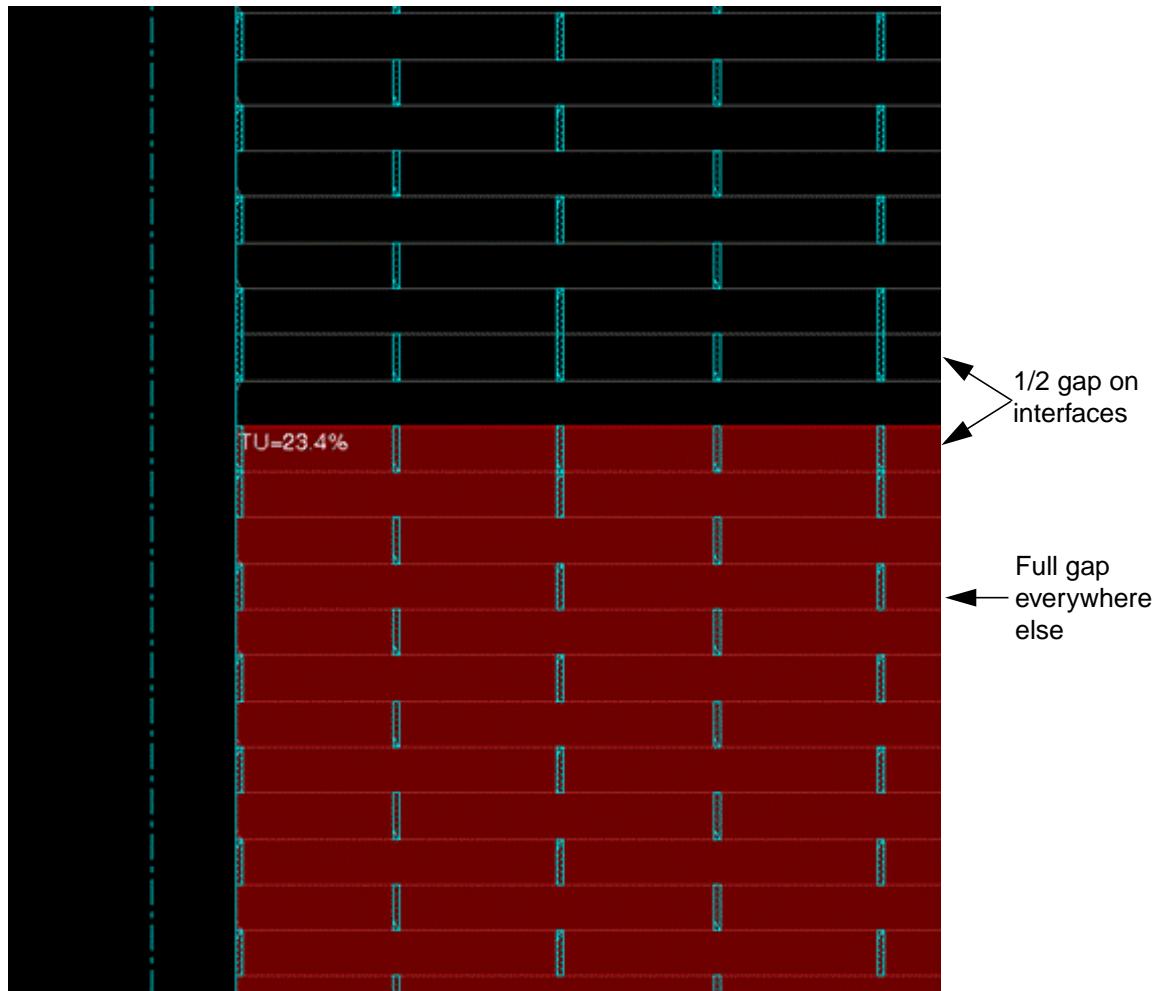
- Add well taps in a checkerboard configuration; for example,

```
addWellTap -cellFILL1 -maxGap 20 -checkerBoard -fixedGap  
addWellTap -cellFILL2 -maxGap 20 -powerDomain PD -checkerboard -fixedGap
```

## Encounter User Guide

### Low Power Design

These commands produce results such as those shown in the following figure:



For well tap cells, you must add stripes to connect the secondary power/ground pins in the vertical or horizontal direction.

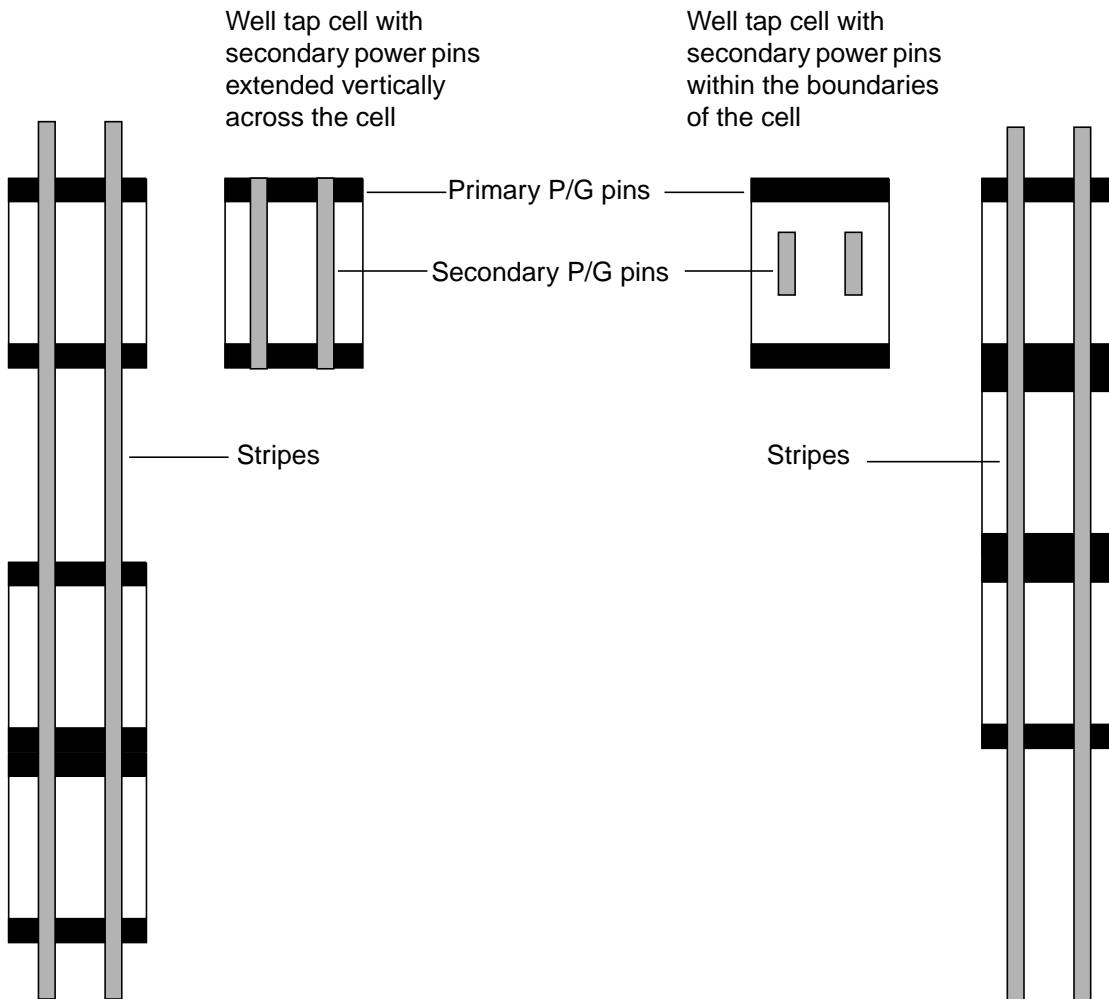
1. Select *Floorplan – Custom Power Planning – Add Stripes*.
2. On the Basic page, select *Over P/G pins*.
3. Click on *Master Name*.
4. Type the master name of the standard cell.
5. Select *Pin Layer*.

The top layer is the default.

## Encounter User Guide

### Low Power Design

The following figures show how well tap cells are connected. The software connects the secondary power/ground pins vertically or horizontally to the nearest secondary power/ground pins, regardless of whether the pins extend fully across the cell.

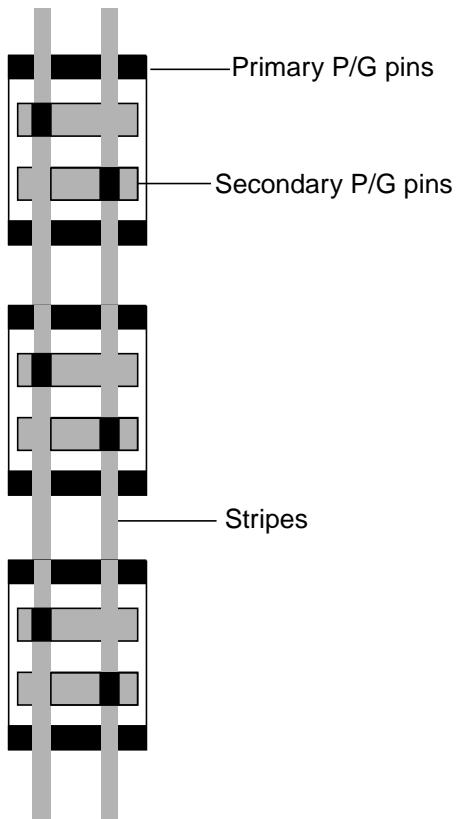


## Encounter User Guide

### Low Power Design

---

Well tap cells with secondary power pins extended horizontally across the cell



## Power Shutdown Techniques

Power shutdown is a coarse-grain methodology for performing power gating. This technique shuts off a specific power domain under certain conditions. There are two styles of this methodology:

- Ring style: All switches are inserted outside the domain.
- Column style: All switches are inserted inside the power domain.

## Power Shutdown Commands

The Encounter software includes two ways to insert power switches:

- Text command:

[addPowerSwitch](#)

- Menu commands:

- For power switch ring forms, select the following from the Encounter main menu:

[Power – Multiple Supply Voltage – Power Switch Insertion – Ring](#)

Forms:

- [Power Switch Insertion - Ring – Sides](#)
    - [Power Switch Insertion – Ring – Switch](#)
    - [Power Switch Insertion – Ring – Filler](#)
    - [Power Switch Insertion – Ring – Buffer](#)
    - [Power Switch Insertion – Ring – Breaker](#)
    - [Power Switch Insertion – Ring – Corner Cells](#)
    - [Power Switch Insertion – Ring – Enable Connection](#)
    - [Power Switch Insertion – Ring – Switch Cell Count](#)
    - [Power Switch Insertion – Ring – Cell Offset](#)
    - [Power Switch Insertion – Ring – Cell Orientation](#)

- For power switch column forms, select the following from the Encounter main menu:

[Power – Multiple Supply Voltage – Power Switch Insertion – Column](#)

Forms:

- [Power Switch Insertion – Column – Switch Cell and Enable Connection](#)
- [Power Switch Insertion – Column – Switch Arrangement](#)

## Data Preparation

For ring-style switch insertion, prepare the data as follows:

- Assign CLASS RING to the power switch cell in the LEF file. No SITE information is required.
- Ensure that there are enable nets to drive the buffer inside of the power switch cell, and acknowledge nets to exit the power switch cell. These nets are used as input to the -enableNetIn and -enableNetOut options to addPowerSwitch.
- Specify the power/ground net and pin connects of the power switch cell.

For column-style switch insertion, prepare the data as follows:

- Assign CLASS CORE and the correct SITE definition for the switch cell in the LEF file.
- Specify the power/ground net and pin connections of the power switch cell.
- Specify the distance between the columns and switches in microns (horizontal pitch value).

For ring style, you need to know the following:

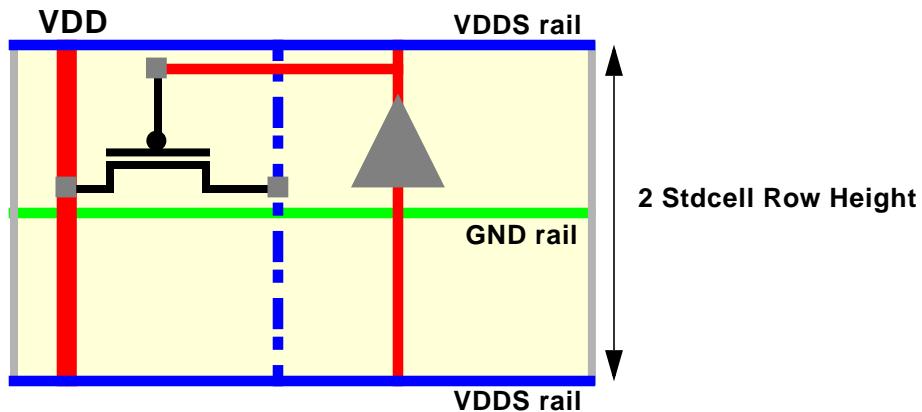
- For power planning, to ensure that the power stripes connect to the power switch cell
- NanoRoute connects enable signals. Abutment depends on the physical layout of the power switch cell.

For column style, you need to know the following:

- In the addPowerSwitch command, the -enableNetOut option can only specify one net name, which will be the net base name. The tool adds the suffix `_columnNumber`.
- The dimension of the power switch cells must be an integer multiple of a single-height standard cell.

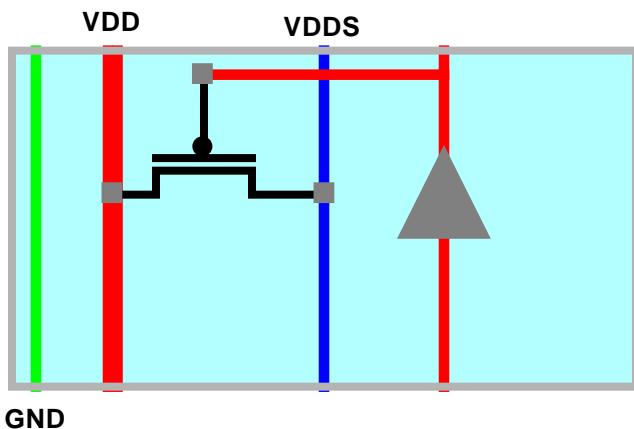
## Buffer Styles

The following figure shows column switch cell, which contains a buffer. This cell has a height of two times the standard cell height.



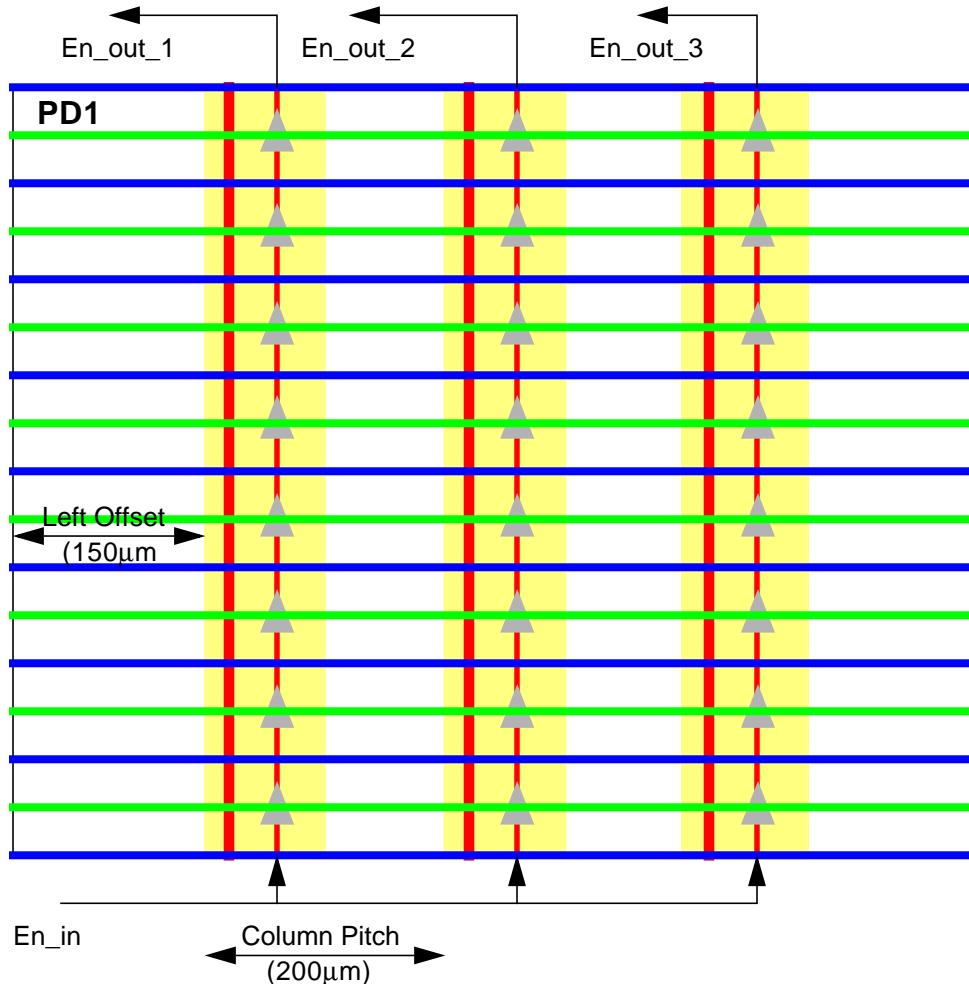
The following figure shows a ring switch cell, which contains a buffer. The cell has the same height as a standard cell. Ring switch cells can also contain two buffers with different directions.

**Ring Switch Cell**



## Adding Column Switches

The column switch methodology adds power switches entirely within the power domain. The following figure shows an example of column switches within a power domain:



To add column switches, use the following command:

```
addPowerSwitch -column
```

The following parameters are required:

- powerDomain
- enablePinIn
- enablePinOut
- enableNetIn
- enableNetOut
- globalSwitchCellName

Optionally, you can specify the following:

## Encounter User Guide

### Low Power Design

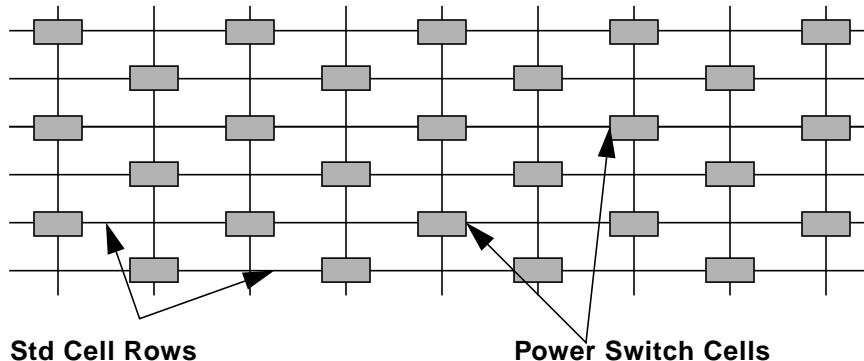
- Offsets from the top, bottom, right, and/or left side of the power domain.
- Area in which the tool can place switches.
- Power/ground pin connections.
- Many other options.

Instead of using the text command, you could use the menu command as follows:

- From the main Encounter window, choose *Power – Multiple Supply Voltage – Power Switch Insertion*, then click on the *Column* button.

With the text command, you can place the switches in a checkerboard pattern as follows:

```
addPowerSwitch -column -checkerboard
```



This command allows you to reserve space for other uses or reduce the number of switches, for example, and possibly reduce leakage power.

## Attaching the Acknowledge Receiver Pin

In CPF, you can specify an input pin that must be connected to an output pin of the last power switch in the chain. This information is specified in CPF as follows:

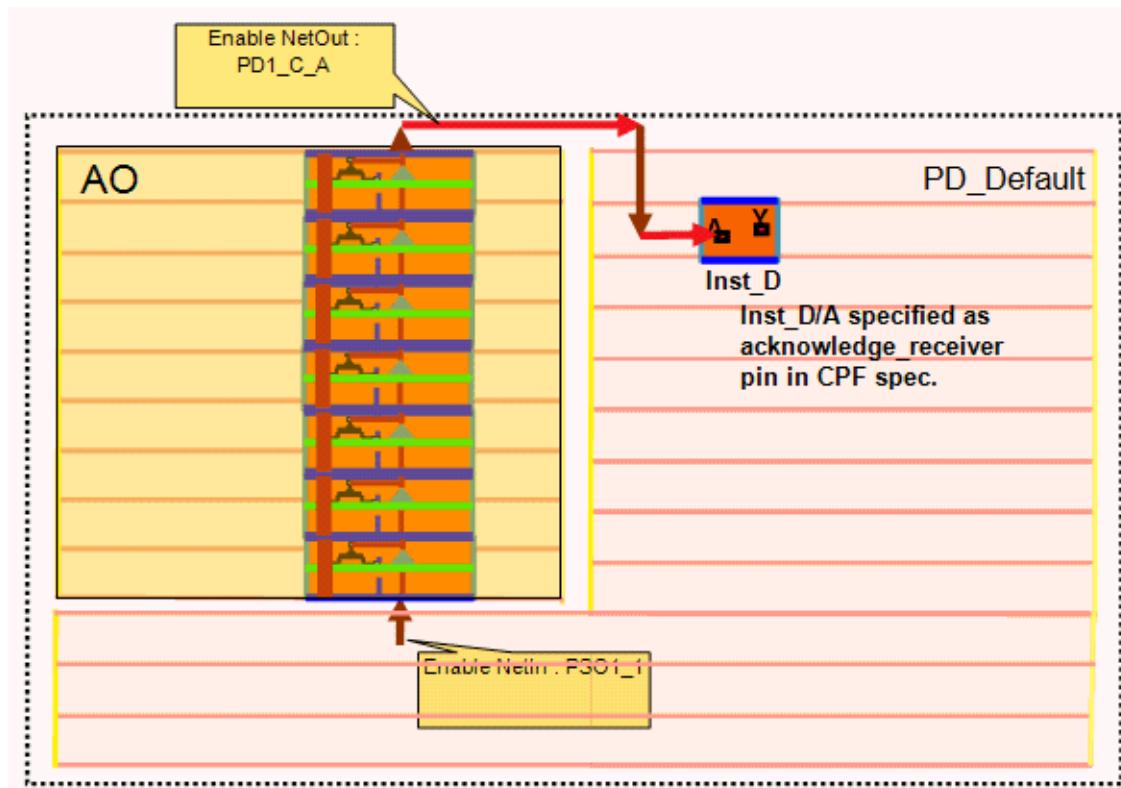
```
update_power_switch_rule -name string  
-acknowledge_receiver pin ...
```

The `addPowerSwitch` command can connect the output pin of the last switch cell to the acknowledge pin specified by `update_power_switch_rule`.

## Encounter User Guide

### Low Power Design

The following figure shows enableNetOut PD1\_C\_A connected to Inst\_D as specified in CPF:



## Example

In the CPF file:

```
create_power_switch_rule -name sw1 -domain PD1 -external_power_net VDDH  
update_power_switch_rule -name sw1 -cells COLUMN_SW -acknowledge_receiver Inst_D/A
```

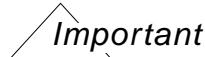
Power switch insertion command:

```
addPowerSwitch -column -powerDomain PD1 -enablePinIn SWIN -enablePinOut SWOUT  
-enableNetIn PS01_1 -globalSwitchCellName COLUMN_SW -leftOffset 3 -horizontalPitch  
100
```

Verilog file after addPowerSwitch is run:

```
BUFXH Inst_D (.Y(switch_out), .A(PD1_C_A));  
...  
COLUMN_SW ps01_PD1_1_COLUMN_SW_9_52_3 (.SWOUT(PD1_C_A),  
.SWIN(psoPSI_PD1_EnNet_1_3_9_49_0));
```

Inst\_D/A is connected to the PD1\_C\_A net and is connected to the SWOUT pin of the last switch.



Do not specify -enableNetOut because this setting interferes with and overrides the -acknowledge\_receiver specification.

## Enable Chaining

By default, the enableNetIn is connected to the bottom of each column and the enableNetOut exits from the top of each column, in parallel. The following commands let you create columns with daisy-chain enables:

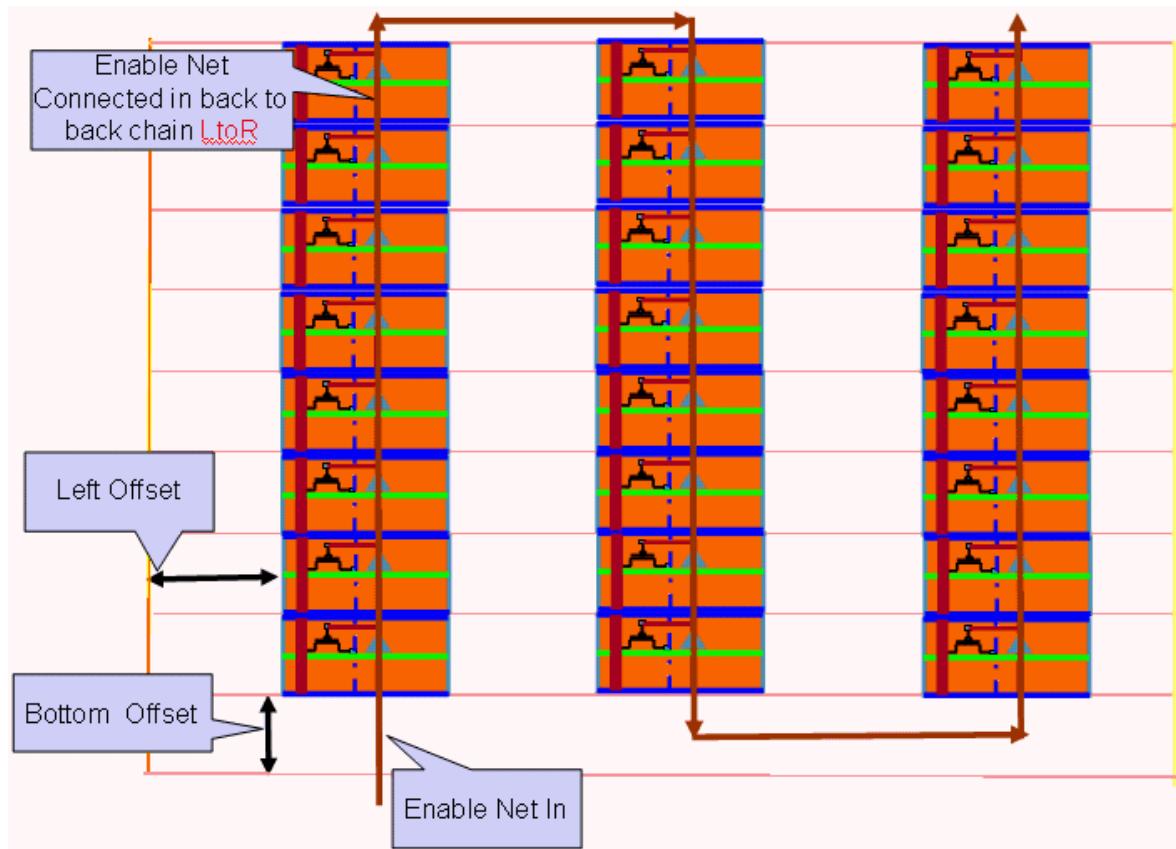
- -backToBackChain

Connects the enableNetOut at the top of a column to the enableNetIn at the top of the next column, and connects the enableNetOut at the bottom of a column to the enableNetIn at the bottom of the next column.

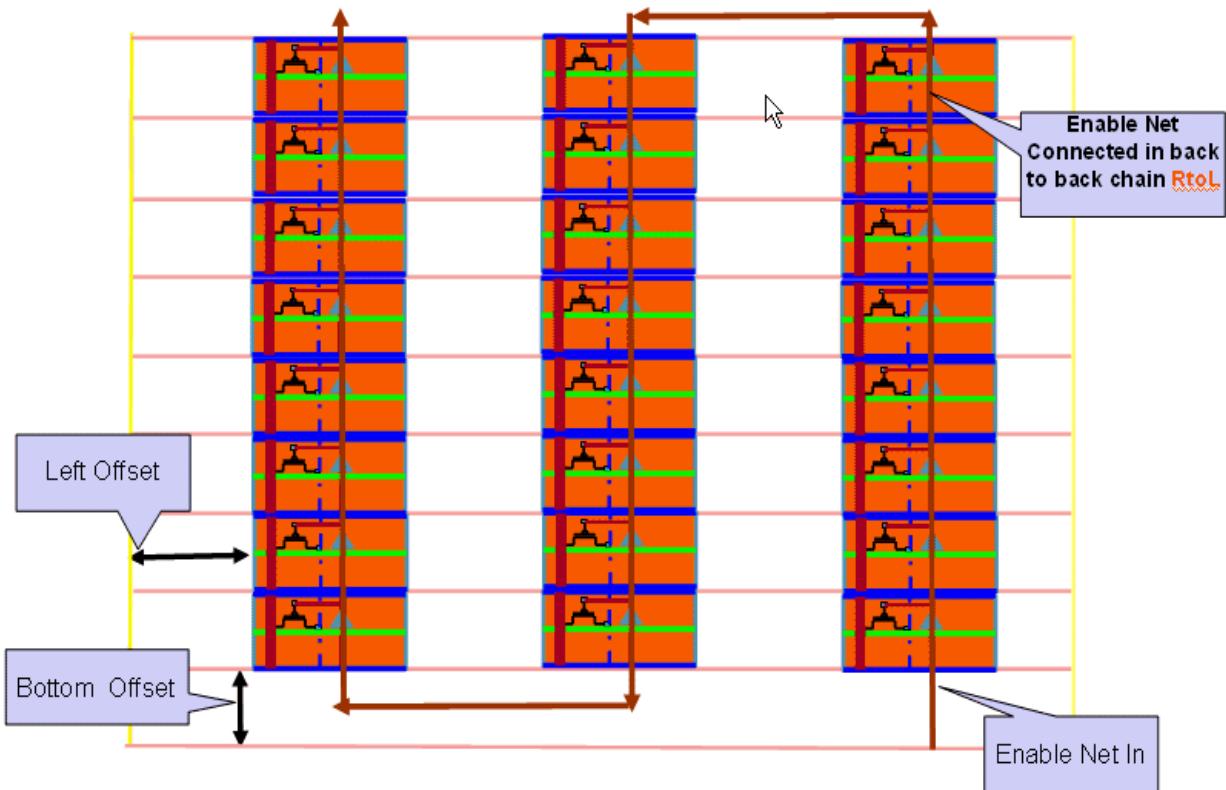
## Encounter User Guide

### Low Power Design

The following figure shows -backToBackChain with the LtoR (left-to-right) option:



The following figure shows -backToBackChain with the RtoL (right-to-left) option:



#### Example:

```
addPowerSwitch \
-column \
-powerDomain DSP \
-switchModuleInstance dummy_dsp_1 \
-enablePinIn {NSLEEPIN2} \
-enablePinOut {NSLEEPOUT2} \
-enableNetIn {UNCONNECTED249} \
-globalSwitchCellName HDRDIHVTD2 \
-enableNetOut {power_out_ack} \
-leftOffset 15.0 \
-bottomOffset 0.0 \
-horizontalPitch 150.0 \
-backToBackChain RtoL
```

- **-loopbackAtEnd**

Connects the enablePinOut of the last cell in the chain to the enablePinIn of the same cell.

In the following example, two -enablePinIn and -enablePinOut pins are specified, so you can use -loopbackAtEnd:

```
addPowerSwitch \
-column \
-powerDomain DSP \
-switchModuleInstance dummy_dsp_1 \
-enablePinIn {NSLEEPIN2 NSLEEPIN1} \
-enablePinOut {NSLEEPOUT2 NSLEEPOUT1} \
-enableNetIn {UNCONNECTED249} \
-globalSwitchCellName HDRDIHVTD2 \
-enableNetOut {power_out_ack} \
-leftOffset 15.0 \
-bottomOffset 0.0 \
-horizontalPitch 150.0 \
-topDown \
-backToBackChain RtoL \
-loopbackAtEnd
```

## Controlling the Maximum Enable Chain Depth

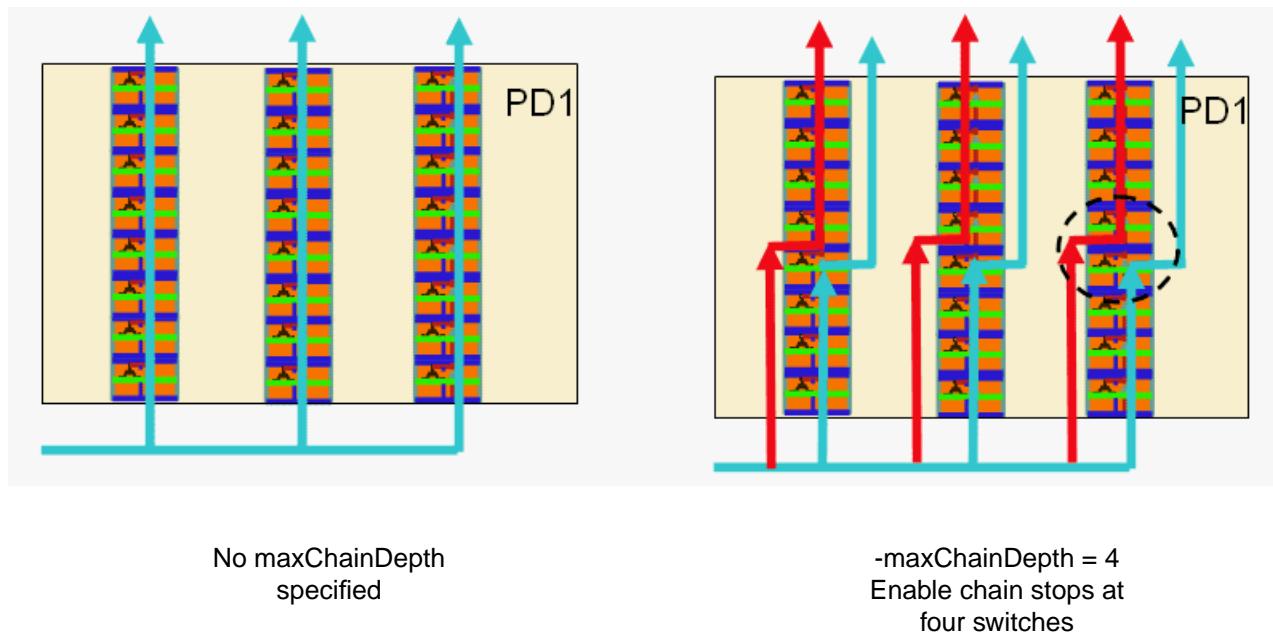
You can control the ramp-up time for the power domain by specifying the number of column switches are allowed in an enable chain before a new chain is started.

- To control the maximum number of switches in an enable chain, use the following parameter:

```
-maxChainDepth integer
```

The software then starts a new enable chain at the next switch, as shown in the following figure:

```
addPowerSwitch -column -powerDomain PD1 -maxChainDepth 4
```



## Synthesizing Acknowledge Trees

The Encounter tool can automatically create acknowledge trees that you would otherwise build manually. The acknowledge tree collects enable signals exiting the power domain and funnels them to an acknowledge receiver pin.

1. Use the following parameters to create the acknowledge tree:

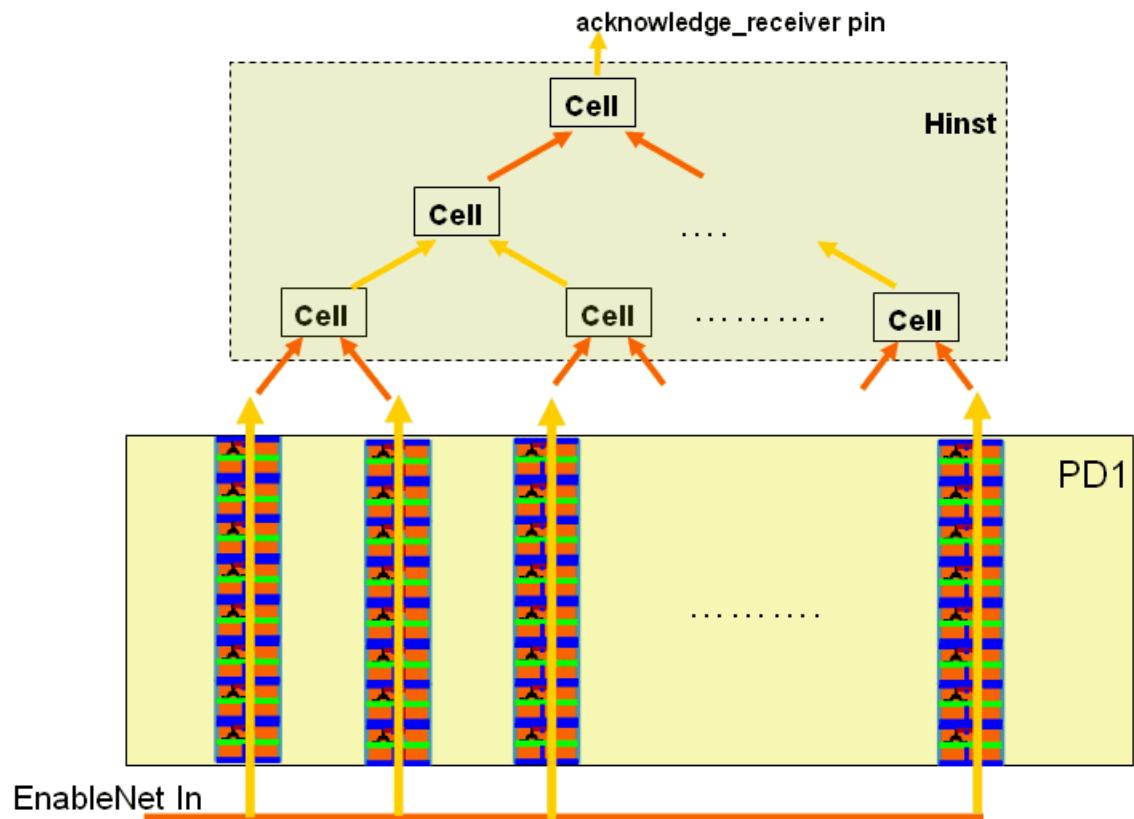
- `-acknowledgeTreeCell`
- `-acknowledgeTreeHierInstance`

If you do not specify `-acknowledgeTreeHierInstance`, the tool places the cells in the top module.

## Encounter User Guide

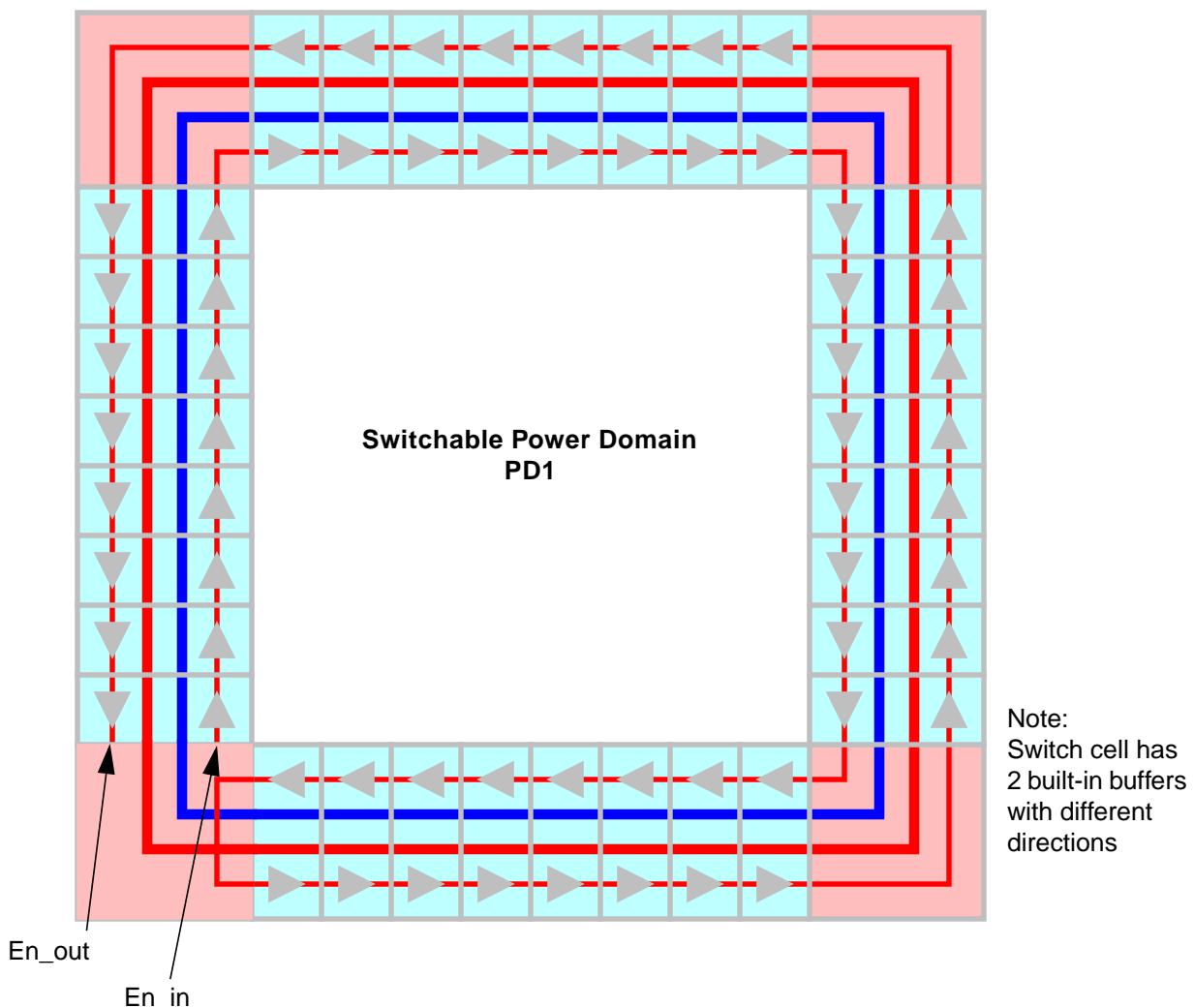
### Low Power Design

The following figure shows an acknowledge tree built from cells of type Cell, placed in hierarchical instance HInst.



## Adding Power Switch Rings

You can add power switches in a ring entirely outside the boundary of the power domain as shown below.



In this figure, the switches abut and connect to the next switch. Because the switches in this example contain two built-in buffers with different directions, the enable net loops around the inner side of the ring, connects at the corner, and loops back around to where it becomes the enable net out.

This technique is useful when the power domain is a pre-designed macro.

To create a power switch ring, use the following command:

## Encounter User Guide

### Low Power Design

---

```
addPowerSwitch -ring
```

The following parameters are required:

```
-powerDomain  
-enablePinIn  
-enablePinOut  
-enableNetIn  
-enableNetOut
```

By default, the tool distributes cells evenly in the ring. To stack cells instead, use the following command:

```
addPowerSwitch -ring -distribute 0
```

Instead of using the text command, you could use the menu command as follows:

- From the main Encounter window, select *Power– Multiple Supply Voltage – Add Power Switch*, then click on the *Ring* tab. Select *Distribute Switches* on the Switch Cell Count form.

With ring options, you have many ways of configuring the switch ring. Among many possibilities, you can do the following:

- Control how switches are distributed around the ring
- Choose the sides on which you want to add switch cells
- Specify the breaker, buffer, filler, switch, and corner cells you want to use on specified sides
- Specify the distance between the power domain and each side of the ring
- Choose the orientation of cells on specified sides
- Arrange the buffer, breaker, filler, and switch cells in a pattern

### Creating Patterns

When you create rings, you can specify a pattern that customizes switch placement. If you do not specify a pattern, the software adds switches evenly around the power domain.

The following command shows a pattern of cells that repeats on all sides:

```
addPowerSwitch -ring \  
-powerDomain TDSP2 \  
-enablePinIn {A0} -enablePinOut {Z0} \  
enableNetIn swcontrol_2 -enableNetOut swack_2  
-specifySides {1 1 1 1 1 1 1 1} \  
-sideOffsetList {3 3 3 3 3 3 3 3} \  
-globalSwitchCellName {{CDN_RING_SW S} {CDN_RING_SW_1 D} {CDS_RING_SW_2 G}} \  
-cornerCellList CDN_RING_CORNER_UL \  
-
```

## Encounter User Guide

### Low Power Design

```
-globalFillerCellName {{CDN_RING_FILLER F}}
-insideCornerCellList CDN_RING_CORNER_InCell \
-globalPattern {S S D D G G F}
```

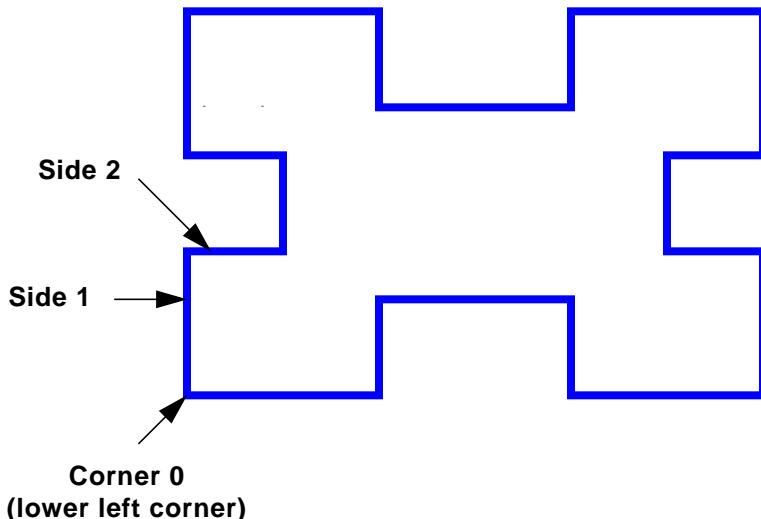
In this example, the command adds power switches in the following pattern:

```
CDN_RING_SW CDN_RING_SW CDN_RING_SW_1 CDN_RING_SW_1 CDN_RING_SW_2 CND_RING_SW_2
CDN_RING_FILLER
```

The command repeats the pattern on each side. If you want to continue the pattern on the next side or edge, use the `-continuePattern` parameter.

## Ring Conventions

The Encounter software supports rectilinear power domains, such as the 20-sided power domain shown below:



The default side/corner numbering is clockwise from the starting corner (Corner 0), which is always the lower left corner of the power domain.

Corner numbering starts with 0. Side numbering starts with 1.

## Specifying Sides in a Switch Ring

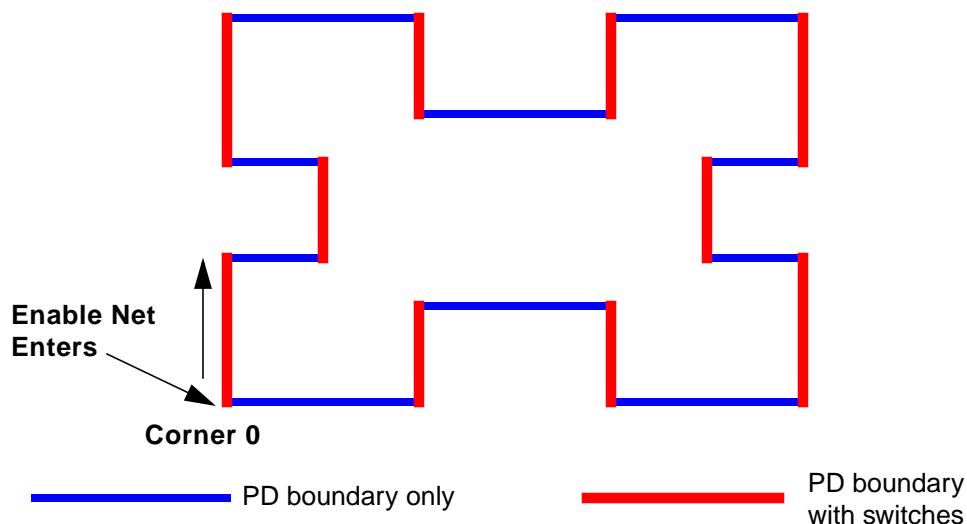
Use `addPowerSwitch -specifySides` to add switches around a power domain of any number of sides.

Each value provided in the `-specifiedSides` parameter corresponds to a side of the power domain. The 1 value indicates that the tool should add switches on a side, and 0

indicates that it should not. For example, for switches on all sides of a 4-sided power domain, use `-specifySides {1 1 1 1}`. By default, the tool adds switches on all sides.

The following example shows how you can place switches on every other side of the 20-sided power domain.

```
addPowerSwitch -ring -specifySides {1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1}
```



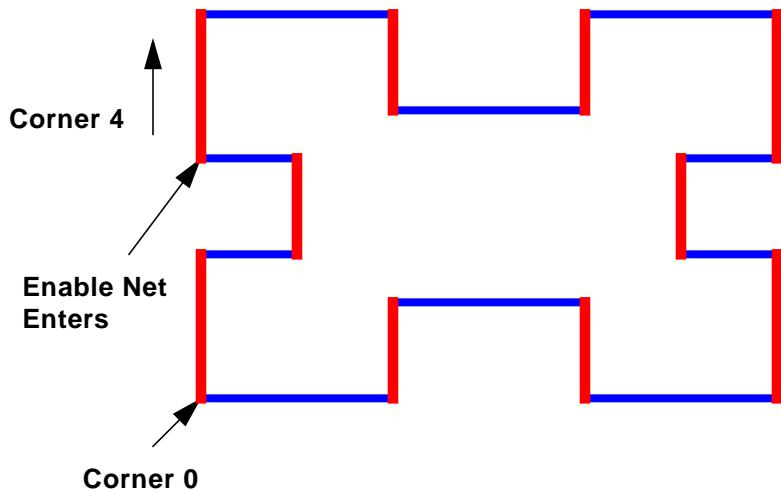
### Starting the Enable Chain at a Different Corner

By default, the enable net enters at Corner 0. To select the corner at which you want the enable net to enter, use the `-startEnableChainAtCorner`. This example does the following:

- Adds power switches on sides 1, 3, 5, 7, 9, 11, 13, 15, 17, and 19
- Starts the enable corner to corner 4.

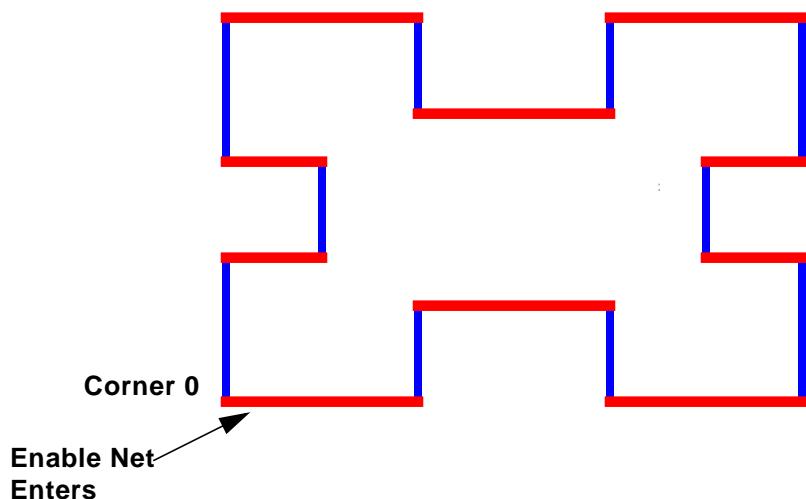
# Encounter User Guide

## Low Power Design



### **Counter-Clockwise**

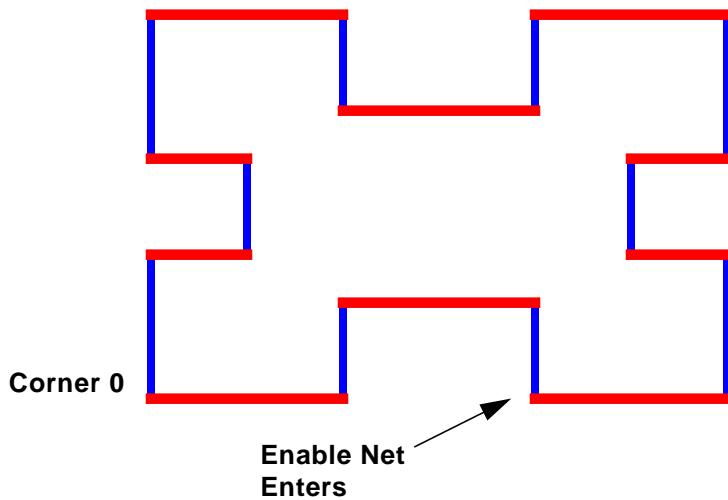
The `-counterclockwise` option reverses the corner/side numbering from Corner 0. What was side 20 in the previous example becomes side 1 in this example.



# Encounter User Guide

## Low Power Design

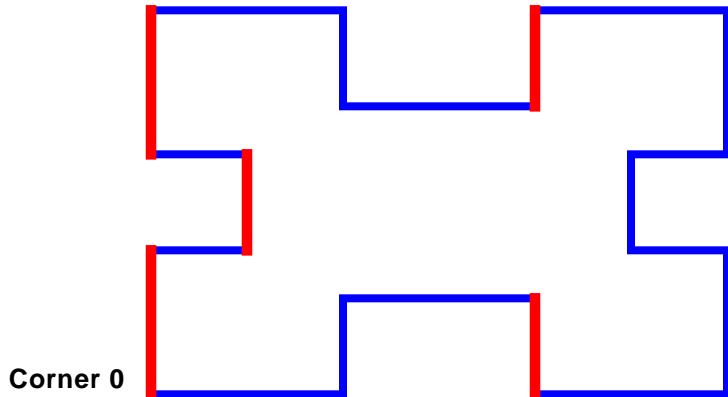
In the following example, side/corner numbering is counterclockwise, and the enable net enters at Corner 4. Note how location of the specified corner differs from the location of the corner specified without the `-counterclockwise` parameter.



## **Left Sides**

The following command adds switches only to the left sides of the power domain: Sides 1, 3, 5, 9, 17.

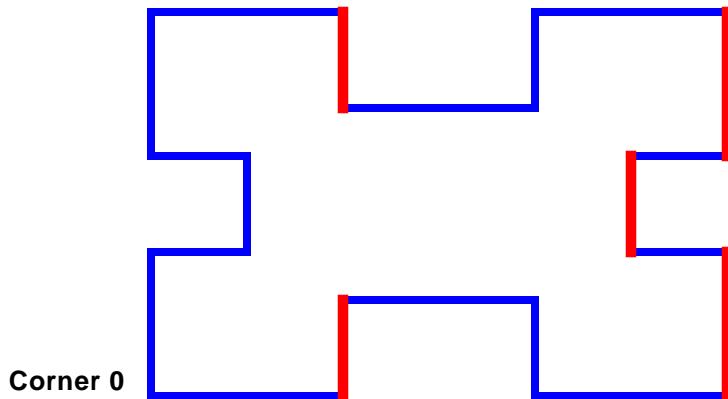
```
addPowerSwitch -ring -leftSide 1
```



## Right Sides

The following command adds switches only to the right sides of the power domain: Sides 7, 11, 13, 15, and 19.

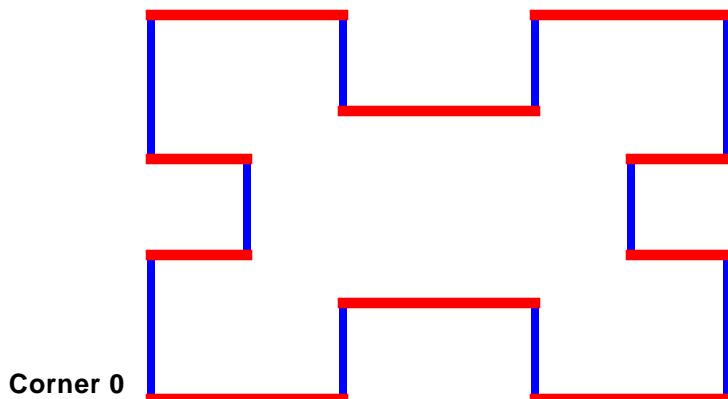
```
addPowerSwitch -ring -rightSide 1
```



## Horizontal Sides

The following command adds switches only to the horizontal sides of the power domain: Sides 2, 4, 6, 8, 10, 12, 14, 16, and 18.

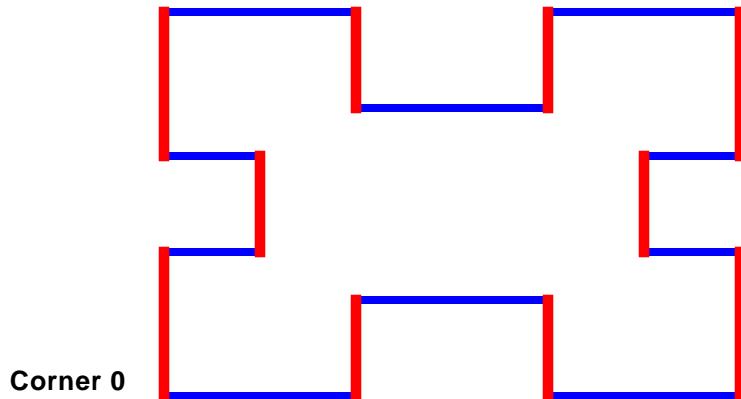
```
addPowerSwitch -ring -horizontalSide 1
```



## Vertical Sides

The following command adds switches only to the vertical sides of the power domain: Sides 1, 3, 5, 7, 9, 11, 13, 15, and 19.

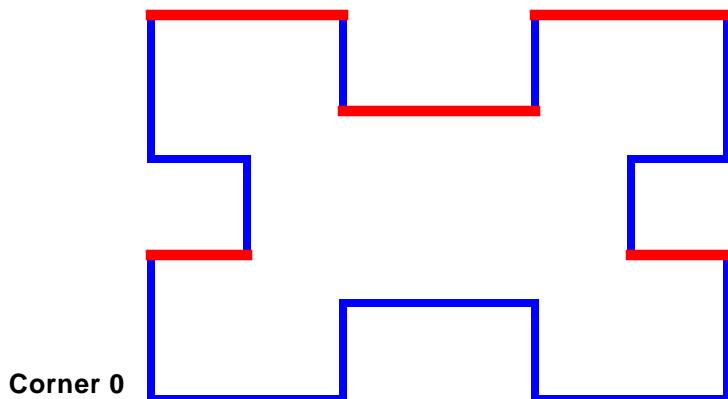
```
addPowerSwitch -ring -verticalSide 1
```



## Top Sides

The following command adds switches only to the top sides of the power domain: Sides 2, 6, 8, 10, and 14.

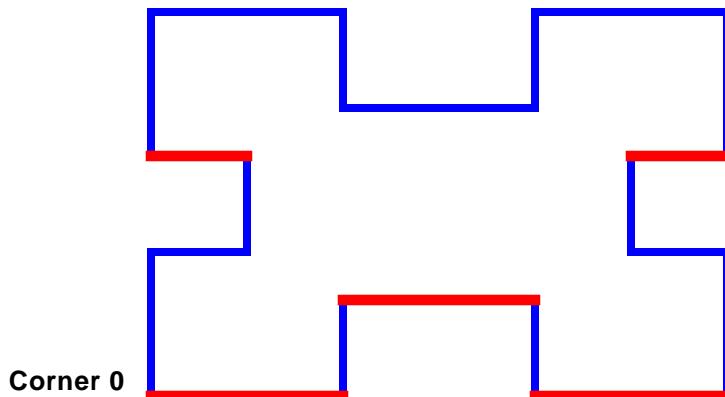
```
-addPowerSwitch -ring -topSide 1
```



## Bottom Sides

The following command adds switches only to the bottom sides of the power domain: Sides 4, 12, 16, 18, and 20.

```
-addPowerSwitch -ring -bottomSides 1
```



## Using Pitch Control and Offsets

You can control switch placement by using the following parameters:

- `-globalOffset`
- `-bottomOffset`
- `-topOffset`
- `-leftOffset`
- `-rightOffset`
- `-horizontalOffset`
- `-verticalOffset`
- `-sideOffsetList {value ...}`
- `-startOffset`
- `-startOffsetBottom`
- `-startOffsetTop`
- `-startOffsetLeft`

## Encounter User Guide

### Low Power Design

---

- -startOffsetRight
- -startOffsetHorizontal
- -startOffsetVertical
- -sideStartOffsetList {value...}
- -endOffset
- -endOffsetBottom
- -endOffsetTop
- -endOffsetLeft
- -endOffsetRight
- -endOffsetHorizontal
- -endOffsetVertical
- -sideEndOffsetList {value...}
- -forceOffset [0|1]
- -switchPitch
- -switchPitchBottom
- -switchPitchHorizontal
- -switchPitchLeft
- -switchPitchRight
- -switchPitchTop
- -switchPitchVertical
- -switchPitchSideList {value...}

### Forcing Offsets

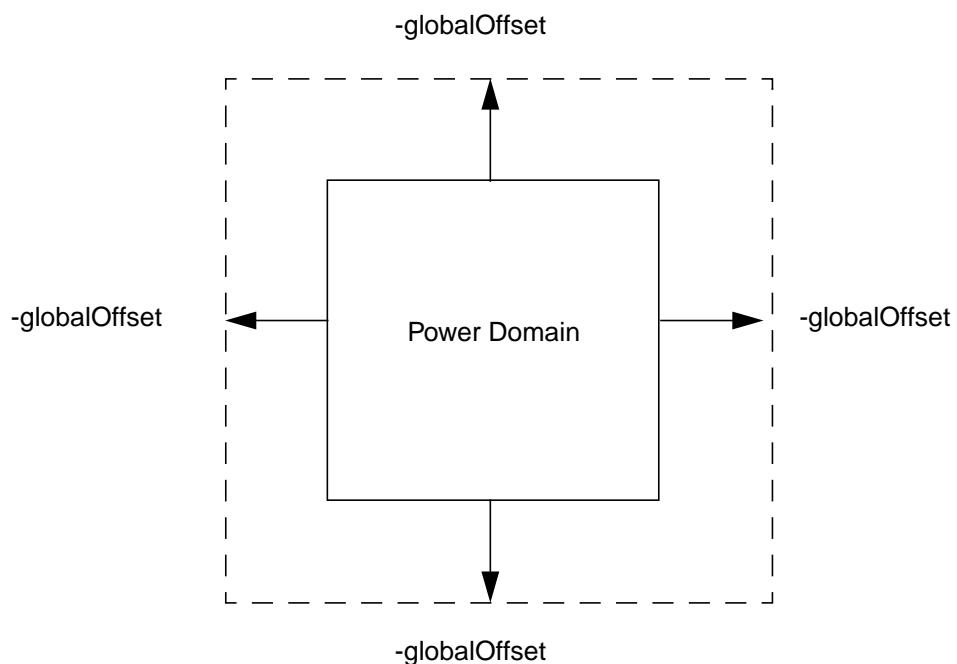
Offsets you specify are the *minimum* offsets you require to complete the power switch ring. Resulting offsets could be quite different from the ones you specify. To force the tool to comply as much as possible to the offset values, specify -forceOffset 1.

## Setting the Global Offset

Instead of placing switches against the power domain boundaries, you can place them away from the boundaries by the specified distances.

- To specify the same offset for all sides, use the `-globalOffset` parameter.  
The default offset value is 0 (no offset).

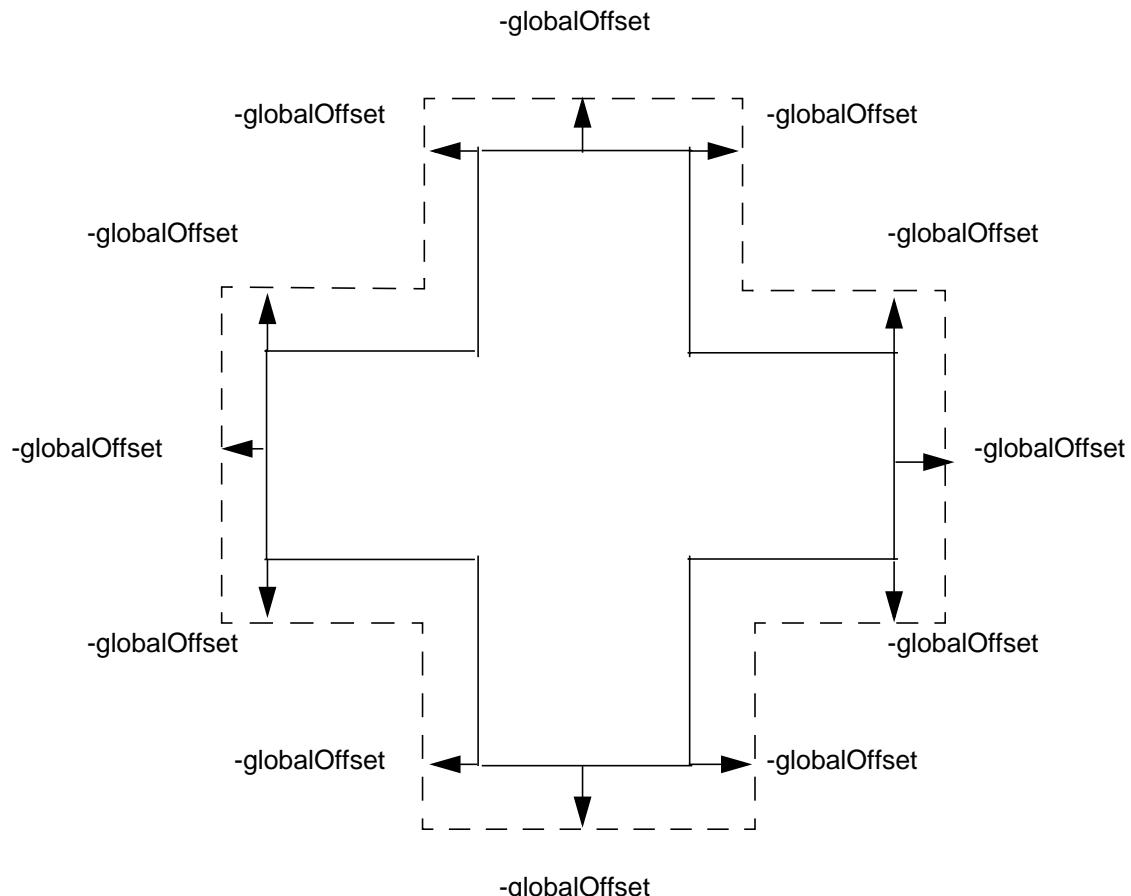
The following figure shows equal offsets for a rectangular power domain if `-forceOffset 1` is specified:



## Encounter User Guide

### Low Power Design

The following figure shows global offsets for a rectilinear power domain:



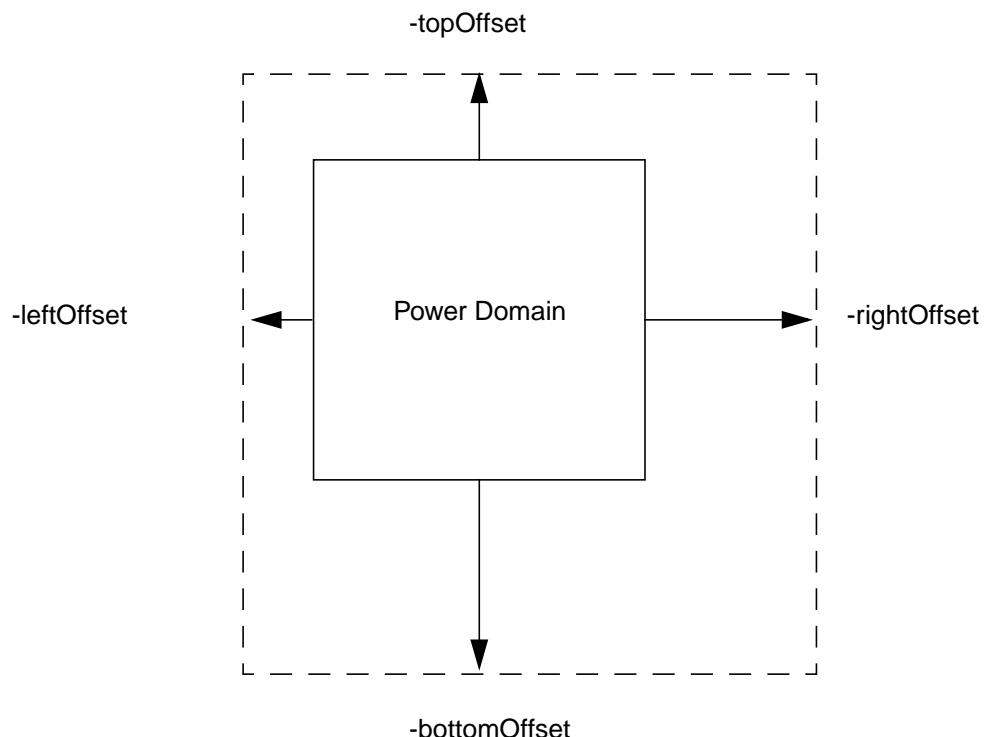
### Setting Different Offsets for Different Sides

- To specify different offsets for different sides, use the `-leftOffset`, `-rightOffset`, `-bottomOffset`, and/or `-topOffset` parameters.

## Encounter User Guide

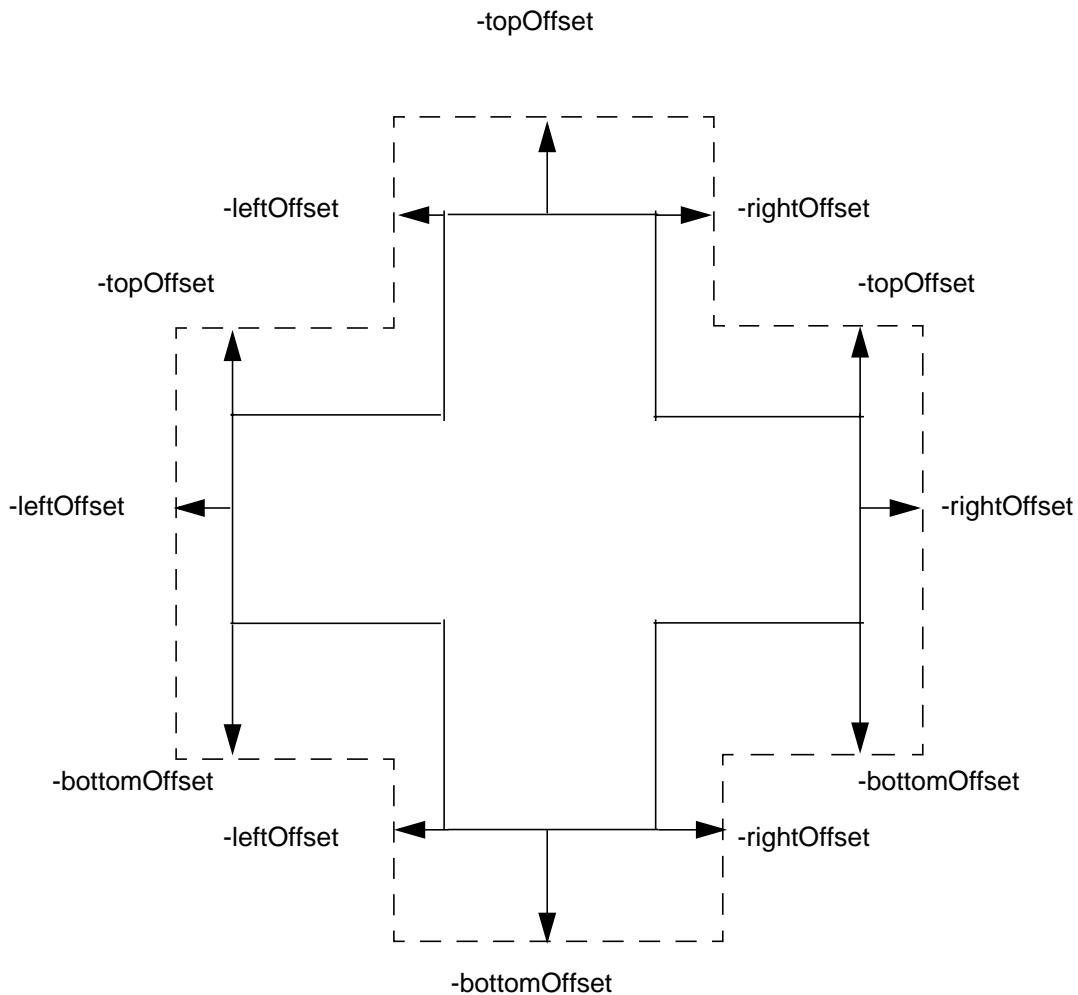
### Low Power Design

The following example shows a different offset for each side if `-forceOffset` is specified:



- To specify offsets for the top and bottom, use the `-horizontalOffset` parameter.
- To specify offsets for the left and right sides, use the `-verticalOffset` parameter.

The following figure shows how `-leftOffset`, `-rightOffset`, `-bottomOffset`, `-topOffset` parameters affects rectilinear power domains:



- To specify offsets for the horizontal sides, use the `-horizontalOffset` parameter.
- To specify offsets for the vertical sides, use the `-verticalOffset` parameter.

### Specifying Switch Location

You can choose the location for placing the first and/or last cell in a power switch row, and the spacing between switches in a row. Use the following parameters:

## Encounter User Guide

### Low Power Design

---

- **-startOffset\***: Places the first cell instance on a side a specified distance from the nearest left, right, top bottom, vertical or horizontal offset (if specified) or the nearest power domain edge.
- **-endOffset\***: Places the last cell a specified distance from the nearest \*Offset (if specified) or the nearest power domain boundary at the end of the side.
- **-switchPitch\***: Specifies the distance from switch to switch (not the spacing between switches).

The following table shows the start, end, and pitch parameters:

<b>Start Offset</b>	<b>End Offset</b>	<b>Switch Pitch</b>	<b>Applies to</b>
-sideStartOffsetList	-sideEndOffsetList	-switchPitchSideList	Specified side(s)
-startOffsetBottom	-endOffsetBottom	-switchPitchBottom	Bottom side(s)
-startOffsetTop	-endOffsetTop	-switchPitchTop	Top side(s)
-startOffsetRight	-endOffsetRight	-switchPitchRight	Right side(s)
-startOffsetLeft	-endOffsetLeft	-switchPitchLeft	Left side(s)
-startOffsetHorizontal	-endOffsetHorizontal	-switchPitchHorizontal	Horizontal side(s)
-startOffsetVertical	-endOffsetVertical	-switchPitchVertical	Vertical side(s)
-startOffset	-endOffset	-switchPitch	All sides equally

You can omit offsets because the default values are 0. You can omit pitch because, by default, the cells abut. The software starts placing cells at corner 0.

- You can combine the following:
  - **-startOffset** with other **-startOffset\*** parameters
  - **-endOffset** with other **-endOffset\*** parameters
  - **-switchPitch** with other **-switchPitch\*** parameters

If there is more than one start or end offset, or switch pitch, on a side, the software always uses the most specific parameter for the side. For example, if both **-startOffset** and **-startOffsetRight** are specified, the tool uses the **-startOffsetRight** value for the right side.

- You can combine the global offsets and pitch with side-specific offsets and pitch.
  - For example, for a rectangular power domain:  
`-startOffsetTop 1 -endOffsetLeft -2 -switchPitch 3`

## Encounter User Guide

### Low Power Design

---

- ❑ For example, for a rectilinear power domain:

```
-startOffset -1 -switchPitchSideList {3, 4, 3, 0, 0, 0, 0, 0, 0, 0}
```

- You can combine any side-specific parameters.

- ❑ For example, for a rectangular power domain:

```
-startOffsetLeft 1 -startOffsetRight 2 -endOffsetTop -2 -switchPitch 3
```

- ❑ For example, for a rectilinear power domain:

```
-startOffsetRight -1 -switchPitchSideList {3, 3, 3, 2, 2, 2, 1, 1, 1, 3}
```

**Note:** All `-startOffset` and `-endOffset` values can be positive or negative, which affect cell placement toward or away from the center of the side. Pitch values can be positive only.

### Example of Offsets

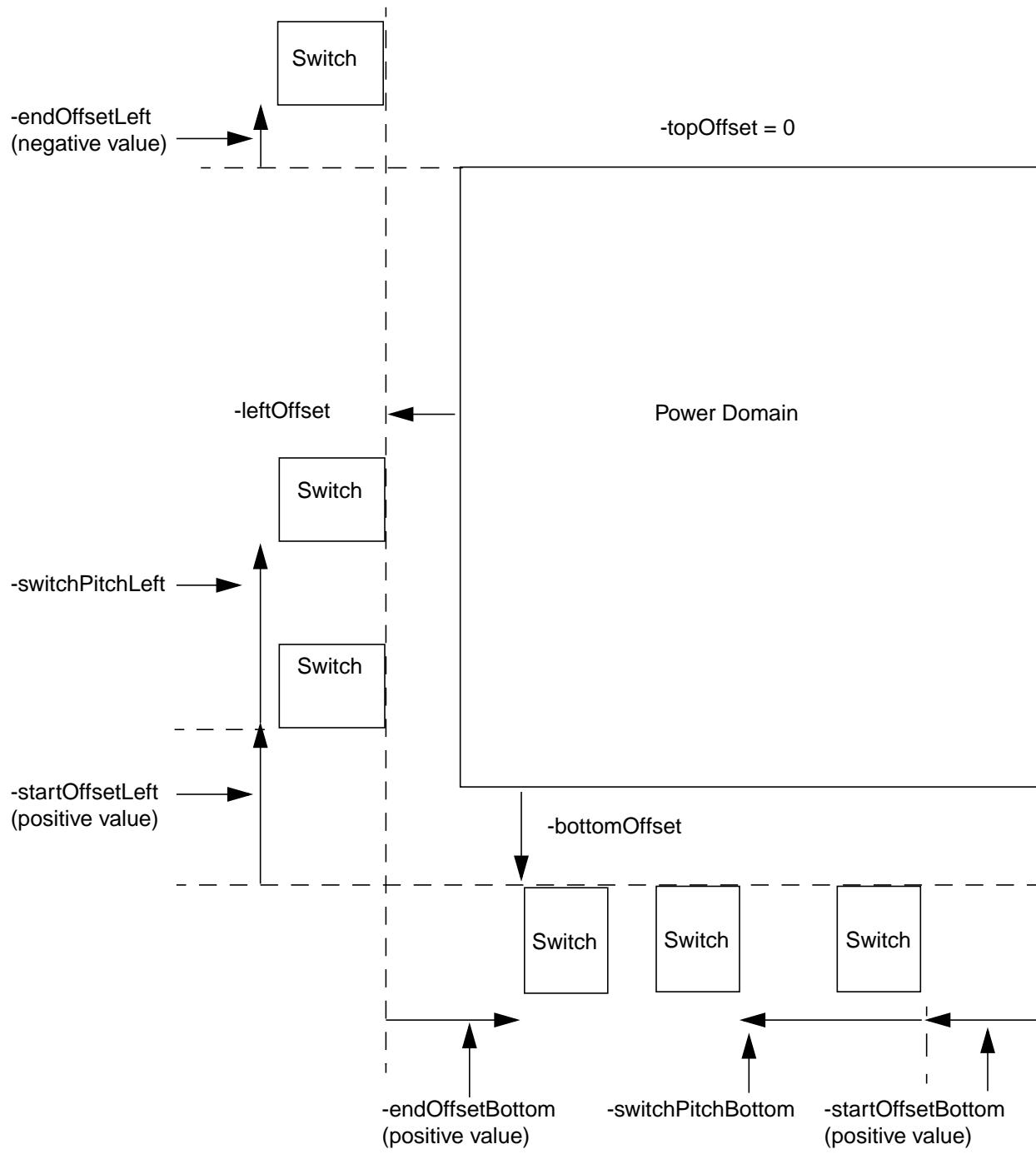
The following example shows a clockwise switch ring with the following parameters:

- `-leftOffset`
- `-bottomOffset`
- `-startOffsetLeft`
- `-startOffsetBottom`
- `-endOffsetLeft`
- `-endOffsetBottom`
- `-switchPitchLeft`
- `-switchPitchBottom`

## Encounter User Guide

### Low Power Design

Different switch pitches are specified for the left and bottom sides, and the start and end offsets are positive or negative.



- The first switch on the left side is placed a distance **-startOffsetLeft** from the edge of the **-bottomOffset** boundary.

## Encounter User Guide

### Low Power Design

---

- The second switch on the left side is placed a distance `-switchPitchLeft` from the bottom edge of the first switch on the side.
- The last switch on the left side is placed a distance `-endOffsetLeft` *above* the `-topOffset` because the `-endOffsetLeft` value is negative. In this case, the `-topOffset` is 0, so the last switch is placed above the top power domain boundary.
- The first switch on the bottom side is placed a distance `-startOffsetBottom` from the right edge of the power domain. There is no `-rightOffset` or `-topOffset` specified.
- The second switch on the bottom side is placed a distance `-switchPitchBottom` from the right edge of the first switch on the side.
- The last switch on the bottom side is placed `-endOffsetBottom` to the *right* of the `-leftOffset` edge. The `-endOffsetBottom` parameter has a positive value.

## Power Switch Optimization

The `optPowerSwitch` command lets you perform power switch optimization in two ways:

- Optimize (reduce) the number of power switches in the ring and columns
- Perform ECOs to replace a filler cell with a switch or replace a switch with a bigger switch

You can use these two features in the following flow:

- Define the floorplan and power domains
- Synthesize the power grid
- Optimize power switches
- Place the design
- Run trialRoute
- Synthesize the clock tree
- Perform power switch ECO
- Perform buffer tree synthesis

## Power Switch Reduction

For power switch optimization (reduction), you can use the following options to `optPowerSwitch`:

- `-readPowerSwitchCell`
- `-commit`
- `-effort`
- `-maxIRDrop`
- `-maxSwitchIRDrop`
- `-net`
- `-padFile`
- `-readInstancePower`
- `-reportFile`

- `-setDontTouchCells`
- `-setDontTouchInstances`
- `-totalPower`

## Power Switch ECO

For power switch ECO, you can use the following options:

- `-vsdgInFile`
- `-reportFile`
- `-fixViolations`
- `-reportViolationsOnly`

You can now fix IR violations due to added power switches.

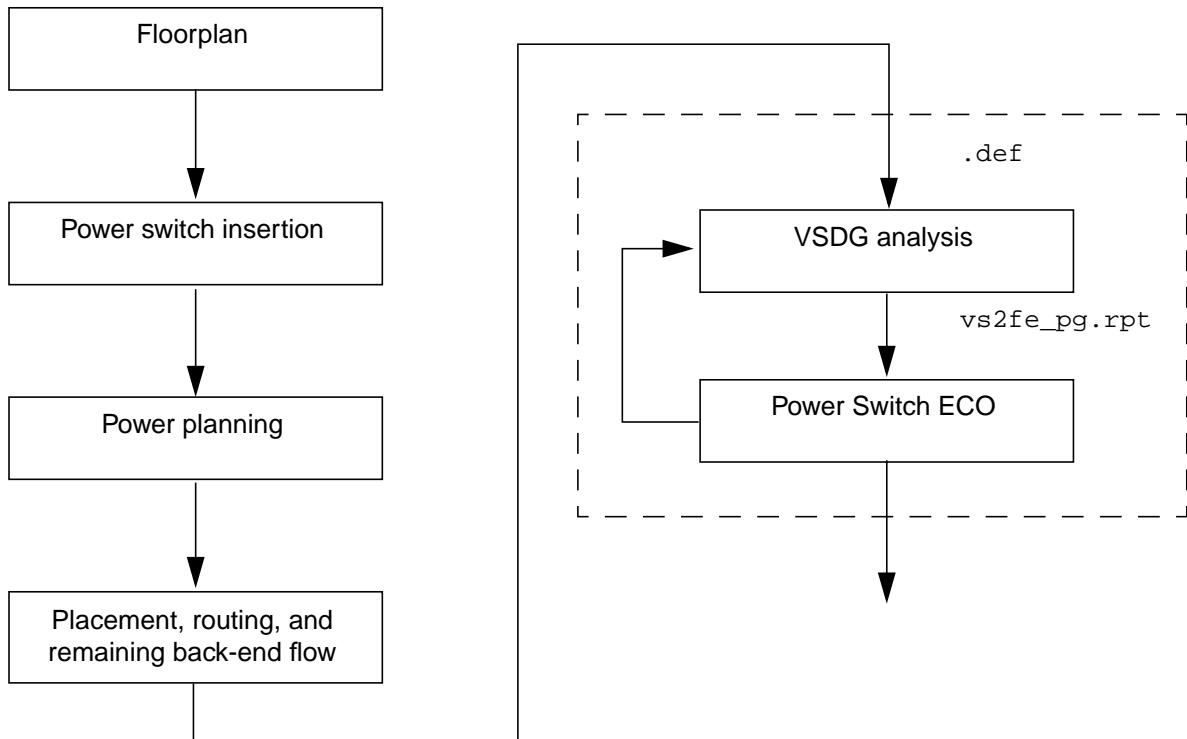
1. Add power switches to your design.
2. Create a complete power structure.
3. Pre-characterize power gating cells in Libgen.
4. Run Voltage Storm analysis to detect IR violations.
5. Use the new `optPowerSwitch` command to repair these violations.

**Note:** In power switch optimization, the inserted power switches could overlap standard cells. After running `optPowerSwitch`, run `refinePlace` to move the standard cells away from the switches.

## Encounter User Guide

### Low Power Design

The following figures shows the power switch optimization ECO flow with VSDG:



## **Encounter User Guide**

### Low Power Design

---

---

## Placing the Design

---

- [Overview](#) on page 538
- [Preparing for Placement](#) on page 538
- [Loading a Design](#) on page 538
- [Guiding Placement With Blockages](#) on page 539
- [Adding Well-Tap Cells](#) on page 541
- [Adding End-Cap Cells](#) on page 542
- [Placing Spare Cells and Spare Modules](#) on page 543
- [Adding Padding](#) on page 550
- [Placing Standard Cells](#) on page 554
- [Running Placement in Multi-CPU Mode](#) on page 555
- [Checking Placement](#) on page 558
- [Adding Filler Cells](#) on page 560
- [Placing Gate Array Style Filler Cells for Post-Mask ECO](#) on page 561
- [Adding Decoupling Capacitance](#) on page 562
- [Adding Logical Tie-Off Cells](#) on page 563
- [Saving Placement Data](#) on page 564
- [Specifying and Placing JTAG and Other Cells Close to the I/Os](#) on page 564
- [Optimizing and Reordering Scan Chains](#) on page 565

## Overview

After floorplanning, place the cells in the design. Placement considers the modules that were placed during floorplanning and takes into account the hierarchy and connectivity of the design. It honors floorplanning constraints, including guides, regions, and fences. For descriptions of the constraint types, see “[Module Constraint Types](#)” on page 360.

After the cells are placed and resulting violations corrected, run pre-CTS optimization.

## Loading a Design

Load a design by using the `restoreDesign` command or by reading in the following files:

- Encounter configuration file (\*.config), with .lib, .lef, timing constraint file, and capacitance table specified
- One of the following files:
  - Floorplan file (\*.fp) with all blocks pre-placed
  - DEF file (the DEF file can contain scan chain information)

For more information, see

- [“Importing and Exporting Designs” on page 119](#)
- [`restoreDesign`](#) in the “Import and Export Commands” chapter of the *Encounter Text Command Reference*.
- [Load and Check Data](#) in the *Encounter Flat Implementation Flow Guide*

## Preparing for Placement

Before placement, run the following commands and correct problems. Some of these commands generate reports you can use as a baseline for comparisons later in the flow.

- Run `runN2NOpt` to remap and reoptimize the gate-level netlist to improve timing and area. For more information, see [`runN2NOpt`](#) in the “Netlist-to-Netlist Command” chapter of the *Encounter Text Command Reference*.
- Run `checkDesign` to check the integrity of the library and design data. For more information, see [`checkDesign`](#) in the “Import and Export Commands” chapter of the *Encounter Text Command Reference*.

- Run `checkPlace` (or `checkDesign -place`) or use the Violation Browser to check for violations caused by preplaced cells or blocks. For more information, see [checkPlace](#) in the “Placement Commands” chapter of the *Encounter Text Command Reference* or [Violation Browser](#) in the “Tools Menu” chapter of the *Encounter Menu Reference*.
- Run `timeDesign -prePlace` to get an idea of Zero Wire Load timing of the design. For more information, see [timeDesign](#) in the “Timing Analysis (Common Timing Engine) Commands” chapter of the *Encounter Text Command Reference*.
- Run `createObstruct` to create blockages (this is usually done during floorplanning). For more information see [Guiding Placement With Blockages](#) on page 539 or [createObstruct](#) in the “Floorplan Commands” chapter of the *Encounter Text Command Reference*.
- Use one of the following methods to place and fix hard blocks. This step is necessary because `placeDesign` does not place blocks in default mode.
  - Run `planDesign`. For information, see [planDesign](#) in the “Floorplan Commands” chapter of the *Encounter Text Command Reference*.
  - Manually place and fix hard blocks.

For more information on preparing the design for placement, see “[Data Preparation](#)” on page 91.

## Guiding Placement With Blockages

Use placement blockages to help guide placement.

Create the blockages during the floorplanning session by using the following command:

`createObstruct`

After creating a blockage, assign an attribute to it by using the Attribute Editor.

Alternatively, you can create placement blockages using the *Placement Blockage Setting* form. For more information, see [Placement Blockage Setting](#) in the “Floorplan Menu” chapter of the *Encounter Menu Reference*.

A placement blockage has one of the following attributes:

**Hard**      The area cannot be used to place blocks or cells. By default, `createObstruct` creates blockages with this attribute.

**Note:** In default mode, `placeDesign` does not place blocks.

**Soft** The area cannot be used to place blocks or cells during placement, but can be used during in-place optimization, clock tree synthesis, ECO placement, or placement legalization (`refinePlace`).

**Partial** Sets a percentage of the area that is unavailable for placement. Use the *Blockage Percentage* pull-down menu to select a percentage. For example, a partial blockage of 75 percent means that up to 25 percent of placement density is allowed in the area.

**Note:** A partial blockage of 100 percent (or 0 percent placement density screen) behaves as a soft blockage.



#### Tip

If the design has routing violations in the small channels between hard blocks, consider running `createObstruct` to add soft placement blockages in these areas. Although the blockages obstruct standard cells during placement, they do not obstruct optimization operations. Using soft blockages can help improve both timing and routability.

For more information, see

- [createObstruct](#) in the “Floorplan Commands” chapter of the *Encounter Text Command Reference*
- [Attribute Editor](#) in the “Tools Menu” chapter of the *Encounter Menu Reference*

## Placement Treatment of Preroutes

Placement treats preroutes the same way it treats routing blockages: It places standard cell instances at legal locations where there should not be any DRC violations against preroutes or routing blockages.

Typically, you use preroutes for special nets that are floorplanned (pre-designed) before placement, such as power, ground, and clock mesh nets, where you do not want any standard cells placed underneath. Instances placed next to power and ground stripes honor the design spacing rule. Instances placed next to routing blockage objects are set adjoined.

By default, the Encounter software blocks the placement of standard cells on *metal2* for a three-metal layer process. The software blocks placement on *metal2* and *metal3* for a four or more metal layer process.

You can change this behavior by using the following command before running placement:

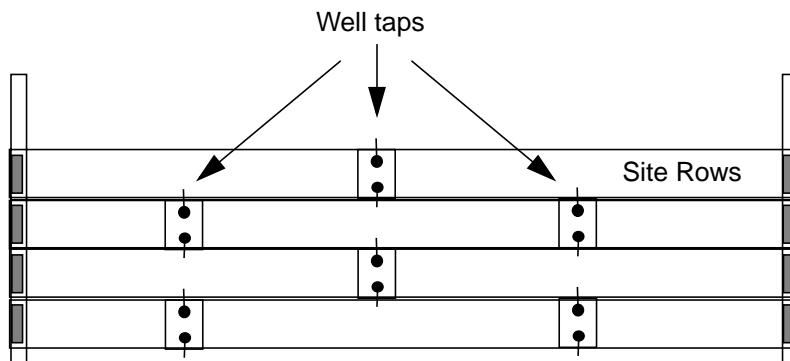
```
setPrerouteAsObs
```

For more information, see [setPrerouteAsObs](#) in the “Placement Commands” chapter of the *Encounter Text Command Reference*.

## Adding Well-Tap Cells

Well taps are physical-only filler cells that are required by some technology libraries to limit resistance between power or ground connections to wells of the substrate. Well-tap cells are placed in a preplaced status, so future placement commands do not move them.

The following diagram shows an example of well-tap cell placement. In this diagram, the cells are staggered in the site rows.



Add well taps after the floorplan is fixed and hard blocks are placed, but before placing standard cells.

Use one of the following methods to add well-tap cells:

- Add Well Tap Instances form
- `addWellTap` command

## Controlling the Distance Between Well-Tap Cells

Use the `addWellTap -cellInterval` or `-maxGap` parameter to specify the maximum distance between well-tap cells in the same row.

- `-cellInterval` measures the distance from the center of one well-tap cell to the center of the next well-tap cell in the same row.
- `-maxGap` measures the distance from the right edge of one well-tap cell to the left edge of the next well-tap cell in the same row.

By default, the software always leaves a distance that is at least 45 percent of the specified maximum distance between well-tap cells in the same row. For example, if the specified maximum distance between same-row well-tap cells is 48.0 microns, the default minimum distance would be 21.6 microns.

## **Adding Well-Tap Cells to MSV Designs**

In cases where there are different voltages in the same design, also known as a multi-voltage (MSV) design, specify the power domain in which to insert the well-tap cells by using the following command:

```
addWellTap -powerDomain
```

## **Deleting Well-Tap Cells**

To remove added well-tap cells, use the Delete Instances form or the `deleteFiller` command. If you specify an area, the `deleteFiller` command deletes only well-tap cells that are completely contained within the area; it does not delete well-tap cells that cross the area boundary.

For more information see the following topics:

- [addWellTap](#) and [deleteFiller](#) in the “Placement Commands” chapter of the *Encounter Text Command Reference*.
- [Add Well Tap](#) and [Delete Filler](#) in the “Placement Menu” chapter of the *Encounter Menu Reference*

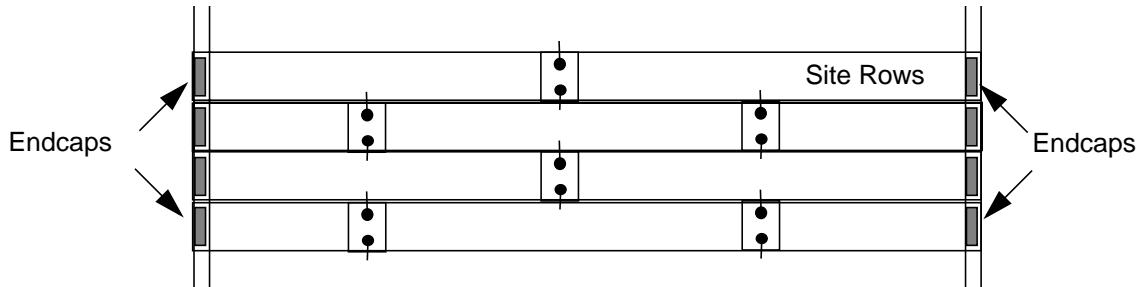
## **Adding End-Cap Cells**

End-cap cells are preplaced physical-only cells that are required to meet certain design rules. They are placed at the ends of the site rows, and are used in some technologies for power distribution. End-cap cells are placed in a preplaced status, so future placement commands do not move them. Add end-cap cells to the design before any other standard cells are placed, but after hard blocks are placed in the floorplan.

## Encounter User Guide

### Placing the Design

The following diagram shows an example of end-cap cell placement. The cells are placed at the ends of each site row.



To add end-cap cells, use the Add End Cap Instances form or the `addEndCap` command.

### Adding End Cap Cells to MSV Designs

In cases where there are different voltages in the same design, specify the power domain in which to insert the end cap cells by using the following command:

```
addEndCap -powerDomain
```

### Deleting End-Cap Cells

To remove end-cap cells, use the Delete Instances form or the `deleteFiller` command. If you specify an area, the `deleteFiller` command deletes end-cap cells that are completely contained within the area; it does not delete end-cap cells that cross the area boundary.

For more information see

- [addEndCap](#) and [deleteFiller](#) in the “Placement Commands” chapter of the *Encounter Text Command Reference*
- [Add End Cap Instances](#) and [Delete Instances](#) in the “Placement Menu” chapter of the *Encounter Menu Reference*

## Placing Spare Cells and Spare Modules

### Placing Spare Cells That Are Included in the Netlist

If spare cell instances are included in the gate-level netlist, the software places them during preplacement processing; however, you must specify them during the floorplanning session.



#### Tip

Cadence recommends that you place clusters of spare cells at different locations within the core area to allow easy access to the cells from different parts of the core.

1. Specify the spare cells by using the following command:

```
specifySpareGate
```

- Use the following parameter to identify the module whose hierarchy contains the spare cell instances: `-hinst`

2. If the design contains hierarchical modules, specify the following `setPlaceMode` parameter to ensure that the spare cells within the modules are kept within bounds of the hierarchy, even if no constraint is set on it:

```
-moduleAwareSpare
```

This parameter is valid whether `setPlaceMode -modulePlan` is true or false.

For more information on using this parameter, see [“Running Hierarchy-Aware Spare Cell Placement”](#) on page 547.

**Note:** In the GUI, select the *Hierarchy Aware Spare Cell Placement* option on the Design – Mode Setup – Placement form.

## Related Topics

To see this step in the design flow, see [Place the Design and Run Pre-CTS Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

For more information, see the following commands in the “Placement Commands” chapter of the *Encounter Text Command Reference*:

- [setPlaceMode](#)
- [specifySpareGate](#)

## Placing Spare Cells That Are Not Included in the Netlist

If the netlist does not include spare cell instances, you must create a spare module and place it before you place the standard cells.

1. Use the following command to create a spare module:

```
createSpareModule
```

2. Use the following command to place the module:

```
placeSpareModule
```

To delete a spare module, use the following command:

```
deleteSpareModule
```

For more information, see the following commands in the “Placement Commands” chapter of the *Encounter Text Command Reference*:

- [createSpareModule](#)
- [placeSpareModule](#)
- [deleteSpareModule](#)

## Spare Cell Placement Behavior

- If there are no floorplanning constraints, or if the design has not been floorplanned, the software places spare cell instances randomly in the core area.
- If a spare cell instance is contained in a fence or a region, the software places the instance randomly in the fence or region that includes the instance.
- If spare cell instances in the netlist are grouped into modules, the software places the modules in a grid fashion in the core area.

**Note:** For information on controlling spare cell placement when hierarchy is an issue, see [“Running Hierarchy-Aware Spare Cell Placement”](#) on page 547.

Spare cell distribution is dependent upon the way spare cells are connected.

- If the spare cells are floating (that is, if they are not connected) or they are connected to power or ground, they are evenly distributed in the placement area.
- If the spare cells have connections to other spare cells, they are treated as a spare cell group and are placed close to one another in the placement area.
- If the spare cells, or a group of spare cells, have a connection to a non-spare cell instance, they are placed close to that instance.

To instruct the software to disregard spare cell connections and distribute the cells evenly in the placement area, complete one of the following steps before running placement:

- Specify the following command:  

```
setPlaceMode -ignoreSpare true
```
- Select the *Ignore Spare Cell Connections* option on the *Placement* page of the Design – Mode Setup form.

## **Encounter User Guide**

### Placing the Design

---

For more information, see [setPlaceMode](#) in the “Placement Commands” chapter of the *Encounter Text Command Reference* or [Mode Setup – Placement](#) in the “Design Menu” chapter of the *Encounter Menu Command Reference*.

## Running Hierarchy-Aware Spare Cell Placement

To control placement of spare cells or modules in the netlist when hierarchy is an issue, use the following commands:

- `specifySpareGate {-hinst | -inst}`
- `setPlaceMode -moduleAwareSpare {true | false}`

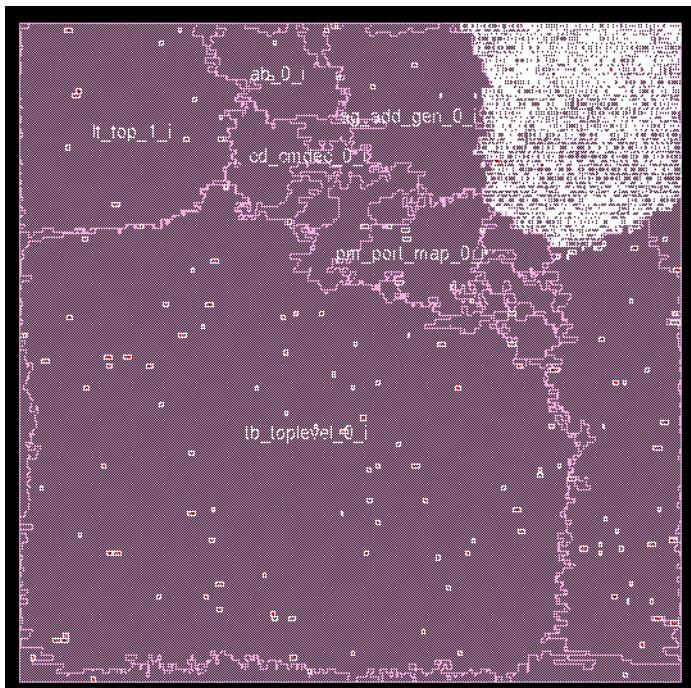
The following examples and figures show how these commands affect placement.

```
specifySpareGate -inst blk1/spare_1/*
# Works also for specifySpareGate -hinst
setPlaceMode -moduleAwareSpare true
#Works also for -moduleAwareSpare true -modulePlan false
placeDesign
```

To place the spare cells evenly in the core, without binding them to the `lt_top_0_i` hierarchy, use the following commands:

```
for {set x 0} {$x < 6} {incr x 1} {
specifySpareGate -inst lt_top_0_i/spare_${x}i/*
}

placeDesign
```



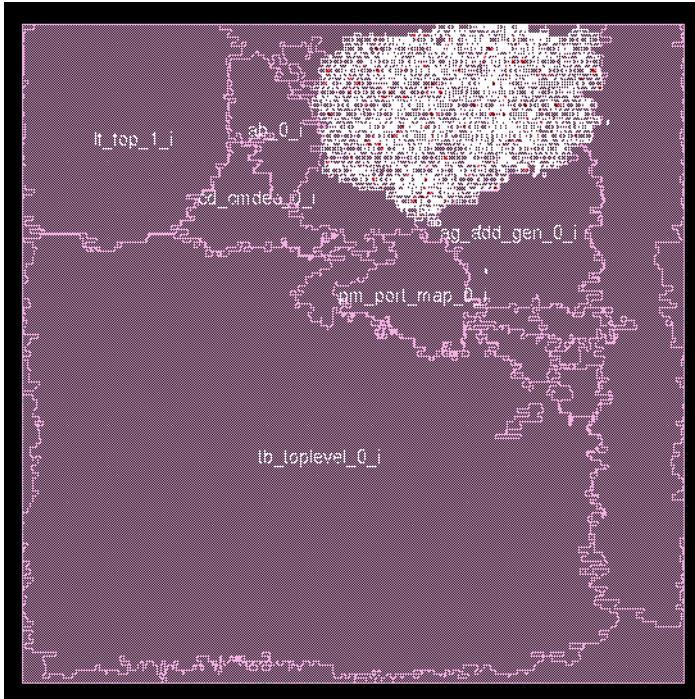
The log file, before Iteration 1, contains the following information: Identified 240 spare or floating instances, with no clusters.

## Encounter User Guide

### Placing the Design

To place the spare cells within the bounds of the `lt_top_0_i` hierarchy, using the following commands:

```
for {set x 0} {$x < 6} {incr x 1} {  
    specifySpareGate -inst lt_top_0_i/spare_${x}i/*  
}  
  
setPlaceMode -moduleAwareSpare true  
placeDesign
```



The log file, before Iteration 1, contains the following information: Identified 240 spares within logical modules.

## Encounter User Guide

### Placing the Design

To spread out the spare cells evenly as six clusters in the core, without binding them to the lt\_top\_0\_i hierarchy, use the following commands:

```
for {set x 0} {$x < 6} {incr x 1} {  
    specifySpareGate -hinst lt_top_0_i/spare_${x}i/*  
}  
  
placeDesign
```



The log file, before Iteration 1, contains the following information: Identified 240 spares or floating instances, where some are grouped into 6 clusters.

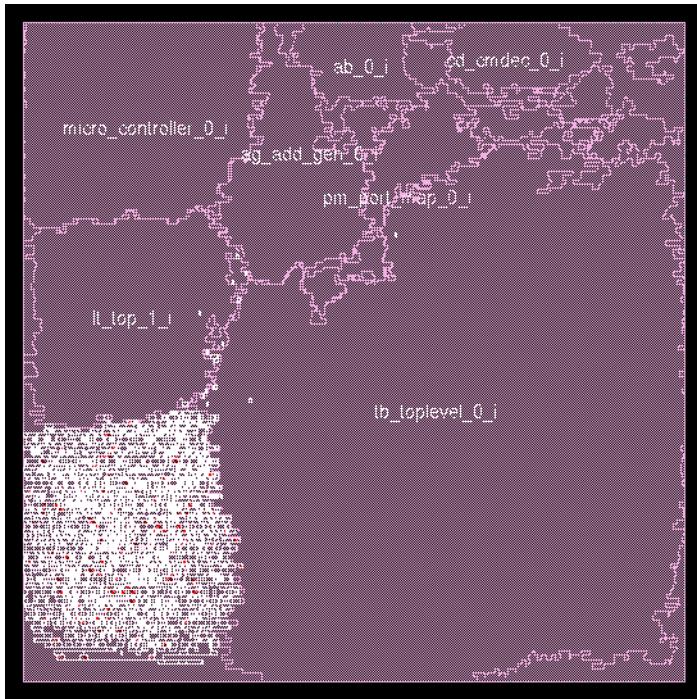
## Encounter User Guide

### Placing the Design

To place the spare cells within the bounds of the `lt_top_0_i` hierarchy, use the following commands:

```
for {set x 0} {$x < 6} {incr x 1} {
    specifySpareGate -hinst lt_top_0_i/spare_${x}i/*
}

setPlaceMode -moduleAwareSpare true
setPlaceMode -modulPlan false
placeDesign
```



The log file, before Iteration 1, contains the following information: Identified 240 spares within logical modules, of which 240 are in 6 spare-only modules.

## Adding Padding

Add padding to reserve placement space for cells or routing added after placement, for example, to make sure there is room to insert clock buffers when running Clock Tree Synthesis (CTS) on a highly localized clock. The software adds the padding on the right side of placed instances at a default `meta/2` pitch dimension.

You can add padding to instances, leaf cells, and hierarchical modules.



If the clock in your design is concentrated in a tight area, reserve three to five percent of the targeted final utilization to add clock buffers. If your initial settings do not provide sufficient space for the buffers, add more padding and rerun placement after CTS or placement optimization.

## Adding Instance or Module Padding

Instance padding and module padding reserve space during global placement so it can be used later in the design flow, for cells added during placement legalization (`refinePlace`), Clock Tree Synthesis (CTS), or timing optimization.

### Adding Instance Padding

Instance padding is specified in terms of the number of sites occupied by each instance. For example, if a row fits 30 instances without padding, you can specify padding of two sites for each instance in the row. In this case, each instance in the row will then occupy two sites, and the row will fit only 15 instances.

1. Specify the following command:

```
specifyInstPad
```



To add extra padding on the most timing-critical instances on timing-critical paths, run the following command before placing standard cells:

```
setPlaceMode -tdInstPadding true
```

2. (Optional) Report instance padding by using the following command:

```
reportInstPad
```

To delete instance padding, use the following command:

```
deleteInstPad
```

For more information, see the following commands in the “Placement Commands” chapter of the *Encounter User Guide*:

- [specifyInstPad](#)
- [reportInstPad](#)
- [deleteInstPad](#)

## Adding Module Padding

To reduce localized congestion, add module padding.

1. Specify the following command:

```
setPlaceMode -modulePadding module factor
```

This command adds padding within hierarchical modules by spreading out the standard cell instances within the modules. The padding is specified in terms of a factor that is applied to the instance area of all the cells within the module. For example, a factor of 1.2 increases the area by 20 percent.

This parameter is disabled when `-modulePlan false` is specified.

**Note:** The software ignores factors that are less than 1.0.

2. Run standard cell placement.

For more information, see [setPlaceMode](#) in the “Placement Commands” chapter of the *Encounter User Guide*.

## Adding Cell Padding

Cell padding adds hard constraints to placement. The constraints are honored by cell legalization, CTS, and timing optimization, unless the padding is reset after placement so those operations can use the reserved space. You can use cell padding to reserve space for routing.

- Specify the following command:

```
specifyCellPad
```

This command adds padding on the right side of library cells during placement. (Padding location is dependent on the orientation of the cell. For example, if the library cell is flipped when it is instantiated, the padding is on the left side.) The padding is specified in terms of a factor that is applied to the *metal2* pitch. For example, if you specify a factor of 2, the software ensures that there is additional clearance of two times the *metal2* pitch on the right side of the specified cells.



To add padding to a cell if any signal pin is near enough to the border to cause DRC violations with any metal geometry, run the following command before placing standard cells:

```
setPlaceMode -padForPinNearBorder true
```

## **Encounter User Guide**

### Placing the Design

---

To delete cell padding, use the following command:

`deleteAllCellPad`

For more information, see the following commands in the “Placement Commands” chapter of the *Encounter User Guide*:

- [specifyCellPad](#)
- [deleteAllCellPad](#)

## Placing Standard Cells

Place standard cells with the `placeDesign` command. By default, the command runs preplacement optimization and standard cell placement. If you specified SDC timing constraints, it runs in timing-driven mode by default. If you specified scan information, it performs scan tracing and reordering by default.



*Important*  
If `placeDesign` does not place the standard cells, for example if all instances are fixed or if there is no placeable area, then `placeDesign` also skips I/O pin assignment.

This command was designed as a super command; that is, with the following exceptions, you can use it to place standard cells without specifying any placement options for your initial placement.

- If the design has more than 1,000 floorplan constraints or other types of complex floorplan constraints, run the following command before placing standard cells:  
`setPlaceMode -modulePlan false`
- If the design has clock-gating cells, run the following commands before placing standard cells:  
`specifyClockTree -clkfile fileName`  
`setPlaceMode -clkGateAware true`
- To reduce switching power on power-critical nets, consider running the following command before placing standard cells:  
`setPlaceMode -powerDriven true`

Tune the initial placement by trying the following techniques:

- If the design is congested, try the following command, then rerun placement:  
`setPlaceMode -congEffort high`
- If the design has local congestion, try the following command, then rerun placement:  
`setPlaceMode -modulePadding module factor`
- If the utilization increased by more than five percent after pre-CTS optimization, compared to what it was after `placeDesign`, try the following command:  
`placeDesign -inPlaceOpt`

The run time for this command is longer than the run time without the `-inPlaceOpt` parameter.

**Related Topics**

- [placeDesign](#) and [setPlaceMode](#) in the “Place Commands” chapter of the *Encounter Text Command Reference*
- [specifyClockTree](#) in the “Clock Tree Synthesis Commands” chapter of the *Encounter Text Command Reference*
- [Place the Design and Run Pre-CTS Optimization](#) in the *Encounter Flat Flow Guide*

## Running Placement in Multi-CPU Mode

The `placeDesign` command supports multi-threading. Multi-threading accelerates placement by splitting a job into two or more tasks that run concurrently on a single machine that has multiple processors. The placement acceleration is not linear, however, because some set-up and synchronization time is required.

Multi-threading requires additional licenses. The number of additional licenses required is dependent on the “base” Encounter license (the base license is the license used to invoke the software) and the number of threads you want to use.

Multi-threading placement has the following limitations:

- It is supported by global placement only, not by placement legalization.
- It is not supported when the `-modulePlan` parameter is set to `false`.
- It is not supported in the blackblob flow.
- The maximum number of threads you can use is eight. If you request more, the software issues a warning and sets the number of threads to eight.

***Important***

To get the greatest benefit from multi-threading, your placement job should be the only job running on your machine—you should avoid all other tasks, even a regular system backup. For example, a machine with four CPUs that is running backup or other system tasks that occupy one CPU might show less speed-up with four threads than a machine running no system tasks that is running global placement with three threads.

## Multi-Threading Placement Steps

To run multi-threading placement, complete the following steps. You can complete these steps before running any commands that run multiple-CPU processing, or before running placement. Because the Encounter software has a common interface for multiple-CPU processing (multi-threading or distributed processing), you need specify these commands only once per session, and any application that can run in multiple-CPU processing mode can use the additional licenses and processors.

1. (optional) Use the following command to specify the number of multiple-CPU licenses to check out and the license check-out order:

```
getMultiCpuLicense [-numThreads integer] [-licOptions licenses]
```

If you do not use this command, the software runs it automatically, using a default check-out order and requesting the appropriate number of licenses based on the parameters you set for `setMultiCpuUsage`.

2. Use the following command to specify the maximum number of threads to use:

```
setMultiCpuUsage -numThreads integer
```

If you request more threads than are available, the software uses the maximum number that are available.

**Note:** It is generally not a good idea to request more threads than the number of CPUs in your machine, as it will slow down the machine and waste licenses.

3. (optional) Use the following command to release the additional licenses after global placement:

```
setReleaseMultiCpuLicense true
```



Alternatively, run the following command *after* placement to release the additional licenses immediately:

```
releaseMultiCpuLicense
```

4. Run global placement.

The log file reports the number of threads used for multi-threading placement just before it shows the global placement iterations, for example:

```
Placement running 2 threads.
```

5. (optional) Check the run-time information at the end of the `placeDesign` section of the log file.

```
(cpu for global=1:25:08) real=0:50:32***  
Placement multithread real runtime: 0:50:32 with 2 threads.
```

## Encounter User Guide

### Placing the Design

---

```
Core Placement runtime cpu: 1:14:24 real: 0:41:42
Starting refinePlace ...
```

The number to look for in the log is Placement multithread real runtime. In the preceding example, the Placement multithread real runtime is 0:50:32.

### Calculating Multi-Thread Speed-Up

The amount of time used for global placement is calculated by using the following formula:

```
Global Placement = Core Placement + Timing Analysis + Congestion Analysis
```

In some designs, when timing and congestion analysis consume a high percentage of run time, the speed-up factor from multi-threading is not significant.

In the preceding example, if only one thread were used, the log would have reported the following:

```
(cpu for global=1:13:53) real=1:15:09***
Core Placement runtime cpu: 1:04:53 real: 1:05:59
Starting refinePlace ...
```

Comparing the times from the two log segments gives the following calculations:

- real time (1 thread) = 1:15:09 = 4509 seconds
- real time (2 threads) = 0:50:32 = 3032 seconds

$4509 / 3032 = 1.49$



If other jobs are running during multi-threading placement, the real time includes the run time for those jobs, so you do not get an accurate speed-up comparison.

### Related Topics

- [Accelerating the Design Process By Using Multiple-CPU Processing](#)
- [“Multiple Processing Commands” chapter in the \*Encounter Text Command Reference\*.](#)
- [“Multiple CPU Processing” in the \*Encounter Menu Reference\*](#)

## Checking Placement

Use the following methods to check placement:

- Amoeba view

For more information, see “[The Main Window](#)” chapter in the *Encounter Menu Reference*.

- `checkPlace` command

For more information, see [`checkPlace`](#) in the “Placement Commands” chapter of the *Encounter Text Command Reference*.

- Placement density map

For information, see the following references:

- “Placement Commands” chapter of the *Encounter Text Command Reference*

- [`getDensityMapMode`](#)
    - [`reportDensityMap`](#)
    - [`setDensityMapMode`](#)

- “Placement Menu” chapter of the *Encounter Menu Reference*

- [Display Density Map](#)

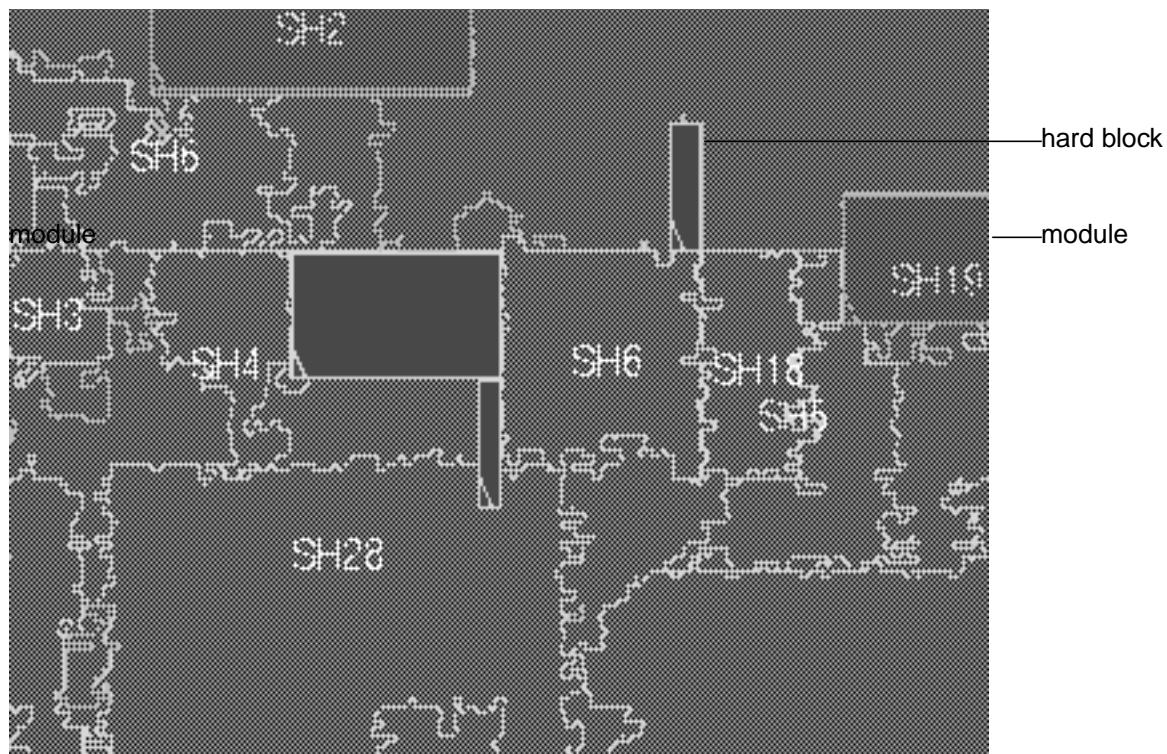
- Violation Browser

**Note:** The Violation Browser does not indicate the layer on which a placement violation occurs.

For information, see [Violation Browser](#) in the “Tools Menu” chapter of the *Encounter Menu Reference*.

## Using the Amoeba View

Use the Amoeba view to see the placement of modules and blocks. For example, in the following figure you can see the outlines of the hard blocks and the modules, and that the instances in each of the modules are placed closely together.



To display the Amoeba view, select the *Amoeba view* widget from the *Views* panel in the main Encounter window.

For more information, see “[The Main Window](#)” chapter in the *Encounter Menu Reference*.

## Using the Density Map

Use one of the following methods to turn the display of the density map on or off:

- Select *Density Map* on the list of Visibility toggles in the main Encounter window.
- Clock the All Colors button to open the Color Preferences form, then select the *View Only* tab, and select *Density Map*, in the *Multi-Color Layers* section.

## Adding Filler Cells

The software uses filler cells to fill the gaps between standard cell instances. Filler cells also provide decoupling capacitance to complete the power connections in the standard cell rows and extend N-well and P-well regions. The reason to add them as the last placement step is that you cannot run in-place optimization after they are added. After routing, the software checks for DRC violations created by the added filler cells.

To add filler cells, use the [Add Filler](#) form or the [addFiller](#) command.



### *Tip*

Provide a list of filler cells so that at least one filler cell can be used to fill the space without causing DRC violations.

Add Filler recognizes whether a filler cell has implant layer geometries and attempts to add fillers that honor the implant layers' width and spacing rules. By judicious selection of filler cells, the software can correct implant layers' minimum spacing errors by putting in same voltage threshold implant layer fillers in spaces between two same implant layer cells. Add Filler also avoids creating implant layer minimum width errors by abutting fillers of same implant layer as the adjacent cells, thus extending the implant layer width.



### *Important*

Add Filler expects to be provided with cells of all types of implant layers to be able to completely fill the design's core area with fillers. For example, if only a low-voltage implant layer filler is provided, and the abutting logical cell has a high-voltage implant layer, then Add Filler places the provided low-voltage implant filler only if its width satisfies the minimum width rule for that implant layer.

## Adding Fillers to MSV Designs

In cases where there are different voltages in the same design, also known as a multi-voltage (MSV) design, you might need to specify the power domain in which the fillers are to be inserted using the `-powerDomain` parameter of the [addFiller](#) command. If this parameter is not specified for MSV designs, the command only inserts default power domain fillers. Any region that does not belong to a specific power domain is assigned to the default power domain.

## Deleting Filler Cells

To remove added filler cells, use the [Delete Instances](#) form or the `deleteFiller` command. If you specify an area, the `deleteFiller` command deletes only filler cells that are completely contained within the area; it does not delete filler cells that cross the area boundary.

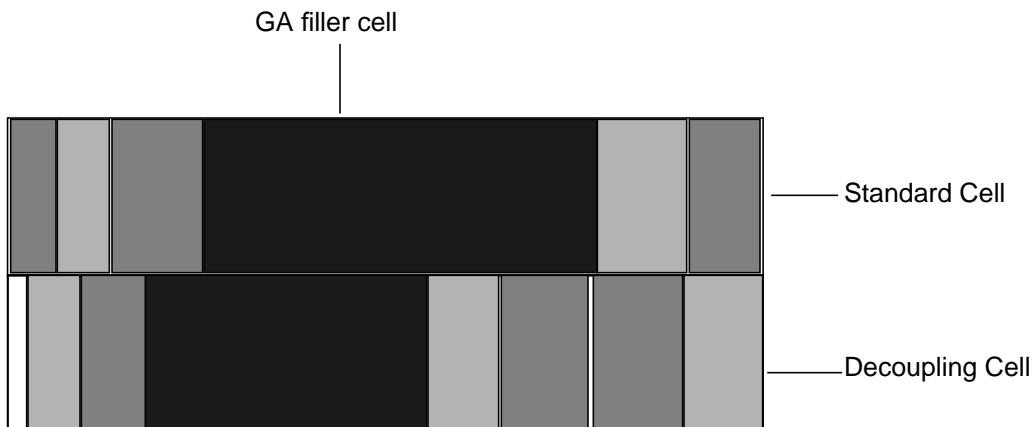
## Placing Gate Array Style Filler Cells for Post-Mask ECO

You can use pre-existing Gate Array (GA) style filler cells in regular CORE sites during a post-mask ECO flow. As such, the placer can add GA fillers at any grid location, rather than in a GA CORE grid.

- Specify the following command:

```
ecoPlace -useGAFillerCells GAFillerCells
```

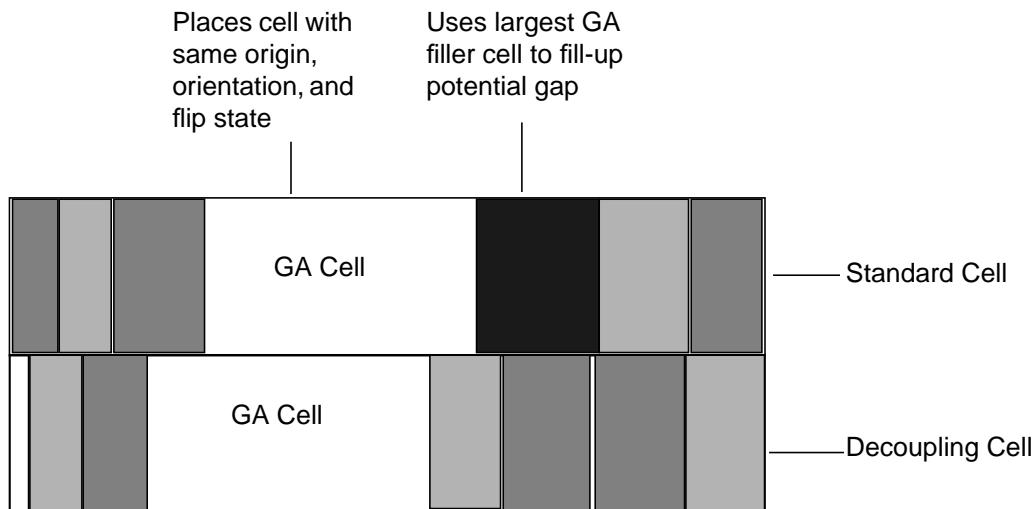
The placer first finds the optimal location each instance based on its connectivity, then searches for the nearest GA filler cell that is equal to or larger in size. A GA cell is placed at the GA filler location, and the original GA filler is deleted.



## Encounter User Guide

### Placing the Design

If the GA filler is larger than the GA cell, the placer creates a new GA filler instance using the list of GA filler cells you provide and places the filler in the gap.



The `ecoPlace` command also contains options that let you map unplaced standard cells to spare cells, and map GA cells to GA core sites. You can specify that instances that are `PLACED` cannot be moved.

For more information, see the following command in the *Encounter Text Command Reference*:

- `ecoPlace` in the “Interactive ECO Commands” chapter

## Adding Decoupling Capacitance

Adding decoupling capacitance to a design can help maintain a stable voltage between power and ground when signal nets switch. This can reduce IR drop for power nets and limit bouncing on ground nets.

The Encounter software adds decoupling capacitance by choosing from the specified available decoupling capacitance cell candidates, and adding enough cells until their combined total capacitance value equals the user-specified value. You can insert decoupling capacitance homogeneously inside a specified area, or based on the peak current density of the instances in the area.

1. To define the cells to use for decoupling capacitance insertion, use the `addDeCapCellCandidates` command.

For example, the following commands define two decoupling capacitance cell candidates: DECAP1 has a capacitance value of 10fF, and DECAP8 has a capacitance value of 5fF.

```
addDeCapCellCandidates DECAP1 10  
addDeCapCellCandidates DECAP8 5:
```

2. To add the specified total decoupling capacitance to the design, use the [addDeCap](#) command.

For example, the following command adds 1000 fF of capacitance to the design using DECAP1 and DECAP8 cells:

```
addDeCap -totCap 1000 -cells DECAP1 DECAP8
```

## Deleting Decoupling Capacitance

- To clear all available decoupling cell candidates, use the [clearDeCapCellCandidates](#) command.
- To delete all of the decoupling capacitance cells in a design, use the [deleteDeCap](#) command.

## Adding Logical Tie-Off Cells

Tie-off cell instances provide connectivity between the tie-hi and tie-lo logical input pins of the netlist instances to power and ground. This connectivity does not cross the hierarchy module boundaries. The number of tie-off instances added can be controlled by setting the distance and fanout constraints using the [setTieHiLoMode](#) command.

To add logical tie-off cells to the design after placing the netlist, use the [Add TieHiLo](#) form or the [addTieHiLo](#) command. To remove added logical tie-off cell instances, you can use the [Delete TieHiLo](#) form or the [deleteTieHiLo](#) command.

## Saving Placement Data

You can save placement data in the Encounter place format or in DEF, PDEF, and TDF placement data formats. This can be done at any time after running placement. To save placement data, use the `savePlace` command or the `saveDesign` command.

## Specifying and Placing JTAG and Other Cells Close to the I/Os

You can constrain the placement of JTAG cells and other cells so they are placed close to the outer core area. Place these cells before you run placement in the rest of the design.

When the software runs JTAG placement, it creates a temporary blockage over the area where the cells must not be placed and removes it after the placement.

You can constrain the placement of instances, hierarchical instances, or cells.

1. Use the following command to constrain placement:

```
specifyJtag
```

To include instances or cells other than JTAG cells, you must identify them with this command.

- a. To undo the specification, use the following command:

```
unspecifyJtag
```

2. To place the instances or cells, use the following command:

```
placeJtag
```

3. (optional) To generate a report of the JTAG placement, use the following command:

```
reportJtagInst
```

To undo JTAG placement, use the following command:

```
unplaceJTAG
```



If you do not want to place regular instances in the JTAG outer core area after running JTAG placement, specify a placement blockage prior to running placement.

## Related Topics

To see this step in the design flow, see [Place the Design and Run Pre-CTS Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

For more information, see the following commands in the “Placement Commands” chapter of the *Encounter User Guide*:

- [specifyJtag](#)
- [unspecifyJtag](#)
- [placeJtag](#)
- [reportJtagInst](#)
- [unplaceJtag](#)
- [traceJtag](#)

## Optimizing and Reordering Scan Chains

The `placeDesign` command reorders scan chains by default, unless it is in prototyping mode. If you decide not to reorder scan cells with `placeDesign`, use the information provided in this section to reorder scan chains.

## Related Topics

To see this step in the design flow, see [Place the Design and Run Pre-CTS Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

## Specifying Scan Cells

Scan cells are usually identified and read automatically from the timing library during design import. Use the [specifyScanCell](#) command to define scan cells that the software cannot retrieve from the library.

You can specify scan chains in a design by defining them in a DEF or TDF file, or by using the [specifyScanChain](#) command.

If scan chains are specified by reading in a DEF file, the software does a native scan trace. The scan DEF file is stored in the database and, when the [scanReorder](#) command runs, the software matches the scans and honors the ordered segments. However, if you run

specify `setPlaceMode -reorderScan false`, the software does not perform scan chain reordering, so the DEF file will not include the `+ ORDERED` statement in the SCANCHAINS section.

## About Scan Chains

If you do not need to retain the scan chain order in your design, you can change the order of the scan flip-flop connections along any or all scan chains. Changing the connection order eases connection constraints on the scan cells, but does not constrain their placement.

To facilitate reordering of the scan nets, uniquify the incoming netlist and make sure that it does not contain Verilog assignment statements involving scan nets. A scan net is a net that resides along the scan datapath—that is, a net that connects the scan flip-flops in a scan chain.

If the netlist is unqualified, but contains Verilog assignment statements involving scan nets, use the following command to insert a temporary buffer into the netlist to enable reordering of these nets:

```
setDoAssign on -buffer bufferName
```

Then load the design in the Encounter software with the `loadConfig` command.

When the Encounter software reads the netlist, it outputs the following messages:

```
Reading netlist ...
Reading verilog netlist ".fileName"
Inserting temporary buffers to remove assignment statements.
```

If buffers were added by `setDoAssign`, these buffers remain in the final netlist and replace the Verilog assignment statements.

## Reordering Scan Chains

Use one of the following approaches to scan chain reordering:

- Native scan reordering

Use this approach in the following conditions:

- Single-clock domain, single-edge chains
- Multiple clock domain chain segments separated by data lockup elements
- Shared functional output signal chains

- ScanDEF-based reordering

## Encounter User Guide

### Placing the Design

---

Use this approach in the following conditions:

- All simple scan chain architectures (handled by the native approach)
- Implied domain transition scan chains (without data lockup elements)
- Scan chains with ordered segments
- Scan chains generated by LogicVision software

After reordering scan chains, save a netlist of the design using one of the following methods:

- Save Netlist form (*Design – Save – Netlist*)
- saveNetlist command

### Native Scan Reordering Approach

Use the native approach to scan chain reordering when you do not have a scanDEF file.

This approach requires that you use the `specifyScanChain` command to identify the START and STOP signals of the top-level chains, or chain segments, in the netlist. Using this information, the software identifies the scan flip-flops along the scan chain when running the `scanTrace` command to analyze the scan flip-flop connections. You can also auto-detect data lockup latch elements using the `scanTrace -lockup` command.

If the scan cells are not listed in the timing library, you must specify them before tracing the scan chains. You can identify scan cells with the `specifyScanCell` command.

After `scanTrace` has identified the elements along the chain, complete the following steps:

1. (Optional) Ignore the scan connections:

```
setPlaceMode -ignoreScan true
```

2. (Optional) Set scan reorder options:

```
setScanReorderMode -skipMode [skipNone | skipBuffer | skipTwoPinCell]
```

3. Run placement:

```
placeDesign
```

The recommended flow for scan chains that have data lockup latches is as follows:

1. Specify a scan chain in the design:

```
specifyScanChain
```

2. Trace the scan chain connection with the automatic detection lockup latch elements:

```
scanTrace -lockup [-verbose]
```

**3. (Optional) Ignore the scan connections:**

```
setPlaceMode -ignoreScan true
```

**4. (Optional) Set scan reorder options:**

```
setScanReorderMode [-skipNone | -skipBuffer | -skipTwoPinCell]
```

**5. Run placement:**

```
placeDesign
```

The recommended flow for scan chains that have data lockup flip-flops is as follows

**1. Specify a scan chain in the design:**

```
specifyScanChain ...
```

**2. Specify a cell or instance as a lockup flip-flop element:**

```
specifyLockupElement ...
```

**3. (Optional) Ignore the scan connections:**

```
setPlaceMode -ignoreScan true
```

**4. (Optional) Set scan reorder options:**

```
setScanReorderMode -skipMode [skipNone | skipBuffer | skipTwoPinCell]
```

**5. Run placement:**

```
placeDesign
```

**Note:** The `scanReorder` command automatically calls `scanTrace` internally if you have not previously run `scanTrace`. By default, this internal `scanTrace` run specifies that the tracing will not detect lockup elements (`-noLockup`); therefore, if you have lockup latches, Cadence recommends using the `scanTrace -lockup` command before `scanReorder`, or `specifyLockupElement` prior to running `scanReorder`.

### **Valid Design Types**

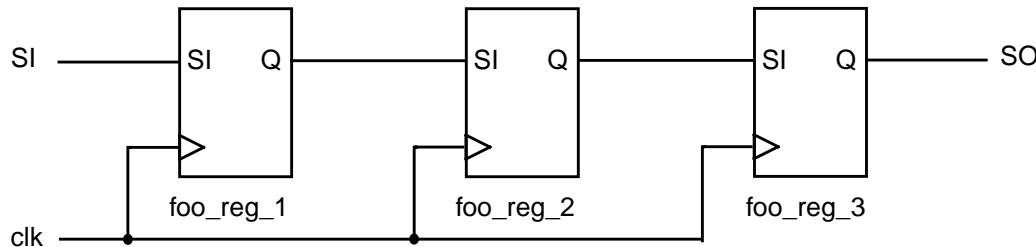
You can use the native approach to scan chain reordering on designs comprising a simple scan chain architecture with the following characteristics:

- Single-clock domain, single-edge chains

## Encounter User Guide

### Placing the Design

In the following figure, all `foo_reg` scan flip-flops are triggered by the same clock domain and phase.

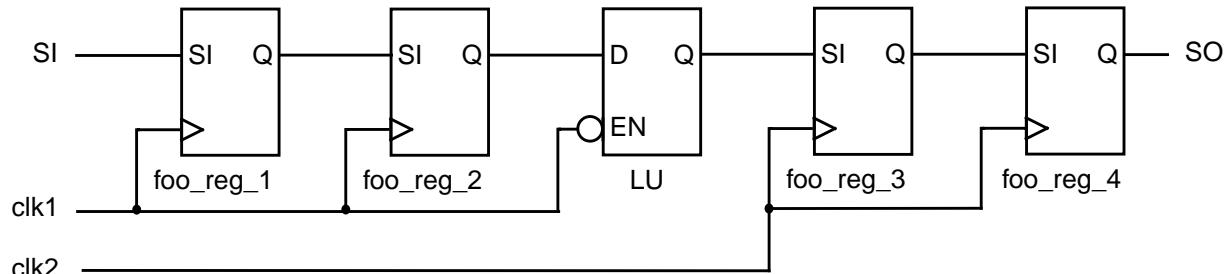


- ❑ `foo_reg_1`, `foo_reg_2`, and `foo_reg_3` scan flip-flops are triggered by `clk1` (positive edge).

```
specifyScanChain chain1 -start SI -stop SO  
scanTrace [-verbose]
```

- Multiple clock domain chain segments separated by data lockup elements

In the following figure, all domain or edge transitions are separated by a data lockup element.



- ❑ `foo_reg_1` and `foo_reg_2` scan flip-flops are triggered by `clk1` (positive edge).
- ❑ `foo_reg_3` and `foo_reg_4` scan flip-flops are triggered by `clk2` (positive edge).
- ❑ LU represents a data lockup element of type latch.

```
specifyScanChain chain1 -start SI -stop SO  
scanTrace -lockup [-verbose]
```

All elements along the scan chain are assumed reorderable from the specified START and STOP signals unless there is a data lockup element in the scan data path. The presence of a data lockup element works as a boundary so that the chain segments on either side of the lockup element are individually reordered. For this example, the top-level chain is reordered as two individual scan chain segments:

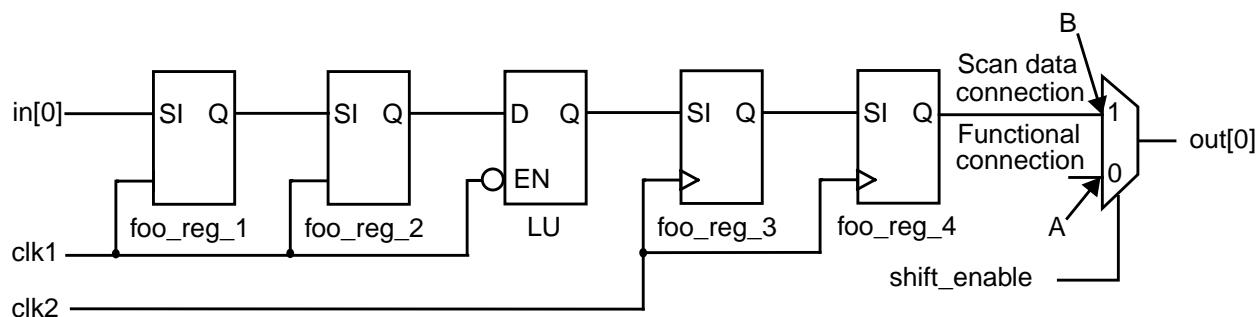
- ❑ reorderable segment 1: `SI` > LU/D
- ❑ reorderable segment 2: LU/Q > SO

## Encounter User Guide

### Placing the Design

#### ■ Shared functional output signal chains

If the STOP signal of the scan chain is also a shared functional output, the endpoint of the scan chain must be specified to the scan input (SI) pin of the last register in the scan chain, or to the data input pin of the multiplexer (MUX), which drives the shared functional output signal. This is necessary because `scanTrace` does not perform the forward trace from the last flip-flop in the scan chain through the MUX instance. The following figure is an example of shared functional output:



The following command sequence performs the forward trace from the last flip-flop in the scan chain to the MUX instance:

```
specifyScanChain chain1 -start in[0] -stop MUX/B  
scanTrace -lockup [-verbose]
```

The following command sequence does not perform the forward trace from the last flip-flop through the MUX instance; `scanTrace` will not succeed:

```
specifyScanChain chain1 -start in[0] -stop out[0]  
scanTrace -lockup [-verbose]
```

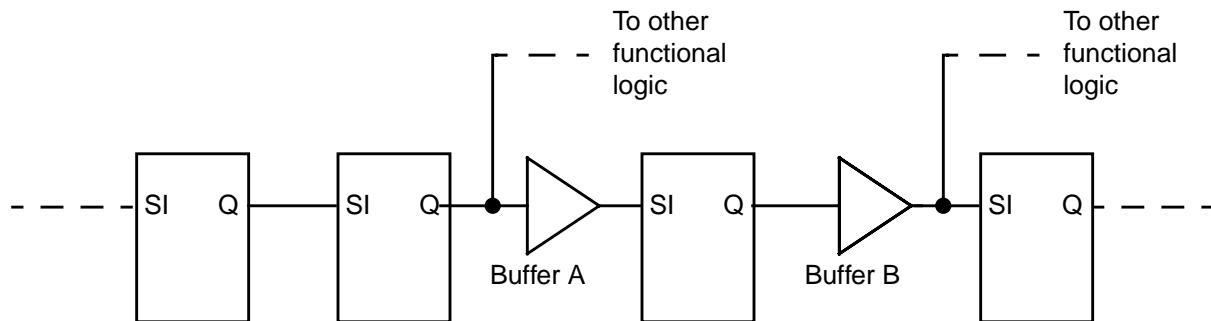
#### **Scan Chains with Two-Pin Logic Cells**

Scan chains often contain two-pin logic cells, usually buffers. The scan tracing algorithm always recognizes and traces through two-pin cells. The `scanReorder` command parameters control whether two-pin cells remain in the scan chain after scan reordering.

## Encounter User Guide

### Placing the Design

In the following scan chain example, buffer A is in the scan chain, but not part of the functional logic of the design, and therefore can be deleted. Buffer B is part of the functional logic, and must not be deleted.



The `scanReorder -skipMode skipNone` command retains all two-pin cells in the scan chain, and reordering changes only the connections to the two-pin cells' outputs. The nets connected to the two-pin cells' inputs will not be modified. In the preceding example, both buffers A and B would be retained, and scan reordering would be performed by rearranging the scan input pin connected to their output nets.

The `scanReorder -skipMode skipBuffer` command reconnects the scan chain so that buffers (as defined by `setBufFootPrint`) are skipped. Buffers that are not part of the functional logic are deleted. This disconnects scan inputs and reconnects them directly to a scan output pin, skipping all buffers. Any two-pin cells that are not buffers are retained in the scan chain in the same manner as `skipNone`.

In the preceding example, buffer A would be deleted and functional buffer B would be retained in the netlist. Scan reordering would disconnect the scan input pin from the output of buffer B, and reconnect some other scan input pin to the input net of buffer B, so buffer B would no longer be in the scan chain.

The `scanReorder -skipMode skipTwoPinCell` command works the same as `scanReorder -skipMode skipBuffer`, except that it is not limited to buffers. Any two-pin cell will be treated as `skipBuffer` would treat a buffer.

#### *Important*

Because `scanReorder -skipMode skipTwoPinCell` does not consider the functionality of cells that it removes from the scan chain, it can change the scan chain in unpredictable ways. For example, if buffer A in the preceding example was an inverter, it would be removed, and the test pattern would have to be changed to account for the loss of inversion.

## scanDEF-Based Reordering Approach

If you have a scanDEF file that describes the set of reorderable scan chains in the design, Cadence recommends using the scanDEF approach. To reorder scan chains with the scanDEF approach, complete the following steps:

1. Read in the scanDEF file:

```
defIn -scanChain
```

**Note:** In the case where a DEF file contains a SCANCHAIN section, the defIn command automatically reads in the scanDEF file, so the -scanChain parameter is not necessary.

2. (Optional) Ignore the scan connections:

```
setPlaceMode -ignoreScan true
```

3. Run placement:

```
placeDesign
```

## Using the scanReorder Command

When running the scanReorder command, the Encounter software uses the begin and endpoints from the scanDEF chains to trace the connectivity of the scan chains in the netlist. This check verifies whether the elements in the netlist scan chains are represented as elements in their respective scanDEF chains. As a result of this check, an internal representation of each scanDEF chain is created in the Encounter database.

When a netlist-to-scanDEF file mismatch occurs, for each instance mismatched, scanReorder issues the following WARNING message:

```
WARNING (SOCSC-5003): The scan chain was found to pass through instance <inst> in the netlist, but this instance does not appear in the DEF scan chain.
```

Mismatches of combinational components (buffers or inverters) in the scan data path can be expected if the netlist has undergone pre-placement optimization, or if the scanDEF file is not properly formatted, as described in Netlist-to-scanDEF mismatch section. Sequential mismatches are tolerated if the mismatch occurs for a scan flop from the FLOATING section only of the scanDEF chain. However, sequential mismatches are not expected and indicate a discrepancy between the scan chains in the netlist, and the scanDEF chains. You should investigate the source of the discrepancy before proceeding with reordering. If necessary, revise the scanDEF description of the scan chains.

Using the internal representation of the scanDEF chains, Encounter issues the following message prior to reordering the chains in the netlist:

```
INFO: Scan reorder based on traced netlist chains.
```

## Encounter User Guide

### Placing the Design

---

```
INFO: Medium effort Scan reorder
INFO: Reordering scan chain <chainName>
```

#### ***Netlist-to-scanDEF Mismatch***

Netlist-to-scanDEF mismatches can occur if a driving scan flip-flop is buffered (or inverted) to the SI pin of the next scan flip-flop in the scan chain. In this situation, the driving scan flop and buffer (or inverter) should be captured to the scanDEF file as an ORDERED segment, rather than capturing the driving scan flip-flop as a freely reorderable element in the FLOATING section of the scanDEF chain. The correct syntax for the FLOATING and ORDERED sections of the scanDEF file is as follows:

```
- chain X
  + START PIN
  + FLOATING
    ....
    next_scan_flop_reg ( IN SI ) ( OUT SO )
  + ORDERED
    driving_scan_flop_reg ( IN SI ) ( OUT SO )
    buf_instance ( IN A ) ( OUT Y )
  + STOP
```

In previous releases of Encounter, when a scanDEF to netlist mismatch occurred, scan reorder would abort. If the mismatches were due to combinational components (buffers or inverters) in the scan data path, you could still proceed with scan reordering by issuing `scanReorder` with the following parameters:

```
scanReorder -defInForce
```

For backward compatibility, these options are maintained in this release of the tool. However, in order to leverage the new netlist-to-scanDEF tracing feature, you should remove these parameters from the `scanReorder` command

The `-defInForce` parameter forces reordering to use the scanDEF file.

#### ***scanDEF File Format***

The scanDEF file follows a pin-based format that describes the set of scan chains or chain segments which are reorderable in the design. The syntax is as follows:

```
SCANCHAINS numScanChains ;
  [- chainName
    [+ COMMONSCANPINS [( IN pin )][( OUT pin )] ]
    [+ START {fixedInComp | PIN} [outPin] ]
    [+ FLOATING {floatingComp [( IN pin )] [( OUT pin )]}...]
    [+ ORDERED
      {fixedComp [( IN pin )] [( OUT pin )]}
      {fixedComp [( IN pin )] [( OUT pin )]}
      {fixedComp [( IN pin )] [( OUT pin )]}...]
    [+ STOP {fixedOutComp | PIN} [inPin] ] ;...]
  END SCANCHAINS
```

## Encounter User Guide

### Placing the Design

The logic synthesis tool writes the input `scanDEF` file after the top-level scan chains are created in the design. Each top-level scan chain can be segmented into multiple `scanDEF` chains because the elements along each `scanDEF` chain must belong to the same clock domain, and be triggered by the same active edge of clock. Scan flip-flops that are freely reorderable along the scan chain are captured to the `FLOATING` section. Fixed segments (a set of connected elements), which are reordered as a fixed entity along the scan chain, are captured to the `ORDERED` section. Each scan chain must also have a `START` and `STOP` signal that defines the reordering start and end points of the scan chain.

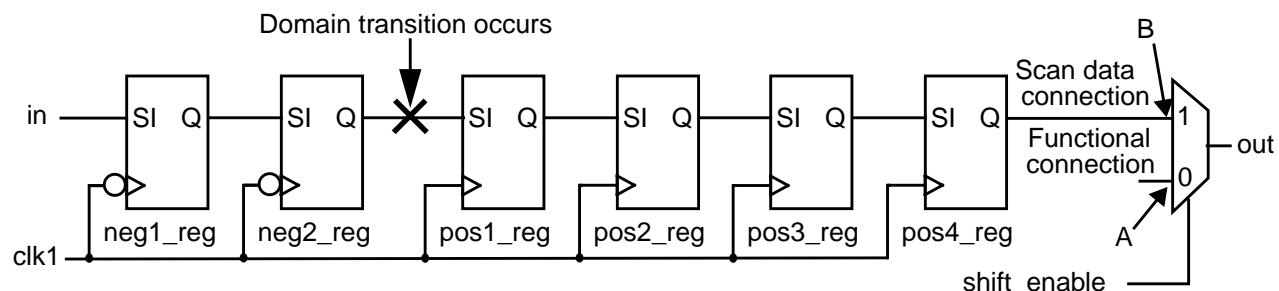
**Note:** You can use the following RTL Compiler command: `write_scandef > fileName`

### Valid Design Types

You can use the `scanDEF` approach to reorder top-level scan chains. This section provides a reordering example for implied domain transition scan chains, and an example of scan chains with fixed-ordered segments. You can also use this approach with all simple scan chain architectures that can use the native approach, as well as scan chains generated by LogicVision software.

#### ■ Implied domain transition scan chains

The scan flip-flops are triggered by alternate active edges of the same clock domain. The negative (positive) edge triggered segment precedes the positive (negative) edge triggered segments, respectively. In the following example, the implied domain transition occurs at `neg2_reg` to `pos1_reg`:



In this example, the two scan chain segments are as follows:

- ❑ `clk1` (negative edge) consisting of elements `neg1_reg` and `neg2_reg`
- ❑ `clk1` (positive edge) consisting of elements `pos1_reg`, `pos2_reg`, `pos3_reg`, and `pos4_reg`

Because the domain transition is done implicitly (without a data lockup element), the scan chain must be segmented to be properly reordered. In the `scanDEF` format, the top-level chain becomes two `scanDEF` chains, segmented by clock domain and clock edge;

the pos1\_reg scan flip-flop is sacrificed to anchor the domain transition. This register becomes an internal end and internal being point of scan DEF chains (chain1 and chain2 respectively):

```
SCANCHAINS 2 ;
- chain1
+ START pin in
+ FLOATING
    neg1_reg ( IN SI ) ( OUT Q )
    neg2_reg ( IN SI ) ( OUT Q )
+ STOP pos1_reg SI
;
- chain2
+ START pos1_reg Q
+ FLOATING
    pos2_reg ( IN SI ) ( OUT Q )
    pos3_reg ( IN SI ) ( OUT Q )
+ STOP pos4_reg SI
;
END SCANCHAINS
```

**Note:** The shared functional output signal (`out`) is not the `STOP` signal of the second scan chain segment. Instead, the scan chain is terminated to the `IN` pin of the last scan flop in the positive-edge triggered segment (BuildGates/PKS), or terminated to the data input pin of the MUX (other third-party tools).

## ■ Scan chains with ORDERED segments

An order segment is a set of connected elements that can be reconnected along the scan chain based on its placement. Reconnection to the fixed segment occurs using the `IN` pin of the first element and the `OUT` pin of the last element of the ordered segment. The connections of the other elements in the ordered segment are presumed connected and remain as intact connections. When an `ORDERED` segment is reconnected in the scan chain, the location of the `ORDERED` segment appears as a comment in the `FLOATING` section and again in the `ORDERED` section in order to correlate the segment to its location in the `FLOATING` section. The notation is as follows:

```
# ORDERED segment integer;
```

The integer corresponds to as many `ORDERED` segments as defined in the original scan chain. For example, a scanDEF chain with one `ORDERED` segment is as follows:

```
SCANCHAINS 1 ;
- chain0
+ START PIN scan_in
+ FLOATING
    out_reg_0 ( IN SI ) ( OUT Q )
    out_reg_1 ( IN SI ) ( OUT Q )
    out_reg_2 ( IN SI ) ( OUT Q )
    out_reg_3 ( IN SI ) ( OUT Q )
+ ORDERED
    out_reg_4 ( IN SI ) ( OUT Q )
    u_buf ( IN A ) ( OUT Y )
+ STOP PIN scan_out ;
END SCANCHAINS
```

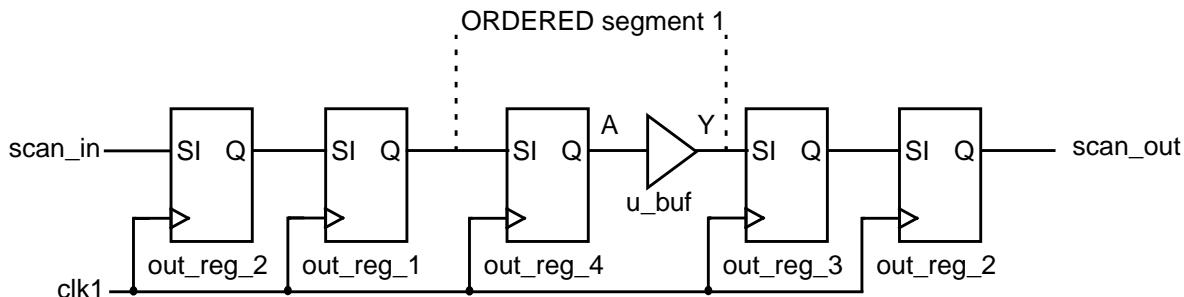
## Encounter User Guide

### Placing the Design

After reordering the output, the scanDEF file is as follows:

```
SCANCHAINS 1 ;
- chain0
  + START PIN scan_in
  + FLOATING
    out_reg_2 ( IN SI ) ( OUT Q )
    out_reg_1 ( IN SI ) ( OUT Q )
  # ORDERED segment 1
    out_reg_3 ( IN SI ) ( OUT Q )
    out_reg_0 ( IN SI ) ( OUT Q )
  + ORDERED
  # ORDERED segment 1
    out_reg_4 ( IN SI ) ( OUT Q )
    u_buf ( IN A ) ( OUT Y )
  + STOP PIN scan_out ;
END SCANCHAINS
```

Therefore, the connectivity of the elements along the reordered scan chain is as follows:



## Saving Scan Files

After scan reorder is run, save a DEF or TDF file using the following command:

```
defOutBySection -noNets -noComps -scanChains
```

With this command, you can view the new order of elements along the scan chain. However, you should use the scanDEF output file for viewing purposes only, not a subsequent reordering pass.

To save scan files, use the Save Scan File form or the defOut or tdfOut commands.

## Loading Scan Files

To load scan files in either DEF or TDF formats, use the Load Scan File form. For DEF, use the defIn command. For TDF, you can also use the tdfIn command.

---

## Synthesizing Clock Trees

---

- [Before You Begin](#) on page 578
- [Results](#) on page 578
- [Understanding CTS Operation Modes](#) on page 579
- [How CTS Calculates Skew Values](#) on page 584
- [Improving Postroute Correlation](#) on page 586
- [Specifying Macro Model Delays](#) on page 587
- [Grouping Clocks](#) on page 590
- [Analyzing Hierarchical Clock Trees](#) on page 591
- [Module Placement Utilization](#) on page 593
- [Clock Designs with Tight Area](#) on page 593
- [Balancing Pins for Macro Models](#) on page 593
- [Timing Model Requirement for Cells](#) on page 593
- [Delay Variation and OCV](#) on page 593
- [Understanding Post-CTS Clock Tree Optimization](#) on page 594
- [Creating a Clock Tree Specification File](#) on page 598
- [CTS Report Descriptions](#) on page 629
- [Supported SDC Constraints](#) on page 633

## Before You Begin

Before you run CTS on your design, make sure the following files are available:

- Clock tree specification file
- Verilog netlist
- GDSII or LEF physical library
- Proper RC model from LEF, Encounter<sup>TM</sup> technology file, or Encounter capacitance table  
For information on RC extraction in Encounter, see [RC Extraction](#) on page 865 of the *Encounter User Guide*.
- Timing constraints file (optional)
- Synopsys .lib file or TLF file with timing models for standard cells and cell footprint names
- Placement information, such as a DEF file or an Encounter placement file

### Related Topics

To see where this step fits in the design flow, see [Run CTS and Post-CTS Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

## Results

After a CTS run, CTS creates reports on the results of the run in ASCII text or HTML format. CTS also creates routing guide files (to guide NanoRoute on routing the clock nets) and macro model files (for partitions or modules).

## Understanding CTS Operation Modes

There are two modes for running CTS: manual and automatic.

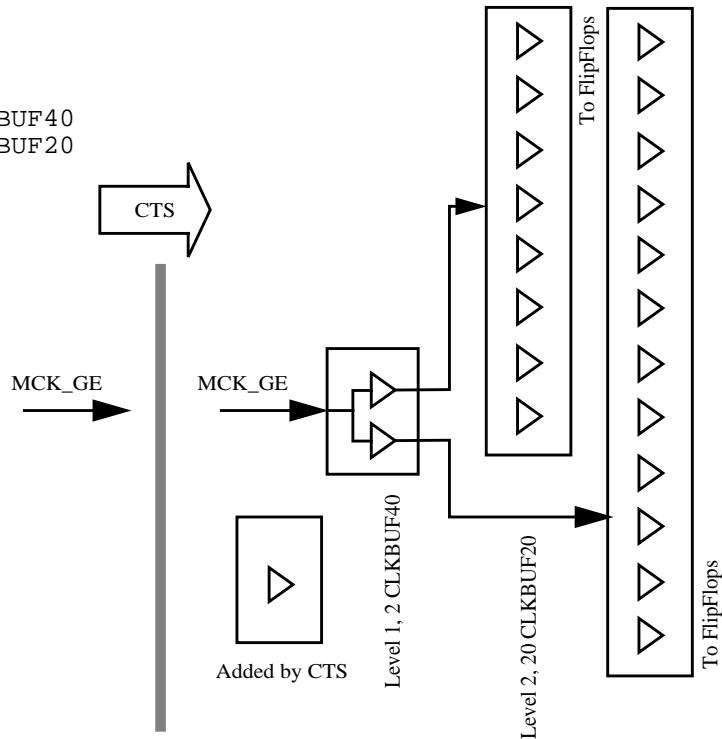
- Manual CTS mode allows you to control the number of levels and the number of buffers, and specify the types of buffers at each level.
- In automatic CTS mode, CTS automatically determines the number of levels and buffers based on the timing constraints in the clock tree specification file, such as the maximum delay and maximum skew.

### Manual CTS Mode

You can run manual CTS on a clock net and specify the levels of clock buffers. CTS builds the clock buffer tree according to the clock tree specification file, generates the clock tree topology, and balances the clock phase delay with inserted clock buffers. However, CTS does not trace the clock net.

The following is an example of clock-tree specification file syntax and a graphic representation of that syntax:

```
ClockNetName  MCK_GE
LevelNumber   2
LevelSpec    1  2  CLKBUF40
LevelSpec    2  20 CLKBUF20
PostOpt      YES
End
```



## **Automatic CTS Mode**

You can run automatic CTS to synthesize the clock design on a clock net or on a gated clock design. However, CTS does not trace the clock net.

### **Automatic CTS on Nets**

For automatic CTS on a net, CTS builds the clock buffer tree according to the clock tree specification file, generates the clock tree topology, and balances the clock phase delay with appropriately sized, inserted clock buffers.

## Encounter User Guide

### Synthesizing Clock Trees

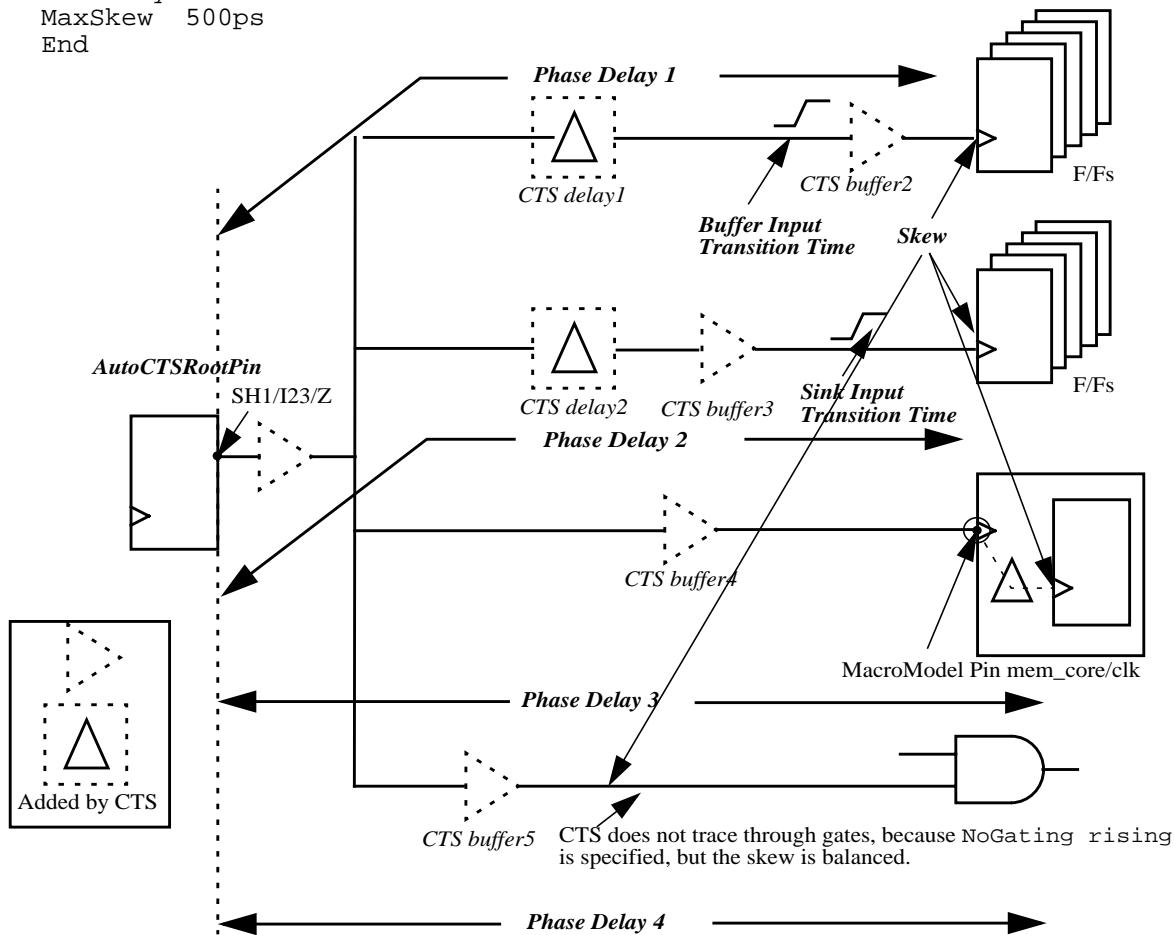
---

The following is an example of clock tree specification file syntax for automatic CTS on a net and a graphic representation of the syntax:

```

MacroModel pin mem_core/clk 20ps 18ps 20ps 18ps 0ff
AutoCTSRootPin SH1/I23/Z
NoGating rising
Buffer INV14 CLKBUF12 CLKBUF40 CLKBUF20 DEL4
MaxDelay 5ns
MinDelay 0ns
MaxSkew 500ps
End

```



### Automatic CTS for Gated Clocks

For automatic-gated CTS, CTS traces the clock tree starting from a root pin. The tracing begins at the root pin, then continues through the buffers, inverters, multi-output cells, and gated instances to establish the clock tree. The tracing stops at

- A clock pin
- An asynchronous set/reset pin

- An input pin without any timing arc to an output pin
- A user-specified leaf pin or excluded pin
- Data pin of registers (or flops)
- Asynchronous set or reset pin of registers (or flops)
- Enable pins of tristate instances

After the tracing, CTS builds the clock buffer tree topology to balance the clock phase delay with inserted clock buffers.

**Note:** Cadence recommends using the `ckSynthesis -check` command to check the gated clock tree of your design before running automatic gated CTS mode. After running the command, review the trace report file, `topCellName.cts_trace`. If tracing fails, the `-forceReconvergent` parameter of the `ckSynthesis` command could resolve tracing failures.

## Encounter User Guide

### Synthesizing Clock Trees

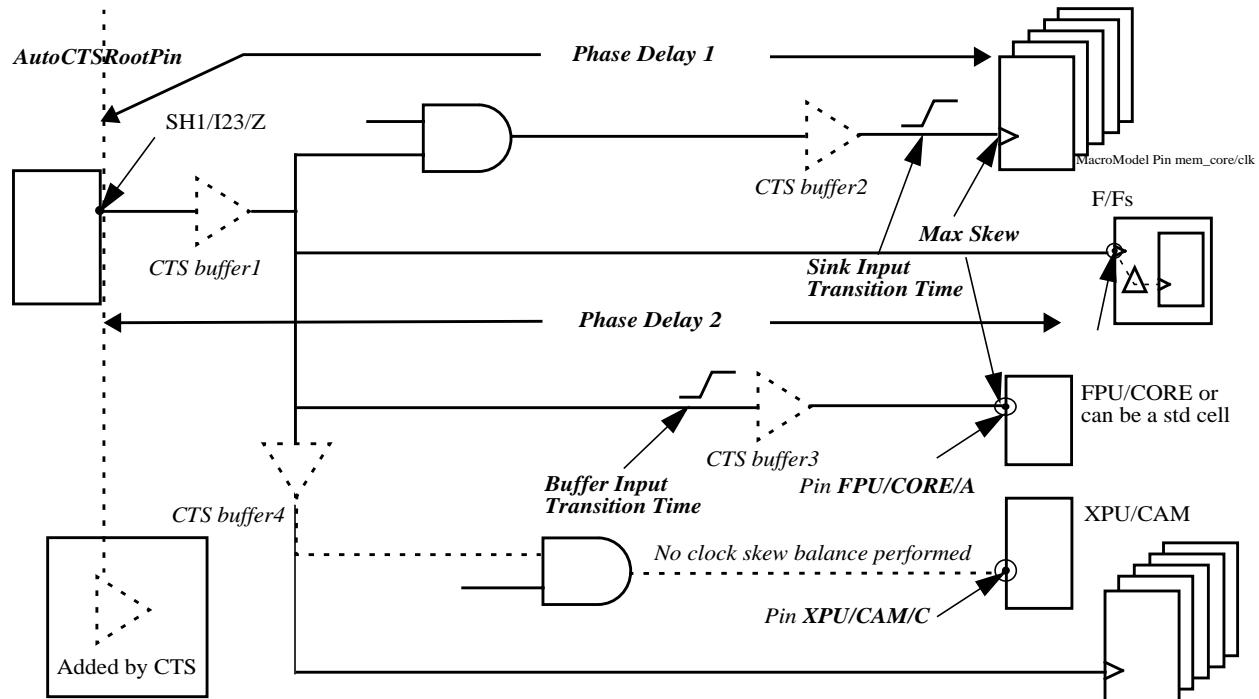
---

The following is an example of clock tree specification file syntax for automatic CTS on a gated clock and a graphic representation of the syntax:

```

AutoCTSRootPin SH1/I23/Z
MaxDelay 5ns
MinDelay 0ns
MaxSkew 500ps
MaxDepth 20
NoGating NO
RouteType CK1
LeafPin
+ FPU/CORE/A rising
ExcludedPin
+ XPU/CAM/C
PreservePin
+ pinA
+ pinB
MaxCap
+ buf1 20ff
+ buf2 50ff
Buffer buf1 buf2 inv1 inv2 dell1
End

```



## How CTS Calculates Skew Values

CTS calculates skew at the edge of the clock root in the following fashion:

- *Rise skew* and *fall skew* are calculated relative to the edge of the clock root—for example, rise skew is calculated based on the rising edge at the clock root.  
**Note:** The edge polarity at the leaf pins can be rising or falling, regardless of whether CTS is reporting on rise skew or fall skew.
- *Rise skew* is the maximum difference of all the arrival times of the clock signal at the leaf inputs, as measured from a rising edge at the clock root.
- *Fall skew* is the maximum difference of all the arrival times of the clock signal at the leaf inputs, as measured from a falling edge at the clock root.
- *Trigger-edge skew* is based on all the arrival times of the active clock signal at the leaf inputs. The calculation considers the trigger-edge polarity of the receiving leaf inputs, and represents the worst-case trigger-edge-to-trigger-edge skew in the design. See the accompanying figure.

**Note:** Trigger-edge skew can be greater or smaller than rise skew or fall skew.

The following example illustrates how CTS calculates various skew values:

Assume that a design has two flip-flops, FF1 and FF2:

FF1 rise: 3.0 ns; fall: 3.6 ns

FF2 rise: 3.4 ns; fall: 4.0 ns

Rise skew is 0.4 ns; fall skew is 0.4 ns.

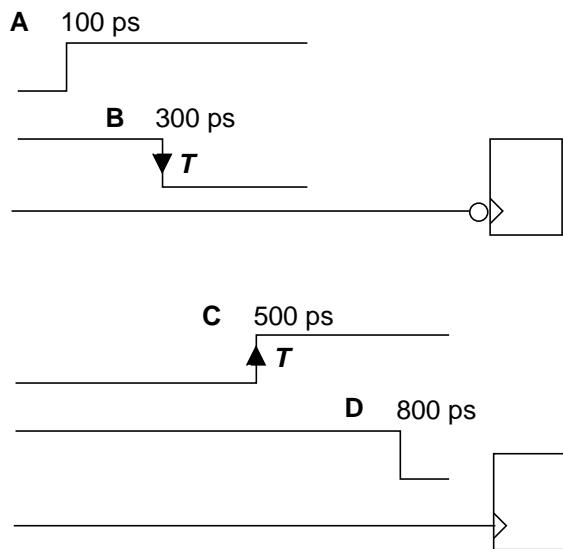
Assume that FF1 is a falling-edge-triggered flip-flop, and that FF2 is a rising-edge-triggered flip-flop. The trigger-edge skew is  $3.6\text{ ns} - 3.4\text{ ns} = 0.2\text{ ns}$ .

Assume that FF1 is a rising-edge-triggered flip-flop, and that FF2 is a falling-edge-triggered flip-flop. The trigger-edge skew is  $4.0\text{ ns} - 3.0\text{ ns} = 1.0\text{ ns}$ .

## Encounter User Guide

### Synthesizing Clock Trees

The following figure illustrates how trigger-edge skew can be smaller than either rise skew or fall skew:



Rise skew (**C - A**):  $500 \text{ ps} - 100 \text{ ps} = 400 \text{ ps}$

Fall skew (**D - B**):  $800 \text{ ps} - 300 \text{ ps} = 500 \text{ ps}$

Trigger-edge skew (**C - B**):  $500 \text{ ps} - 300 \text{ ps} = 200 \text{ ps}$

**T** = Trigger edge

## Improving Postroute Correlation

You can create a routing guide file that CTS automatically uses to direct the global detailed routing of clock nets. This process helps achieve tighter correlation between preroute (Steiner tree) and postroute topologies.

There are two methods of improving postroute correlation with a routing guide file. The difference between the two methods is this: With Method 2, CTS reports on the clock tree *before* you complete the detailed routing on the design.

### Method 1

1. In the Encounter console, type `setCTSMode -routeGuide true`.
2. In the clock tree specification file, include `RouteClkNet YES`, or use the text command `createClockTreeSpec -routeClkNet`.
3. In the Encounter console, type `ckSynthesis`.
4. Check your run directory for the clock tree timing report (`top_level_cell.ctsrpt`).

### Method 2

1. In the clock tree specification file, include `RouteClkNet NO`.
2. In the Encounter console, type `ckSynthesis`.
3. In the Encounter console, type `routeClockNetWithGuide [-clk clock_root_pin_name]`
4. Check your run directory for the clock tree timing report (`top_level_cell.ctsrpt`).

## Specifying Macro Model Delays

You use the MacroModel statement to specify pin delays. A macro model is a block with synthesized clock trees, and thus has delays that have been identified.

There are three ways to define the macro model:

- Cell or port delay specification—All instantiations of cells have the same pin delay.

```
MacroModel port cellName/portName maxRiseDelay minRiseDelay  
          maxFallDelay minFallDelay extraCap
```

where *cellName* is the cell type name and the *portName* is the port name. For example:

```
MacroModel port ram256x64/clk 10ns 80ns 110ns 7ns 0.35ff
```

- Pin instance delay specification—This specification can supersede a cell delay or port delay specification.

```
MacroModel pin leafPinName maxRiseDelay minRiseDelay  
          maxFallDelay minFallDelay extraCap
```

where the *leafPinName* is the leaf pin instance name. For example:

```
MacroModel pin mem_pin/clk 20ps 18ps 20ps 18ps 0.29ff
```

Delay units for MacroModel statements must be specified in nanoseconds (ns) or picoseconds (ps), for example, 200ps, 1ns.

- INSERTION\_DELAY statement in the TLF file.

```
INSERTION_DELAY(CLK FAST 01 01 DELAY(InsDelay0) SLEW(InsSlew1))  
INSERTION_DELAY(CLK SLOW 01 01 DELAY(InsDelay0) SLEW(InsSlew1))
```

For information on TLF, see the *Timing Library Format Reference*.

For illustrations of MacroModel behavior, see [Automatic CTS on Nets](#) on page 580 and [Automatic CTS for Gated Clocks](#) on page 581.

## Dynamic Macro Model

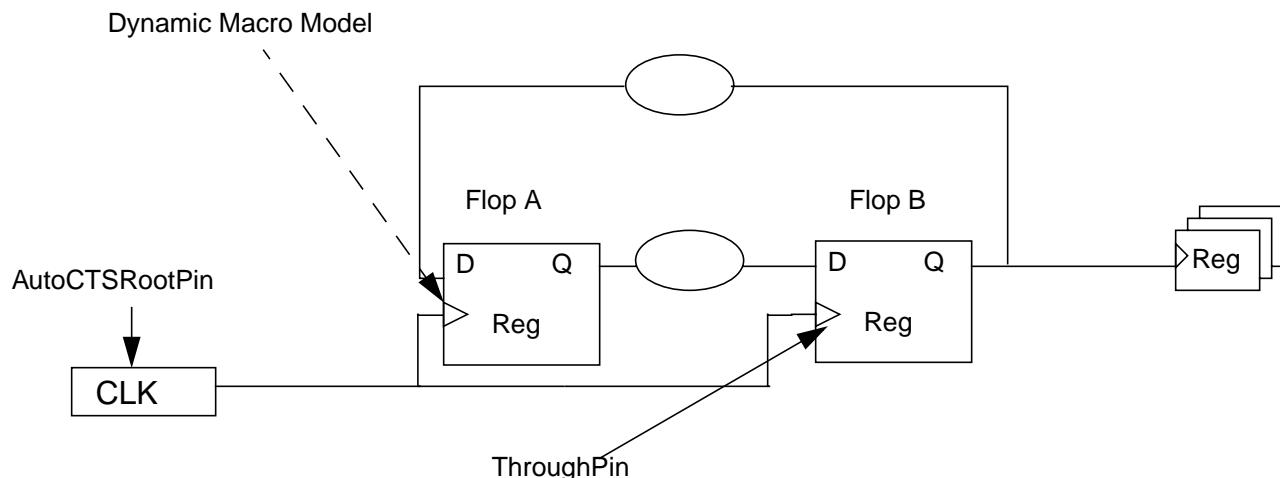
A dynamic macro model is used to minimize the skew between the reference pin and the target pin during CTS. The reference pin is a clock instance pin along a clock path. The target pin must be a leaf pin.

```
DynamicMacroModel ref refInstPinName pin targetInstPinName [offset delayNumber]
```

where *refInstPinName* is the reference instance pin name, *targetInstPinName* is the target instance pin name, and *delayNumber* specifies the offset arrival delay in nanoseconds or picoseconds.

As an example, the `DynamicMacroModel` statement can be used when your design contains clock dividers. The following figure contains two flops, A and B. A `ThroughPin` has been defined in the clock pin of Flop B. So, the clock pin of Flop A is balanced with the group of flops and not with the clock pin of Flop B because of the `ThroughPin` that has been defined in Flop B. Using a dynamic macro model in Flop A, you can balance the skew between the two flops. You can then specify clock pin of Flop B as a reference pin and clock pin of Flop A as the target pin so that the clock pin of flop A is balanced with the clock pin of flop B. The `DynamicMacroModel` statement minimizes the skew between these two flops to avoid timing violation on the data path.

The following figure illustrates how the skew is minimized between the reference pin and the target pin using dynamic macro model.



If there are multiple `DynamicMacroModel` statements where the target pin is referenced to more than one reference pin, the latter overrides the previous statement.

The `offset` parameter is optional. Instead of minimizing the difference between the arrival delays at the reference and the target pins, the `offset` parameter allows you to specify the offset arrival delay for the target pin. The offset arrival delay can be a positive or a negative value. A positive `offset` number indicates a shorter clock path for the target pin as compared to that of a reference pin. The default value of the `offset` parameter is 0.

## Example

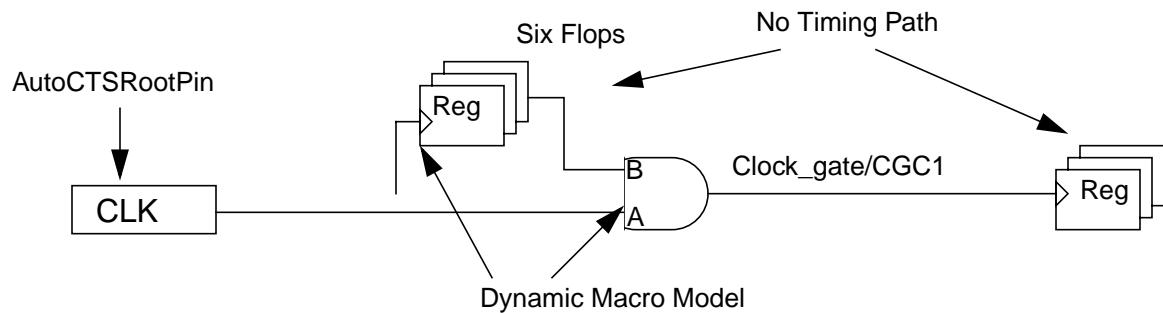
In the following example, a clock tree is built by minimizing the skew between the reference pin (`Clock_gate/CGC1/CLK`) and the target pins (`fsm_reg1/CLK`, `fsm_reg2/CLK`, and so on).

```
..  
DynamicMacroModel ref Clock_gate/CGC1/A pin fsm_reg1/CLK offset 1
```

## Encounter User Guide

### Synthesizing Clock Trees

```
DynamicMacroModel ref Clock_gate/CGC1/A pin fsm_reg2/CLK offset 1  
..
```



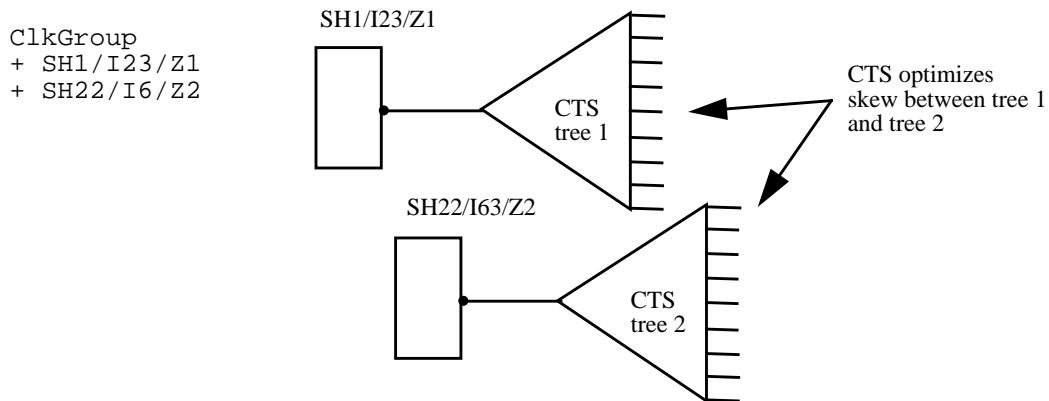
## Grouping Clocks

Clock grouping is available in automatic CTS mode. All clock root pin names entered into a clock group that will have their sinks meet the maximum skew as specified in the clock tree specification file. CTS balances the clock tree roots as if they were one tree.

The sinks of all clock root pins listed in a `ClkGroup` statement will meet the maximum skew value set in the clock tree specification file. Clock grouping inserts delays to balance the clocks, and attempts to meet clock skew for all clocks.

**Note:** You can define more than one clock group in the clock tree specification file.

The following is an example of clock group syntax and its graphical representation:



**Note:** All `ClkGroup` statements must be specified in lines following macro model line(s), and before any clock specification.

## Analyzing Hierarchical Clock Trees

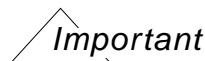
Encounter designs clock trees in a two-step, bottom-up fashion.

Within Encounter, the designing of the clock tree is done bottom-up in two steps. After partitioning the design, you can run CTS on each partition individually. Once the partitions are synthesized, the top-level partition runs CTS hierarchically. So CTS runs at the top-level partition, and the partitions' clock tree results are treated as macro model instances.

To generate the partition macro models, use the Synthesize Clock Tree form (*Clock – Synthesize Clock Tree*) or the following command when running CTS for the partition:

ckSynthesis -macromodel *fileName*

The rise time, fall time, and input capacitance for the clock pins are characterized, and the *fileName* output model file is used when creating the top-level partition's clock tree specification file. Running CTS for the top-level partition balances the clock phase delay between the top-level and the partitions.



The macro model specifications for each partition are at the top of the clock tree specification file.

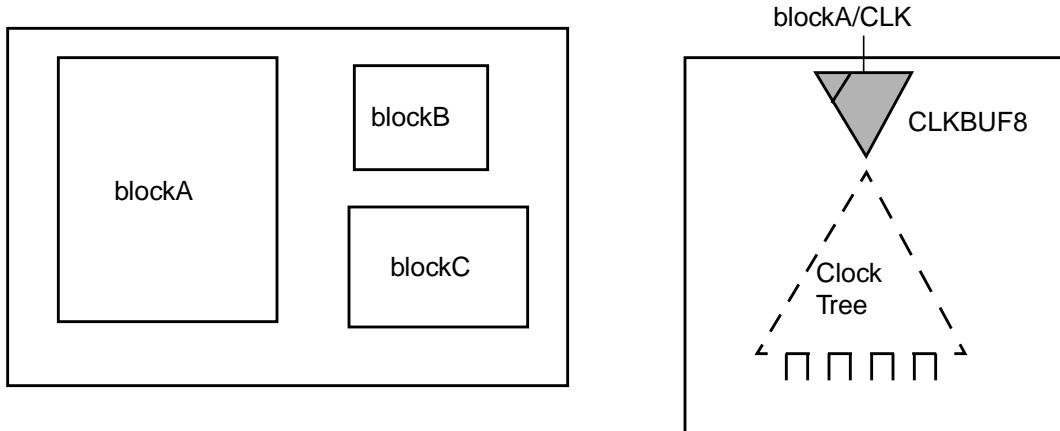
For example, in a design with three partitions (blockA, blockB and blockC), you should first synthesize the partitions individually. To run CTS on the partition's blocks, you should add the AddDriverCell statement in the clock tree specification file. Use the `AddDriverCell driver_cell_name` statement for block-level CTS to place a driver cell name at the closest possible location to the clock port location. For example:

```
AutoCTSRootPin blockA/clk
...
...
AddDriverCell CLKBUF8
End
```

## Encounter User Guide

### Synthesizing Clock Trees

CTS adds buffer CLKBUF8 after the input pin, as shown in the following figure:



After running CTS on the blocks, run CTS on the top level of the design. To run top-level CTS, you must include all the macro models from block-level CTS in the clock tree specification file.

#### *Important*

If your top-level design has a large amount of blockage and is limited in routing resources, you should add the `Obstruction Yes` statement in the clock-tree specification file. That statement instructs CTS to run the detail maze router to detect the obstruction (which increases CTS runtime). Use this statement only when routing resources are extremely limited, such as in top-level CTS.

The following example shows the `Obstruction Yes` command in the clock specification file:

```
MacroModel port blockA/clk 900ps 800ps 900ps 800ps 17ff  
MacroModel port blockB/clk 1100ps 1000ps 1100ps 1000ps 18ff  
MacroModel port blockC/clk 500ps 400ps 500ps 400ps 19ff  
AutoCTSRootPin clk  
....  
...  
Obstruction Yes  
End
```

## Module Placement Utilization

Make sure the modules' placement utilization, which contains the clock nets is set to 5–7 percent less than the desired final chip utilization (placement density). This provides placement resources for adding clock buffers during CTS.

## Clock Designs with Tight Area

For a clock design that is limited to a tight area, use the Specify Cell Padding form (*Place – Specify – Cell Padding*) to create placement resources near clocked flip-flop cell types.

## Balancing Pins for Macro Models

CTS can balance a pin of a macro model. These macro models are user specified. CTS balances the phase delay of all leaf pins in the clock tree, including leaf pins of macro models.

The timing models for macro models are defined in the clock tree specification file `MacroModel` statement.

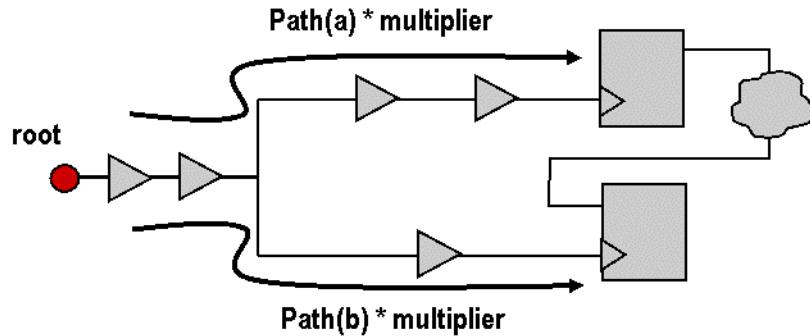
## Timing Model Requirement for Cells

Make sure that all cells have a timing model. If a cell does not have a timing model, CTS will not trace through the gate, and may set the gate's input pin as a leaf pin.

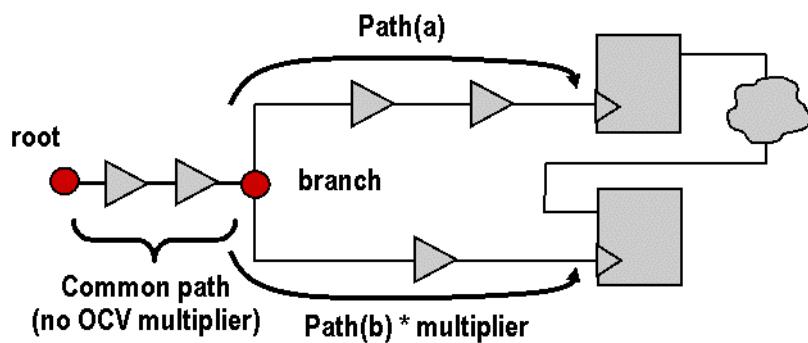
## Delay Variation and OCV

CTS can analyze your design for delay variation and on-chip variation (OCV). CTS identifies adjacent registers and then calculates the delay, using derating factors that you set with the specify with the `setAnalysisMode` and `setTimingDerate` commands.

The following diagram illustrates how CTS applies wire and cell derating factors to the whole path in a clock tree, including the common path.



The following diagram illustrates how CTS applies cell and wire derating factors only to the paths after a branch point—and not to the common path.



## Understanding Post-CTS Clock Tree Optimization

### Using the ckECO Command for Post-CTS Clock Tree Optimization

Use the `ckECO` command for post-CTS optimization of clock tree(s) in the same Encounter session as the `ckSynthesis` command was run, or in a new session. The sole aim of the `ckECO` command is to improve the skew of each clock and clock group, and to resolve minimum phase delay violations. The `ckECO` command does not attempt to correct any design rule violations. However, in trying to improve skew, the `ckECO` command does not significantly worsen maximum transition or maximum capacitance violations.

The `ckECO` command performs resizing and buffer insertion or dummy buffer insertion to improve skew. In addition, the `ckECO` command might move gating cells when the `ckECO` command runs `refinePlace`.

## Support for Local Skew Optimization

The `ckECO` command also supports local skew optimization (with the `-localSkew` parameter). Local skew optimization considers the skew between adjacent flip-flops that have data path connection (from a Q-pin of one flip-flop to the D-pin of another flip-flop).

Load the timing constraints into the Encounter session (design import stage) when you want to perform local skew optimization.

At a minimum, the clock roots need to be defined in the SDC file so the `ckECO` command can identify the adjacent register pairs.

## Command Modes for the `ckECO` Command

The `ckECO` command can be run in three modes:

- `ckECO -preRoute`
- `ckECO -clkRouteOnly`
- `ckECO -postRoute`

It is important to choose the correct mode, based on the state of the design. Otherwise CTS might use the wrong RC model, which can lead to quality-of-result (QOR) and correlation problems.

## Using a SPEF File with the `ckECO` Command for RC Estimation

As an alternative to using CTS estimation for RCs it is possible to load a SPEF file.

If you use an external SPEF file (`spefIn`) you just use the `ckECO -postRoute` command, and CTS does *not* create a clock tree report after the `ckECO -postRoute` process is done. You will need to re-extract the RC values for all the wires.

After you create a new SPEF file, load this new file into the Encounter session. Then run `reportClockTree -postRoute`. CTS then creates the clock report.

If clock nets are in a routed state you run the `ckECO` command, any wires that are disturbed by the `ckECO` command will automatically be rerouted using NanoRoute in an ECO mode.

## Running Post-CTS Optimization with the ckECO Command

You must load the clock tree specification file to run the `ckECO` command. (The clock tree specification file defines the clocks which you want to optimize.)

Whether the `ckECO` command performs resizing or ECO routing depends on these clock tree specification file or `setCTSMODE` settings:

- `RouteClkNet` YES | NO
- `setCTSMODE -routeClkNet {true | false}`
- `PostOpt` YES | NO
- `setCTSMODE -opt {true | false}`

The following examples show the usage of the `ckECO` command for two general design states.

### Example 1

If your design has the following characteristics—

- Clock tree is already inserted
- Clocks are routed
- Signal nets are not routed

—use this command sequence:

```
specifyClockTree  
ckECO -clkRouteOnly
```

### Example 2

If your design has the following characteristics—

- Clock tree is already inserted
- Clock nets are not routed
- Signal nets are not routed

—use this command sequence:

```
specifyClockTree  
ckECO
```

**Note:** When you specify the `ckECO` command without a parameter, you instruct CTS to use the default mode for the command, which is `-preRoute`.

## Guidelines for Using the `ckECO` Command

- If the design contains signal routes routed by Trial Route or NanoRoute, you must specify `ckECO -postRoute`; otherwise, the software generates an error message and stops. When you specify `ckECO -postRoute` on designs with Trial-Routed nets, the software removes any Trial-Routed nets before running the command. After the command has run, the software calls Trial Route to replace the Trial-Routed nets.

**Note:** The new Trial-Routed nets are *not* guaranteed to be identical to those before you ran the `ckECO` command.

- By default, the `ckECO` command inserts a maximum of 50 buffers. The number of buffers is controlled by the [OptAddBufferLimit](#) statement in the clock tree specification file.
- Often it is possible to make incremental improvements to skew by running the `ckECO` command multiple times.

For details on the `ckECO` command and its parameters, see “Clock Tree Synthesis Commands” in the *Encounter Text Command Reference*.

## Creating a Clock Tree Specification File

Before you can run CTS, you must create a clock tree specification file by:

- Using the Create Clock Tree Spec form
- Using the `createClockTreeSpec` command
- Using the `specifyClockTree` command with the `-template` parameter

The `specifyClockTree` command creates the basic clock tree specification file template file `template.ctstch` in the directory in which you are running the Encounter software. Edit the template file with any text editor.

This file contains the following information on the clock or clocks you want to analyze with CTS:

- Timing constraint file (optional)
- Naming attributes (optional)
- Macro model data (optional)
- Clock grouping data (optional)
- Attributes used by NanoRoute™ Ultra SoC routing solution (optional)
- Requirements for manual CTS or automatic, gated CTS

You can create a clock tree specification file for all the clocks in your design, for a subset of clocks in your design, or for a single clock.



The general sections of the clock tree specification files must appear in the order given above. However, the individual statements within each section can appear in any order.

## Using the Automatic Clock Tree Specification File Generator

You can specify `setCTSMode -specMultiMode true`, to enable the automatic clock tree specification file generator, which generates clock tree specification files that are more timing aware than regular specification files.

The auto clock tree specification file generator can be used only with the `clockDesign` command, and can be used in two ways:

- By specifying `clockDesign -genSpecOnly`, to only generate the timing aware clock tree specification file.
- By specifying `clockDesign` on its own, to generate the specification file, and then running clock tree synthesis.

The auto clock tree specification file generator works differently depending on whether the software is in multi-mode multi-corner (MMMC) analysis mode.

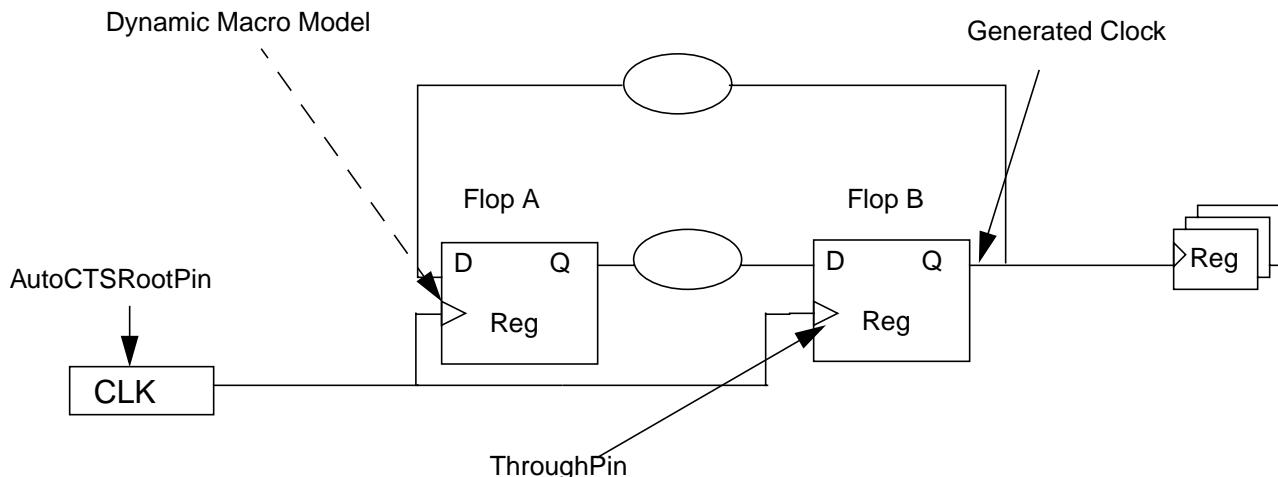
If you specify the following commands when the software is not in MMMC analysis mode:

```
setCTSMode -specMultiMode true
addCTSCellList "BUFX2 BUFX4 BUFX8"
clockDesign -genSpecOnly
```

The auto clock tree spec generator determines whether certain groups of flops that form a clock divider circuit need to be balanced separately. Additionally, it handles case analysis constraints, disabled timing, and loop breaking in the SDC file.

The auto clock tree spec generator also:

- Generates a spec file with dynamic macro models to help close on timing issues between the flops (which function as clock dividers) that are responsible for providing the generated clock. It adds dynamic macro models when appropriate.



- Groups clocks that communicate with each other in the spec file (`ClkGroup`) when appropriate.

**Note:** If you run `clockDesign` without the `-genSpecOnly` parameter, the automatic clock spec generator generates the new spec file with dynamic macromodels and grouped clocks, then runs synthesis.

If you specify the same commands when the software is in MMMC analysis mode, the auto

clock tree spec generator generates multiple specification files as above, using the naming convention *fileName.modeName*.

If you run `clockDesign` without the `-genSpecOnly` parameter, it generates the multiple specification files, and also performs the following CTS functions on all active analysis views, sequentially:

- Creates and loads a clock tree specification file for each mode.
- Runs clock tree synthesis.
- Determines which instances of the synthesized clock tree to preserve for building the next clock tree.
- Removes the clock tree spec file from memory.

After the software runs clock tree synthesis on all of the views, it runs `timeDesign`.

## **Example of a Clock Tree Specification File**

The following example illustrates the content of a clock tree specification file:

```
UseSingleDelim YES
NameDelimiter |
MacroModel pin freq/mod004048/CLK 20ps 18ps 20ps 18ps 0.29ff

ClkGroup
+ CGEN_1
+ CGEN_2

UseCTSRouteGuide NO

#Start RouteTypeName section
RouteTypeName CK1
NonDefaultRule rule1
PreferredExtraSpace 1
TopPreferredLayer 5
BottomPreferredLayer 4
Shielding VSS
End
#End RouteTypeName section

#Example of manual CTS specification information
ClockNetName CK
LevelNumber 2
LevelSpec 1 1 BUFX2
```

## Encounter User Guide

### Synthesizing Clock Trees

---

```
LevelSpec 2 2 BUFX2
PostOpt YES
OptAddBuffer YES
End

AutoCTSRootPin cgen/i_5/Y
MaxDelay 5.0ns
MinDelay 0ns
MaxFanout 30
MaxSkew 250ps
SinkMaxTran 550ps
BufMaxTran 550ps
RootInputTran XYZ
NoGating NO
DetailReport YES
Obstruction YES
RouteType CK1
LeafRouteType specialRoute
CellRouteType
+ BUFX2 routeName1
+ BUFX4 routeName2
CellLeafRouteType
+ BUFX8 routeName3
+ BUFX12 routeName4
RouteClkNet YES # Turns on NanoRoute. The default value is NO
PostOpt YES # Turns on optimization. The default value is YES
OptAddBuffer YES
OptAddBufferLimit 100
Buffer BUFX2 BUFX4 BUFX8 BUFX12 INVX1
MaxCap
+ BUFX2 1pf
+ BUFX4 1pf
+ BUFX8 1pf
+ BUFX12 1pf
ForceMaxTran NO
ThroughPin
+ df/mod000446/CK Y
ThroughPort
+ df/mod002300/ax2
LeafPin
+ PCLK66_gate_i/A rising
LeafPort
+ ssfd2s/D rising
KeepPortPolarity
```

## Encounter User Guide

### Synthesizing Clock Trees

---

```
+ MOD1/PORT1
PreservePin
+ cgen/mod000043/A
ExcludedPin
+ freg/mod004048/CLK
ExcludedPort
+ DFF_B/CLK
GatingGrpInstances
+ cg1/an cg1/i_0
+ cg2/an cg2/i_0
+ cg1/an cg3/i_0
+ cg4/an cg4/i_0
GatingGrpModule
+ grp_module1
+ grp_a*
CellHalo
+BUFX411
End
```

```
MasterGateTargetFanout32
AutoCTSRootPin clk
MaxDelay 3.1ns
MinDelay 0ns
MaxSkew 100ps
SinkMaxTran 200ps
BufMaxTran 200ps
Buffer CLKBUFX4
NoGating NO
MasterGateInst
+ 7 clk_latch
End
```

```
AutoCTSRootPin clk
MaxDelay 3.1ns
MinDelay 0ns
MaxSkew 100ps
SinkMaxTran 200ps
BufMaxTran 200ps
Buffer CLKBUFX4
NoGating NO
EquivGateInst
+ G1
+ G2
EquivGateInst
```

## **Encounter User Guide**

### Synthesizing Clock Trees

---

```
+ clk_and1 clk_neg1  
+ clk_and2 clk_neg2
```

## Naming Attributes Section

The following table describes the entries for the naming attributes section:

NameDelimiter *delimiter*

Allows you to customize the name delimiter that CTS uses when inserting buffers and updating clock root and net names. There are no restrictions on the type of characters that you specify for the name delimiters in the clock tree specification file but you must specify only a single character. For example:

`NameDelimiter # creates names with the format  
clk##L3#I2, rather than the default format, clk__L3_I2.`

Insert the `NameDelimiter` statement after `MacroModel` and `ClkGroup` statements but before an `AutoCTSRootPin` statement.

**Note:** If you have multiple `NameDelimiter` statements in the clock tree specification file, CTS uses only the last `NameDelimiter` statement in the file.

**Note:** The `NameDelimiter` and `UseSingleDelim` statements are independent of each other.

UseSingleDelim YES | NO

Instructs CTS whether to use single name delimiters after the first element in clock root and net names. For example:

`UseSingleDelim YES creates clock and net names with the format clk_L3_I2, rather than the default format, clk__L3_I2.`

By default, CTS always inserts double (or, in some cases, multiple) name delimiters after the first element of clock root or net names. The `UseSingleDelim YES` statement overrides this behavior by instructing CTS to use *only* a single delimiter after the first element of the name:

Insert the `UseSingleDelim` statement after `MacroModel` and `ClkGroup` statements but before an `AutoCTSRootPin` statement.

**Note:** The `UseSingleDelim` and `NameDelimiter` statements are independent of each other.

## NanoRoute Attribute Section

The following table describes the entries for the section of the clock tree specification file that deals with attributes that CTS passes to NanoRoute Ultra.

`UseCTSRouteGuide YES | NO`

Specifies whether NanoRoute should route clock nets with the routing guide file generated by CTS.

NanoRoute will produce better pre- and postroute correlations, though at the expense of longer run time.

If you specify `UseCTSRouteGuide YES` and `RouteClkNet YES`, clock nets will be routed based on the route guide file created by CTS.

If you specify `UseCTSRouteGuide YES` and `RouteClkNet NO`, CTS creates a route guide file. But because you instructed CTS not to route the clock nets, the guide file is not used. If you subsequently use the `routeClockNetWithGuide` command, CTS generates a new routing guide file that is based on the synthesized clock structure, and uses it to automatically route the clocks.

`RouteTypeName name` Specifies the routing type for which you are defining routing attributes.

**Note:** `RouteTypeName` statements must appear in the clock tree specification file before their corresponding `RouteType` statements.

`NonDefaultRule ruleName`

Specifies the LEF NONDEFAULTRULE statement that the router should use.

*Default:* If you do not use this statement, the router uses the default routing rule.

`PreferredExtraSpace [0-3]`

Specifies the spacing attribute, with which to add space around clock wires.

*Default:* If you do not use this statement, CTS uses a preferred extra space value of 1.

`TopPreferredLayer number`

Specifies the top-most preferred routing layer.

*Default:* 4

BottomPreferredLayer *number*

Specifies the bottom-most preferred routing layer.

*Default:* 3

Shielding *GNetName*

Defines the ground net name.

AssumeShielding YES | NO

Instructs CTS to assume that unshielded wires are shielded when CTS estimates wire loading.

*Default:* If you omit this statement, CTS behaves as if you had specified AssumeShielding NO.

End

Marks the end of the NanoRoute Ultra attribute section.

 **Important**

In all clock tree specification file sections that contain delay values or capacitance values, you *must* include the appropriate unit with the values you specify. The unit designations are case-insensitive. For example, pF, pF, Pf, and PF are all valid unit designations for picofarads.

## Macro Model Data Section

The following table describes the entries for the macro model *port* data section:

MacroModel port *cellName\_or\_portName*  
*delay\_and\_capacitance\_values*

Specifies the name of the macro model cell or port.

*maxRiseDelay{ns | ps}* Specifies the maximum rise delay in nanoseconds or picoseconds.

*minRiseDelay{ns | ps}* Specifies the minimum rise delay in nanoseconds or picoseconds.

*maxFallDelay{ns | ps}* Specifies the maximum fall delay in nanoseconds or picoseconds.

*minFallDelay{ns | ps}* Specifies the minimum fall delay in nanoseconds or picoseconds.

The following table describes the entries for the macro model *pin* data section:

<i>MacroModel pin pinName delay_and_capacitance_values</i>	Specifies the name of the macro model pin.
<i>maxRiseDelay{ns   ps}</i>	Specifies the maximum rise delay in nanoseconds or picoseconds.
<i>minRiseDelay{ns   ps}</i>	Specifies the minimum rise delay in nanoseconds or picoseconds.
<i>maxFallDelay{ns   ps}</i>	Specifies the maximum fall delay in nanoseconds or picoseconds.
<i>minFallDelay{ns   ps}</i>	Specifies the minimum fall delay in nanoseconds or picoseconds.

The following table describes the entries for the global directive section.

 **Important**

Global directive statements can not be used between `AutoCTSRootPin` and `End` statements.

`GlobalLeafPin`  
+ *pinName rising | falling*  
+ ...

Marks the input pin as a “leaf” pin for non-clock-type instances globally (throughout the design), stops tracing, and balances clock skew.

**Note:** Use the `GlobalLeafPin` statement *only* with input pins. CTS ignores `GlobalLeafPin` statements that are associated with output pins.

Choose one of the following:

`rising`      CTS treats the input pin as a rising-edge-triggered flip-flop clock pin.

## Encounter User Guide

### Synthesizing Clock Trees

---

**falling** CTS treats the input pin as a falling-edge-triggered flip-flop clock pin.

`GlobalLeafPort`

+ `cellType`/`inputPinName` `rising` | `falling`  
+ ...

Marks the pin as a “leaf” pin for cells globally (throughout the design), stops tracing, and balances clock skew.

Choose one of the following:

**rising** CTS treats the pin as a rising-edge-triggered flip-flop clock pin.

**falling** CTS treats the pin as a falling-edge-triggered flip-flop clock pin.

`GlobalExcludedPin`

+ `pinName`  
+ ...

Treats the pin as a non-leaf pin globally (throughout the design), and prevents tracing and skew analysis of the pin.

`GlobalExcludedPort`

+ `cellType`/  
`inputPinName`  
+ ...

Treats the pin of particular cell type as a non-leaf pin globally (throughout the design), and prevents tracing and skew analysis of the pin.

`GlobalThroughPin`

+ `pinName` [`outputPinNames`]  
+ ...

Traces through the pin globally (throughout the design), even if the pin is a clock pin.

For multi-output cells, you must use `outputPinName` to instruct CTS which output pin to trace through.

`GlobalPreservePin`

+ `inputPinName`  
+ ...

Preserves the netlist for the pin and pins below the pin in the clock tree globally (throughout the design). However, CTS considers any synchronized pins after the pin when computing skew.

## Encounter User Guide

### Synthesizing Clock Trees

---

DontTouchNet  
+ *netName1*  
+ *netName2*  
+ ...

Specifies a list of nets for which the `ckSynthesis` and `ckEco` commands should not insert buffers. The `deleteClockTree` command does not delete buffers if their input or output nets have the `DontTouchNet` attribute.

**Note:** The `DontTouchNet` statement is not a physical parameter; any net specified in this statement can still be routed.

DontTouchFromToPin  
+ *pinName1 pinName2*

Instructs the `ckSynthesis` and `ckEco` commands to not insert buffers for nets that are between the specified start instance pin and end instance pin. Any nets between these pins are considered to have the `DontTouchNet` attribute.

DontAddNewPortModule  
+ *moduleName1*  
+ *moduleName2*  
+ ...

Does not add a new port to the specified logical modules at their given hierarchical levels.

**Note:**

- The `DontAddNewPortModule` statement applies only to the specified modules but not the nested submodules. When you specify this statement for any given module, only that module does not have a new port added to it but the nested submodule still might have a new port added.
- The `DontAddNewPortModule` statement might increase the latency of the clock tree and more buffers could be added because due to this restriction, the instances of different modules cannot be shared by the same buffer.

## Clock Grouping Data Section

The following table describes the entries for the clock grouping data section:

ClkGroup	Specifies two or more clock domains for which you want CTS to balance the skew.
+ <i>clockRootPinName</i>	
+ <i>clockRootPinName</i>	
...	

## Clock-Tree Topology Section

The clock-tree topology section provides a method for you to manually define buffers at particular levels. The following table describes the entries for the clock-tree topology section:

ClockNetName <i>netName</i>	Specifies the name of the clock net.
LevelNumber <i>totalNumberOfClockTreeLevels</i>	Specifies the total number of clock tree levels.
LevelSpec <i>levelNumber</i> <i>numberOfBuffers</i> <i>bufferType</i>	Provides the specifications for an individual clock level. Specify all the following information: <i>levelNumber</i> Sets the level number in the clock tree. <i>numberOfBuffers</i> Specifies the total number of buffers CTS should allow on the specified level of the clock tree. <i>bufferType</i> Specifies the buffer type (based upon the LEF file).
End	Marks the end of a clock tree topology section.

## Automatic Gated CTS Section

The following table describes the entries for the automatic gated CTS section:

AutoCTSRootPin <i>clockRootPinName</i>
--

	Specifies the name of the clock root pin name from which to start tracing.
MaxDelay <i>number{ns ps}</i>	Specifies the maximum phase delay constraint.  <i>Default:</i> If you do not use this statement, CTS uses a maximum phase delay constraint of 10 ns.
MinDelay <i>number{ns ps}</i>	Specifies the minimum phase delay constraint.  <i>Default:</i> If you do not use this statement, CTS uses a minimum phase delay constraint of 0.0 ns.
SinkMaxTran <i>number{ns ps}</i>	Specifies the maximum input transition time constraint for sinks (clock pins). CTS does not allow you to specify a value greater than 10,000 ns.  <i>Default:</i> If you do not use this statement, CTS uses a maximum sink transition time constraint of 400 ps.
BufMaxTran <i>number{ns ps}</i>	Specifies the maximum input transition time constraint for buffers. CTS does not allow you to specify a value greater than 10,000 ns.  <i>Default:</i> If you do not use this statement, CTS uses a maximum buffer transition time constraint of 400 ps.
MaxGateRatio <i>ratio</i>	Specifies the maximum gate ratio for the clock tree. Gate ratio is defined as:  sum of all cell delay / clock path delay (from clock root to all registers)  <i>Default:</i> 1
MinGateRatio <i>ratio</i>	Specifies the minimum gate ratio for the clock tree. Gate ratio is defined as:  sum of all cell delay / clock path delay (from clock root to all registers)  <i>Default:</i> 0
MaxDeltaGateRatio <i>delta_ratio</i>	

Specifies the maximum delta gate ratio for the clock tree. Delta gate ratio is defined as the difference between fmGateRatio and toGateRatio, where:

- fmGateRatio is the sum of all cell delay divided by the clock path delay from the branching point to the source register.
- toGateRatio is the sum of all cell delay divided by the clock path delay from the branching point to the destination register.

**Note:** fmGateRatio and toGateRatio are local measurements: the starting point is at the input pin of the last buffer of the common clock path.

*Default:* 1

**MinDeltaGateRatio *delta\_ratio***

Specifies the minimum delta gate ratio for the clock tree. Delta gate ratio is defined as the difference between fmGateRatio and toGateRatio, where:

- fmGateRatio is the sum of all cell delay divided by the clock path delay from the branching point to the source register.
- toGateRatio is the sum of all cell delay divided by the clock path delay from the branching point to the destination register.

**Note:** fmGateRatio and toGateRatio are local measurements: the starting point is at the input pin of the last buffer of the common clock path.

*Default:* 1

## Encounter User Guide

### Synthesizing Clock Trees

---

**SrcLatency number{ns}** Specifies an external latency value for each clock.

CTS adds this value to the clock tree latency when grouping and balancing clocks. Consider the following clock tree specification file extracts:

```
AutoCTSRootPin clock1
MaxDelay 5.0ns
MinDelay 4.5ns
SrcLatency 2.0ns
MaxSkew 300ps
...
End
```

```
AutoCTSRootPin clock2
MaxDelay 2.0ns
MinDelay 1.5ns
SrcLatency 5.0ns
MaxSkew 300ps
...
End
```

The values for MaxDelay, MinDelay, and SrcLatency *must* satisfy the following relationship for all clocks listed in a clock tree specification file ClkGroup statement for that ClkGroup statement to be valid:

$$\begin{aligned} \text{SrcLatency}_{\text{clock1}} + \text{MinDelay}_{\text{clock1}} &= \\ \text{SrcLatency}_{\text{clock2}} + \text{MinDelay}_{\text{clock2}} &= \\ \text{SrcLatency}_{\text{clockn}} + \text{MinDelay}_{\text{clockn}} & \end{aligned}$$
$$\begin{aligned} \text{SrcLatency}_{\text{clock1}} + \text{MaxDelay}_{\text{clock1}} &= \\ \text{SrcLatency}_{\text{clock2}} + \text{MaxDelay}_{\text{clock2}} &= \\ \text{SrcLatency}_{\text{clockn}} + \text{MaxDelay}_{\text{clockn}} & \end{aligned}$$

**Default:** 0ns

**RootInputTran number{ns|ps}**

Specifies the input transition time for an input pin or an input port. Use this statement when you do not want CTS to use the default Encounter input transition time.

**Note:** If your SDC file contains a `set_input_transition` constraint, the `RootInputTrans` statement overrides that constraint.

 *Important*

You *cannot* use this statement with output pins or output ports.

`MaxSkew number{ns|ps}[viewName]`

Specifies the maximum skew between sinks (clock pins).

You can specify different skew limits to be used for specific active analysis views (`viewName`) for multi-mode CTS.

*Default:* If you do not use this statement, CTS uses a maximum skew constraint of 300 ps.

`DefaultMaxCap integer`

Specifies the default maximum capacitance value for all buffers, inverters, and gating cells that do not have a specific maximum capacitance value specified with the `MaxCap` and `PinMaxCap` statements.

`MaxFanout integer`

Limits the number of clock buffer fanouts to the number you specify.

If you specify the `MaxFanout` statement and the `setCTSMODE -useLibMaxFanout true` parameter, CTS uses the worst case constraint specified.

CTS does not support a `fanout_load` that is not equal to 1.

**Note:** CTS considers the `MaxFanout` statement to be a soft constraint. CTS will try to meet it, but might need to make adjustments in order to meet physical constraints.

`MaxNumLevel number`

Specifies the maximum number of buffer levels that CTS builds in a clock tree that has no gating components.

`ClockGatingProb number`      Specifies the probable amount of time that the clock is turned on.

The `ClockGatingProb` value is used for statistical estimation of power. CTS calculates internal power by multiplying the activity of the clock net by the `ClockGatingProb` value. (The activity of the clock net is taken from the `clock Period` statement in the clock tree specification file.)

If you have a single clock gate at the root, and all of the buffers are after the clock gating cell, CTS calculates the activity rate as:

$(\text{activity of clock net at gate's input}) \times \text{ClockGatingProb}$

CTS computes the activity rate of a net driven by a buffer in the clock path as:

$(\text{activity of clock net at buffer's input}) \times \text{ClockGatingProb}$

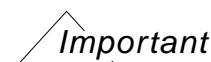
*Default:* 1 (clock is always on)

`LevelBalanced YES | NO`

Instructs CTS to build a clock tree in which all paths from the root to the leaves have the same number of levels. (Equal numbers of levels can be helpful in high-performance designs, such as designs incorporating structured ASICs.)

**Note:** Be aware of the following requirements:

- This statement does not optimize for skew: It only builds trees with equal numbers of levels. (If you want CTS to optimize for skew, specify `LevelBalanced NO` after initial CTS.)
- To avoid creating maximum capacitance violations, include the `YES` statement in the clock tree specification file.



You cannot use this statement for gated clocks.

## Encounter User Guide

### Synthesizing Clock Trees

---

Period <i>value</i>	Specifies the clock period of a root pin.  <i>Default:</i> If you omit this parameter, CTS uses a value of 10 ns.
NoGating {rising   falling   NO}	Sets the criteria for tracing through logic gates.  Choose only one of the following: <ul style="list-style-type: none"><li>rising      Stops tracing through a gate (including buffers and inverters) and treats the gate as a rising-edge-triggered flip-flop clock pin.</li><li>falling     Stops tracing through a gate (including buffers and inverters) and treats the gate as a falling-edge-triggered flip-flop clock pin.</li><li>NO          Allows CTS to trace through clock gating logic.  <i>Default:</i> This is the default behavior for gated-clock designs. (If you omit the NoGating statement, CTS traces through clock gating logic.)</li></ul>
AddDriverCell <i>driver_cell_name</i>	Places a driver cell name at the closest possible location to the clock port location for block-level CTS.
MaxDepth <i>number</i>	Sets the maximum depth of clock tree tracing.  <i>Default:</i> If you do not use this statement, CTS limits the number of levels of clock tree tracing to 1024.
RouteType <i>routeTypeName</i>	Specifies the name of the routing attributes definition.  <b>Note:</b> CTS uses the RouteType statement <i>only</i> if you specify RouteClkNet YES.  <b>Note:</b> If you use a RouteType statement, you must also use a corresponding <i>routeTypeName</i> .
LeafRouteType <i>leafRouteTypeName</i>	

Specifies the route type name for which you are defining a routing attribute. Use this statement when you want to define a particular routing type for nets that connect to leaf pins.

The `LeafRouteType` statement applies to that part of a net that runs from the last non-leaf cell to leaf cell(s). The `LeafRouteType` statement is useful for high-fanout designs.

`cellRouteType`  
+ `cellName RouteTypeName`  
+ ...

Specifies the routing attribute for the output net driven by the specified cell.

`cellLeafRouteType`  
+ `cellName RouteTypeName`  
+ ...

Specifies the routing attribute for the output net driven by the specified leaf cell. This applies only to the leaf nets.

`DetailReport YES | NO`

Determines whether CTS provides a detailed report. The detailed report includes timing information for every component in the design. The non-detailed report contains only summary information for the design.

*Default:* If you do not use this statement, CTS does not provide a detailed report.

`Obstruction YES | NO`

Specifies whether CTS should take design obstructions into account during flip-flop clustering.

## Encounter User Guide

### Synthesizing Clock Trees

---

RouteClkNet YES | NO

Specifies whether CTS routes clock nets.

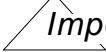
The following combinations of the RouteClkNet specification file statement and the `setCTSMode -routeGuide true` command are possible:

- If you specify RouteClkNet NO, CTS does not route clock nets. Timing is based on the pre-routed design using a steiner model. The RouteClkNet NO statement is useful for rapid prototyping.
- If you specify RouteClkNet YES together with the `setCTSMode -routeGuide false` command, CTS routes clocks using NanoRoute but CTS does *not* use a routing guide file. This combination of settings is useful for straightforward designs for which pre-route–post-route correlation is not an issue. CTS runs more quickly than when it uses a guide file. Note that NanoRoute routes the clocks but does not follow the pattern determined by CTS. Thus, this method cannot take advantage of the balanced routing that takes place during CTS.
- If you specify RouteClkNet YES together with the `setCTSMode -routeGuide true` command, CTS routes the clocks with NanoRoute and the CTS routing guide file. This method ensures appropriate pre-route–post-route correlation but at the expense of longer runtime.

*Default:* If you do *not* use this statement, CTS does not route clock nets. In other words, the default behavior is as if you had specified RouteClkNet NO.

## Encounter User Guide

### Synthesizing Clock Trees

PostOpt {YES   NO}	<p>Specifies whether CTS resizes buffers or inverters, refines placement, and corrects routing for signal and clock wires.</p> <p><i>Default:</i> YES</p> <p>When used together, the PostOpt and OptAddBuffer statements try to meet the trigger edge skew constraints as defined in the clock tree specification file. However, phase delay, buffer transition time, and sink transition times are not necessarily improved by using these two statements.</p> <p><b>Note:</b> If you specify PostOpt NO, CTS does <i>not</i> resize gating components. If you specify PostOpt YES, CTS <i>attempts</i> to resize gating components, though it may not do so.</p>
OptAddBuffer {YES   NO}	Controls whether CTS adds buffers during optimization.
	<p> <i>Important</i></p> <p>The OptAddBuffer statement is effective <i>only if you specify PostOpt YES</i>.</p> <p>When used together, the PostOpt and OptAddBuffer statements try to meet the trigger edge skew constraints as defined in the clock tree specification file. However, phase delay, buffer transition time, and sink transition times are not necessarily improved by using these two statements.</p>
OptAddBufferLimit <i>positive_integer</i>	<p>Specifies the maximum number of buffers that CTS can add to a clock tree during optimization. The number you specify can be no smaller than 1. However, the higher the number you specify, the longer the runtime.</p> <p><i>Default:</i> If you do not use this statement, CTS inserts no more than 50 buffers.</p>

AddSpareFF *cellName* *number*

Specifies the maximum number of flip-flops to add to the lowest level of the clock tree.

The inputs of the flip-flops are tied off and the outputs are left floating.

Adding extra flip-flops during CTS ensures that if a spare flip-flop needs to be connected at some later time (for example, by a metal-only ECO change), the existing clock network will not be disturbed.

*cellName*      Represents the name of the flip-flop(s) to be inserted in the clock root.

*number*      Represents the maximum number of flip-flops to insert. You can specify 1 or more flip-flops.

Buffer *cell1* *cell2* *cell3* ...

Specifies the names of buffer cells to use during automatic, gated CTS.

**Note:** To turn on the buffer insertion mechanism during the optimization process, you must have these statements in your clock tree specification file:

- PostOpt YES
- OptAddBuffer YES

DummyBuffer *cell1* *cell2* *cell3* ...

Specifies the names of dummy buffer cells to use during the optimization process.

Use this statement if you want CTS to use buffers other than those specified in the Buffer statement. (Buffers defined in the Buffer statement might be too large, or have an input capacitance value that is too small.)

**Note:** To turn on the buffer insertion mechanism during the optimization process, you must have these statements in your clock tree specification file:

- PostOpt YES
- OptAddBuffer YES

## Encounter User Guide

### Synthesizing Clock Trees

---

**LeafBuffer *buffer\_list***      Specifies a list of one or more buffer cells or inverters for CTS to use when inserting buffers at the leaf level of the clock tree.

CTS ignores the **LeafBuffer** statement if:

- The netlist has an inverter driving a small group of flops, and the design rule constraints are not violated.
- The gating cells (such as AND gates) are physically closed to the flops, and have sufficient drive strength to drive the flops at its fanout zone.

**LeafPin**  
+ *pinName* rising | falling  
+ ...

Marks the input pin as a “leaf” pin for non-clock-type instances, stops tracing, and balances clock skew.

**Note:** Use the **LeafPin** statement with an input pin.

Choose one of the following:

rising      CTS treats the input pin as a rising-edge-triggered flip-flop clock pin.

falling      CTS treats the input pin as a falling-edge-triggered flip-flop clock pin.

**LeafPort**  
+ *cellType/inputPinName* rising | falling  
+ ...

Marks the pin of a particular cell type as a “leaf” pin for non-clock-type instances, stops tracing, and balances clock skew.

Choose one of the following:

rising      CTS treats the pin as a rising-edge-triggered flip-flop clock pin.

falling      CTS treats the pin as a falling-edge-triggered flip-flop clock pin.

**ExcludedPin**  
+ *pinName*  
+ ...

Treats the pin as a non-leaf pin, and prevents tracing and skew analysis of the pin.

## Encounter User Guide

### Synthesizing Clock Trees

ExcludedPort + <i>cellType/inputPinName</i> + ...	Treats the pin of a particular cell type as a non-leaf pin, and prevents tracing and skew analysis of the pin.
KeepPortPolarity + <i>moduleName/portName</i>	Preserves the polarity of specified module ports while running the <code>deleteClockTree</code> command.
ThroughPin + <i>pinName [outputPinNames]</i> + ...	Traces through the pin, even if the pin is a clock pin.  For multi-output cells, you must use <i>outputPinName</i> to instruct CTS which output pin to trace through.
ThroughPort + <i>cellType/inputPinName</i> + ...	Traces through the pin of a particular cell type, even if the pin is a clock pin.
SetDPinAsSync YES   NO	Determines whether CTS automatically treats the Data pins of flip-flops as synchronous pins, instead of as excluded pins.  Data pins include: <ul style="list-style-type: none"><li>■ Data pins</li><li>■ Enable pins</li><li>■ Scan-in pins</li><li>■ Scan-enable pins</li><li>■ Synchronous set and reset pins</li></ul> This parameter does not control tristate control pins. By default, the software treats them as synchronous pins.  <i>Default:</i> If you omit this statement, CTS treats the Data pins of flip-flops as excluded pins.
SetIoPinAsSync YES   NO	Determines whether CTS automatically treats I/O pins as synchronous pins, instead of as excluded pins.  This parameter does not control tristate control pins. By default, the software treats them as synchronous pins.  <i>Default:</i> If you omit this statement, CTS treats I/O pins as excluded pins.

## Encounter User Guide

### Synthesizing Clock Trees

---

PreservePin  
+ *inputPinName*  
+ ...

Preserves the netlist for the pin and pins below the pin in the clock tree. However, CTS considers any synchronized pins after the pin when computing skew.

PinMaxCap  
+ *instanceName/pinName1 capValue1 {pf | ff}*  
+ *instanceName/pinName2 capValue2 {pf | ff}*  
...

Specifies the maximum capacitance value for the specified pins.

*Default:* If there is no PinMaxCap statement specified in the clock tree spec file, but there is a DefaultMaxCap statement specified, the software uses the DefaultMaxCap value to constrain pins.

If there are no PinMaxCap or DefaultMaxCap statements specified in the clock tree spec file, but you specify setCTSMODE -useLibMaxCap true, the software uses maximum capacitance values in the timing library.

If there are no PinMaxCap or DefaultMaxCap statements specified in the clock tree spec file and you do not specify setCTSMODE -useLibMaxCap true, the software does not apply a maximum capacitance constraint.

MaxCap  
+ *bufferName1 capValue1{pf | ff}*  
+ *bufferName2 capValue2{pf | ff}*  
+ *clockGatingCell1 capValue1{pf | ff}*  
+ *clockGatingCell2 capValue2{pf | ff}*  
+ ...

Specifies a maximum capacitance value for the specified buffers, inverters, or gating cells.

*Default:* If there is no `MaxCap` statement specified in the clock tree spec file, but there is a `DefaultMaxCap` statement specified, the software uses the `DefaultMaxCap` value to constrain buffers, inverters, or gating cells.

If there are no `MaxCap` or `DefaultMaxCap` statements specified in the clock tree spec file, but you specify `setCTSMODE -useLibMaxCap true`, the software uses maximum capacitance values in the timing library.

If there are no `MaxCap` or `DefaultMaxCap` statements specified in the clock tree spec file and you do not specify `setCTSMODE -useLibMaxCap true`, the software does not apply a maximum capacitance constraint.

*bufferName*      Specifies the name of the buffer for which you specify a maximum capacitance value.

*capValue*      Specifies the maximum capacitance for the buffer, in picofarads (pF) or femtofarads (fF).

`ExcludedBuffer cell`

Specifies the buffer to insert to isolate the clock path that is excluded during clock tree tracing. The `ExcludedBuffer` statement can be used to separate the normal path to leaf pins and the excluded path to non-leaf pins.

**Note:** Excluded pins can be identified by clock tree tracing, as well as by being specified with the `ExcludedPin` statement in the clock tree specification file.

*Default:* If you do not specify the `ExcludedBuffer` statement, the software chooses a buffer to insert from the `Buffer` statement.

## Encounter User Guide

### Synthesizing Clock Trees

ForceMaxTran YES | NO Increases the number of buffers to ensure that the clock net has no maximum transition violations.

**Note:** If you specify ForceMaxTran YES and run ckECO -fixDRVOnly, the software attempts to correct maximum transition violations.

ForceReconvergent YES | NO Controls whether CTS allows or prevents reconvergence conditions for individual clocks.

- YES indicates that you want CTS to allow reconvergence on a particular clock. CTS synthesizes such clocks.
- NO indicates that you do not want CTS to allow reconvergence on a particular clock. When CTS synthesizes such a clock, the software stops and issues a warning if a reconvergence condition is found for the clock.

The following example illustrates how to specify different reconvergence conditions on clocks clk and clk2:

```
AutoCTSRootPin clk
...
ForceReconvergent YES
...
END
AutoCTSRootPin clk2
...
ForceReconvergent NO
...
END
```

**Note:** You can override any ForceReconvergent statement by specifying ckSynthesis -forceReconvergent. In this case, CTS always synthesizes clocks that have reconvergence conditions.

*Default:* If you omit this statement from the clock tree specification file, CTS behaves as if you had specified ForceReconvergent NO.

GatingGrpInstances

```
+ instanceName instanceName ...
+ instanceName instanceName ...
+ ...
```

Instructs CTS to group instances. CTS will not insert buffers between the instances you specify.

- Each line that begins with a + represents a group of instances.
- Instances that you specify on the same line are in the same group.
- Each instance list must include at least one gate and one latch.
- You can specify no fewer than two, and no more than five instances, on each line.
- Each latch and gate you specify must be connected together.

GatingGrpModule

```
+ moduleName
+ moduleName
+ ...
```

Instructs CTS to treat all the instances within a specified module as if you had included the module's instances individually in a GatingGrpInstances statement.

- Each line that begins with a + represents a module and its instances.
- You can specify only one module on each line.
- You can use the wildcards \* and ?.

# **Encounter User Guide**

## Synthesizing Clock Trees

**CellHalo**  
+ *cellName* *xHalo* *yHalo* Provides spacing rules between various clock instances. It can be specified on buffers and inverters which are used by CTS to build a clock tree. It can also be specified on the gating cells.

For example, if you specify a *xHalo* of 3  $\mu\text{m}$  for BUFX4 and if there are two BUFX4 instances placed close to each other, then the x-direction spacing must be equal to or greater than 3  $\mu\text{m}$ .

If you specify 3  $\mu\text{m}$  for BUFX4 and 4  $\mu\text{m}$  for BUFX6 and if the two buffers are placed close to each other, then the spacing between these buffers is 4  $\mu\text{m}$  or greater.

**Note:** CellHalo applies to clock instances. It does not apply between a clock tree buffer and the instance which is not a part of the clock tree.

## MasterGateInst

```
+ numClusters1 instance1
+ numClusters2 instance2
+ numClusters3 instance3 instance4
+
+ ...
```

Controls instance cloning.

**Note:** To clone gating cells in groups, specify more than one instance name.

*numClusters* Indicates how many times each instance is to be cloned.

*instance* Represents the instance name.

## MasterGateModule

```
+ numClusters1 module1  
+ numClusters2 module2  
+ ...
```

## Controls module cloning:

*numClusters* Indicates how many times each module is to be cloned.

*module* Represents the module name.

**MasterGateTargetFanout** *number*

	Limits the number of cloning cells.
	The number of cloning cells is equal to the number of fanouts divided by the master gate target fanout number.
	<b>Note:</b> <code>ckCloneGate</code> uses this constraint to limit the number of cloning cells. However, the final fanout of gating cells might differ depending on the loads the gating cells drive.
EquivGateInst + <i>instance1</i> + <i>instance2</i> + <i>instance3</i> <i>instance4</i> + ...	Specifies the instances to be cloned. <code>ckDeCloneGate</code> merges the instances you specify into one instance.
End	To declone groups of gating cells, specify more than one instance on one line.
	Marks the end of an automated gated CTS section.

## Log File Headings

Given particular settings for the `RouteClkNet` and `PostOpt` statements, and for the `reportClockTree` command, the `encounter.log` file reports results in the following sections of the log file:

Clock tree specification file statements	Example clock tree text command sequences	Encounter log file section heading: <i>Clock</i> <i>clockName</i> plus—
<code>RouteClkNet NO</code>	<code>ckSynthesis</code> <code>globalDetailRoute</code> <code>reportClkTree -clk</code> <i>clock</i>	Pre-Route Timing Analysis
<code>RouteClkNet YES</code>	<code>ckSynthesis</code> <code>globalDetailRoute</code> <code>reportClkTree -clk</code> <i>clock</i> <code>-clkRouteOnly</code>	Clk-Route-Only Timing Analysis
<code>RouteClkNet YES</code> <code>PostOpt YES</code>	<code>ckSynthesis</code> <code>globalDetailRoute</code> <code>reportClkTree -clk</code> <i>clock</i> <code>-postRoute</code>	Post-Route Timing Analysis

## CTS Report Descriptions

The standard CTS report (*top\_level\_cell.ctsrpt*) contains several sections by default, and several sections that are controlled by the settings of various CTS text commands.

**Note:** The report extracts that follow are based on various designs, and should not be construed as results from a single CTS run.

### General Information

The beginning of each CTS report contains the following general information about the clock tree:

#### ■ Library information

```
#####
# Complete Clock Tree Timing Report
#
# CLOCK: cgen/i_5/Y
#
# Mode: preRoute
#
# Library Name      : slow
# Operating Condition : slow
# Process           : 1
# Voltage           : 1.62
# Temperature       : 125
#
#####
#
```

#### ■ Clock tree structure information

Nr. of Subtrees	:	1
Nr. of Sinks	:	343
Nr. of Buffer	:	9
Nr. of Level (including gates)	:	2
Max trig. edge delay at sink(F):	TPRAM/mod166798/CK	477.7(ps)
Min trig. edge delay at sink(R):	TPRAM/mod167332/CK	459.6(ps)

#### ■ Delay, skew, and transition information

	(Actual)	(Required)
Rise Phase Delay	:	459.6~477.7(ps)
		0~5000(ps)

# Encounter User Guide

## Synthesizing Clock Trees

---

Fall Phase Delay	: 432.8~446.7(ps)	0~5000(ps)
Trig. Edge Skew	: 18.1(ps)	250(ps)
Rise Skew	: 18.1(ps)	
Fall Skew	: 13.9(ps)	
Max. Rise Buffer Tran	: 238.5(ps)	550(ps)
Max. Fall Buffer Tran	: 141.4(ps)	550(ps)
Max. Rise Sink Tran	: 366.2(ps)	550(ps)
Max. Fall Sink Tran	: 204.5(ps)	550(ps)
Min. Rise Buffer Tran	: 120(ps)	0(ps)
Min. Fall Buffer Tran	: 120(ps)	0(ps)
Min. Rise Sink Tran	: 340.6(ps)	0(ps)
Min. Fall Sink Tran	: 192(ps)	0(ps)

### ■ Maximum transition time violations

\*\*\*\*\* Max Transition Time Violation \*\*\*\*\*

Pin Name	(Actual)	(Required)
reg/CK	[406 353.5](ps)	400(ps)
reg2/CK	[406 353.4](ps)	400(ps)
clk0__L6_I2/A	[345.5 288.1](ps)	300(ps)
clk0__L7_I4/A	[346.2 296.3](ps)	300(ps)
clk0__L9_I11/A	[351.6 299.9](ps)	300(ps)
clk0__L9_I10/A	[361.5 305.9](ps)	300(ps)

### ■ Skew distribution information

cgen/i\_5/Y delay[0 0] ( CK\_\_L1\_I0/A )

\*\*\*\*\* Skew Distribution \*\*\*\*\*

LEVEL 1 Buffer:

Input Delay Range	Nr of Buffers
[0.6 0.6]	1
(max, min, avg, skew) = (0.6(ps) 0.6(ps) 0.6(ps) 0(ps))	

-----

Output Delay Range                           Nr of Buffers

[195.5 195.5]	1
(max, min, avg, skew) = (195.5(ps) 195.5(ps) 195.5(ps) 0(ps))	LEVEL 2 Buffer:

-----

Input Delay Range                           Nr of Buffers

## Encounter User Guide

### Synthesizing Clock Trees

```
[212.8 212.8]           1
(max, min, avg, skew) = (212.8(ps) 212.8(ps) 212.8(ps) 0(ps))

-----
Output Delay Range          Nr of Buffers
-----
[414.5 414.5]           1
(max, min, avg, skew) = (414.5(ps) 414.5(ps) 414.5(ps) 0(ps))
```

## Macro Model Information

Information on macro models appears in the standard report:

```
Max trig. edge delay at sink(R): AClk 4971(ps) *Mmodel*
Min trig. edge delay at sink(R): AClk 4671(ps) *Mmodel*
```

\*Mmodel\* Trig. edge delay calculation uses MacroModel for the sink pin.

## Power Information

Power information appears at the end of the general section of the CTS report, immediately after the transition information. The software reports power information only if you specify `setCTSMode -powerAware true` and include the `Period` statement in the clock tree specification file.

CTS reports the total net switching power, internal clock instances power, internal leaf pin power, and leakage power for the clock.

CTS calculates internal power by multiplying the activity of the clock net by the `ClockGatingProb` statement value. The activity of the clock net is taken from the clock `Period` statement in the clock tree specification file. The `ClockGatingProb` statement value is specified in the clock tree specification file, and is used for statistical estimation of power.

For example,

```
AutoCTSRootPin sysclk
MaxDelay      4ns
MinDelay      0ns
MaxSkew       0.2ns
...
ClockGatingProb 0.1
Period        100ns
```

END

If you do not specify a value for the `ClockGatingProb` statement, CTS uses the default value of 1.

If you have a single clock gate at the root, and all of the buffers are after the clock gating cell, CTS calculates the activity rate as:

*(activity of clock net at gate's input) x ClockGatingProb*

CTS computes the activity rate of a net driven by a buffer in the clock path as:

*(activity of clock net at buffer's input) x ClockGatingProb*

CTS uses the same net activity for calculating the switching power of the net.

NetSwitchPower	:	0.000997444 (mW)
InstInternalPower	:	0.00409096 (mW)
InstLeakagePower	:	1.9129e-06 (mW)
LeafPinInternalPower	:	0.0013708 (mW)
Total Power	:	0.00646111 (mW)

## AC Current Density Violations

Information on AC current density violations appears in the standard CTS report only if your LEF file contains an `ACCURRENTDENSITY` statement.

AC Irms Limit Violation : 0.387332(mA) (17216.7%)

Information on AC current density violations appears in a special violations section:

***** AC Irms Limit Violation *****			
Pin Name	(Actual)	(Required)	(Violation)
ClkMux/Y	0.389581(mA)	0.00224974(mA)	0.387332(mA)
scanClk/Y	0.156409(mA)	0.00224974(mA)	0.154159(mA)

## Supported SDC Constraints

Clock Tree Synthesis supports only the following SDC timing constraints during tracing (`setCTSMode -traceHonorConstants true`):

- `set_logic_one`
- `set_logic_zero`
- `set_case_analysis`
- `set_disable_timing`

The `createClockTreeSpec` command automatically translates certain SDC constraints into clock tree specification file statements. The following table shows the SDC constraints that the `createClockTreeSpec` command automatically translates, and also shows the default values in those statements if an SDC constraint does not exist:

<b>SDC Constraint</b>	<b>Clock Tree Specification File Statement (default)</b>
<code>create_clock</code>	<code>AutoCTSRootPin</code>
<code>set_clock_transition</code>	<code>SinkLeafTran</code> and <code>BufMaxTran</code> (Default: 400 ps)
<code>set_clock_latency value</code>	<code>MaxDelay</code> (Default: clock period) <code>MinDelay</code> (Default: 0)
<code>set_clock_latency -source value</code>	<code>SrcLatency value ns</code>
<code>set_clock_uncertainty</code>	<code>MaxSkew</code> (Default: 300 ps)
<code>create_generated_clock</code>	Add <code>ThroughPin</code> when necessary

## **Encounter User Guide**

### Synthesizing Clock Trees

---

---

## Working with Clock Mesh Structures

---

- [Overview](#) on page 636
- [Clock Meshes Versus Clock Trees](#) on page 636
- [Creating Clock Meshes](#) on page 639
  - [Determining the Mesh Structure](#) on page 639
    - [Supported Mesh Styles](#) on page 639
    - [Clock Mesh Structure Characteristics](#) on page 641
    - [Multilevel Structure of a Mesh](#) on page 643
  - [Implementing the Clock Mesh](#) on page 644
  - [Analyzing the Clock Mesh](#) on page 645
    - [Pre-Route Wire Estimation](#) on page 645
    - [RC Extraction](#) on page 645
    - [Computing Mesh Delays](#) on page 646

## Overview

The Encounter software provides a semi-automatic way to synthesize clock mesh structures by automating the tasks of netlist update, physical implementation and analysis.

Clock meshes typically provide tighter skew control and limit the impact of process variation compared to clock trees. However, these advantages might be offset by increased routing resource usage and increased power dissipation due to larger switching capacitance.

## Clock Meshes Versus Clock Trees

There are advantages and drawbacks to using clock meshes instead of clock trees. Consider the following factors when determining whether a clock mesh is appropriate for a given block or clock domain.

- Skew

Clock meshes can often deliver lower skew than a clock tree. By their nature and design, clock meshes deliver low skew to their leaf inputs. However, if any leaves require different clock arrival times (such as macro-models with different built-in insertion delays or useful skew), then clock mesh alone will not deliver a good overall solution. Clock mesh does not directly support early or late clocks.

**Note:** It might be possible to implement early and late clocks by hand for a limited number of leaves.

- Insertion delay

Because meshes can use multiple buffers driving in parallel, they can potentially fan out to the clock inputs with fewer stages and lower insertion delay than a tree.

If clock tree synthesis can meet the performance targets (skew and insertion delay), considering the effects of on-chip and process variation, there might not be a compelling reason to consider a clock mesh. CTS is currently more highly automated than clock mesh synthesis, and typically uses less power. However, if CTS cannot meet the skew or insertion delay constraints, it might be worth considering a clock mesh.

- Tolerance to variation

Mesh structures are generally less sensitive to on-chip variations (OCV) and process variations than corresponding trees.

- Power

## Encounter User Guide

### Working with Clock Mesh Structures

---

Generally meshes can consume somewhat more power than trees, although the amount of extra power is highly design dependent. In some cases, clock meshes might actually consume less power than trees. Because most power typically is consumed at the final (leaf) level, using local distribution can significantly narrow the power gap between mesh and tree implementations.

#### ■ Gating

Clock mesh structures are not as flexible as clock trees, with respect to gating. Because the final mesh stage is a single net, gating must be implemented either at the root level, or the local level.

#### ■ Floorplan limitations

Clock meshes rely on arranging buffers and routing in regular symmetric patterns in order to achieve good skew. When constructing a mesh, the software tries to adjust positions of individual trunks and branches slightly in order to avoid conflicts and violations with existing placement or routing blockages, or pre-routes such as power stripes. Therefore, clock meshes work best in floorplan areas that are rectangular and relatively free of obstructions. Highly non-rectangular floorplans, such as L-shaped areas, or areas with macros or obstructions placed in the middle of the floorplan, can make it difficult or impossible to implement a mesh. Always evaluate the floorplan to determine if it looks feasible to insert a mesh.

#### ■ Degree of automation

Clock tree synthesis is usually 100 percent automatic; you specify the constraints, and the tool does all of the work. Currently, clock mesh is not fully automatic. You must choose the structure of the clock mesh in terms of style, number of levels, and so on. Also, depending on the floorplan and obstructions, you might need to experiment with different implementation parameters in order to find a feasible solution.

#### ■ Available Types of Driver Cells

Although the clock mesh feature is intended to work with arbitrary driver cells, some buffer or inverter cells are better suited as mesh drivers than others. The ideal situation is to have specially designed mesh drivers, but if such cells are not available, usually the best approach is to select from the normal clock buffer cells used for CTS.

There are two primary aspects to consider when designing or selecting a mesh driver cell: its electrical or timing characteristics, and the geometry of its output pad.

- Electrical Characteristics

Clock mesh driver cells have the following electrical characteristics:

- Drive strength

## Encounter User Guide

### Working with Clock Mesh Structures

---

Clock mesh nets typically have higher loading than the nets in clock trees. Even though meshes can arrange many drivers in parallel to drive large loads, it is usually preferable to have higher strength drivers available for clock meshes. In addition to simplifying the design, having fewer drivers in parallel often improves the performance and memory usage of delay calculation and timing analysis.

- Multiple row cells

Very strong mesh drivers draw large currents from the power supply rails. Arranging drivers to span multiple rows can reduce their impact on the power distribution system.

- Decoupling capacitors

Consider including decoupling capacitors inside the mesh driver to help supply transient current when the driver is switching.

- Output via stacks

Consider integrating via stacks up to mesh routing layers within the mesh driver cell. Including the stack within the cell can help ensure that it has sufficient current carrying ability and has minimal impact on routing resources.

- Balanced rise and fall

As with normal clock buffers, balanced rise and fall characteristics are important for clock mesh drivers to maintain duty cycle, and so on.

- Output Pad Geometry

The geometry of driver output pads is important because the clock mesh relies on direct driver-trunk connections. It does this by placing the driver underneath the trunk in such a way that either the wire directly connects to the output pad shape, or it crosses above it on a higher layer and connects with a stacked via.

Consider the following output pad shape guidelines when choosing (or designing) a driver cell:

- Use a single rectangle pad

Complex pad geometries composed of many shapes usually make it more difficult to locate and drop a clean via stack onto the pad.

- Use pads on the mesh layer

Connections to pads on the trunk or branch routing layers are easier because they do not require additional via stacks.

- Be careful when input pin is on the mesh layer

If the input pin is on the mesh routing layer, it must be possible to connect to the output without shorting to the input. For example, consider a driver with input and output both on  $M2$ . If the mesh is using a vertical trunk on  $M2$  and the input and output are aligned vertically, it might not be possible to make the trunk connection without a short or spacing violation.

## Creating Clock Meshes

Creating a clock mesh generally consists of the following tasks:

1. [Determining the Mesh Structure](#) on page 639
2. [Implementing the Clock Mesh](#) on page 644
3. [Analyzing the Clock Mesh](#) on page 645

### Determining the Mesh Structure

Similar to clock tree synthesis, the clock mesh feature uses a “specification” to define the scope of the clock domain, express constraints, and control the structure of the mesh. In addition to loading and saving specification files, you can also interactively edit the specification using the [Edit Clock Mesh Specification](#) form. This makes it easier to experiment with different clock mesh structures.

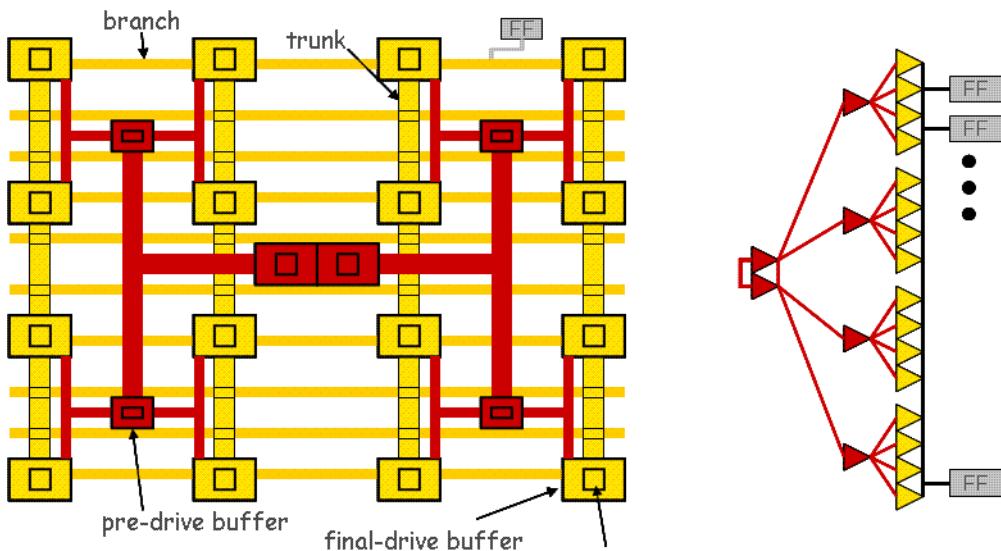
### Supported Mesh Styles

There is a wide variety of mesh styles currently in use. The Encounter software can synthesize the following mesh styles:

- H-tree + Mesh

This classic style uses a multilevel H-tree pre-drive, followed by a general mesh final stage. Although the pre-drive is a tree structure, it can still employ multiple drivers on a single net. The final stage consists of a rectangular grid of final drivers feeding a rectangular mesh grid of trunks and branches.

**Figure 17-1 H-Tree + Mesh Style**

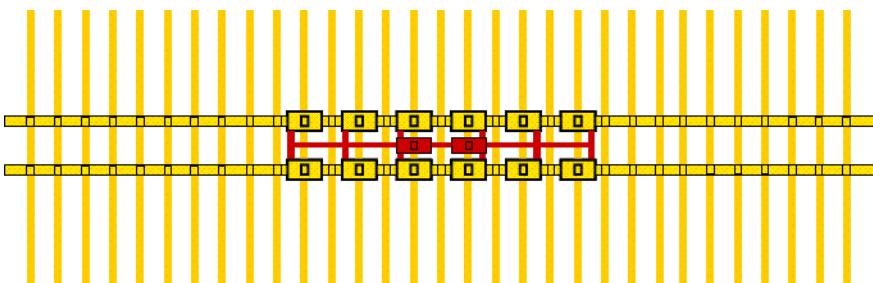


The advantage to this style is that the pre-drive is highly symmetrical and can achieve good skew control for large clock domains with many flip-flops. The drawback is that it can require higher power than other mesh styles.

#### ■ Fishbone

The final stage of a basic fishbone structure uses multiple drivers feeding a single trunk (spine) that, in turn, drives a number of orthogonal branches (bones). Pre-drive stages consist of multi-driven pre-drive trunks placed sufficiently near the next-stage driver inputs.

**Figure 17-2 Single and Double Fishbone Style**



Additionally, there is a double-fishbone variant, in which the final stage uses two parallel trunks. The fishbone style can be a very good style for smaller clock domains.

## Clock Mesh Structure Characteristics

There also is a wide variety of meshed clock distribution structures currently in use. The clock mesh structures synthesized by the Encounter software have the following basic characteristics:

- Symmetry

Meshes rely heavily on symmetry of netlist structure and routing pattern to achieve tight skew control and tolerance to variation.

- Routing patterns and topologies

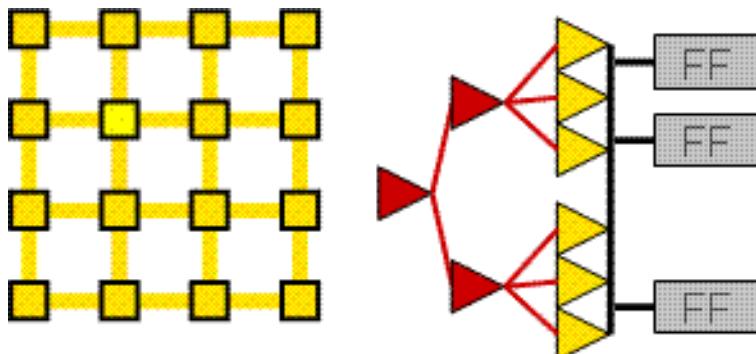
A major difference between clock meshes and clock trees are their routing patterns. Mesh routing commonly has the following characteristics:

- Makes heavy use of non-tree routing topologies containing cycles or loops.
- Uses wide wires to achieve low resistance and well controlled capacitance. For example it is not unreasonable for a mesh trunk to be 10  $\mu\text{m}$  wide.
- Routing is implemented using a combination of special and regular routes.

- Parallel drive

Clock meshes typically rely on multiple buffers or inverters working in parallel to drive a given net in the pre-drive and final-drive stages, whereas clock trees always use exactly one driver per net. Using multiple parallel drivers makes it possible to drive heavily loaded nets from multiple points. This allows a mesh to fan out with fewer stages and better skew control than a tree.

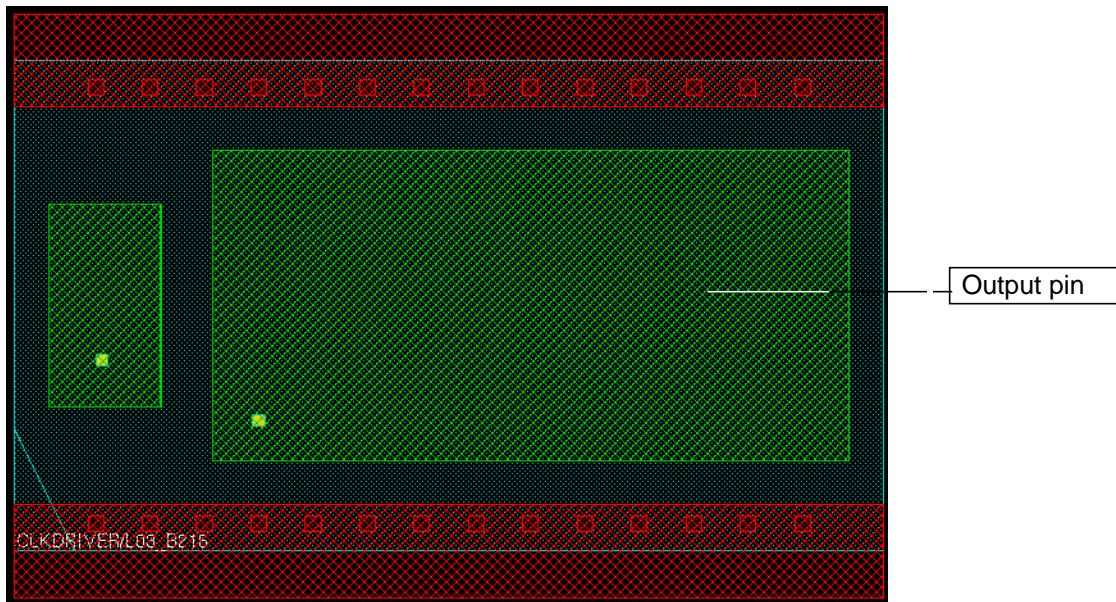
**Figure 17-3 Parallel Drive with Cycle Routing Pattern**



- Driver output connections by abutment

The connections between the mesh driver output pins and trunks are made by placing the trunk wire directly over the output pad, and possibly dropping a stacked via if required. This method ensures a strong connection that is capable of carrying the current supplied by the driver.

**Figure 17-4 Driver Cell with Large Output Pin for Abutment Connection**



- Input connections completed with regular routing

In contrast to driver output pins, which are connected directly to mesh trunks, input pins, such as driver inputs and flip-flop inputs in the final stage, are connected to mesh trunks and branches by regular routes created by the detail router (NanoRoute). Because individual input connections do not need to carry large currents, minimum width connections are generally sufficient. Nondefault rules can be used if wider widths or larger spacings are desired.

- Drivers placed in core area

Mesh driver cells are placed in the core area in rows like other standard cells, rather than being specially placed macro block cells. Large drivers can span multiple rows. Because mesh driver connections are made by abutment, driver cell placement is typically “fixed” to keep them from being unintentionally moved (and disconnected) during subsequent operations.

## Multilevel Structure of a Mesh

Like clock trees, clock meshes typically use a multilevel structure to fan the clock signal out from the root to all the clock input pins (flip-flops, memories, and so on). The Encounter software creates multilevel meshes that can include the following sections:

- Top-level chain

The top-level chain is a cascaded buffer chain from the mesh root to the first level of mesh pre-drive buffers. Chains can be used either to supply a suitable input transition to the mesh pre-drive, or to pad the mesh with extra insertion delay.

The routing for mesh chain nets is handled entirely by the NanoRoute router rather than by using a combination of special and regular routes. If wide routing is required for mesh chain routes, use LEF nondefault rules. Meshes are not required to include a top-level chain; the root can connect directly to the first pre-drive stage, if it is suitable.

- Global mesh

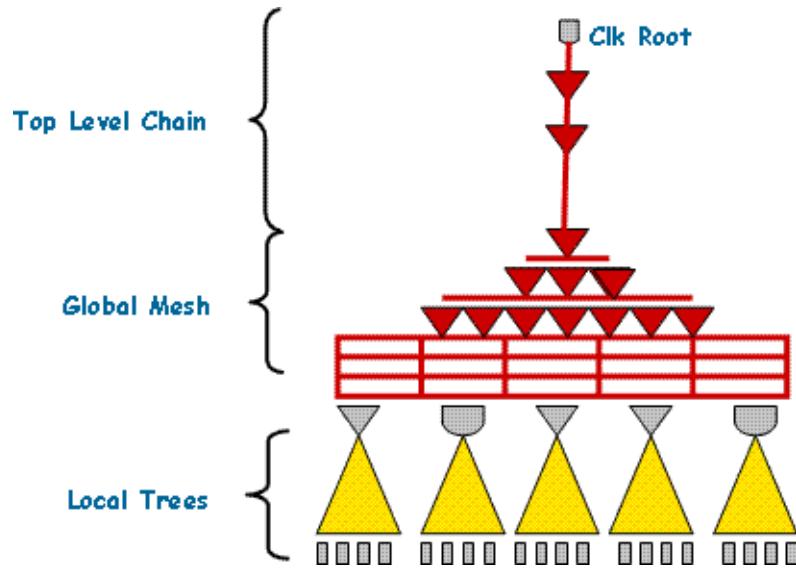
The purpose of the global mesh is to distribute a single clock signal across the entire clock domain with good insertion delay and skew control. The global mesh consists of multiple (zero or more) pre-drive stages, followed by a single final-drive stage. While the pre-drive stages can have multiple nets at a given level (for example in an H-tree), the final stage always drives a single final global mesh net. This final mesh net can connect directly to the clock input pins, or there can be an additional level of local distribution.

- Local distribution

Local distribution is an optional section in which multiple small trees distribute the single global signal of the final mesh drive net to individual flip-flops or memory inputs. The simplest form of local distribution consists of multiple small single-buffer “local trees,” each driving a cluster of flip-flops. NanoRoute handles all of the local-level routing; there are no special routes.

Using local distribution can have advantages over direct mesh-to-flip-flop connections in the final stage. For example, local distribution significantly reduces the loading in the final mesh stage, and can allow for a mesh structure with lower overall power consumption. Also, for large clock domains with non-uniform flip-flop distributions, adding extra local buffers can help to balance the load seen by the mesh, and allow for better skew control.

**Figure 17-5 Multilevel Mesh Structure**



## Implementing the Clock Mesh

Implementing a clock mesh involves various tasks, such as inserting drivers into the netlist, placing and “fixing” the drivers, creating mesh routes, and so on. The Encounter clock mesh feature includes the following major implementation capabilities:

- Synthesis

The `synthesizeClockMesh` command implements the clock mesh according to the current specification, by inserting and placing buffers, and creating mesh special routes as needed.

- Mesh routing

The `routeClockMesh` command uses the native NanoRoute™ router to complete detailed routing connections for driver and flip-flop inputs, top-level cascade chains, and local trees.

- Wire trimming

After mesh routing is complete, the `trimClockMesh` command removes unused portions of the mesh trunks and branches, eliminates antenna violations, and reduces overall mesh capacitance.

## Analyzing the Clock Mesh

The Encounter software can analyze clock meshes at different stages of the implementation process (pre-route or post-route), using default extraction, detail extraction, or externally supplied RC data, and using either delay calculation technology or a circuit simulator. Various reports are available. To help visualize mesh quality, delay or transition information can be displayed graphically using a color scale.

Clock meshes have several characteristics that make them more difficult to analyze than clock trees:

- Meshes use a mixture of special routing and regular routing, which makes estimating and extracting wire RC information more challenging.
- Nets with multiple drivers need special consideration during delay calculation.
- A final stage clock mesh net might have thousands of drivers and tens of thousands of receivers, resulting in tens (or hundreds) of millions of driver-to-receiver timing arcs. This huge number of arcs can potentially lead to memory or performance problems during timing analysis.

The following sections describe how the Encounter software addresses some of the unique challenges to analyzing clock mesh timing.

### Pre-Route Wire Estimation

The software's clock mesh synthesis inserts and places buffers and draws mesh trunks and branches, but it does not immediately start NanoRoute to complete detail routing connections for mesh driver inputs and flip-flops; this is handled by a separate command. Therefore, some wire estimation is needed to complete pre-route clock mesh analysis.

For nets without any special net routing, such as top-chain nets and local distribution nets, the software uses a simple two-layer Steiner routing estimation based on the routing preferences given in the clock mesh specification.

For global nets, the software builds an RC network for the existing special net routing and then estimates nearest connections for each unconnected driver input and flip-flop. For these nets, nearest point-to-point connections are more appropriate than Steiner routing estimates because NanoRoute will connect them using “pattern trunk” connections.

### RC Extraction

There are several different ways to get RC information for clock mesh analysis:

## Encounter User Guide

### Working with Clock Mesh Structures

---

#### ■ Default extraction

Default extraction is simple 1-D extraction based on per-unit-distance resistance and capacitance values for mesh routes. Default extraction is most appropriate before full detailed routing is complete.

**Note:** Default extraction is the only available method to analyze pre-route clock mesh nets.

#### ■ Detail extraction

The Encounter software detail extraction can be used when mesh routing is complete.

**Note:** The clock mesh analysis commands will not automatically start detailed net extraction; to use this method, you must directly set the extraction mode then run the extractRC command prior to mesh analysis.

#### ■ External SPEF

To use RC data from an external parasitic extraction tool, load the SPEF prior to clock mesh analysis.

## Computing Mesh Delays

There are two ways of computing mesh delays and transitions times: delay calculation and circuit simulation.

#### ■ Delay Calculation

The fact that clock meshes rely on multiple buffers driving a single net presents some challenges for delay calculation. Although not all delay calculators can handle it, Encounter clock mesh uses delay calculation technology that supports multi-driven nets. For delay calculation to be accurate, all drivers on a given net must have similar input waveforms. This restriction is consistent with a well-designed mesh and probably does not present a major limitation in practice. The software generates warnings when the skew at multiple driver inputs exceeds a given fraction of their input slew.

#### ■ Circuit Simulation

The software supports simulation with Virtuoso® UltraSim™ fast SPICE simulator as an alternative to delay calculation for analyzing clock mesh delays and slews. Virtuoso® UltraSim™ simulator is a fast-SPICE simulator that is well-suited for analyzing clock meshes, even with post-route RC data. Virtuoso UltraSim simulator is not packaged with the Encounter software, therefore to use it, you must have the Virtuoso UltraSim simulator executable in your path, and an appropriate license.

## **Encounter User Guide**

### Working with Clock Mesh Structures

---

The software also supports simulation with external SPICE-like simulators, but the flow is not as convenient as with Virtuoso UltraSim simulator. The steps are as follows:

- a. Dump a SPICE netlist for the clock mesh.
- b. Manually run the simulator outside of the Encounter software.
- c. Backannotate the simulation measurement results. Encounter clock mesh provides commands to write the spice netlist and backannotate the results.

To simulate clock meshes, either with Virtuoso UltraSim simulator or with an external simulator, the Encounter software needs to have driver sub-circuit and model information available. This information must be provided via a plain-text cdB database file.

**Encounter User Guide**  
Working with Clock Mesh Structures

---

---

## Editing Wires

---

- [Overview](#) on page 651
- [Before You Begin](#) on page 652
- [Results](#) on page 652
- [Using Keyboard Shortcuts](#) on page 652
- [Moving Wires](#) on page 655
- [Adding Wires](#) on page 657
- [Deleting Wires](#) on page 655
- [Cutting Shielding Wires](#) on page 662
- [Trimming Antennas on Selected Stripes](#) on page 662
- [Changing Wire Width](#) on page 663
- [Repairing Maximum Wire Width Violations](#) on page 664
- [Duplicating Wires](#) on page 664
- [Stretching Wires](#) on page 665
- [Changing Wire Layers](#) on page 665
- [Splitting and Merging Wires](#) on page 666
- [Adding Vias](#) on page 666
- [Changing Vias](#) on page 667
- [Moving Vias](#) on page 668
- [Reshaping Routes](#) on page 668
- [Controlling Cell Blockage Visibility](#) on page 669
- [Editing Wires with Virtuoso Chip Assembly Router](#) on page 670

## **Encounter User Guide**

### Editing Wires

---

- [Running Virtuoso CAR in Batch Mode](#) on page 670
- [Running Virtuoso CAR in Interactive Mode](#) on page 671

## Overview

You can edit the wires and vias in your design manually by using the [Edit Route](#) form, the wire editing commands, and a combination of keyboard shortcuts (bindkeys) and tool widgets.

For signal wires you can perform the following actions:

- Add wires
- Cut wires
- Move wires
- Change the wire to another layer
- Delete wires
- Merge selected wires
- Force wires to use specified widths
- Add vias

For power wires, you can perform all of the actions available for signal wires, as well as the following additional actions:

- Trim selected wires
- Split selected wires
- Duplicate selected wires
- Change wire width
- Fix wires wider than the maximum width
- Force wires associated with special nets to be created as signal wires
- Change vias

## Before You Begin

Before you can use the wire editing features, load the design into the current Encounter session.

## Results

After you use the wire editing features, the Encounter software saves the new and modified wires and vias in the database.

## Using Keyboard Shortcuts

The Encounter software includes keyboard shortcuts for use with the wire editing features. Type the keyboard shortcuts while the main Encounter window is active and the cursor is in the design display area. Some of the keyboard shortcuts provide functionality that is not available through the Edit Route form or the wire editing commands.



To display a list of all the Encounter keyboard shortcuts, open the Binding Key form. Select *Design – Preferences* from the Encounter menu, then click the *Binding Key* button on the Preferences form.

### Keyboard Shortcuts That Open Forms

Click in the design display area, then use one of the following shortcuts:

- d    Opens or closes the Select/Delete/Deselect Route form
- e    Opens or closes the Edit Route form

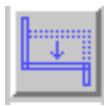
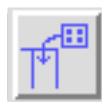
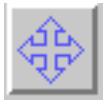
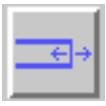
### Keyboard Shortcuts That Are Equivalent to Tool Widgets

- a        *Select*

## Encounter User Guide

### Editing Wires

---

- A  *Add Wire*
- m  *Move Wire*
- o  *Add Via*
- R  *Move/Resize/Reshape  
(non-connectivity-based move)*
- S  *Stretch Wire*
- X  *Cut Wire*

For more information, see “[Tool Widgets](#)” in the *Encounter Menu Reference*.

### Keyboard Shortcuts Used in Auto Query Mode

- n Toggles to next object under cursor.
- p Toggles to previous object under cursor.
- s Populates the Edit Route form with net name, width, layers, and shape of highlighted (queried) wire or pin. The *Nets* field of the Select/Delete Routes form is also populated.  
  
If the queried object is a pin, the layer and width information is set for both horizontal and vertical directions. If the queried object is a wire, the width information is set for both horizontal and vertical directions, but only one of the layers is set. That is, only the horizontal layer is set for a horizontal wire and only the vertical layer is set for vertical wires. This keyboard shortcut does not populate the form with spacing information.

Control-w

Deletes the queried segment or via.

These keyboard shortcuts work only while you are in *auto query* mode—they do not work while you are drawing a wire. For more information, see “[Auto Query](#)” in the *Encounter Menu Reference*.

## Keyboard Shortcuts Used in Add Wire Mode

d	Changes the added wire to the layer below the current layer.
u	Changes the added wire to the layer above the current layer.
Control-w	Deletes the last segment created in the design area. This allows you to remove one segment of the route at a time.
Esc	Removes the entire route.
Number keys	Change the added wire to a specific layer number. If you want the wire to be added to metal layer 1, use the 1 keyboard shortcut, use the 2 keyboard shortcut for metal layer 2, and so forth.
Single-click	Ends the segment, allowing you to continue the route in either the same direction or the orthogonal direction.
Double-click	Ends the route.

## Keyboard Shortcuts Used in Stretch Wire Mode

- 1 Stretches or reduces horizontal wires from the left and vertical wires from the bottom.
- 2 Stretches or reduces horizontal wires from the right and vertical wires from the top.

For more information, see “[Stretching Wires](#)” on page 665.

## Keyboard Shortcuts Used to Change Vias

- N Changes the selected via to the next available via.  
P Changes the selected via to the previous available via.

For more information, see “[Changing Vias](#)” on page 667.

## Selecting Wires

1. Click the *Move Wires* widget in the Tool Widgets area of the Encounter main window or press the **m** keyboard shortcut while the cursor is in the design display area.
2. Click a wire.

## Deleting Wires

To delete a wire without deleting the vias connected to it, complete the following steps:

1. Turn on *Auto Query*.
2. Move the cursor over the wire to delete.
3. Use the **n** (next) or **p** (previous) keyboard shortcut to move select the correct wire.
4. Press **Control-w** or the **DEL** key.

During post-mask ECO, you can freeze wires and vias by changing their status to COVER. The Encounter software does not delete wires or vias with status COVER. Type the following commands to freeze the wires and vias on metal layers 1 and 2:

```
deselectAll  
editselect -layer {M1 M2}  
editSelectVia -cut_layers V12  
editChangeStatus -to COVER
```



You cannot change the wires and vias back to their non-COVER status, so you should issue the **saveSpecialRoute** and **saveRoute** commands before changing the status.

## Moving Wires

You can use the mouse or the keyboard arrow keys (in conjunction with the **Shift** key) to move wires in the orthogonal direction.

### Using the Mouse to Move Wires

1. Click the *Move Wires* icon in the Tool Widgets area of the Encounter main window.

The equivalent keyboard shortcut is **m**.

2. Click the wire to be moved.

The selected wire is highlighted.

3. Move the cursor slightly within the selected wire.

The cursor changes to a circle shape.

4. Click and release the mouse.

The wire moves with the cursor in the orthogonal direction (up or down for a horizontal wire, left or right for a vertical wire). Wires connected to the moved wires stretch to maintain connectivity.

5. Click the mouse again to place the wire in the new location.

**Note:** To cancel the move before you click the mouse, press the `Esc` key. The wire returns to its original location.

**Note:** If you select the *Snap to Track* option, the wire automatically snaps to the appropriate routing track.

## Using Arrow Keys to Move Wires

1. Choose *Edit - Edit Route* from the menu.

The Edit Route form opens.

The equivalent keyboard shortcut is `e`.

2. Click the *Route* tab.

3. Specify a value, in microns, in the *Arrow increment* field.

This value defines the distance that the wire is to move each time you press an arrow key while holding the `Shift` key. You can specify either a positive or negative number.

4. Click the *Move Wire* icon in the Tool Widgets area of the Encounter main window.

The equivalent keyboard shortcut is `m`.

5. Click the wire to be moved.

The selected wire is highlighted.

6. Hold the `Shift` key, then press the up or down arrow key for a horizontal wire, or the left or right key for a vertical wire.

The selected wire moves in the direction of the arrow.

## Moving Selected Wires or Vias

To copy, paste, and move selected wires or vias, complete the following steps:

1. Select wires or vias.
2. Type the `editDuplicate` command (or use the `C` keyboard shortcut) to copy the objects.
3. Use the `R` keyboard shortcut to switch to *unconn move* mode.
4. Move the mouse over any of the selected objects. A black dot appears.
5. Click once to start moving the selected objects.
6. Click again to place the objects in the desired location.

**Note:** To cut, paste, and move the wires or vias, skip step 2.

## Adding Wires

You can add one or more wires interactively to single or multiple nets. When you add wires, the flight lines to routed pins are displayed in the pin color (by default, yellow) and flight lines to unrouted pins are displayed in the wire color (by default, blue).

By default, the routing status for newly added signal wires is **FIXED**. A **FIXED** routing status means that the automated routers do not rip up and reroute preroutes. Signal wires that are moved, cut, or otherwise changed by wire editing commands maintain the routing status that was set before the wire editing commands were issued.

### Adding a Wire for a Single Net

1. Click the *Add Wires* widget.

This places the Encounter software in *add wire* mode and the mouse cursor changes to a pencil. In addition, Encounter is automatically placed in *auto query* mode, even if the `Q` widget below the design display area is not enabled.

The equivalent keyboard shortcut is `A`.

2. If pins are not visible, use the *Layers* tab of the [Color Preferences](#) form to make pins visible.
3. Place the cursor over the pin or wire at the starting point for the wire to be drawn, and then type `S` while the cursor is in the design display window.

## Encounter User Guide

### Editing Wires

---

This populates the *Edit Route* form with the net name, layer, and width information that is used in creating new wires.

**Note:** If multiple objects exist at a particular point, use the `n` or `p` keyboard shortcut to cycle through the objects.

4. (Optional) Choose *Edit – Edit Route* from the menu or use the `e` keyboard shortcut.

The *Edit Route* form opens, and has been automatically populated with the net name, layers, and widths. The form is not populated with spacing information, which only applies while editing multiple nets.

5. (Optional) Click the *Route* tab on the *Edit Route* form and adjust the values in the Layer and Width fields.

6. (Optional) Click the *Misc* tab and select a shape from the pulldown menu.

**Note:** Shapes are only defined for power wires. This value is ignored for signal wires.

7. Click the startpoint for the wire you want to add, then move the mouse to a new point.

The wire is drawn interactively as you move the mouse.

8. (Optional) Click a new location to change the direction of the wire or continue in the same direction with a different segment.

**Note:** If there is a layer change, a via is automatically created.

9. (Optional) Press a number key to change the layer of the wire being added.

When the software is in *add wire* mode, number keys can be used as keyboard shortcuts, with the number indicating the layer number of the wire being drawn. For example, if you press the number 2, the segment is added to metal layer 2. Alternatively, you can use the `u` or `d` keyboard shortcuts to change the layer of the segment. The `u` keyboard shortcut changes the segment to the next higher layer and the `d` keyboard shortcut changes the segment to the next lower layer.

10. (Optional) Press the `Backspace` key to erase the most recently drawn segment.

You can do this for as many segments as needed.

11. Double-click the mouse.

The wire ends at the location of the cursor.

**Note:** After double-clicking, you cannot use the `Backspace` key to erase segments that you drew. Instead, click the undo widget to remove the entire route, or use the *Edit Delete* form.

## Adding Wires for Multiple Nets

To add parallel wires to multiple nets at the same time, complete the following steps:

1. Click the *Add Wires* widget.

This places the Encounter software in *add wire* mode and changes the mouse pointer to a pencil.

2. Choose *Edit – Edit Route* from the menu.

The *Edit Route* form opens.

3. Click the *Nets* tab on the *Edit Route* form and enter the net names into the *Nets* field.

**Note:** You can also specify a file that contains a list of nets. See “[Adding Wires that Automatically Extend to a Target](#)” on page 660 for more information.

4. (Optional) Click the *Route* tab, then select horizontal and vertical layer names and specify width and spacing values.

**Note:** To use different width or spacing values for different nets, use the *Override* tab. See “[Using Override to Add Wire Groups with Multiple Widths and Spacing](#)” on page 661 for more information.

5. (Optional) Specify a value in the *Drawing wire* field.

This specifies which of the nets (specified in the *Nets* field) corresponds to the mouse pointer location. By default, this value is 1, meaning the mouse position corresponds to the position of the left-most or bottom-most net of the group.

For example, if the *Nets* field contains VSS VDD VDDA VSSA, the VSS net is the bottom-most net for horizontal segments, and the left-most net for vertical segments. If the value in the *Drawing wire* field is set to 1, the mouse location corresponds to wires on the VSS net.

6. (Optional) Click the *Misc* tab and select a shape from the pulldown menu.

**Note:** Shapes are only defined for power wires. This value is ignored for signal wires.

7. Click the startpoint for the wires you want to add, then move the mouse to a new point.

The wires are drawn interactively as you move the mouse.

8. (Optional) Click a new location to change the direction of the wires or to continue in the same direction with a different segment.

**Note:** If there is a layer change, a via is automatically created.

9. (Optional) Press the Backspace key to erase the most recently drawn set of segments.

You can do this for as many sets of segments as needed.

**10.** Double-click the mouse.

The wires end at the location of the cursor.

**Note:** After double-clicking, you cannot use the `Backspace` key to erase segments that you drew. Instead, click the undo widget to remove the entire route, or use the `Edit Delete` form.

## Adding Wires that Automatically Extend to a Target

To add wire groups to multiple nets that automatically extend to targets, complete the following steps:

**1.** Click the *Add Wires* widget.

This places the Encounter software in *add wire* mode and changes the mouse pointer to a pencil.

**2.** Choose *Edit – Edit Route* from the menu.

The Edit Route form opens.

**3.** Create a text file that contains the names of multiple nets.

Make sure that each line in the file contains the name of one net, and that the nets are listed in the order in which you want to create the wire group.

**4.** Click the *Nets* tab on the *Edit Route* form.

**5.** Click the *Read* button.

This opens the *Select A File* form.

**6.** Select the file you created in step 3, then click *OK*.

The *Nets* field now contains the net names in the file.

**7.** Click the *Route* tab, then select horizontal and vertical layer names and specify width and spacing values.

**Note:** To use different widths or spacing values for different nets, use the *Override* tab. See “Using Override to Add Wire Groups with Multiple Widths and Spacing” on page 661 for more information.

**8.** Click the *Misc* tab, and select the *Extend Start Wires* and *Extend End Wires* options.

These options extend both ends of the wires until they connect to a target.

9. Click the point in the design display area where the left-most or bottom-most wire should start.

**Note:** The startpoint does not have to be at a target.

10. Move the mouse horizontally or vertically.

The wires are drawn interactively.

11. Double-click the mouse.

The startpoint and endpoint of the wire extend until they connect to a target. If no target is present, the wire does not extend.

## Using Override to Add Wire Groups with Multiple Widths and Spacing

To add pairs of power and ground wires, where wires have different widths and spacing, complete the following steps:

1. Click the *Add Wires* widget.

This places the Encounter software in *add wire* mode and changes the mouse pointer to a pencil.

2. Choose *Edit – Edit Route* from the menu.

The Edit Route form opens.

3. Click the *Nets* tab on the *Edit Route* form and enter the net names into the *Nets* field.

**Note:** You can also specify a file that contains a list of nets. See [“Adding Wires that Automatically Extend to a Target”](#) on page 660 for more information.

4. Click the *Route* tab, then select horizontal and vertical layer names and specify width and spacing values.

5. Click the *Override* tab and enter a set of width and spacing values for the nets that do not have default width and spacing values.

For example, if you want wires for the third and fourth nets to have a wider width and larger spacing, specify the following values

3 6 4  
4 6

The first line indicates the third net (3) has a width of 6 microns and that the spacing value between the third and fourth net is 4 microns.

The second line indicates that the fourth net (4) has a width of 6 microns.

**Note:** To specify a value of less than 1, you must include a 0 before the decimal point. For example, a value of .6 is not valid, and must be expressed as 0.6.

6. Click the point in the design display area where the left most stripe should start.

**Note:** The startpoint does not have to be at a target.

7. Double-click the mouse.

The wires end at the location of the cursor.

## Cutting Shielding Wires

1. Click the *Cut Wires* widget.

The cursor changes to the shape of a scissors, indicating that the Encounter software is in *cut wire* mode.

2. Click the location at which you want to start cutting the shields.
3. Move the mouse so that the drawn line is touching or overlaps the wire orthogonally.
4. Click to complete the cut.



The cursor changes to an arrow shape.

6. Click the piece of wire to be deleted.

The selected piece of wire is highlighted.

**Note:** If multiple objects exist at the location of the cursor, press the spacebar to toggle the selection between them. To select multiple objects, press the Shift key while clicking.

7. Use the D keyboard shortcut to delete the selected objects.

**Note:** Wires can only be cut in the orthogonal direction. If you cut multiple wires, including wires in the same direction as the cut, the cut only affects wires in the orthogonal direction to the cut. Once cut, signal wire pieces maintain a 1/2 wire width extension, but power wires are not extended.

## Trimming Antennas on Selected Stripes

If your completed power structure contains stripes in a mesh configuration, physical antennas might remain on some stripes.

1. Use the `d` keyboard shortcut to display the Select/Delete Routes form.
2. Choose *Select* from the *Action* pulldown menu.
3. (Optional) Click *Nets*, then specify one or more nets for the wires to be trimmed.
4. (Optional) Click *Direction*, then click *H* to trim horizontal wires or *V* to trim vertical wires.
5. Click *Shapes*, then select *STRIPE*.
6. Click *Apply*.

The selected wires are highlighted in the design display area.

7. Use the `T` keyboard shortcut to trim the selected wires.

The selected power wires are trimmed back to the last connection point and deselected.

**Note:** Signal wires cannot be trimmed.

## Changing Wire Width

After running power analysis, you might need to increase the width of some power stripes to alleviate any IR drop or EM issues.

1. Make sure the software is in *select* mode (you can use the `w` keyboard shortcut), then click the wire segment to be widened.

2. Use the `e` keyboard shortcut.

This displays the Edit Route form without placing the software in *add wire* mode.

3. Click the *Route* tab on the Edit Route form, and enter values in the *Width* fields.

Specify a width value in the *Horizontal* section for horizontal wires and a width value in the *Vertical* section for vertical wires.

4. Use the `w` keyboard shortcut.

This changes the width of the selected wire. Any via connected to that the wire is also updated based upon the new width.

**Note:** Widths for signal wires depend on the applicable LEF rule, no matter what value is populated in the GUI. To specify a wire width that is different from the default wire width value, select both *Force Special* or a value other than *DEFAULT* from the *Rule* pulldown menu.

## Repairing Maximum Wire Width Violations

Violations occur if you specify wires widths greater than the maximum width defined by the `maxWidth` rule in the LEF file.

1. Use the `e` keyboard shortcut.

This displays the Edit Route form without placing the software in *add wire* mode.

2. Click the *Fix wires wider than max width* widget.

This executes the `editFixWideWires` command, which finds any wires violating the `maxWidth` rule, and splits up both the wires and the associated vias as minimally as possible while maintaining the same footprint.

## Duplicating Wires

After running power analysis, you might need to add some power stripes to alleviate any IR drop or EM issues. Instead of creating new wires interactively, you can duplicate existing wires.

1. Make sure the software is in *select* mode (you can use the `a` keyboard shortcut), then click the wire segment to duplicate.

2. Use the `e` keyboard shortcut.

This displays the Edit Route form without placing the software in *add wire* mode.

3. Click the *Route* tab on the Edit Route form and specify the vertical and horizontal layers for the duplicated wires.

**Note:** The widths of the duplicated wires are always the same as the original wires, but the layers are the ones specified in the form.

4. Click the *Duplicate selected wires* widget or use the `c` keyboard shortcut.

The duplicated wires are automatically selected and placed directly on top of the original wires.

**Note:** To duplicate a wire and change the layer, use the `editDuplicate` command and specify the layer for the duplicate wire. For more information, see `editDuplicate` in the “Wire Edit Commands” chapter of the *Encounter Text Command Reference*.

5. Use the `m` keyboard shortcut.

This places the software in *move* mode, allowing you to use the mouse or the arrow keys (while holding down the Shift key) to move the newly created wires to the desired location.

## Stretching Wires

1. Click the *Select* widget in the Tool Widgets area of the Encounter main window.

The cursor shape is an arrow, indicating that Encounter software is in *select* mode. The equivalent keyboard shortcut is **w**.

2. Click the wire to stretch.

The selected wire is highlighted.

3. Click the *Stretch Wire* widget in the Tool Widgets area of the Encounter main window.

The equivalent keyboard shortcut is **t**.

4. Move the cursor to the end point of the wire to be stretched.

The cursor changes to a T shape.

5. Click the end point, then release the mouse button and move the cursor to a new location and click again.

The wire stretches to the new location.

Alternatively, you can use the Shift key in conjunction with the arrow keys to stretch or shrink the wire. When the software is in *stretch wire* mode, you can use 1 and 2 as keyboard shortcuts to set the edge of the wire to be stretched. By default, the wire is stretched from the top or the right. To stretch the wire from the bottom or the left, use the 1 keyboard shortcut. The *Stretch Wires* widget reverses so that the outer arrow points to the left. To return to stretching wires from the top or right, use the 2 keyboard shortcut. The arrows on the *Stretch Wires* widget change so that the outer arrow points to the right.

## Changing Wire Layers

You may need to change sections of wires to different layers in order to relieve congestion on a specific layer or to fix process antenna violations.

1. Make sure the software is in *select* mode (you can use the **w** keyboard shortcut), then click the wire segment to be updated.
2. Use the **e** keyboard shortcut.

This displays the *Edit Route* form without placing the software in *add wire* mode.

3. Click the *Route* tab on the *Edit Route* form, and enter values in the *Layer* fields.

Specify a layer value in the *Horizontal* section for horizontal wires and a layer value in the *Vertical* section for vertical wires.

4. Use the `L` keyboard shortcut.

This changes the layer of the selected wire. Any via connected to that wire is also updated based on the new layer.

## Splitting and Merging Wires

Stripes that spread over the entire die may need to be altered in specific locations. In this case, a stripe that is represented as a single piece of wire must be split into multiple segments. You can split a wire to automatically cut stripes at each crossover:

1. Make sure the software is in *select* mode (you can use the `w` keyboard shortcut), then click the wire segment to be split.
2. Use the `Control-s` keyboard shortcut.

This automatically splits the single wire segment into multiple segments at points connected to other wires.

After splitting a wire, you can merge those wire segments that align back into a single segment.

1. Select a single segment.
2. Use the `M` keyboard shortcut.

This merges the wire segments into a single segment.

## Adding Vias

1. Select the *Add Via* widget. The equivalent keyboard shortcut is `o`.
2. Press the `F3` key.

This displays the *Edit Via* form.

3. Select *Geometry* in the *Create Via by* field.

4. Fill all of the fields in the form. For more information, see [Edit Via](#) in the *Encounter Menu Reference*.

5. Move the cursor to the location to which the via is to be added, then click the mouse.

A via with the exact configuration specified in the Edit Via form is added at that location.

## Changing Vias

### ■ Using the `editChangeVia` command

You can change one or more vias using this command. For example, to change all `VIA_XX` vias of a specified net located within a specified region to `VIA_YY` vias, type the following command:

```
editChangeVia -net netName -area {x1 y1 x2 y2} -from VIA_XX -to VIA_YY
```

For more information, see [editChangeVia](#) in the *Encounter Text Command Reference*.

### ■ Using keyboard shortcuts

You can change one via at a time using keyboard shortcuts.

- a. Place the cursor on the via to change
- b. Use the `n` or `p` keyboard shortcut to select the correct via if multiple vias exist in the same location on different layers.
- c. Without moving the mouse, use the `N` (next) or `P` (previous) keyboard shortcut to display a via that has the same LEF rule as the selected via.
  - If a via is available, the display is updated with the new via when you press the keyboard shortcut.
  - If another via is not available, you will hear a warning beep when you press the keyboard shortcut. This can occur when only one via is defined in the LEF file, when the currently queried object is not a via, or when no object is currently queried.

### ■ Using the Edit Power Vias form

For information, see [Edit Power Vias](#) in the *Encounter Menu Reference*.

**Note:** You cannot change vias using the Edit Route form.

## Moving Vias

You move vias the same way you move wires. The Encounter software moves vias without considering connectivity. For more information, see [“Moving Wires” on page 655](#).

## Reshaping Routes

You can reshape routes by specifying that wires at the corner of a route are to be trimmed after adding wires within an area that makes the existing corner wires obsolete. In addition, if you add a wire that circumvents an existing path, the existing route is trimmed after the new wires are added.

1. Click the *Add Wires* widget.

This places the Encounter software in *add wire* mode and changes the mouse pointer to a pencil. In addition, it places the software in *auto query* mode.

The equivalent keyboard shortcut is A.

2. Select the *Edit Route* form from the *Edit* menu and click the *Misc* tab.

The equivalent keyboard shortcut is e.

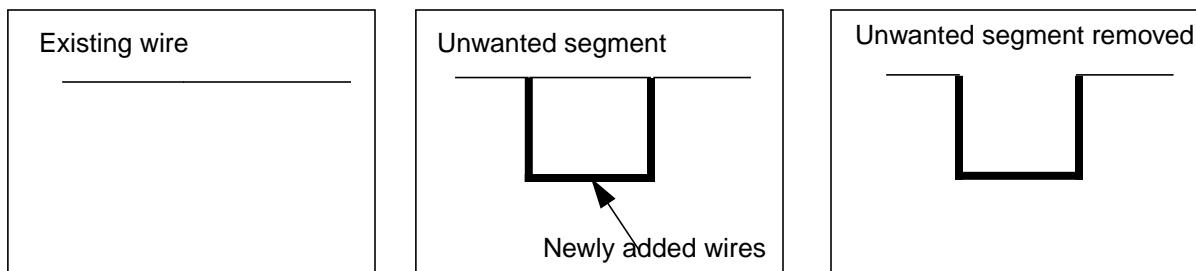
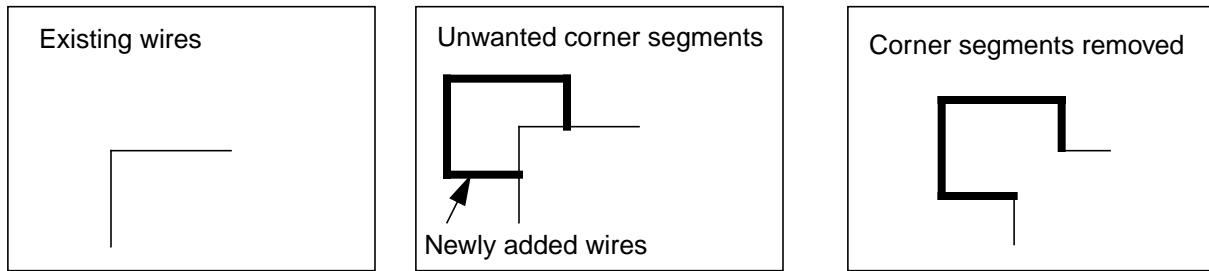
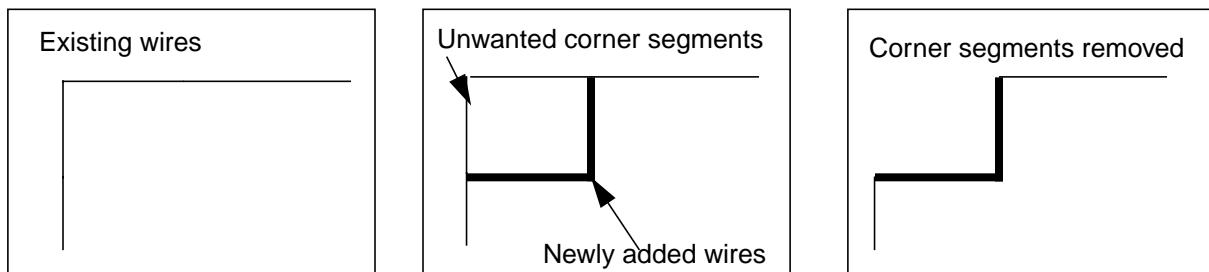
3. Select the *Reshape* option on the form.

4. Add wires to the design, as described in [“Adding Wires” on page 657](#).

## Encounter User Guide

### Editing Wires

The following illustrations show the results of using the *Reshape* option:



## Controlling Cell Blockage Visibility

If you see a spacing violation when adding or editing a via or wire, it might be caused by a cell blockage that is not currently visible.

To see cell blockages, select the *Cell Blkg* option on the *Routing color control* display (click the sidebar to display this option). Alternatively, you can click the *All Colors* button to display the Color Preferences form, then select the *Cell Blockage* visibility option. In addition, depending on whether the blockage is outside or inside a cell, you must do one of the following:

- Cell blockages outside a cell are visible by default. To control the visibility of these blockages for particular layers, click the *Wire Layers* tab of the Color Preferences form. Use the buttons in the fifth column, *Blkg*, to deselect the visibility of blockages for particular layers. By default, all layers are selected.
- Cell blockages within a cell are not visible by default. To see these cell blockages, click the *Wire Layers* tab of the Color Preferences form, then use the buttons in the sixth column, *V Blkg*, to select the visibility of blockages for each cut layer that you want to see. By default, all layers are deselected.

## **Editing Wires with Virtuoso Chip Assembly Router**

You can use the Virtuoso® CAR environment to route specified nets of a design created with the Encounter software. You can also reroute nets to correct violations. The CAR interface can be used at any stage in the design process after placement. Trial route data is ignored in the CAR environment. Power routing should be done before you edit using Virtuoso CAR. For more information about the CAR environment, see the *IC Shape-Based Technology Chip Assembly User Guide*.

Before you can use the Virtuoso CAR Interface form, the UNIX path variable must have access to the Encounter and CAR executables.

After using the CAR software, your design contains all routes that were present before using the CAR, as well as the new and updated routes that were added within the CAR environment.

You can use the CAR in batch or interactive mode.

### **Running Virtuoso CAR in Batch Mode**

When you run the CAR in batch mode, the software automatically performs the actions specified in the .do file. To run the CAR in batch mode, complete the following steps:

1. Select *Virtuoso CAR Interface* from the *Route* menu.  
This opens the Virtuoso CAR Interface form.
2. Specify the nets to be routed.

**3. Specify a CAR .do file.**

**Note:** If you do not specify a .do file, the software automatically generates a default .do file. The following statements are in the default .do file:

```
set rotate_via off  
groute 4  
route 25  
protect all wires  
save_oa export  
quit
```

If you create your own .do file, you should include the following statements:

```
set rotate_via off  
protect all wires
```

**4. Select *Batch* as the Operation Mode.**

**5. Specify all additional options on the Virtuoso CAR Interface form.**

**6. Click *OK* or *Apply*.**

The CAR GUI does not display, but the design within Encounter contains the changes to the nets that were specified in the .do file.

## **Running Virtuoso CAR in Interactive Mode**

When you run the CAR in interactive mode, your design is transferred from the Encounter environment into the CAR environment, where you can use the CAR editing features. This mode is asynchronous, and does not wait for the CAR to complete. To run the CAR in interactive mode, complete the following steps:

**1. Select *Route – Virtuoso CAR Interface* from the menu.**

This opens the Virtuoso CAR Interface form.

**2. Specify the nets to be routed with the CAR.**

**3. (Optional) Select *Specify Route Area*, then click the *Mouse* widget.**

**4. Select *Interactive* as the Operation Mode.**

**5. Specify all additional options on the Virtuoso CAR Interface form.**

**6. Click *OK* or *Apply*.**

The GUI displays the design that you transferred from the Encounter environment.

## **Encounter User Guide**

### Editing Wires

---

7. After you use the CAR editing features, choose *File – Save* or *File – Save As* from the CAR menu.
8. Choose *Design – Load – OA Database* from the Encounter menu.

This opens the Load OA Database form, which you can use to import the design as an Open Access design. The equivalent text command is `oaIn`.

The updated design is displayed in the Encounter environment.

---

## Using Trial Route for Congestion and Timing Analysis

---

- [Overview](#) on page 674
- [Data Preparation](#) on page 674
- [Routing A Flat Design](#) on page 675
- [Routing a Partitioned Design](#) on page 676
- [Routing Two-Metal Layer Designs](#) on page 678
- [Loading and Saving Route Data](#) on page 678
- [Analyzing Route Data](#) on page 678
- [Improving Route Congestion](#) on page 687
- [Using Bus Guides](#) on page 688
- [Additional Information](#) on page 689

## Overview

Trial Route performs quick global and detailed routing for estimating routing-related congestion and capacitance values. It also incorporates any changes made during placement, such as scan reorder.

You can use Trial Route results to estimate and view routing congestion, and to estimate parasitic values for optimization and timing analysis. When used during prototyping, Trial Route creates actual wires, so you can get a good representation of RC and coupling for timing optimization at an early stage in the flow. Trial Route also produces a congestion map you can view to get early feedback on whether the design is routable.

Trial Route results can also be used for pin assignment when you commit partitions.

**Note:** Trial Route does not guarantee DRC-clean routing results. Do not perform signal integrity analysis on a design that has been routed using Trial Route, because the routes are only used to estimate parasitic values for timing analysis. Route designs with NanoRoute or WRoute, if you want to perform signal integrity analysis.

You can use Trial Route during virtual prototyping, hierarchical floorplanning, block implementation, and top-level implementation.

### Related Topics

To see where this step fits in the design flow, see [Place the Design and Run Pre-CTS Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

## Data Preparation

- The design must be successfully placed.



If you make any changes after running Trial Route that affect the placement data—for example, floorplan changes—you must run placement before rerunning Trial Route.

- The design must be loaded into the current Encounter session.

The following optional input files are only required as necessary:

- DEF file

## Encounter User Guide

### Using Trial Route for Congestion and Timing Analysis

---

- Top Design Format (TDF) file containing routing data
- Routing guide file

## Routing A Flat Design

- Initial Design Routing

Run Trial Route for the first time to gauge the routability of the design. You can then examine the congestion map and congestion distribution report to identify congested areas that might cause routing problems later in the design session.

- a. Choose *Route – Trial Route*.

This opens the Trial Route Form.

- b. Select the *Prototyping* effort level and click *OK*.

You can also issue the following command:

```
trialRoute -floorplan
```



The prototyping (-floorplan) mode runs Trial Route quickly, which is important when prototyping large designs. However, note that components in your design might not be routed at legal locations.

- Post Clock Tree Synthesis Routing

Run Trial Route after clock-tree synthesis to recheck the routing congestion and to estimate parasitic values for timing analysis.

- a. Choose *Route – Trial Route*.

- b. Select the *Medium Effort* (default) or *High Effort* effort level on the Trial Route form.

- c. (Optional) Select the *Use Routing Guide* option, and specify the name of a guide file in the corresponding field. Trial Route follows the routing regions defined in the guide file and honors the specified pre-routed nets.

- d. Click *OK*.

Alternatively, you can issue the following command:

```
trialRoute -highEffort -guide my_chip.rguide
```

## Routing a Partitioned Design

In a flat design, Trial Route can route through guides, regions, and fences, as long as there are no routing blockages or hard blocks. However, fences are often defined as partitions, which become blocks after the design becomes hierarchical. Once partitions become blocks, the routes are no longer allowed, unless they use a proper feedthrough mechanism, such as inserted buffers or routing feedthroughs.

In channel-based routing designs, all top-level routing use channels to route around partitions. In a partitioned design in which the partitions have not been committed, you can use the Trial Route `-handlePartition` and `-handlePartitionComplex` parameters to force the routing into channels, simulating a channel-based design.

- Issue one of the following commands:

```
trialRoute -highEffort -handlePartition
```

or

```
trialRoute -highEffort -handlePartitionComplex
```

Use the `-handlePartition` parameter to route nets that are only connected within partitions. Use the `-handlePartitionComplex` to route nets that belong to more than one partition, so that the routing does not violate partition boundaries.

When the `-handlePartition` or `-handlePartitionComplex` parameter is specified, Trial Route works in three phases:

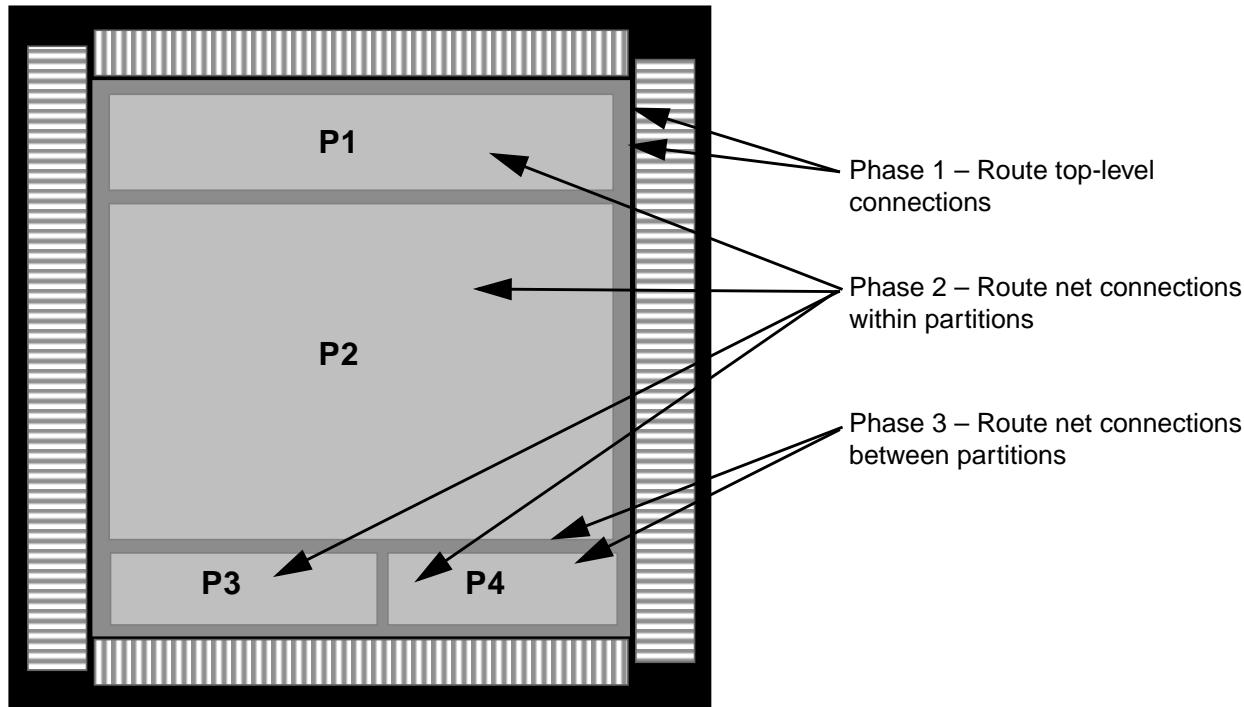
- Phase 1 – Routes the top-level connections
- Phase 2 – Routes net connections within partitions
- Phase 3 – Routes net connections between partitions

For example, the design in [Figure 19-1](#) on page 677 has five metal layers, a top-level partition and four flattened partitions: P1, P2, P3, and P4. P1 reserves the *metal1*, *metal2*, and *metal3* layers for partitions, and P2, P3, and P4 reserve layers *metal1* through *metal5* for partitions.

## Encounter User Guide

### Using Trial Route for Congestion and Timing Analysis

**Figure 19-1**



If you specify `-handlePartition` or `-handlePartitionComplex`, Trial Route performs the following tasks to route the net connections:

1. Trial Route applies all of the metal blockages defined for each partition. The layers *metal1*, *metal2*, and *metal3* are blocked for P1, and *metal1* through *metal5* are blocked for the other partitions.
2. Trial Route then routes nets connecting only at the top level. No connections to partitions or cells within the partitions are made.
3. Trial Route blocks all areas outside of the defined partitions on all routing layers. For P1, Trial Route applies a routing blockage for layers *metal4* and *metal5*.
4. Trial Route routes all nets within P1, P2, P3, and P4 on the available routing layers. This means that even a cell at the lower-left corner of P2 that connects to a cell at the upper-right corner of P2 is routed within P2, regardless of any congestion.
5. Trial Route removes the routing blockages and routes the net connections between partitions.

If an I/O at the top level connects to P2:

- If you specify `-handlePartition`, Trial Route uses all metal layers to route through P1.

- ❑ If you specify `-handlePartitionComplex`, Trial Route uses only layers *meta14* and *meta5* to route through P1.

## Routing Two-Metal Layer Designs

Trial Route can route designs with only two metal layers defined in the LEF file. Trial Route automatically detects when a design is *M1/M2* only, and uses wires from the *M1* and *M2* layers for routing.

Trial Route can also perform two-layer routing on designs that have more than two metal layers defined in the LEF file by setting the `trialRoute` or `setTrialRouteMode -maxRouteLayer` parameter to 2. Trial Route then uses wires from the *M1* and *M2* layers for routing.

All pins of the nets to be routed must be on layers *M1* and *M2*.

## Loading and Saving Route Data

After initially running the Trial Route program, you can load or save Trial Route data at any time during an Encounter session.

- To load route data using the Load Route File form, choose *Design – Load – Route*.
- To save route data using the Save Route File form, choose *Design – Save – Route*.

**Note:** You can also save route data by selecting the *Save Routing to* option on the Trial Route form and specifying a filename.

## Analyzing Route Data

After running Trial Route, you can analyze the results to check if your design is routable for back-end detailed routing tools.

1. Visually check the route congestion markers.

The red diamond-shaped congestion markers should not be very dense in a local area. These markers contain an overflow value to identify the number of tracks required for that grid, and the actual number of tracks available.

See “[Congestion Markers in the Display](#)” on page 679 for more information.

2. In the log file, inspect the Trial Route contents in the congestion distribution table.

## Encounter User Guide

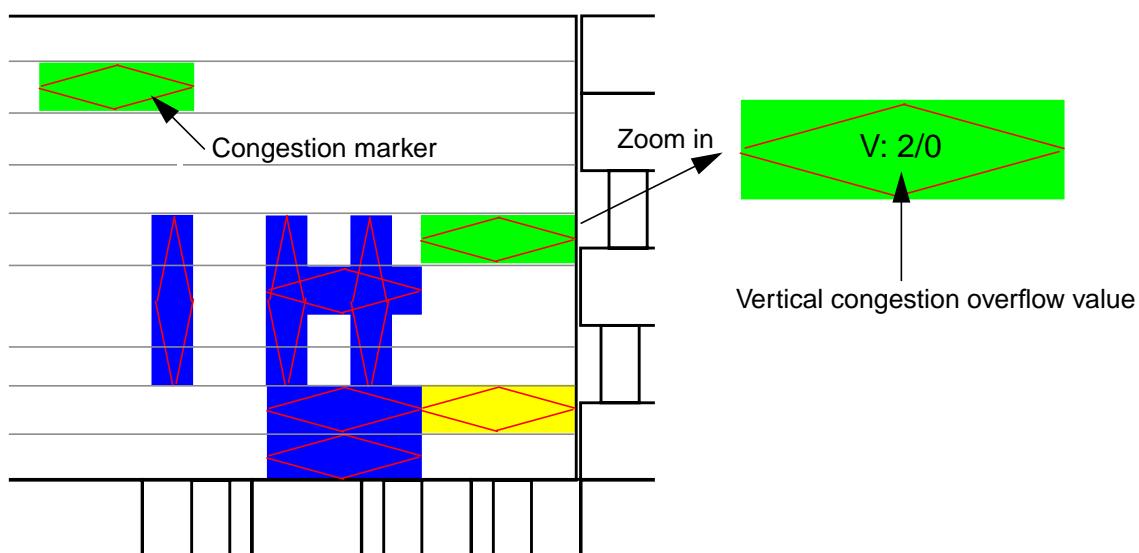
### Using Trial Route for Congestion and Timing Analysis

See “[Congestion Distribution Report](#)” on page 681 for more information.

When these two tests are satisfactory, the design is routable by a detail router.

### Congestion Markers in the Display

You can visually check the Trial Route congestion statistics in the design display area of the main Encounter window to identify the tight clusters of congestion markers. Check the design display area to make sure there are no markers grouped closely together. These usually occur around blocks or between large blocks. The indicators are diamond shaped and red by default. Zoom into the area to display the vertical and horizontal congestion overflow values, as shown in the following figure.



Congestion markers contain a vertical or horizontal overflow value to identify the number of tracks required for that grid, and the actual number of tracks available. For example, in the above illustration, the vertical overflow is 2/0, which indicates that two additional tracks are required, and 0 tracks are available.

Congestion marker values are based on an integer number of adjacent gcells that are grouped together to form a “super gcell.” Horizontal congestion super gcells are tall, narrow boxes that typically have a height of four gcells and a width approximately equal to the height of a vertical congestion super gcell. Vertical congestion super gcells are short, wide boxes that typically have a height of one gcell and a width approximately equal to the height of a horizontal congestion super gcell.

Vertical and horizontal overflow values are calculated separately for better accuracy. The overflow value is the amount by which the track demand exceeds the track supply. The

## Encounter User Guide

### Using Trial Route for Congestion and Timing Analysis

required track value is calculated by totalling the number of required tracks in the super gcell. That is, the value is the sum of the number of required tracks in all of the adjacent gcells that form the super gcell. The available track value is calculated by totalling the number of available tracks in the super gcell.

**Note:** Congestion markers can display different congestion information than that contained in the default congestion distribution report. The information in the congestion distribution report is based on the congestion of each gcell instead of the super gcells. To create a congestion report based on the congestion of the super gcells, use the [describeCongestion](#) command.

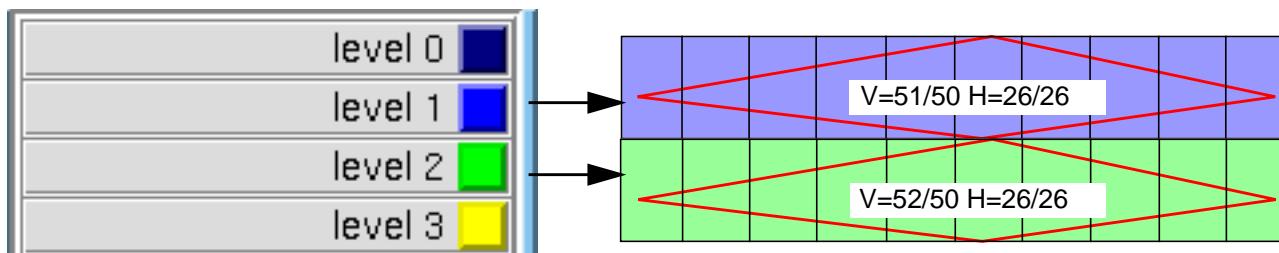
To change the size of super gcells, define the following variable:

```
set rdaSuperGcellSize n
```

The value you specify for *n* must be greater than or equal to 0 and less than or equal to 10. If you specify a value of 1, a super gcell becomes a regular gcell, and the displayed congestion marker information matches the congestion information provided in the report. If you specify a value of 0, the super gcells become square.

### Congestion Marker Color Boxes

By specifying the *HCongest* and or *VCongest* colors in the *Color* panel, you can also add a color box to the congestion marker that indicates the severity of the overflow level (that is, the number of overflow tracks in a one-unit area). Usually, a one-unit area contains 10 global cells (gcells) horizontally. If there are 50 vertical tracks available in that area, and Trial Route requires 51 vertical tracks, the congestion marker color box is blue (by default), indicating a one-track overflow. If Trial Route requires 52 vertical tracks, the congestion marker color box is green (by default), indicating a two-track overflow. An example of this is shown in the following figure.



## Encounter User Guide

### Using Trial Route for Congestion and Timing Analysis

---

The following table shows the default congestion marker colors and their corresponding overflow values:

Level	Color	Overflow Value
level 1	Blue	1 (One more track required)
level 2	Green	2 (Two more tracks required)
level 3	Yellow	3 (Three more tracks required)
level 4	Red	4 (Four more tracks required)
level 5	Magenta	5 (Five more tracks required)
level 6 and higher	Gray to white	6 or greater (Six or more tracks required)

For more information, see [“Multicolor Layers.”](#)

## Congestion Distribution Report

After Trial Route completes, a congestion distribution report is created in the Encounter log file. The congestion distribution report provides usage and routing overflow percent values, as well as gcell overflow information (that is, the internal supply and demand for each gcell). There are two types of congestion distribution reports that can be generated: a default congestion distribution report; and a detailed congestion distribution report.

**Note:** The congestion information contained in the congestion distribution report can differ from the congestion information displayed in congestion markers in the Encounter window. For more information, see [“Congestion Markers in the Display”](#) on page 679.

### Default Congestion Distribution Report

By default, Trial Route generates a congestion distribution report that summarizes congestion information for the entire chip.

### Usage and Routing Overflow

The following example illustrates the section of the congestion distribution report that summarizes the usage and routing overflow percent values:

```
Phase 1f route (0:00:02.0 105.9M):
Usage: (24.2%H 35.8%V) = (8.695e+06um 1.314e+07um) = (1734514 486668)
OvInObst: 21 = 21/5777 (0.4% H) + 0/1587 (0.00% V)
```

## Encounter User Guide

### Using Trial Route for Congestion and Timing Analysis

---

```
Overflow: 192 = 1 (0.0% H) + 191 (0.07% V)
```

The Usage statement summarizes horizontal and vertical tracks used in gcells. In the above example, there are 1,734,514 horizontal tracks used in all gcells and 486,668 vertical tracks used in a gcells. Of the available horizontal tracks, 24.2 percent were used for horizontal routing (that is, wires), and 35.8 percent were used for vertical routing. The total horizontal wire length used equals  $8.65e+06 \mu\text{m}$ , and the total vertical wire length used equals  $1.314e+07 \mu\text{m}$ .

The OvInObst statement summarizes obstructed gcell information. In the example, there are 5,777 horizontally obstructed gcells (where there are no available tracks). The approximate number of wires over horizontally obstructed gcells equals  $21/5,777$ .

The Overflow statement summarizes all overflowed gcells. In the example, there is one horizontally overflowed gcell, and 191 vertically overflowed gcells.

### **Gcell Overflow**

The following example illustrates the section of the congestion distribution report that summarizes gcell overflow information. A gcell has overflow if its demand exceeds its supply. Supply is the available routing resource, and demand is the amount of routing resource assigned to the gcell. Typically, the supply is the number of unobstructed tracks crossing the gcell, and the demand is the number of wires assigned to it.

Remain	cntH	cntV	
<hr/>			
-6:	0	0.00%	1
-5:	2	0.00%	0
-3:	10	0.00%	26
-2:	510	0.03%	830
-1:	8100	0.47%	17618
<hr/>			
0:	78504	4.59%	178501
1:	102934	6.02%	214588
2:	76165	4.46%	185168
3:	72832	4.26%	179080
4:	81555	4.77%	180443
5:	92704	5.42%	158498
6:	106167	6.21%	137707

## Encounter User Guide

### Using Trial Route for Congestion and Timing Analysis

---

The following table defines the columns in the congestion report:

Column	Definition
Remain	The track supply minus the track demand.
cntH	When <code>Remain</code> is positive, the number and percentage of gcells where the horizontal track supply exceeds the horizontal track demand. When <code>Remain</code> is negative, the number and percentage of gcells where the horizontal track demand exceeds the horizontal track supply. When <code>Remain</code> is 0 (zero), the number and percentage of gcells where the horizontal track demand is equal to the horizontal track supply.
cntV	When <code>Remain</code> is positive, the number and percentage of gcells where the vertical track supply exceeds the vertical track demand. When <code>Remain</code> is negative, the number and percentage of gcells where the vertical track demand exceeds the vertical track supply. When <code>Remain</code> is 0 (zero), the number and percentage of gcells where the vertical track demand is equal to the vertical track supply.

The following line from the example shows that there are 8,100 gcells (.47 percent of the total number of gcells) where the demand exceeds the supply by one track in the horizontal direction, and 17,618 gcells (1.05 percent of the total number of gcells) where the demand exceeds the supply by one track in the vertical direction:

-1: 8100 0.47% 17618 1.05%

The following line shows that there are 78,504 gcells where the track supply is equal to the track demand in the horizontal direction, and 178,501 gcells where the track supply is equal to the track demand in the vertical direction:

0: 78504 4.59% 178501 10.63%

### Detailed Congestion Distribution Report

You can create a detailed congestion distribution report that writes out information about sections of the chip. These sections can be either fixed quadrants defined by the middle horizontal and vertical lines, or user-defined sections specified by rows and columns.

To create a report for quadrants, specify `trialRoute -printSections` or `setTrialRouteMode -printSections true`. The report formats the information section-by-section, with each section containing congestion information for each layer in the section.

## Encounter User Guide

### Using Trial Route for Congestion and Timing Analysis

---

To create a report for user-defined sections, define the following two variables before running trialRoute -printSections or setTrialRouteMode -printSections true:

```
set trgNrSecRows number  
set trgNrSecCols number
```

These variables define the number of equal-size sections into which to divide the chip when reporting the congestion information. For example, the following two variable definitions divide the chip area into 3 x 2 sections of equal size:

```
set trgNrSecRows 2  
set trgNrSecCols 3
```

The detailed congestion distribution report is divided into six categories of information for each section of the chip:

- Virtual (global) wire length
- Range of tracks in a gcell
- Number of gcells with remaining tracks, including blocked gcells
- Number of gcells with remaining tracks, excluding blocked gcells
- Track usage in gcells
- Real wire length

These categories are described below.

#### ***Virtual (global) wire length***

This section summarizes the number of tracks used, the estimated wire length, the percentage of overflow, and the percentage of blocked gcells for each layer. The following example illustrates this section of the congestion distribution report:

```
***Virtual wire length:  
M2: tracks used = 17.8% = 7754505/43513490 est wire length = 1.707e+07um  
      overflow = 0.0% (0.0%) = 369/13471232 blk = 65.5%  
M3: tracks used = 19.1% = 4452746/23334471 est wire length = 1.959e+07um  
      overflow = 0.0% (0.0%) = 515/13471232 blk = 65.1  
M4: tracks used = 13.1 % =14991505/114837671 est wire length = 3.298e+07um  
      overflow = 0.0% (0.0%) = 237/13471232 blk = 0.4%
```

## Encounter User Guide

### Using Trial Route for Congestion and Timing Analysis

---

#### **Range of tracks in a gcell**

This section summarizes the range of the number of tracks used in a gcell on each layer, and the average number of tracks used per gcell on the layer. The following example illustrates this section of the congestion distribution report:

Range of number of tracks in a Gcell in layer M2: [5:10], avg: 3.2

Range of number of tracks in a Gcell in layer M3: [1:29], avg: 1.7

Range of number of tracks in a Gcell in layer M4: [1:29], avg: 8.5

Range of number of tracks in a Gcell in layer M5: [3:6], avg: 5.0

The first line of this example shows that there are between 5 and 10 tracks used in a gcell on layer *metal2*, and that the average number of tracks used in a gcell is 3.2.

#### **Number of gcells with remaining tracks, including blocked gcells**

This section summarizes the number of gcells, including blocked gcells, with remaining tracks (or the internal supply and demand for gcells) for each layer. A gcell has overflow if its demand exceeds its supply. Supply is the available routing resource and demand is the amount of routing resource assigned to the gcell. Typically, the supply is the number of unobstructed tracks crossing the gcell, and the demand is the number of wires assigned to it.

The following example illustrates this section of the congestion distribution report:

Table for number of Gcells with remain tracks (includes blocked Gcells):

Remain	M2	M3	M4	M5	...
<hr/>					
-6:	0	0.00%	0	0.00%	0
-5:	0	0.00%	0	0.00%	3
-4:	0	0.00%	0	0.00%	5
-3:	0	0.00%	0	0.00%	37
-2:	10	0.00%	28	0.00%	264
-1:	355	0.00%	476	0.00%	1254
<hr/>					
0:	8855290	65.73%	8858305	65.76%	70715
1:	91903	0.68%	269941	2.00%	124321
2:	166878	1.24%	441388	3.28%	192300
3:	250040	1.86%	559047	4.15%	347255
4:	319497	2.37%	734307	5.45%	738688
					5.48% 2369132 17.59%

The Remain column is the track supply minus the track demand. When this value is a positive number, it is the number and percentage of gcells where the track supply exceeds the track demand on each layer. When it is a negative number, it is the number and percentage of gcells where the track demand exceeds the track supply on each layer. When it is 0 (zero), it

## Encounter User Guide

### Using Trial Route for Congestion and Timing Analysis

---

is the number and percentage of gcells where the track demand is equal to the track supply on each layer.

The following line from the example shows that there are 8,855,290 gcells (65.73 percent of the total number of gcells) where the track supply is equal to the track demand on layer *meta/2*, and 8,858,305 gcells (65.76 percent of the total number of gcells) where the track supply is equal to the track demand on layer *meta/3*.

Remain	M2	M3	M4	...		
0:	8855290	65.73%	8858305	65.76%	70715	0.52%

The following line from the example shows that there are 91,903 gcells (.68 percent of the total number of gcells) where the track supply exceeds the track demand on layer *meta/2*, and 269,941 gcells where the track supply exceeds the track demand on layer *meta/3*.

Remain	M2	M3	M4	...		
1:	91903	0.68%	269941	2.00%	124321	0.92%

### ***Number of gcells with remaining tracks, excluding blocked gcells***

This section summarizes the number of gcells, excluding blocked gcells, with remaining tracks for each layer, and follows the same format as the table that reports the number of gcells with remaining tracks, including blocked gcells. The following example illustrates this section of the congestion distribution report:

Table for number of Gcells with remain tracks (excludes blocked Gcells):

Remain	M2	M3	M4	M5	...	
-6:	0	0.00%	0	0.00%	0	0.00%
-5:	0	0.00%	0	0.00%	0	0.00%
-4:	0	0.00%	0	0.00%	0	0.00%
-3:	0	0.00%	0	0.00%	0	0.00%
-2:	10	0.00%	28	0.00%	6	0.00%
-1:	355	0.00%	476	0.00%	229	0.00%
0:	34536	0.74%	83715	1.78%	18240	0.14%
1:	91903	1.98%	269941	5.75%	124321	0.93%
2:	166878	3.59%	441388	9.40%	192300	1.43%
3:	250040	5.38%	559047	11.90%	347255	2.59%

## Encounter User Guide

### Using Trial Route for Congestion and Timing Analysis

---

#### **Track usage in gcells**

This section summarizes the percentage of tracks used for routing per gcell for each layer. It also reports the average number of tracks used in a gcell on each layer. The following example illustrates this section of the congestion distribution report:

Table for track usage in Gcells

Usage(%)	All	M	M2	M3	M4	M5	...
#Gcells	avg	trks:					
0%-50%	97.31%	30.20%	29.25%	93.68%	88.09%		
	13108628	4068834	3940947	1260035	11866924		
50%-60%	1.64%	0.33%	3.00%	1.81%	7.37%		
	221249	44376	404372	243607	9933345		
60%-70%	0.78%	1.81%	0.08%	2.08%	0.16%		
	104771	243711	10795	279931	22125		
70%-80%	0.24%	1.24%	1.85%	1.20%	3.34%		
	32323	166753	249401	161193	450375		
80%-90%	0.03%	0.68%	0.05%	0.71%	0.07%		
	3796	91903	6908	95516	9724		

The fourth line from this example shows that 70 percent to 80 percent of the tracks in 1.24 percent of the gcells (out of a total of 166753 gcells) on layer *metal2* are used.

#### **Real wire length**

This section summarizes the total real wire length used and number of vias for each layer in the section. The following example illustrates this section of the congestion distribution report:

```
***Real Wire Length:  
Total: 1.396e+08um, total number of vias: 3547270  
M1(V): 9.076e+04um  
M2(H): 1.490e+07um, number of M1/M2 vias: 1560437  
M3(V): 1.652e+07um, number of M2/M3 vias: 1273925  
M4(H): 3.241e+07um, number of M3/M4 vias: 360881  
M5(V): 4.233e+07um, number of M4/M5 vias: 242047  
M6(H): 2.492e+07um, number of M5/M6 vias: 75288  
M7(V): 5.399e+06um, number of M6/M7 vias: 26668  
M8(H): 3.013e+06um, number of M7/M8 vias: 8034
```

## **Improving Route Congestion**

For a module or submodule that has route congestion, complete one of the following actions to improve congestion, depending on the type and severity of the violation:

## Encounter User Guide

### Using Trial Route for Congestion and Timing Analysis

#### 1. Change the block pin orientation.

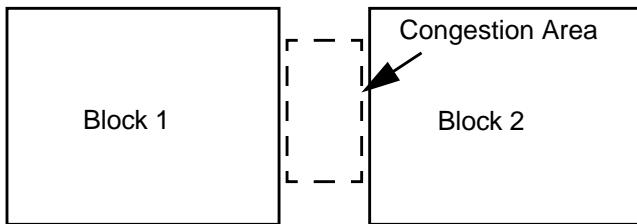
Route congestion usually occurs around blocks that have their pins facing incorrectly. You can identify these blocks by clicking on them in the design display area to see where the flight lines terminate. When the block pins are visible, you can rotate or flip the block(s) to alleviate the congestion.

For information on changing block orientation, see [“Flip/Rotate Instances.”](#)

#### 2. Add density screens.

You can use density screens to control standard cell placement density in certain areas where there is high routing congestion. Use the Add Density Screens tool to create a screen over the highly congested area.

Congestion is more severe if it spans between two blocked areas, as illustrated in the following figure.



This figure represents a vertically congested area between two blocks that are placed close to one another. This routing bottleneck is more severe than local congestion. Assigning a density screen alleviates this congested area.

For information on adding density screens, see *Add Partial Placement Blockage* under in the Floorplan Widgets section of [“Tool Widgets.”](#)

## Using Bus Guides

Trial Route uses bus guide information created with the bus planning feature to better control the routing of nets and nets groups. Bus guides can be created to control routing by area, layer, and net names. Trial Route automatically checks the bus guide information, then routes nets and net groups based on their defined bus guide constraints. Trial Route with bus guide routing can be run before or after assigning pins for a hierarchical partition or a black box design. When you run Trial Route on a design containing a bus guide, all the routing nets belonging to the specified net group are highlighted.

Bus guide routing enables groups of nets to be routed, through larger areas. This requires the gcells allowed for the nets to cover a larger area, based on the bus guide width. Nets can go

## Encounter User Guide

### Using Trial Route for Congestion and Timing Analysis

---

outside of a specified bus guide, but must be at least partially within the gcells covered by the bus guide. If the bus guide area is blocked, or too narrow, nets can go completely outside of the bus guide area. In this case, the Encounter software issues a warning message. To display warning messages, set `setTrialRouteMode -printWiresOutsideBusguide` parameter to `true` (or specify `trialRoute -printWiresOutsideBusguide`).

For more information on creating bus guides, see "Bus Planning" chapter in the *Encounter User Guide*.

The following limitations exist for the Trial Route bus guide feature:

- Bus bit ordering is not guaranteed; it depends on the pin ordering and topology requirements.
- Bus guides must be complete, and must cover the pins of the net they are to guide.

## Additional Information

### Wire Overlap

In certain situations, wire overlap can occur when using Trial Route.

A wire can overlap a routing blockage boundary if the blockage only partially covers the gcell. The covered tracks are counted as blocked tracks that are not available during global routing. However, Trial Route does not record the exact track location, which can result in wires being placed on a track which is already occupied by a routing blockage.

When using nondefault rules, wire width and space are considered during the global routing phase for congestion calculation, before track assignment. Because they are not considered during track assignment, overlapping nondefault wires can occur. However, because spacing was considered during congestion calculation, the routing congestion information is correct.

**Encounter User Guide**  
Using Trial Route for Congestion and Timing Analysis

---

---

## Using the NanoRoute Router

---

- [About NanoRoute Routing Technology](#) on page 694
- [Routing Phases](#) on page 694
  - [Global Routing](#) on page 694
  - [Detailed Routing](#) on page 695
  - [NanoRoute Router in the Encounter Flow](#) on page 696
- [NanoRoute Router in the Encounter Flow](#) on page 696
- [Before You Begin](#) on page 696
  - [Checking Your LEF Files](#) on page 696
  - [Checking for Problems with Cells, Pins, and Vias](#) on page 697
  - [Generating Tracks](#) on page 698
  - [Specifying Routing Layers](#) on page 698
- [Interrupting Routing](#) on page 700
- [Using the routeDesign Supercommand](#) on page 700
- [Results](#) on page 702
- [Running the NanoRoute Router with Encounter Menu Commands and Forms](#) on page 703
  - [Running the NanoRoute Router with Encounter Menu Commands and Forms](#) on page 703
  - [Running the NanoRoute Router with Encounter Text Commands](#) on page 703
  - [Running the NanoRoute Router in Standalone Mode](#) on page 704
- [Using NanoRoute Parameters](#) on page 705
  - [Using Attributes and Options Together](#) on page 706

## **Encounter User Guide**

### Using the NanoRoute Router

---

- [Accelerating Routing with Multi-Threading and Superthreading](#) on page 708
  - [When to Accelerate Routing](#) on page 709
  - [Superthreading Log File Excerpts](#) on page 710
- [Following a Basic Routing Strategy](#) on page 712
  - [Using the Encounter Text Commands](#) on page 712
  - [Using the Encounter GUI](#) on page 713
- [Checking Congestion](#) on page 716
  - [Using the Congestion Analysis Table](#) on page 716
  - [Using the Congestion Map](#) on page 718
- [Resolving Open Nets](#) on page 722
  - [Log File Examples](#) on page 722
  - [Diagnosing Problems Using verifyTracks](#) on page 723
  - [Resolving Additional Open Net Problems](#) on page 723
- [Running Timing-Driven Routing](#) on page 725
  - [Input Files](#) on page 725
  - [Using the CTE and the NanoRoute Router in Native Mode](#) on page 725
  - [Using the CTE and Standalone NanoRoute](#) on page 726
- [Routing Clocks](#) on page 728
  - [Setting Attributes for Clock Nets](#) on page 728
  - [Routing Clock Nets Using the GUI Forms](#) on page 729
  - [Running Postroute Optimization](#) on page 729
- [Preventing and Repairing Crosstalk Problems](#) on page 730
  - [Crosstalk Prevention Options](#) on page 732
- [Running ECO Routing](#) on page 734
  - [ECO Limitations](#) on page 734
  - [ECO Flow](#) on page 735
- [Evaluating Violations](#) on page 736

# **Encounter User Guide**

## Using the NanoRoute Router

- ❑ [Violations on Upper Metal Layers](#) on page 740
    - ❑ [Violations in Timing-Driven Routing](#) on page 742
    - ❑ [Deleting Violated Nets](#) on page 743
    - ❑ [Using Additional Strategies to Repair Violations](#) on page 743
  - [Concurrent Routing and Multi-Cut Via Insertion](#) on page 744
  - [Postroute Via Optimization](#) on page 744
  - [Optimizing Vias in Selected Nets](#) on page 745
  - [Via Optimization Options](#) on page 746
  - [Performing Shielded Routing](#) on page 747
    - ❑ [Shielding Option](#) on page 747
    - ❑ [Performing Shielded Routing Using the GUI](#) on page 748
    - ❑ [Performing Shielded Routing Using Text Commands](#) on page 749
    - ❑ [Interpreting the Shielding Report](#) on page 749
  - [Routing Wide Wires](#) on page 750
    - ❑ [Using Non-Default Rules](#) on page 751
  - [Repairing Process Antenna Violations](#) on page 753
    - ❑ [Changing Layers](#) on page 754
    - ❑ [Using Diodes](#) on page 754
    - ❑ [Process Antenna Options](#) on page 755
    - ❑ [Examples](#) on page 755
  - [Using a Design Flow that Includes Astro or Apollo](#) on page 757
  - [Troubleshooting](#) on page 759

## About NanoRoute Routing Technology

The NanoRoute® router performs concurrent signal integrity, timing-driven, and manufacturing aware routing (SMART routing) of cell, block, or mixed cell and block level designs. The router is optimized for routing designs with the following features:

- More than 300K instances or nets and at least five routing layers
- 180 nanometer or smaller process technology
- Signal integrity critical
- Timing critical
- Detailed-model (full-model) abstracts

**Note:** The WRoute router is also included in the Encounter® software. Your routing results might be better with the WRoute router when the technology is 180 nm or larger, and you have fewer than five routing layers and 300K instances. For information on using the WRoute router, see the [Ultra Router Reference](#).

### Related Topics

To see where routing is used in the design flow, see [Route the Design and Run Postroute Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

## Routing Phases

Full routing consists of global and detailed routing. You can repeat detailed routing incrementally on a routed database. Incremental detailed routing is not the same as ECO routing. For information, see [Global Routing](#) on page 694 and [Detailed Routing](#) on page 695.

ECO routing consists of incremental global and detailed routing passes on a routed design. During ECO routing, the router completes partial routes and makes minimal changes to existing wire segments. For information, see [Running ECO Routing](#) on page 734.

### Global Routing

During this phase, the router breaks the routing portion of the design into rectangles called global routing cells (gcells) and assigns the signal nets to the gcells. The global router attempts to find the shortest path through the gcells, but does not make actual connections or assign nets to specific tracks within the gcells. It tries to avoid assigning more nets to a

gcell than the tracks can accommodate. The detailed router uses the global routing paths as a routing plan.

The router can generate a map of the gcells, called a congestion map, that you can examine to see the approximate number of nets assigned to the gcells. The congestion map uses colors to indicate whether there are too few, too many, or the correct number of nets assigned to the gcells. If the router assigns too many nets to a gcell, it marks the gcell as over-congested. You can also read the Congestion Analysis Table in the Encounter log file to learn the distribution and severity of the congestion after global routing.

## Related Topics

- For more information on gcells, see “[GCell Grid](#)” in the “DEF Syntax” chapter of the *LEF/DEF Language Reference*.
- For more information on the congestion map and table, see “[Checking Congestion](#)” on page 716.

## Detailed Routing

During this phase, the NanoRoute router follows the global routing plan and lays down actual wires that connect the pins to their corresponding nets. The detailed router creates shorts or spacing violations rather than leave unconnected nets.

You can run detailed routing on the entire design, a specified area of the design, or on selected nets. In addition, you can run incremental detailed routing on a database that has already been detail routed.

The router runs search-and-repair routing during detailed routing. During search and repair, it locates shorts and spacing violations and reroutes the affected areas to eliminate as many of the violations as possible. The primary goal of detailed routing is to complete all of the required interconnect without leaving shorts or spacing violations.

During detailed routing, the router divides the chip into areas called switch boxes (SBoxes), which align with the gcell boundaries. The SBoxes can be expressed in terms of gcells; for example, a 5x5 SBox is an SBox that encompasses 25 gcells. The SBoxes overlap with each other and their size and amount of overlap might vary during search-and-repair iterations.

The router also runs postroute optimization as part of detailed routing. During postroute optimization, it runs more rigorous search and repair steps. Detailed routing stops automatically if it cannot make further progress on routing the design.

The routed data is saved as part of the Encounter database.

## NanoRoute Router in the Encounter Flow

The NanoRoute router is part of the block implementation and the top-level implementation stages of the Encounter flow.

Run the router early in the design flow to identify and fix routability problems or avoid them altogether. You can run the router in non-timing-driven mode after the default parasitic extraction step to establish a baseline for future steps. If the design is congested or unrouteable, stop and resolve problems before continuing.

### Related Topics

- “[Following a Basic Routing Strategy](#)” on page 712
- [\*Encounter Flat Implementation Flow Guide\*](#)

## Before You Begin

The NanoRoute router reads designs directly from Encounter. Before running the router, ensure your design meeting the following conditions:

- It is fully placed and the placement is legal, without any overlaps.  
    Use the [checkPlace](#) command to check for overlaps.
- (Optional) Run the [verifyGeometry](#) command and fix any problems. In general it is easier to fix geometry problems before routing than after routing.
- Power is routed.  
    Use the [sroute](#) command to route power structures.

## Checking Your LEF Files

You can avoid violations and save time if you ensure your LEF files are optimized for routing. Check the following statements and edit the files with a text editor if necessary:

- MINSIZE  
    The router does not support specifying MINSIZE without specifying AREA. MINSIZE allows a geometry that is smaller than AREA.
- UNITS

The router does not support a value of 100 for DATABASE MICRONS in the UNITS statement. If the LEF file specifies DATABASE MICRONS 100, issue the following command before you import the design:

`setImportMode -minDBUPerMicron 1000`

■ **MANUFACTURINGGRID**

The router requires that you define the manufacturing grid.

■ **MACRO**

To improve pin access, ensure that all standard cell macros are defined as CLASS CORE.

You must use real shapes—not block-style abstracts—for the shapes on the layers where you expect the router to connect to pins of standard cell macros.

■ **VIA**

The TOPOFSTACKONLY keyword is unnecessary if there are LEF LAYER AREA statements, because the router automatically derives TOPOFSTACKONLY vias based on the AREA statements. If a default via satisfies the AREA statement, the router tags it internally as a TOPOFSTACKONLY via.

If there is no AREA statement for a routing layer, the router looks for TOPOFSTACKONLY vias that go up to the next metal layer. If TOPOFSTACKONLY vias exist, it derives the AREA rule from those vias—the smallest area of the bottom layer metal of all such vias becomes the AREA rule. This feature provides backward compatibility with LEF files that do not have AREA rule support.

### Related Topics

- “[Unsupported LEF and DEF Syntax](#)” in the *Encounter User Guide*
- “[LEF Syntax](#)” chapter of the *LEF/DEF Language Reference*.

### Checking for Problems with Cells, Pins, and Vias

- Make sure that all power and ground pins in the SPECIALNETS section of the DEF file are marked + USE POWER or + USE GROUND.
- Overlapping cells

Overlapping cells make pins short each other and create violations on *metal1*. Check for overlaps by using one of the following commands:

□ `verifyGeometry -wireOverlap`

□ [checkPlace](#)

■ Pins underneath power routes

Pins that are underneath power routes are inaccessible and cause violations on *meta1* and *meta2*. Check for pins underneath power routes by using the *Auto Query* feature.

For more information, see “[Auto Query](#)” in the *Encounter Menu Reference*.

■ Lack of rotated vias

Rotated vias help reduce design rule violations by making pins accessible. The router does not rotate vias automatically and creates violations on *meta1* when it cannot access the pins.

Define rotated vias in the LEF file. For more information, see the *Encounter Library Development Guide*.

## Generating Tracks

In the Encounter environment, the router generates tracks automatically, based on the routing pitch, layer width and spacing, and minimum via widths.

If you import a DEF file, run the `generateTracks` command prior to global routing to correct faulty track definitions and tune the tracks to routing.

### Related Topics

- For more information, see [generateTracks](#) in the *Encounter Text Command Reference*.
- For information on importing DEF files, see “[Import and Export Commands](#)” in the *Encounter Text Command Reference*.

## Specifying Routing Layers

By default, the router uses all possible routing layers for routing wires. In some situations, you might want to limit routing to a layer range that does not include all routing layers. For example, you might want to reserve the top layers for power and ground stripes or perform ECO routing on a few layers only. You can specify hard limits for routing all nets within a layer range or you can specify soft limits to route specified nets within a layer range.

## Specifying Hard Layer Limits

When you specify hard layer limits, the router routes all nets within those limits. If there is a pin outside the limits you specify, the router uses vias, including stacked vias, to access the pin.

Use the following setNanoRouteMode parameters to specify hard layer limits:

- `-routeBottomRoutingLayer`
- `-routeTopRoutingLayer`

At times it might not be possible to route the nets within the limits without creating violations. For example, assume two pins, `pin_a` is on `meta/8` and `pin_b` is on `meta/7`. The pins overlap in the X and Y direction. If you specify that the top routing layer is `meta/6`, the router connects to `pin_a` by using stacked vias, creating a short with `pin_b`.

## Specifying Soft Layer Limits

When you specify soft layer limits, the router attempts to route specific nets within a layer range, but might route some nets outside the layer range if necessary to complete routing without creating violations. In addition, you can specify the effort level for staying within the range. You can also route specific nets within the layer range and others outside the layer range. For example, you can route critical nets within a narrower layer range than you route the rest of the nets in order to improve timing.

Use the following setAttribute parameters to specify soft layer limits and set the effort level toward honoring the limits:

- `-bottom_preferred_routing_layer`
- `-top_preferred_routing_layer`
- `-preferred_routing_layer_effort`

You can also use the `PREFERLAYER RANGE` property in the DEF file for the design to set soft layer limits for specific nets. For example, if you want the router to route `net_a` on metal layers 4, 5, and 6, add the following property to the net definition:

```
+ PROPERTY PREFERLAYER RANGE "4:6" ;
```

In the example above, 4 is the preferred bottom layer. It corresponds to using the following command:

```
setAttribute -net net_a -bottom_preferred_routing_layer 4
```

## Interrupting Routing

To interrupt routing, press **Ctrl-C**. The `routeDesign` or `globalDetailRoute` command continues to run until the database is in a state where the command can stop safely.

When the software stops, it prompts you to confirm that you want to interrupt the command.

- To confirm, type **Y**.

The software returns you to the Encounter prompt and the command does not continue to run.



***When you interrupt routing with Ctrl-C, the database will be in a state that is useful for debugging purposes only, and not one that you should save and continue to use in the design flow.***

- To continue running the command, type **N**.

## Using the `routeDesign` Supercommand

The recommended Cadence design flows use the `routeDesign` command to run global and detailed routing and to optimize vias and wirelength after routing.

`routeDesign` honors the `setNanoRouteMode` and `setAttribute` settings and has the following advantages over using the `globalRoute` and `detailRoute` or `globalDetailRoute` commands:

- It runs SMART routing by default; that is, it runs in both timing- and signal integrity-driven mode by default.

The other routing commands are not timing- or signal-integrity driven by default, but you can use the following `setNanoRouteMode` parameters to turn on timing- and signal-integrity-driven routing for those commands:

- ❑ `-routeWithTimingDriven true`
- ❑ `-routeWithSiDriven true`

- It changes the status of clock nets from `FIXED` to `ROUTED` so it can modify them during routing and routes them before routing other nets.

- ❑ To keep clock nets' status `FIXED`, run the following command before running `routeDesign`:

## Encounter User Guide

### Using the NanoRoute Router

---

```
setNanoRouteMode -routeDesignFixClockNets true
```

- ❑ To stop the router from routing clock nets first, run the following command before running `routeDesign`:

```
setNanoRouteMode -routeDesignRouteClockNetsFirst false
```

- It runs a placement check prior to routing to ensure that the placement is clean.

To turn off the placement check, specify the following `routeDesign` parameter:

```
-noPlacementCheck
```

- It checks for conflicts in `setNanoRouteMode` settings and issues warning messages when it detects problems. In some cases, it resets a mode in order to continue processing. For example, trying to fix postroute lithography problems and optimize vias concurrently can cause conflicts. If `routeDesign` detects requests for both types of operation, it issues a warning, turns off via optimization, and proceeds with fixing lithography problems.
- It has parameters that simplify via and wire optimization after routing. In addition, some `setNanoRouteMode` parameters work with `routeDesign`, but not with other routing commands.
  - ❑ The `routeDesign` parameters for via and wire optimization are `-viaOpt` and `-wireOpt`.
  - ❑ The `setNanoRouteMode` parameters that work only with `routeDesign` are `-routeDesignFixClockNets` and `-routeDesignRouteClockNetsFirst`.

## Related Topics

- To see this step in the design flow, see [Route the Design and Run Postroute Optimization](#) in the *Encounter Flat Implementation Flow Guide*.
- For more information, see the following commands in the “[Route Commands](#)” chapter of the *Encounter Text Command Reference*:
  - ❑ [detailRoute](#)
  - ❑ [globalDetailRoute](#)
  - ❑ [globalRoute](#)
  - ❑ [routeDesign](#)
  - ❑ [setAttribute](#)
  - ❑ [setNanoRouteMode](#)

## Results

The NanoRoute router outputs can include the following (depending on the run-time options you set):

- Section in the Encounter log file
- Routed DEF file
- GDSII file
- SDF or SPEF file

For information on outputting GDSII and DEF files, see “[Importing and Exporting Designs](#)” on page 119.

For information on outputting an SDF or SPEF file, see “[Timing Analysis](#)” on page 899.

- The following reports:

- Routing statistics

For information, see the `reportRoute` command in “[Route Commands](#)” in the *Encounter Text Command Reference*.

- Routing connectivity

For information, see the `checkRoute` command in “[Route Commands](#)” in the *Encounter Text Command Reference*.

- Wire statistics, including wirelength

For information, see the `reportWire` command in “[Route Commands](#)” in the *Encounter Text Command Reference*.

- Shielding statistics

For information, see “[Interpreting the Shielding Report](#)” on page 749.

- Timing analysis

For information, see “[Timing Analysis](#)” on page 899.

- Capacitance

For information, see “[RC Extraction](#)” on page 865.

- Design rule checking (DRC) and layout versus schematic (LVS)

- ❑ Process antenna violations

For information on DRC, LVS, and process antenna reports, see “[Verifying Violations](#)” on page 1103.

- ❑ Signal integrity

For information on signal integrity reports, see “[Analyzing and Repairing Crosstalk](#)” on page 1061.

## Use Models

### Running the NanoRoute Router with Encounter Menu Commands and Forms

Use the following forms to route the design.

- [Design – Mode Setup – NanoRoute](#)

Use this form to specify the router’s run-time options.

- [NanoRoute/Attributes](#)

Use this form to specify attributes for nets.

- [NanoRoute](#)

Use this form to set routing options.

- [Set Congestion Map Style – NanoRoute](#)

Use this form to customize the congestion map.

### Running the NanoRoute Router with Encounter Text Commands

Use the following commands to set NanoRoute attributes and options, generate tracks, LEF files, and vias that are optimized for the router, route the design, and optimize vias and wirelength after routing. The text commands include some NanoRoute options that are not included on the forms.

- [generateTracks](#)

Generates optimized tracks for the router (only necessary if you import a non-native DEF) file.

■ [generateLef](#)

Generates an optimized LEF file (only necessary if you import a non-native or non-current LEF file).

■ [generateVias](#)

Generates vias that are optimized for the router (useful if you import an incomplete or non-current LEF file).

■ [getAttribute](#)

[setAttribute](#)

Display and set net attributes.

■ [getNanoRouteMode](#)

[setNanoRouteMode](#)

Display and set run-time options for the router.

■ [globalRoute](#)

[detailRoute](#)

[ecoRoute](#)

[globalDetailRoute](#)

[routeDesign](#)

Route the design.

The recommended design flows use `routeDesign`. For more information, see [Using the routeDesign Supercommand](#) on page 700.

## Running the NanoRoute Router in Standalone Mode

The commands and options for running the NanoRoute router in standalone mode are not described in this document. The standalone NanoRoute router has its own GUI and command syntax. The commands, options, and attributes used by the standalone router are described in the *NanoRoute Technology Reference*.

### Related Topics

■ [NanoRoute Technology Reference](#).

## Using NanoRoute Parameters

The NanoRoute router has two kinds of parameters: attributes and options.

- Attributes assign characteristics to nets.

For example, use attributes to specify nets that have the following attributes: they are routed first (or last), they are routed on certain layers, they are protected by extra spacing, they are shielded (or act as shields), and signal integrity violations that affect them are repaired after routing.

- Options determine run-time operations.

For example, use options to perform the following run-time operations: run global or detailed routing, route selected nets only, repair antenna or design-rule violations, run timing driven or signal integrity driven routing, and specify the number of processors to use.

The following table lists attribute and option characteristics:

Characteristic	Attributes	Options
Application	Apply locally to a net object	Apply globally to a process or command
Specification	<ul style="list-style-type: none"><li>■ NanoRoute/Attributes form</li><li>■ <code>setAttribute</code> command</li><li>■ Some attributes can only be specified by <code>setAttribute</code>.</li></ul>	<ul style="list-style-type: none"><li>■ NanoRoute form</li><li>■ <code>setNanoRouteMode</code> command</li><li>■ Some options can only be specified by <code>setNanoRouteMode</code>.</li></ul>
Persistence	Saved with the database. When you set an attribute and save the database and exit, the attribute setting is saved. If you reload the database, the object retains the attribute setting.	Saved with the database. When you set an option, save the database and exit, the option setting is saved. If you reload the database, the router retains the option value.

## Encounter User Guide

### Using the NanoRoute Router

---

Characteristic	Attributes	Options
Format	<p><code>-attribute_name</code></p> <ul style="list-style-type: none"> <li>■ Example: <code>-avoid_detour</code></li> <li>■ Case sensitive (all lower case)</li> <li>■ Mandatory underscores separate words</li> <li>■ Native and standalone NanoRoute formats are the same</li> </ul>	<p><code>-optionName</code></p> <ul style="list-style-type: none"> <li>■ Example: <code>-drouteAutoStop</code></li> <li>■ Case sensitive (mixed case). Each word starts with an uppercase letter</li> <li>■ No underscores.</li> <li>■ Native and standalone NanoRoute formats are different (standalone options are all lowercase; words are separated by underscores; options do not start with a leading hyphen)</li> </ul>
See settings for this run ...	Use the <a href="#"><u>getAttribute</u></a> command	Use the <a href="#"><u>getNanoRouteMode</u></a> command
More information available at ...	<a href="#"><u>setAttribute</u></a> in the <i>Encounter Text Command Reference</i>	<a href="#"><u>setNanoRouteMode</u></a> in the <i>Encounter Text Command Reference</i>

## Using Attributes and Options Together

You can use attributes and options together. For example, to run global and detailed routing and repair signal integrity violations on a specified net during postroute optimization, set the `-si_post_route_fix` attribute for the net and route the design with the `-routeWithSiPostRouteFix` option set to `true`.

Using text commands, issue the following commands:

```
setAttribute -net net1 -si_post_route_fix true
setNanoRouteMode -routeWithSiPostRouteFix true
globalDetailRoute
```

## Encounter User Guide

### Using the NanoRoute Router

---

Using the GUI, make the following selections:

- On NanoRoute/Attributes form,
  - a. Type the name of the net in the *NetName(s)* text box.
  - b. Select *SI Post Route Fix True*.
- On the NanoRoute form,
  - a. Select both *Global Route* and *Detail Route*.
  - b. Select *Post Route SI*.

## Accelerating Routing with Multi-Threading and Superthreading

Encounter products accelerate routing by using more than one processor in the same machine and by distributing routing to multiple machines.

Both signal routers, the NanoRoute router and the WRoute router, can use more than one processor in the same machine. This is called multi-threading. For more information, see “Running Multi-Threading” in Accelerating the Design Process By Using Multiple-CPU Processing.

The NanoRoute detail router accelerates routing even more by distributing detailed routing across the network to multiple machines. This capability combines multi-threading with distributed processing, and is called Superthreading. When used with a gigabit Ethernet connection, Superthreading makes data communication time negligible.

Superthreading has the following features:

- Uses available Encounter licenses. No special licenses are necessary.
- Platform independent.
  - Different operating systems—including Solaris, Linux, and HP-UX—can be used in the same job.
  - Different hardware—including Sun, IBM, and HP—can be used in the same job.
  - 64-bit and 32-bit versions of the NanoRoute router can be used in the same job. For example, you can start a large job on a 64-bit server and run the job on smaller 32-bit clients.
- Can run using the `rsh` command, and with LSF, Sun Grid, or SSH configurations.
  - The RSH and SSH method tie multi-threaded jobs together.
  - The LSF and Sun Grid methods tie single jobs together.

### Related Topics

- [Accelerating the Design Process by Using Multiple CPU Processing](#) chapter in the *Encounter User Guide*
- [Multiple-CPU Commands](#) chapter in the *Encounter Text Command Reference*
- [“Multiple CPU Processing”](#) form in the Design Menu chapter of the *Encounter Menu Reference*

## When to Accelerate Routing

Not all designs or network topologies can take advantage of accelerated routing. Consider the following issues, and use single threading, multi-threading, or Superthreading when appropriate:

- Small (10k), simple designs or designs that do not have a lot of violations
  - Small jobs or designs that are easily routed probably do not need multiple CPUs or machines.
- Slow networks
  - The speed (10 Mb, 100 Mb, or 1,000 Mb) and type (LAN or WAN) of the network affect Superthreading speed.
- Loaded networks
  - Sharing CPU cycles with other processes increases Superthreading run time.
- Full or pending LSF queues or queues configured for one job
  - A queue that is set up to run only one job decreases efficiency.

## Usage Notes

- If you use the `rsh` command for Superthreading, you must be able to run the remote shell from the server machine to the client machines without a password prompt.
- The NanoRoute software must be accessible to the server and client machines.
- Client machines must be able to access the same version of the NanoRoute software.
- If you run the router in native mode, it will be the server program and the standalone router will be the client program.
- Start your routing job on the fastest multi-threaded machine available.
- Include the host machine as a client, otherwise it will be a server only and will not perform any routing jobs.
- If any CPUs crash, your job will not complete.

If there is a crash, most likely it will happen during the client routing stage, and the server will continue to run. The database on the server will be maintained in the state it had prior to the crash.

Check the messages in the log file to determine the problem and zoom into the area of the crash to see a graphical representation of the cause of the failure. After you fix the problem, you can continue routing from the crash point.

- If your job includes both Sun and Linux clients, include a different path to each executable in the command script or configuration file.
- You can run a job that uses both a Sun queue and a Linux queue.

## Superthreading Log File Excerpts

The following excerpts from a log file show progress during Superthreading. The software uses the following definitions to calculate the time:

- `client CPU time` is the CPU time on clients only.
- `cpu time` is the server CPU time plus the `client CPU time`
- `elapsed time` is the complete run time (the total elapsed time).

The first file fragment shows that the job is running with RSH, with two threads on the same host. The NanoRoute router pauses as the data on the server is synchronized.

```
#server my_machine is up on port 123456 waiting for connection .....
# client 2thread 1 from host machine_1
# client 2thread 2 from host host_machine_1
# Sync client 2 data ...
# cpu time = 00:00:03, elapsed time = 00:04:18, memory = 561.87 (Mb)
```

The second fragment shows that only 86 percent of the `client CPU time` is being used. Another process (in addition to the route job) is using CPU resources.

```
# client 3thread 1 from host machine_2
# client 3thread 2 from host machine_2
# Sync client 3 data ...
# cpu time = 00:00:03, elapsed time = 00:04:31, memory = 561.87 (Mb)
#
#Start Detail Routing.
#Start initial detail routing ...
# completing 10% with 0 violations
...
# completing 90% with 14 violations
# elapsed time = 00:12:29, memory = 606.02 (Mb)
# completing 100% with 10 violations
# elapsed time = 00:12:53, memory = 567.24 (Mb)
# number of violations = 0
# client cpu time = 00:03:12, memory 562.70 (Mb), util = 86%
#cpu time = 00:01:21, elapsed time = 00:123:54, memory = 566.24 (Mb)
...
```

## Encounter User Guide

### Using the NanoRoute Router

---

The third fragment shows that the job took less elapsed time than cpu time. The elapsed time is less than the cpu time because two clients are being used to route one job.

```
#Total number of violations on LAYER M8 = 4  
#Total number of violations on LAYER M9 = 1  
#Total number of violations on LAYER M10 = 0  
#Client cpu time = 17:38:54  
#Client peak memory = 795.22 (Mb)  
#Cpu time = 19:18:40  
#Elapsed time = 10:15:51
```

The final fragment shows the time the job completed.

```
#Increased memory = 92.98 (Mb)  
#Total memory = 628.17 (Mb)  
#Peak memory = 1019.30 (Mb)  
#Complete global_detail_route on Fri Apr 16 10:14:33 2004
```

## Following a Basic Routing Strategy

In general, the first time you route a design, you should be able to accept the default values on the NanoRoute form. You can look at the Encounter log file to see the processes that the NanoRoute router runs and the problems it encounters. Then you can adjust the net attributes or run-time options to improve your results.

The strategy presented in this section shows how you can break the routing processes into steps, so you can analyze and solve problems easily. After each step, check for data problems and congestion and make repairs. Repeat the step and repair remaining violations. Continue this process until the design is free of violations before going to the next step.

### Using the Encounter Text Commands

The following commands show the basic routing strategy using the Encounter text commands.

1. The router globally routes the design:

```
globalRoute
```

2. The router does the initial detailed routing (iteration 0 does not include a search-and-repair step), and saves the design as droute0:

```
setNanoRouteMode -drouteStartIteration 0
setNanoRouteMode -drouteEndIteration 0
detailRoute
saveDesign droute0
```

3. The router does the first search-and-repair iteration and saves the design for analysis:

```
setNanoRouteMode -drouteStartIteration 1
setNanoRouteMode -drouteEndIteration 1
detailRoute
saveDesign droute1
```

4. The router does the second to nineteenth search-and-repair iterations and saves the design for analysis. The switch box grows larger as the iteration number increases.

```
setNanoRouteMode -drouteStartIteration 2
setNanoRouteMode -drouteEndIteration 19
detailRoute
saveDesign droute19
```

5. The router runs postroute optimization (drouteEndIteration default) and additional search-and-repair operations and saves the design as droute:

```
setNanoRouteMode -drouteStartIteration 20
setNanoRouteMode -drouteEndIteration default
detailRoute
saveDesign droute
```

## Using the Encounter GUI

The following section describes the basic routing strategy using the GUI.

### Run Global Routing

1. Choose *Route – NanoRoute*.
2. Select *Global Route*.
3. Click *OK*.
4. Save as `groute`.
5. Check the congestion map.

If you see congested areas after global routing, your design is unroutable.

### Run Initial Detailed Routing

1. Choose *Route – NanoRoute*.
2. Set the following options on the NanoRoute form:

- Detail Route*
- Start Iteration 0*
- End Iteration 0*

The router builds the initial detailed routing database, but does not do any search and repair during this step.

3. Click *OK*.
4. Save the design as `droute0`.
5. Check the violations in the log file.

If you have many violations on *metal1* and *metal2*, you probably have pin-access problems, incorrect track settings, or overlapped cells. Check your LEF file and correct any problems.

See “[Evaluating Violations](#)” on page 736 for an excerpt of a log file from a design with many violations on *metal1* and *metal2*.

For information on the LEF file, see the [\*LEF/DEF Language Reference\*](#) or the [\*Encounter Library Development Guide\*](#).

## Run Search and Repair

Break search and repair into two phases. Check congestion after each phase and repair violations.

To run the first phase of search and repair, complete the following steps:

1. Choose *Route – NanoRoute*.
2. Set the following options on the NanoRoute form:

- Detail Route*
- Start Iteration 1*
- End Iteration 1*

During this phase, the router makes local changes to the database. It does not do detailed or global routing.

3. Click *OK*.
4. Save the design as `droute1`.
5. Check the violations in the log file and graphically.

To run the second search-and-repair phase, complete the following steps:

1. Choose *Route – NanoRoute*.
2. Set the following options on the NanoRoute form:
  - Detail Route*
  - Start Iteration 2*
  - End Iteration 19*

In this phase, the router makes additional search-and-repair passes. It reroutes nets with violations within a local area (a switch box). In each successive pass, the size of the switch box size increases, so the router can make the repairs over larger areas.

3. Click *OK*.
4. Save the design as `droute19`.
5. Check congestion.

If you still have many violations (more than 1,000) or an unbalanced distribution of violations, you might still have a problem with the data or a congested design.

For help resolving the violations, see “[Evaluating Violations](#)” on page 736.

## Run Postroute Optimization

Ensure your data and library are violation-free before you run postroute optimization, or the router might spend a lot of time trying to repair violations that it cannot repair. Postroute optimization takes longer than any of the other steps because the router does more rigorous search and repair during postroute optimization than previous steps.

To run postroute optimization, complete the following steps:

1. Choose *Route – NanoRoute*.
2. Set the following options on the NanoRoute form:
  - Detail Route*
  - Start Iteration* 20
  - End Iteration* default

**Note:** In general, do not set *Start Iteration* or *End Iteration* higher than 20 because it does not increase the quality of results.

3. Click *OK*.
4. Save the design as `droute`.

During postroute optimization, the router runs both global and detailed routing and makes global changes to repair violations.

## Checking Congestion

Check congestion in your design after global routing by using the Congestion Analysis Table in the Encounter log file and the congestion map in the Encounter main window.

### Related Topics

To see where this step fits in the design flow, see [Place the Design and Run Pre-CTS Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

## Using the Congestion Analysis Table

The congestion analysis table shows the distribution and severity of congestion in global routing cells (gcells) on each routing layer.

**Note:** For information on global routing and on gcells, see “[Global Routing](#)” on page 694.

Following is an example of a Congestion Analysis table:

Congestion Analysis:					
Layer	OverCon	OverCon	OverCon	OverCon	%Gcell OverCon
	#Gcell (1-2)	#Gcell (3-4)	#Gcell (5-6)	#Gcell (7-12)	
Metal 1	22(0.01%)	10(0.00%)	0(0.00%)	0(0.00%)	(0.01%)
Metal 2	5531(2.39%)	1680(0.73%)	370(0.16%)	123(0.05%)	(3.33%)
Metal 3	4114(1.78%)	19(0.01%)	0(0.00%)	0(0.00%)	(1.79%)
Metal 4	1333(0.58%)	137(0.06%)	0(0.00%)	0(0.00%)	(0.64%)
Metal 5	5852(2.53%)	4(0.00%)	0(0.00%)	0(0.00%)	(2.53%)
Metal 6	27(0.01%)	0(0.00%)	0(0.00%)	0(0.00%)	(0.01%)
Total	16879(1.22%)	1850(0.13%)	370(0.03%)	123(0.01%)	(1.39%)

- The first column, Layer, lists the metal layers that have over-congested gcells. The NanoRoute router marks a gcells as over-congested if the global router has assigned more nets to the gcell than the gcell has available tracks.
- The second through fifth columns, labelled OverCon #Gcell, list the number and percentage of gcells on each layer that are over-congested.
- The numbers in parentheses after OverCon #Gcell indicate how many additional tracks within the gcell are needed to accommodate the global routing assignments. For example, OverCon #Gcell (1-2) means that one or two additional tracks are needed to accommodate all the nets that the global router has assigned the gcells listed in the column. As you move from left to right in the table, congestion increases because the difference between the number of nets assigned to the gcell by the global router and number of available tracks within the gcell increases.

- The number of columns in the table is determined by the number of additional tracks needed by the gcells with the worst congestion. For example, if the most over-congested gcells need only four additional tracks, the table would include columns for 1-2 and 3-4 tracks, but not for 5-6 or more tracks.
- The NanoRoute router creates only one column for gcells that need seven or more additional tracks. In the example, all gcells that need seven to 12 additional tracks are listed in the column labelled OverCon #Gcell (7-12).
- The NanoRoute router displays the maximum number of tracks needed in the last OverCon #Gcell column. In the example, the maximum number of tracks needed is 12. If some gcells needed 14 more tracks, the column would be labelled OverCon #Gcell (7-14). If the maximum number of tracks needed were only eight, the column would be labelled OverCon #Gcell (7-8).

Within each column, the table does not indicate exactly how many additional tracks are needed. For example, in the column labelled OverCon #Gcell (7-12), The NanoRoute router does not distinguish between gcells that need seven, eight, nine, ten, 11, or 12 additional tracks.

- The last column, %Gcell OverCon, lists the percentage of all gcells on the layer that are over-congested. In the example, on layer Metal 1, only 0.01% of the gcells are over-congested.
- The last row of the table, Total, lists the total number and percentage of over-congested gcells in each column. In the example, 1,850 gcells in the design, or 0.13% of all gcells, need three or four more tracks.
- The last row of the last column displays the overall percentage of over-congested gcells in the design. In the example, 1.39% of all cells are over-congested.

### Interpreting the Table

- Read the table horizontally to see the distribution and percentage of gcells on each layer that have a greater demand for tracks than they have supply of tracks.
- Read the table vertically to see which layers have the most over-congested gcells and how severe the congestion is.
- The table does not show how closely the over-congested gcells are clustered. Look at the congestion map in the GUI to see clusters of congestion and their exact location.
- There is no “magic number” that determines whether the design is routable. In general, the more columns, and the more the percentages increase toward the right side of the table, the worse the congestion.

## Using the Congestion Map

Check obstructions and congestion in your design graphically by analyzing a congestion map. The information in the map is directly extracted from the router after you run global routing. You choose the layers to display on the map. The Encounter software displays the congestion map in the main window when you complete the following steps:

1. Globally route the design.
2. Select *Physical view* in the *Views* area of the Encounter main window.
3. Move the slider under *All Colors* to the left. This displays *General color control*.
4. Make *Congestion* viewable.
5. Click the *All Colors* button.
6. Select both *Horizontal Congest* and *Vertical Congest*.
7. Click *Update Display*.

For more information on selecting the objects and colors, see “[The Main Window](#)” in the *Encounter Menu Reference*.

## Interpreting the Congestion Map

In the map, blue or black indicate an acceptable level of congestion; white indicates an unacceptable level. However, this depends on your design. For example, a design that is mostly uncongested might have small areas (often called hot spots) that are highly congested. You must look at the overall congestion graphically to assess routability.

The following table explains the meaning of the default colors in the congestion map:

Color	Explanation
Black	No congestion: You have at least two tracks that are under-used.
Blue	No congestion: You probably have one track that is under-used.
Green	No congestion: All the tracks are used.
Yellow	Low congestion: You probably have one track that is over-used.
Red	Some congestion: You probably have two tracks that are over-used.
Magenta	Moderate congestion: You probably have three tracks that are over-used.

## **Encounter User Guide**

### Using the NanoRoute Router

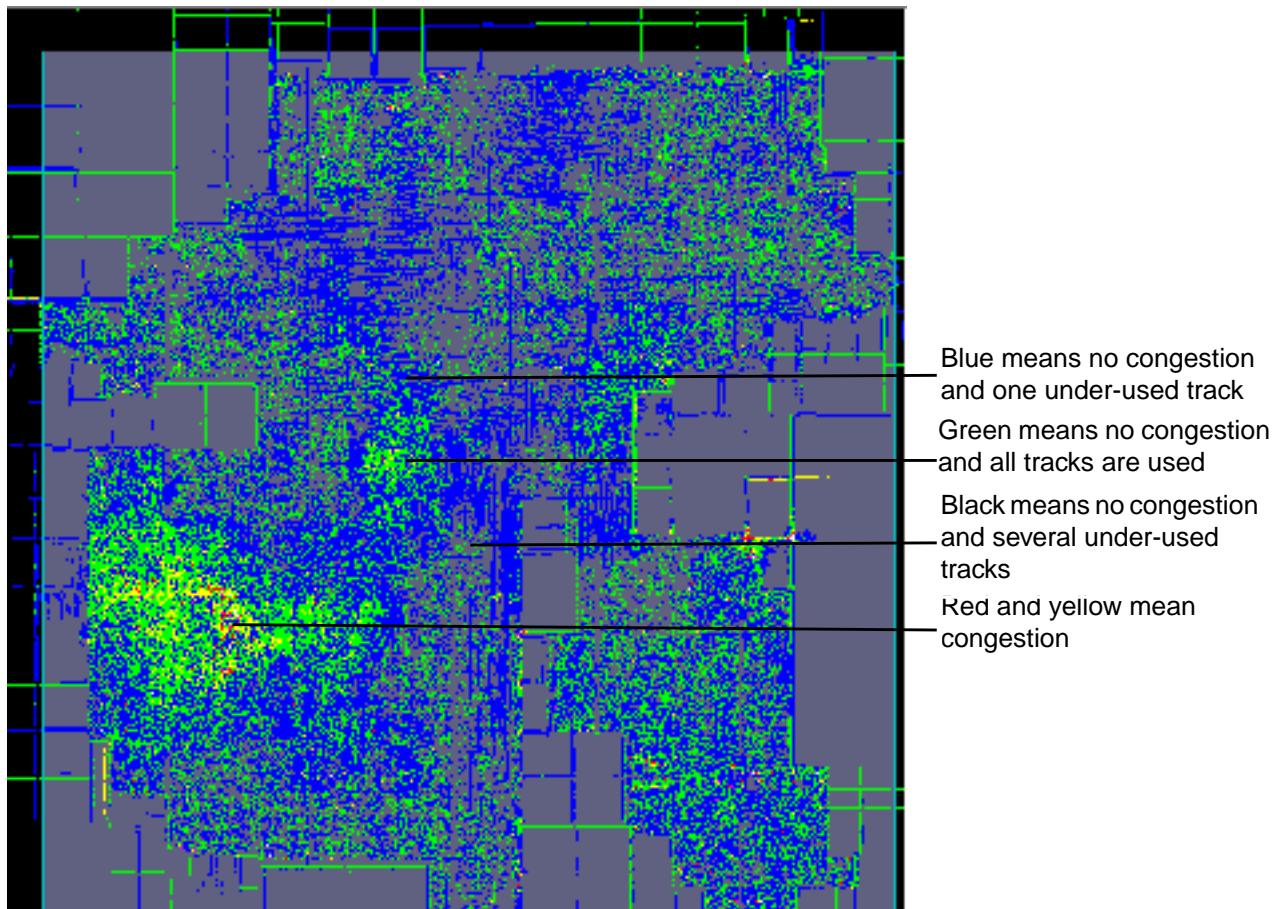
---

<b>Color</b>	<b>Explanation</b>
White	High congestion: You probably have at least four tracks that are over-used.

## Encounter User Guide

### Using the NanoRoute Router

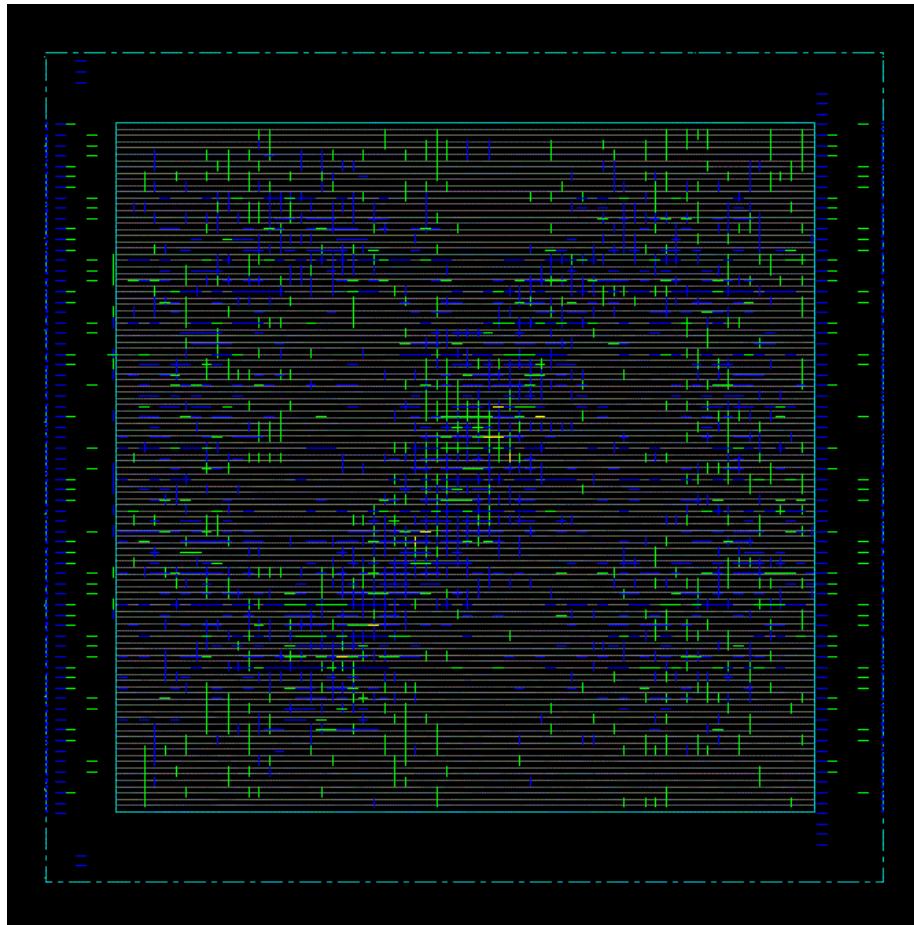
In the congestion map shown below, there is a congested area (a hot spot) in the lower left quadrant.



## Encounter User Guide

### Using the NanoRoute Router

In the congestion map shown below, the design is not congested.



## Resolving Open Nets

If the router cannot complete the connection of a net during routing, it generates an open net warning message in the Encounter log file and sets the net status to open. Additionally, the log file provides a list of open nets in summary format.

- During detailed routing, problems with pin modelling, routing track definitions, floorplanning, or conflicts between `setNanoRouteMode` option settings can cause open nets.
- During global routing, missing power or ground routing can cause open nets.

To resolve open net problems, complete the following steps:

1. Run `verifyTracks` to diagnose a subset of open net problems in standard cells. This command generates a report in the Encounter log file. Use the report to determine the specific cause of the open net. For more information, see “[Diagnosing Problems Using verifyTracks](#)” on page 723.
2. Determine the cause of the remaining problems—mostly those caused by option conflicts or libraries—by manual analysis. For more information, see “[Resolving Additional Open Net Problems](#)” on page 723.
3. Resolve the problems.
4. Re-run global and detailed routing.

## Log File Examples

The following examples show sections of an Encounter log file that includes five open net warning messages generated during detailed routing:

```
#Start Detail Routing.  
#start initial detail routing ...  
#WARNING (NR) Fail to route NET example56/cp_aclk_2 in region ( 302.295 272.894  
331.695 306.495) Set net status to open.  
#WARNING (NR) Fail to route NET example56/cp_aclk_3 in region ( 302.295 272.894  
331.695 306.495) Set net status to open.  
...  
#start 1st optimization iteration ...  
#WARNING (NR) Fail to route NET example12/cp_bclk_5 in region ( 402.295 372.894  
431.695 406.495) Set net status to open.  
#WARNING (NR) Fail to route NET example12/cp_bclk_6 in region ( 402.295 372.894  
431.695 406.495) Set net status to open.  
...  
#start 2nd optimization iteration ...  
#WARNING (NR) Fail to route NET example99/cp_cclk_8 in region ( 502.295 472.894  
531.695 506.495) Set net status to open.  
...
```

The following section of the same log file includes the open net summary:

```
# number of violations = 0
#cpu time = 00:00:01, elapsed time = 00:00:01, memory = 51.15 (Mb)
#Complete Detail Routing.
#WARNING (NR) There are 5 open nets.
#Please refer to Encounter User Guide for details of open net messages and possible
root causes.
#After resolving it, please re-run globalDetailRoute command.
#List of open nets :
# example56/cp_aclk_2
# example56/cp_aclk_3
# example12/cp_bclk_5
# example12/cp_bclk_6
# example99/cp_cclk_8
#
#Total wire length = 340827 um.
#Total half perimeter of net bounding box = 298122 um.
```

## Diagnosing Problems Using verifyTracks

The `verifyTracks` command reports the following types of problems in the Encounter log file:

- Pins that are too far inside a blockage

For more information, see [Macro Obstruction Statement](#) syntax and the accompanying figures in the “LEF Syntax” chapter of the *LEF/DEF Language Reference*.

- Pins that are not aligned with routing tracks

Align pins with routing tracks to assure the maximum number of pickup points. For more information, see the [“NanoRoute Ultra Guidelines”](#) chapter in the *Encounter Library Development Guide*.

- Pins that are above or underneath power stripes on the adjacent metal layer

The router might not be able to access a pin if it is blocked by a power stripe.

For more information, see [verifyTracks](#) in the *Encounter Text Command Reference*.

## Resolving Additional Open Net Problems

If the router generates an open net message after you correct the problems reported by `verifyTracks`, or if `verifyTracks` does not report any problems, check for the following additional problems:

- Pin modelling or library problems

- Pins without physical geometry

## Encounter User Guide

### Using the NanoRoute Router

---

- ❑ Pins that are less than the minimum width
  - ❑ Minimum-width pins that are placed off the manufacturing grid
  - ❑ Pins that are blocked for planar access, and are not accessible through a via without violating the adjacent-cut rule
  - ❑ Pins that trigger multiple-cut vias, but no multiple-cut vias are specified in the LEF file
- Floorplanning problems
- ❑ Cell overlaps introduced during placement
    - Use the `checkPlace` command to check for cell overlaps. For information, see [checkPlace](#) in the *Encounter Text Command Reference*.
- Problems caused by `setNanoRouteMode` option settings or conflicts between option settings and library specifications
- ❑ No via access in pin but `-routeWithViaOnlyForStandardCellPin true` is specified
  - ❑ No via access in pin but `-routeBottomRoutingLayer` is too high or `-routeTopRoutingLayer` is too low for the router to connect without using a via
  - ❑ Via stacking is not allowed but `-routeBottomRoutingLayer` is higher than the pin layer (or `-routeTopRoutingLayer` is lower than the pin layer) so via stacking is required to reach the pin
- Problems caused by missing power or ground routing
- ❑ Missing special routes for stripes or followpins to connect tie-high or tie-low nets causes open power or ground nets during global routing.

The global router issues open net warning messages such as the following:

```
#WARNING (NR) There is no prerouted stripe wire within routing layer range  
1:9 for special net VSS.  
#WARNING (NR) Please reroute special net wires before running NR.
```

## Running Timing-Driven Routing

In the Encounter environment, during timing-driven routing, the router uses the Common Timing Engine (CTE) by default. All the related tasks (route estimation for the timing graph, capacitance extraction, timing analysis, timing graph generation) are executed within the Encounter environment.

Timing-driven routing might cause longer run time and more violations than nontiming-driven routing. For information, see [“Violations in Timing-Driven Routing” on page 742](#).

### Input Files

To run timing-driven routing you need the following files:

- Physical libraries in LEF
- Timing library in .lib format
- Timing constraints in .sdc format or a timing graph

For information on the timing constraints that are compatible with the Encounter CTE, see [“Data Preparation” on page 91](#).

- Extended capacitance table generated by the Encounter software
- Netlist in DEF or Verilog format
- Placed design in DEF

### Using the CTE and the NanoRoute Router in Native Mode

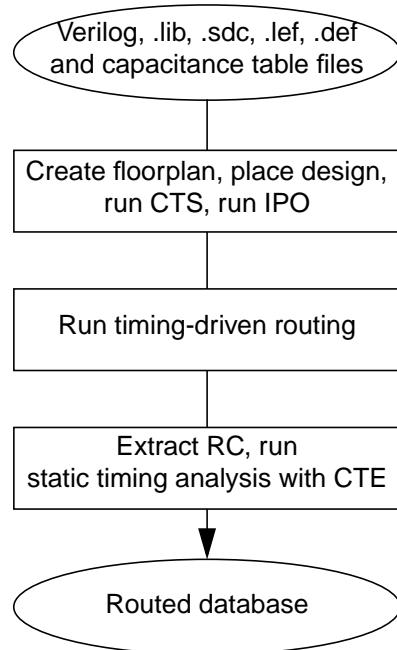
[Figure 20-1 on page 726](#) shows the design flow for routing with NanoRoute in native mode using the CTE. In native mode, the router uses the CTE as its default timing engine.

In native mode, the following commands use the CTE:

```
setNanoRouteMode -routeWithTimingDriven true  
globalDetailRoute
```

**Figure 20-1**

Native NanoRoute-CTE timing-driven routing flow



## Using the CTE and Standalone NanoRoute

Figure 20-2 on page 727 shows the design flow for standalone NanoRoute using the CTE. The standalone NanoRoute router is loosely integrated with the CTE.

In standalone mode, the following commands use the CTE:

```
pdi set_option timing_engine external_timing_graph  
pdi set_option route_with_timing_driven true  
pdi global_detail_route
```

The flow is shown in three main steps:

1. Generating a timing graph

To generate a timing graph, load your design into the Encounter software and use the `Encounter writeDesignTiming` command.

```
writeDesignTiming design.tif
```

2. Routing

When you route the design, type the following commands:

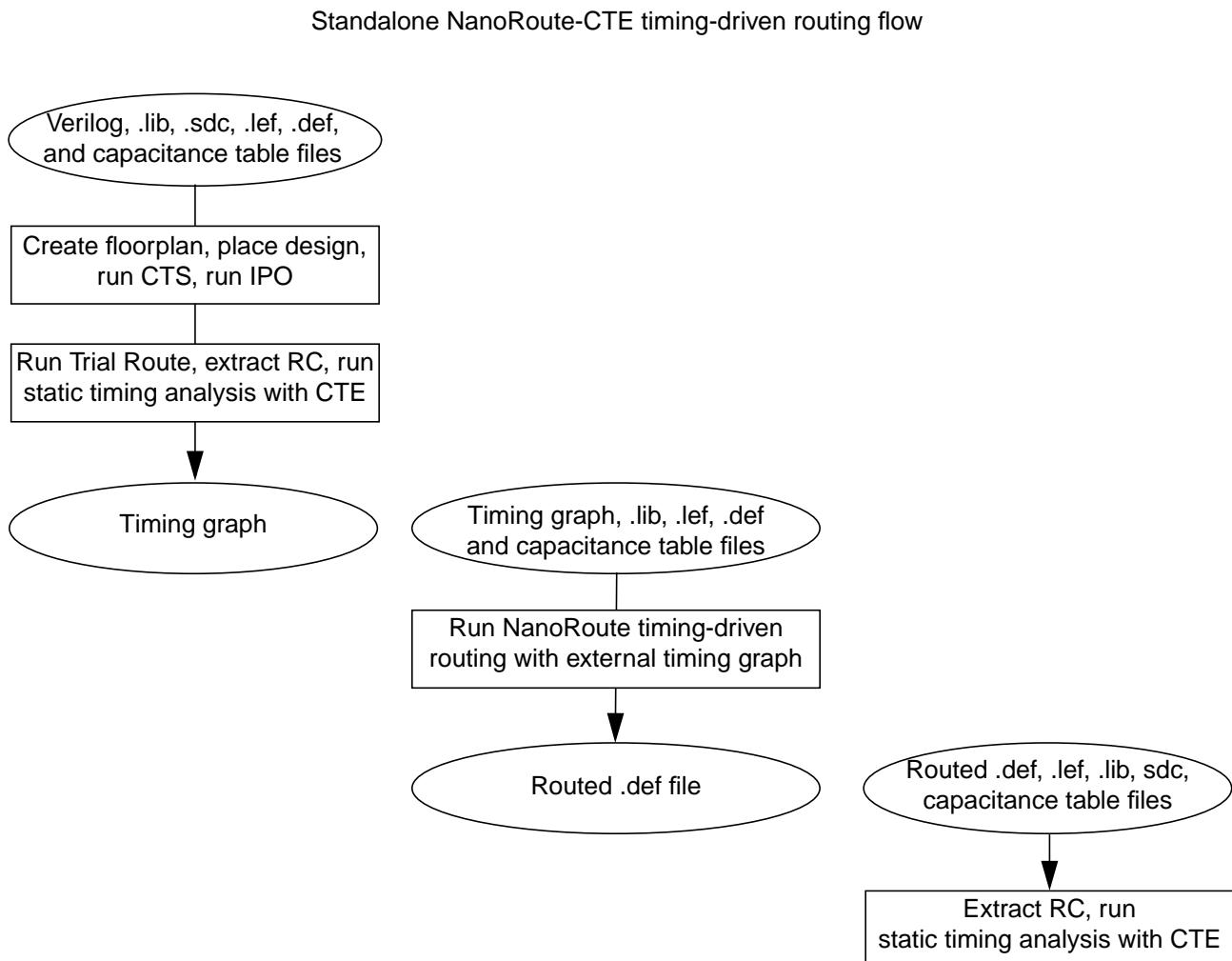
## Encounter User Guide

### Using the NanoRoute Router

```
pdi set_option timingEngine design.tif
```

### 3. Extracting capacitance and analyzing timing

**Figure 20-2**



## Routing Clocks

Route clock nets before routing the rest of the signal nets. If you are using the `routeDesign` supercommand, the NanoRoute router changes the status of clock nets from `FIXED` to `ROUTED` so they can be moved and routes them before routing other nets.

This section gives additional information on you can use to route clocks manually.

Layer assignments for clock nets might not correlate in global and detailed routing. For tight control over clock timing, run global and detailed routing on clock nets before routing other nets. Fix the locations of the nets during detailed routing and unfix them during postroute optimization. Use net weights to ensure priority during search and repair.

### Setting Attributes for Clock Nets

If clock nets are marked `USE CLOCK` in the DEF file or you have defined a clock net in the Encounter database, the router automatically sets the following values. You can change the values by setting attributes on the NanoRoute Attributes form. If the clock nets are not defined, type the name of a clock net in the *Net Name(s)* text box to set attributes for the net.

- *Weight*

The default net weight for clock nets is 10 to give clock nets priority during global routing (the default net weight for other nets is 2). During global routing, the router goes from global routing cell to global routing cell within each switch box, and routes the nets with the highest weight first.

- *Bottom Layer*

The default bottom layer for routing clock nets is 3, to ensure that the router has access to *meta1* pins during routing. This attribute sets a soft limit, and the router might route some nets on lower layers, if necessary to complete the routing.

- *Top Layer*

The default top layer for routing clock nets is 4. This attribute sets a soft limit, and the router might route some nets on lower layers, if necessary to complete the routing.

- *Avoid Detour*

*Avoid Detour* is *True* for clock nets, so they are routed as straight as possible.

Set the following attribute in the Encounter console, using the `setAttribute` command

- `-preferred_extra_space 1`

`-preferred_extra_space` adds spacing around the clock nets, which improves coupling capacitance. It is not included on the NanoRoute/Attributes form.

For information on `setAttribute -preferred_extra_space`, see “[Route Commands](#)” in the *Encounter Text Command Reference*.



Select *SI Prevention True* to set *Weight*, *Avoid Detour* and `-preferred_extra_space` all at once. *SI Prevention True* sets *Weight* to 10, *Avoid Detour* to *True*, and `-preferred_extra_space` to 1 for clock nets.

## Routing Clock Nets Using the GUI Forms

- Specify the following options on the NanoRoute form:

- *Selected Nets*

Specify *Selected Nets* to route the clock nets first. Unlike the *Weight* attribute, which gives priority to routing nets within a switch box, *Selected Nets* is a global option that routes whole nets.

- *Global Route*

- *Detail Route*

Specify *End Iteration* 19 to stop routing before the postroute optimization step.

## Running Postroute Optimization

To prevent rip-up and rerouting of clock nets during postroute optimization, specify the following:

- On the NanoRoute/Attributes form, keep the attributes you have already set, and specify *Skip Routing True*.
- On the NanoRoute form, specify *Start Iteration* 20 and *End Iteration* default.

## Related Topics

- [Using the routeDesign Supercommand](#) on page 700

- [routeDesign](#) in the “Route Commands” chapter of the *Encounter Text Command Reference*.
- “[Synthesizing Clock Trees](#)” on page 577

## Preventing and Repairing Crosstalk Problems

During SMART routing, the NanoRoute router automatically prevents crosstalk problems by wire spacing, net ordering, minimizing the use of long parallel wires, and selecting routing layers for noise-sensitive nets. The router performs these operations concurrently with other operations.

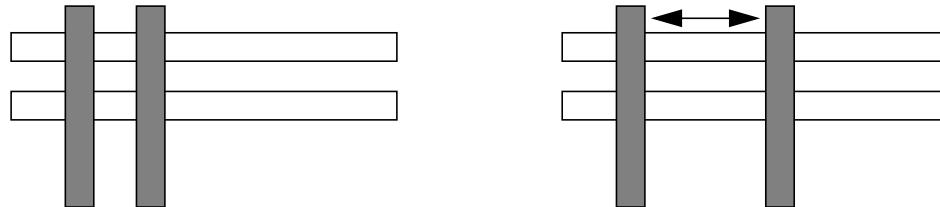
In addition to the operations it performs automatically, the router also performs shielded routing to protect critical wires from crosstalk.

During postroute signal integrity repair, the router performs these same operations.

The following sections describe the crosstalk prevention and repair operations the router performs, and whether you can set net attributes to control them.

- Wire spacing

The router automatically adds extra space between critical nets. You can also use the `-preferred_extra_space` attribute to add space. For information on this attribute, see [setAttribute](#) in the *Encounter Text Command Reference*.



- Net ordering

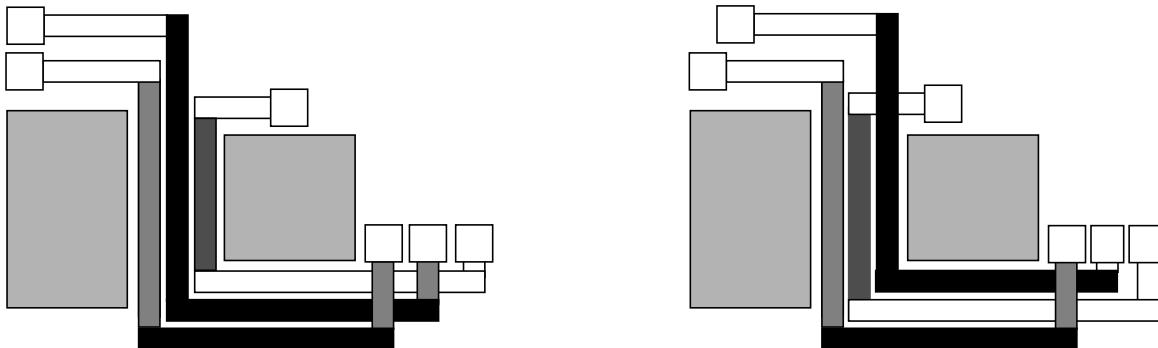
The router automatically routes critical nets first and avoids detours on those nets so they are as short as possible.

- You can also use the `-weight` attribute to give priority to critical nets within a switch box, so they are routed first.

## Encounter User Guide

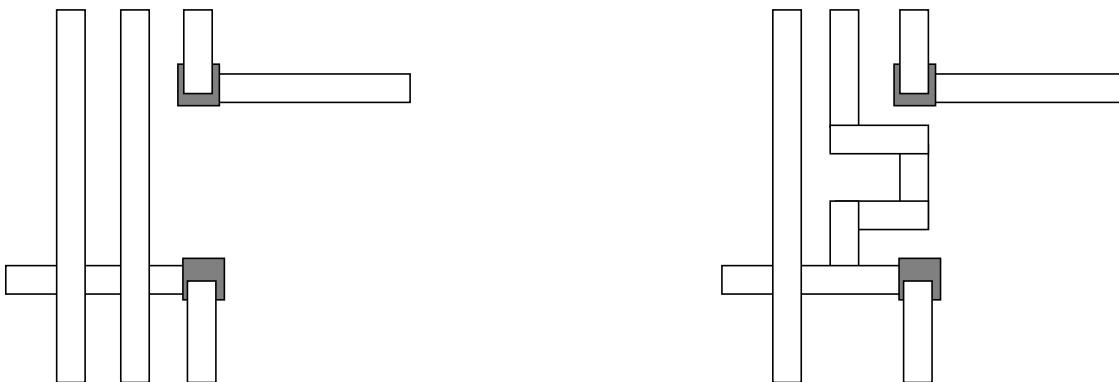
### Using the NanoRoute Router

- You can use the `-avoid_detour` attribute to ensure that critical nets are routed as short as possible.



#### ■ Minimizing the use of long parallel wires

The router automatically minimizes the use of long parallel wires, based on an internal algorithm. You cannot set an attribute to control this feature.

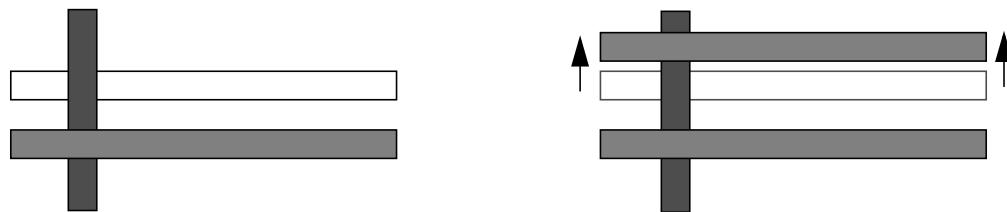


#### ■ Selecting routing layers

The router automatically restricts routing layers for critical nets to reduce both coupling and resistance. It routes clocks on layers 3 and 4.

- You can set the `-bottom_preferred_routing_layer` and `-top_preferred_routing_layer` attributes to specify preferred layers for critical nets.

- ❑ You can specify how strictly to enforce these attributes by specifying the -preferred\_routing\_layer\_effort attribute.



- Shielding

The router can shield critical nets with power or ground wires to protect them from coupling. Shielding is not an automatic operation—you control it with the -shield\_net attribute.

### Related Topics

- [Performing Shielded Routing](#) on page 747.

## Crosstalk Prevention Options

To minimize problems caused by crosstalk, set the following NanoRoute options:

```
setNanoRouteMode -routeWithTimingDriven true  
setNanoRouteMode -routeWithSiDriven true
```

These options specify timing-driven and signal integrity-driven routing.

Optionally, you can also set the following options:

```
setNanoRouteMode -routeTdrEffort  
setNanoRouteMode -routeSiEffort
```

These options fine-tune the priorities the router assigns to timing, signal integrity, and congestion. Use these options together to minimize crosstalk. After meeting the timing requirements of your design, adjust the values and rerun routing, following these guidelines:

- If your design is congested, use a low timing-driven effort.
- If your design is not congested, use a high timing-driven effort.



*Tip*

Because designs with severe signal-integrity problems are usually not congested, use a high timing-driven effort for those designs.

- If increasing the timing-driven effort creates a jump in the number of timing violations, decrease the timing-driven effort.

For more information on these options, see [setNanoRouteMode](#) in the *Encounter Text Command Reference*.

For more information, see “[Analyzing and Repairing Crosstalk](#)” on page 1061.

## Running ECO Routing

The NanoRoute router performs ECO routing by completing partial routes with added logic while maintaining the existing wire segments as much as possible. ECO routing is useful in cases such as the following:

- After the chip is initially routed, the customer or chip owner gives you a new netlist with minor changes.
- After the chip is initially routed, buffers were added to repair setup or hold violations or DRVs during physical optimization.
- Buffers were added or gates were resized during hand editing of a routed design.
- Antenna diodes were added interactively after routing to repair process antenna violations.
- After metal fill is added to the design.

During ECO routing, the router does the following:

- Reroutes partial routes and nets without routing.

You can use wire editing commands to partially preroute wires to guide global ECO routing. The router does not globally reroute nets that are automatically prerouted, such as clock nets, but it might make minor routing changes to preroutes to increase routability of the design. Examples of minor routing changes include the following:

- Completely moving a preroute
  - Changing the routing topology within the current routing switch box.
- Retains fully prerouted nets and pin-to-pin paths.
- Might use dangling paths in order to complete routes, but removes dangling wires left after global routing.
- Keeps connectivity within the bounding box, but does not constrain layers or positions.

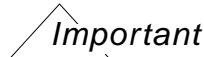
## ECO Limitations

- Do not use the `globalRoute` command in ECO mode. To route in ECO mode, use `globalDetailRoute`.
- If more than 10 percent of the nets are new or partially routed, run full global and detailed routing instead of ECO routing.

## ECO Flow

To perform ECO routing, specify the following commands and options:

```
setNanoRouteMode -routeWithEco true  
globalDetailRoute
```



The `-routeWithEco` option constrains changes but might lead to violations or long run times if it causes the router to move more signals to resolve the routing.

### Specifying Nets for ECO Routing

The router automatically identifies the nets that need changes during ECO routing.

To route only a few nets, and skip routing on all the other nets, specify the following commands:

```
setAttribute -net @PREROUTED -skip_routing true  
setAttribute -net eco_net_name1 -skip_routing false  
setAttribute -net eco_net_name2 -skip_routing false
```

### ECO Routing After Multiple-Cut Via Insertion

If your design is already fully routed and multiple-cut vias have been inserted for manufacturing, specify the following commands for ECO route:

```
setNanoRouteMode -routeWithEco true  
setNanoRouteMode -drouteUseMultiCutViaEffort low  
globalDetailRoute
```

For more information on using Encounter ECO commands and flows, see “[Interactive ECO](#)” on page 1045.

## Evaluating Violations

After you run several search-and-repair iterations, look at the number and distribution of violations remaining to determine whether the violations are caused by congestion or by design or library problems. If you see many violations (more than 1,000) or the number of violations does not decrease with each search-and-repair iteration, stop search and repair and check congestion graphically.

**Note:** Use the `Ctrl-C` key combination to stop search and repair. For information on using `Ctrl-C`, see [“Interrupting Routing”](#) on page 700.

The router marks four types of violations:

- Horizontal

A violation that falls on a horizontal wire is a horizontal violation.

- Vertical

A violation that falls on a vertical wire is a vertical violation.

- Via

A violation that falls on a via is a via violation. The router places via violation markers on the lower layer of the via, even when the violations are on the upper layer. For example, if the router finds a spacing violation between an M1-M2 via's *metal2* layer and another *metal2* shape, the violation marker on the via will be at *metal1* layer. If your routed design has via violations, the router corrects the violations during search and repair.

- Minimum area rule

If the `AREA` rule for the layer has not been satisfied, the router marks a minimum area rule violation.

For details on the violations, use the [verifyGeometry](#) command.

If you import a routed design from standalone mode, the Encounter software imports the violation markers as well. To delete the violations, rerun detailed routing. The software deletes the violations during search and repair.

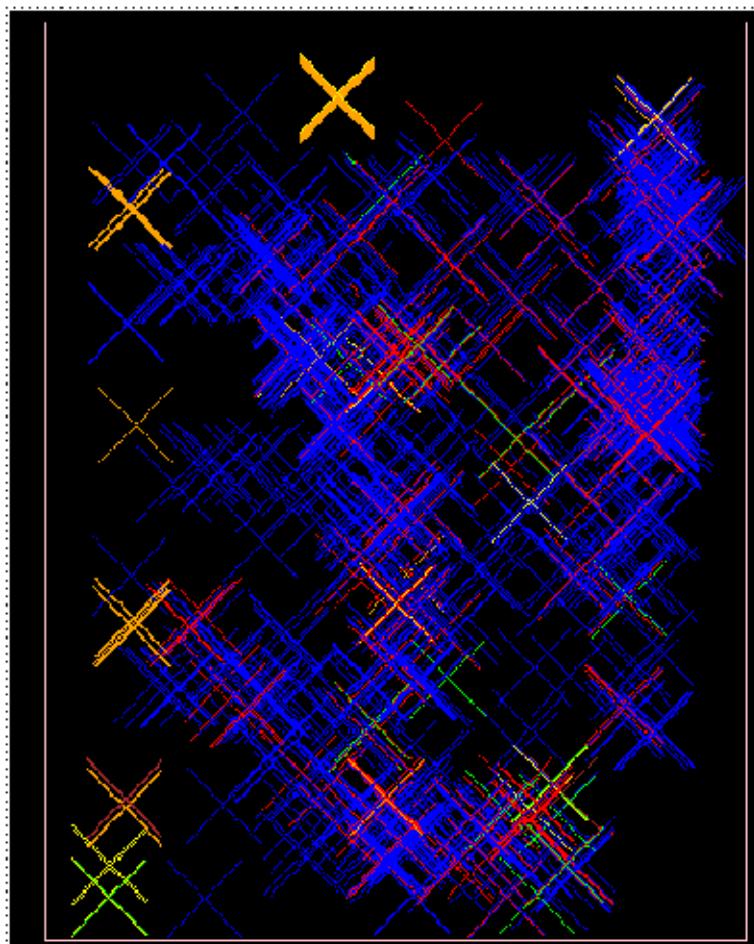
The following excerpt is from a log file from a design after the nineteenth iteration of search and repair. There are many violations, mostly on layers *metal1*, *metal2*, and *metal6*.

```
#Total number of DRC violations = 1426
#Total number of violations on LAYER Metal1= 876
#Total number of violations on LAYER Metal2= 275
#Total number of violations on LAYER Metal3= 84
#Total number of violations on LAYER Metal4= 38
#Total number of violations on LAYER Metal5= 18
```

```
#Total number of violations on LAYER Metal6= 135  
#Total number of violations on LAYER Metal7= 0
```

Figure 20-3 on page 737 illustrates this stage in the design. Violations on different layers are shown by different-colored markers.

**Figure 20-3**



Designs with violations like those in the preceding illustration often have library or design problems, such as overlapping pins, improperly defined tracks, or an insufficient number of rotated vias.

## Encounter User Guide

### Using the NanoRoute Router

---

Use the guidelines in the following table to help evaluate violations:

Violations/Warnings	Check for ...
Many violations on <i>metal1</i>	<ul style="list-style-type: none"><li>■ Improper setup of routing tracks</li><li>■ Overlapping cells</li><li>■ Insufficient via rotation, causing inaccessible pins</li></ul>
	<p>For information on fixing these problems, see the <a href="#"><u>Encounter Library Development Guide</u></a>.</p>
Many violations on <i>metal1</i> and <i>metal2</i>	<ul style="list-style-type: none"><li>■ Offgrid pins</li><li>■ Overlapping tracks</li><li>■ Overlapping cells</li><li>■ Improper X offset, causing offgrid standard cell pins</li><li>■ Pins buried under power routing</li></ul>
	<p>For information on fixing these problems, see the <a href="#"><u>Encounter Library Development Guide</u></a>.</p>
Open net warning messages	<ul style="list-style-type: none"><li>■ Pin modelling, track definition, data, floorplanning problems</li><li>■ Conflicts between option settings or options and library specifications</li></ul>
An open net is one that the router cannot route because it cannot complete the connection of a net.	<p>For information on fixing these problems, see “<a href="#"><u>Resolving Open Nets</u></a>” on page 722.</p>

## **Violations/Warnings**

Violations not decreasing after first iteration of detailed routing

### **Check for ...**

- Offgrid pins
- Overlapping tracks
- Overlapping cells
- Improper X offset, causing offgrid standard cell pins
- Pins buried under power routing

For information on fixing these problems, see the [Encounter Library Development Guide](#).

Violations on upper layers, such as via-to-wire violations or shorts

- Improper routing pitch (not line-to-via)

For information, see “[Violations on Upper Metal Layers](#)” on page 740. For more information on correcting the routing pitch, see the [Encounter Library Development Guide](#).

Localized congestion (also called hot spots)—areas in the congestion map that are red, magenta, or white. The congestion might be caused by one of the following:

- Limited pin access to a block
- Congestion around the corner of a block and in the middle of the standard cells

- Improper placement or floorplanning

For information on placement see “[Placing the Design](#)” on page 537. For information on floorplanning, see “[Floorplanning the Design](#)” on page 353

False violations

- Violation markers on the lower metal layer of a via, even though the actual violation is on the upper layer.

When it flags a via violation, the router places the violation marker on the lower metal layer of the via, whether the actual violation is due to a problem on the lower layer or the upper layer. To repair the violation, rerun detailed routing. The router finds and repairs the violation, even when the marker was reported on the incorrect layer.

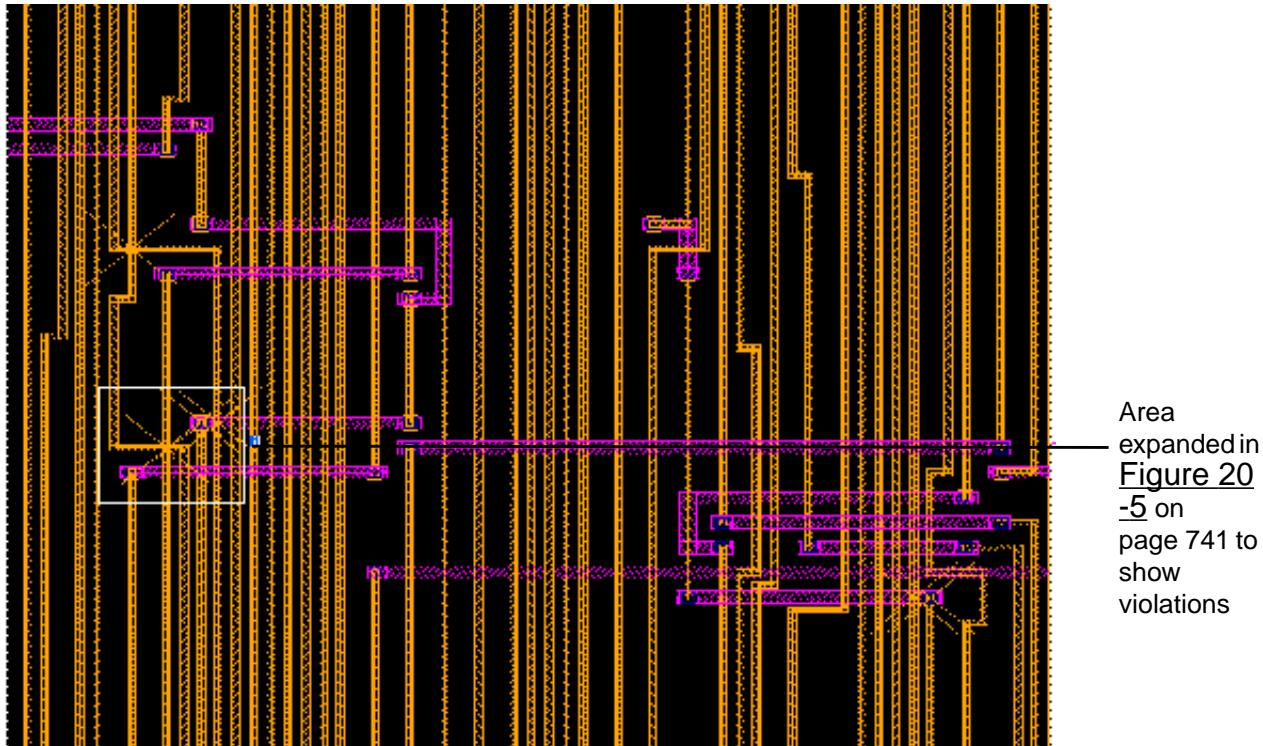
## Violations on Upper Metal Layers

Upper layers are typically used to route on top of macros where only a few routing layers are allowed. These upper layers typically have larger vias than lower layers. When the routing pitch is not set at line-to-via distance, two types of violations are likely to occur:

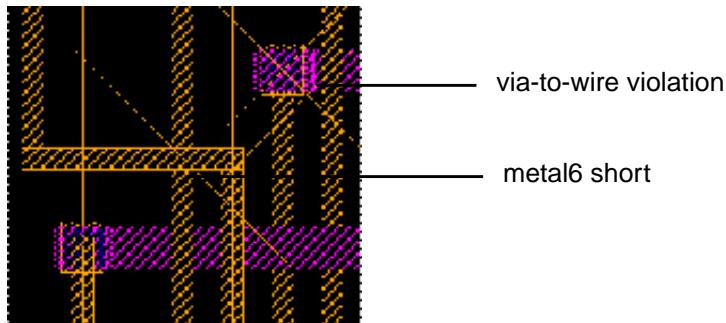
- Via-to-wire violations
- Shorts

[Figure 20-4 on page 740](#), [Figure 20-5 on page 741](#), and the LEF and DEF file excerpts that follow show a design with many violations on *meta16*.

**Figure 20-4**



**Figure 20-5**



The relevant LEF file excerpt is:

```
LAYER Metal6
  TYPE ROUTING ;
  PITCH 0.46 ;
  WIDTH 0.2 ;
  SPACING 0.21 ;
  DIRECTION VERTICAL ;
END Metal6
LAYER Metal7
  TYPE ROUTING ;
  PITCH 0.82 ;
  WIDTH 0.4
  SPACING 42 ;
  DIRECTION HORIZONTAL ;
END Metal7
VIA via6 DEFAULT
  LAYER Metal6 ;
  RECT -0.19 -0.23 0.19 0.23 ;
  LAYER Via6 ;
  RECT -0.18 -0.18 0.18 0.18 ;
  LAYER Metal7 ;
  RECT 0.29 -0.2 0.29 0.2 ;
  RESISTANCE 0.68
END via6
```

The relevant DEF file excerpt is:

```
TRACKS X -4749270 D0 6324 STEP 460 LAYER Metal6
```

To repair the shorts and via-to-wire violations, align the tracks as much as possible without sacrificing them. Change the TRACKS statement in the DEF file to have at least line-to-via STEP (pitch).

The line-to-via calculation for *metal6* is:

$$\begin{aligned} \text{Line to via metal6} &= 1/2 \text{ Width} + \text{Spacing} + 1/2 \text{ Via} \\ &= 0.1 + 0.21 + 0.19 \\ &= 0.5 \end{aligned}$$

## Violations in Timing-Driven Routing

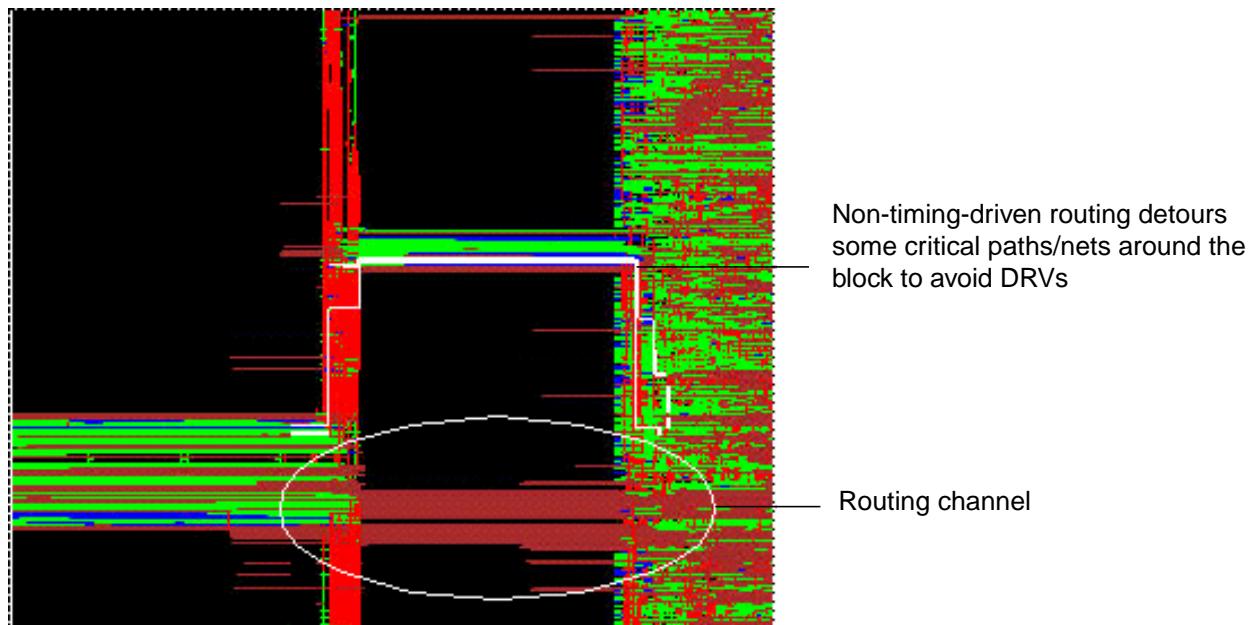
Run time and the number of violations often increase during timing-driven routing because the router restricts the routing of timing-critical nets.

During non-timing-driven routing, the router might detour some nets in order to avoid creating violations. In timing-driven mode, however, the router does not detour timing-critical nets. Instead, it forces them to be routed as short as possible, which can create congestion in the channels. Later, when design-rule checking takes precedence, the router detours timing-critical nets in overly congested channels.

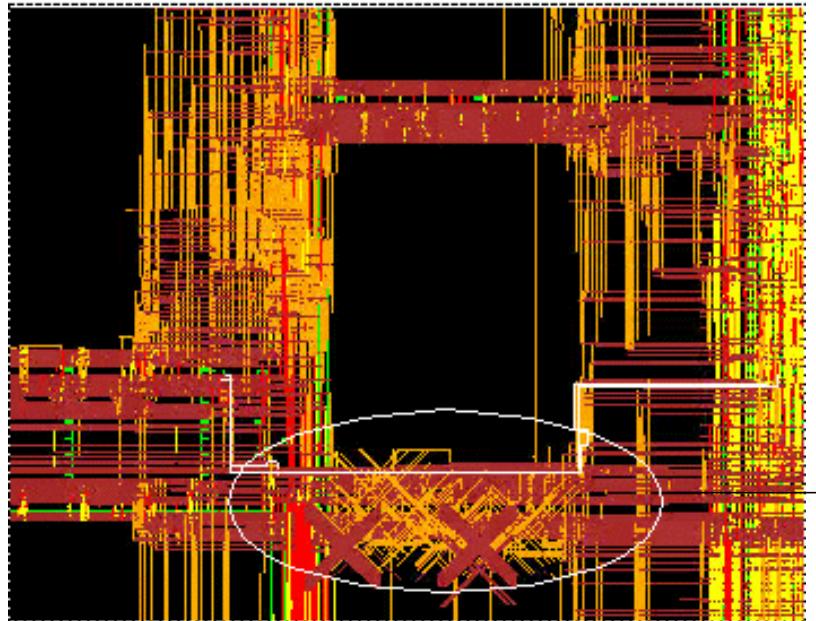
For information on the timing-driven routing flow, see [“Running Timing-Driven Routing”](#) on page 725.

[Figure 20-6](#) on page 742 and [Figure 20-7](#) on page 743 illustrate non-timing-driven and timing-driven routing results for the same design.

**Figure 20-6**



**Figure 20-7**



## Deleting Violated Nets

To delete violated nets, use the `editDeleteViolations` command. After deleting the nets, use ECO routing, detailed routing, or the `globalDetailRoute` command to re-route the design.

### Related Topics

- [detailRoute](#)
- [ecoRoute](#)
- [globalDetailRoute](#)
- [setNanoRouteMode -routeWithEco](#)

## Using Additional Strategies to Repair Violations

### Process Antenna Violations

Repair process antenna violations with antenna repair options or the wire editing commands.

- For information on verifying process antenna violations, see “[Verifying Process Antennas](#)” on page 1114.
- For information on process antenna options, see [Repairing Process Antenna Violations](#).
- For information on wire editing, see “[Editing Wires](#)” on page 649.

### Core Congestion

Ensure that blocks are placed in corners and near boundaries to help ease core congestion.

## Concurrent Routing and Multi-Cut Via Insertion

The NanoRoute router can insert multiple-cut vias during detailed routing in order to achieve a high ratio of multiple-cut to single-cut vias, minimize the number of vias in the design, and increase yield.

To specify the effort level for inserting multiple-cut vias and route the design concurrently, type the following commands:

```
setNanoRouteMode -drouteUseMultiCutViaEffort {medium | high}  
detailRoute
```

For more information on this parameter, see [setNanoRouteMode](#) in the *Encounter Text Command Reference*.

## Postroute Via Optimization

The NanoRoute router can optimize vias on a fully routed design by replacing single-cut vias with multiple-cut vias or with fat vias (single or multi-cut vias with an extended metal overhang). The router does not replace multiple-cut vias during this step.

The router replaces vias by substituting vias in the following order:

1. Fat double-cut vias
2. Normal double-cut vias
3. Fat single-cut vias

Ensure the following before replacing the vias:

- Double-cut vias and fat vias are automatically generated or defined in the LEF file.  
    Use the [generateVias](#) command to generate vias.

- The design is completely global and detailed routed.  
If you delete any wires after routing, reroute the design before replacing the vias.
- The design is free of all DRC violations.

Complete the following steps:

1. To run postroute via reduction, type the following commands:

```
setNanoRouteMode -drouteMinSlackForWireOptimization slack  
setNanoRouteMode -droutePostRouteMinimizeViaCount true  
routeDesign -viaOpt
```

**Note:** When you run these commands, the software replaces all multiple-cut vias with single-cut vias. Use these commands only if no concurrent via optimization was done.

2. To run postroute single-cut to multiple-cut via swapping, complete one of the following steps:

- a. To run postroute single-cut via to multiple-cut via swapping, type the following commands:

```
setNanoRouteMode -drouteMinSlackForWireOptimization slack  
setNanoRouteMode -droutePostRouteSwapVia multiCut  
routeDesign -viaOpt
```

- b. To run non-timing-driven postroute single-cut via to multiple-cut via swapping, type the following commands:

```
setNanoRouteMode -routeWithTimingDriven false  
setNanoRouteMode -droutePostRouteSwapVia multiCut  
routeDesign -viaOpt
```

## Related Topics

- [Using the routeDesign Supercommand](#) on page 700
- [routeDesign](#) in the “Route Commands” chapter of the *Encounter Text Command Reference*

## Optimizing Vias in Selected Nets

To optimize vias in selected nets, set the `-skip_routing` attribute to `true` for all nets, then set the attribute to `false` for nets with vias you want to optimize.

```
setAttribute -net * -skip_routing true  
setAttribute -net ... -skip_routing false  
globalDetailRoute
```

## Via Optimization Options

- -droutePostRouteSwapVia
- -drouteUseBiggerOverhangViaFirst
- -drouteUseMultiCutViaEffort
- -droutePostRouteMinimizeViaCount
- -routeConcurrentMinimizeViaCountEffort

To ensure that the router chooses vias with the largest overhang first, specify the following option:

```
setNanoRouteMode -drouteUseBiggerOverhangViaFirst true
```

To minimize the number of vias, specify the following options:

```
setNanoRouteMode -droutePostRouteMinimizeViaCount true  
setNanoRouteMode -drouteEndIteration default
```

For more information on these options, see “[Route Commands](#)” in the *Encounter Text Command Reference*.

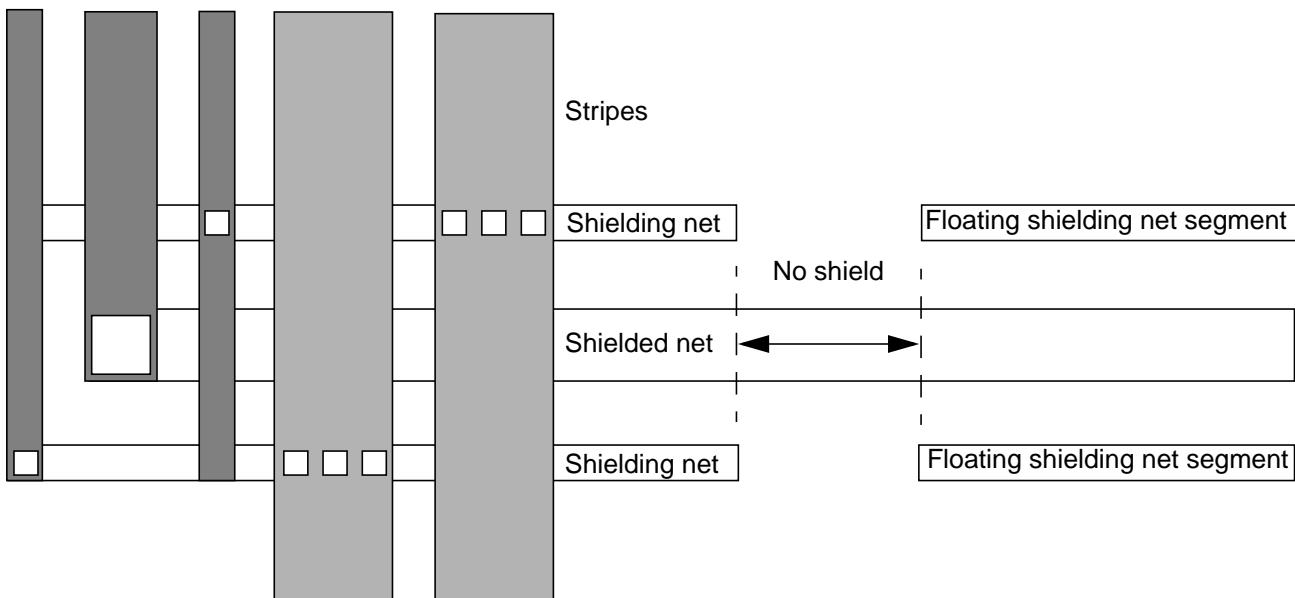
## Performing Shielded Routing

The NanoRoute router can protect noise-sensitive nets, such as a clock nets, from crosstalk by shielding them with power or ground wires. You typically route shielded nets before routing other nets. At the end of routing, the router deletes shielding wires that are not connected to power or ground wires. NanoRoute automatically generates a shielding statistics report after routing. For information on the report, see “[Interpreting the Shielding Report](#)” on page 749.

[Figure 20-8 on page 747](#) shows a section of a design with a shielded net. In the figure,

- The signal net is shielded by a power net on one side and a ground net on the other side.
- Multiple vias can be dropped where a stripe crosses the shielding net at a right angle, if the stripe is wide enough to accommodate them.
- A segment of the signal net is not shielded.
- There are some floating shielding net segments.

**Figure 20-8**



### Shielding Option

Use the following option to control shielding:

- `setNanoRouteMode -routeMinShieldViaSpan`

This option controls the distance between vias and special nets that are used as shield wires.

## Performing Shielded Routing Using the GUI

1. From the main menu, choose *Route – NanoRoute/Attribute*.

This opens the NanoRoute/Attributes form.

2. On the NanoRoute/Attributes form, enter the name of the net to shield (this is the shielded net in the figure) in the *Net Name(s)* field.
3. Enter the name of the power ground net (or both) in the *Shield Net(s)* field. These are the shielding nets in the figure.
  - To shield both sides with ground wires, enter the name of the ground net.
  - To shield one side with a ground wire and one side with a power wire, enter both the ground and the power net names.

4. Click *OK* or *Apply*.

5. Use the Encounter `selectNet` command to specify the net to shield. It must be the same as the net you specified on the NanoRoute/Attributes form.

6. From the main menu, choose *Route – NanoRoute*.

This opens the NanoRoute form.

7. On the NanoRoute form, select the following:

- a. In the *Job Control* area, select *Selected Nets*.
- b. In the *Mode* area select both *Global Route* and *Detail Route*.

8. Click *OK* or *Apply*.

To route the remaining nets, complete the following steps:

1. On the NanoRoute/Attributes form, set the *Skip Routing True* for the shielded nets.



You can also skip routing on prerouted nets by issuing the following command:

```
setAttribute -net @PREROUTED -skip_routing true
```

`@PREROUTED` applies to a net that has any wiring, including partial wiring.

2. On the NanoRoute form, deselect *Selected Nets Only*.
3. Click *OK* or *Apply* to reroute the design.

## Performing Shielded Routing Using Text Commands

- The following commands shield net1 with both power and ground wires, and shield net2 with a ground wire:

```
setAttribute -net net1 -shield_net vdd \
             -shield_net vss
setAttribute -net2 -shield_net vss
globalDetailRoute
```

- The following commands show how to shield two nets (do not shield more than one net with the same command):

```
setAttribute -net net1 -shield_net abc_gnd
setAttribute -net net2 -shield_net abc_gnd
```

## Interpreting the Shielding Report

The software generates a shielding report for the NanoRoute router when you run the `reportShield` command. You can customize the report to output information on the whole design or on selected nets, and you can report per-layer statistics.

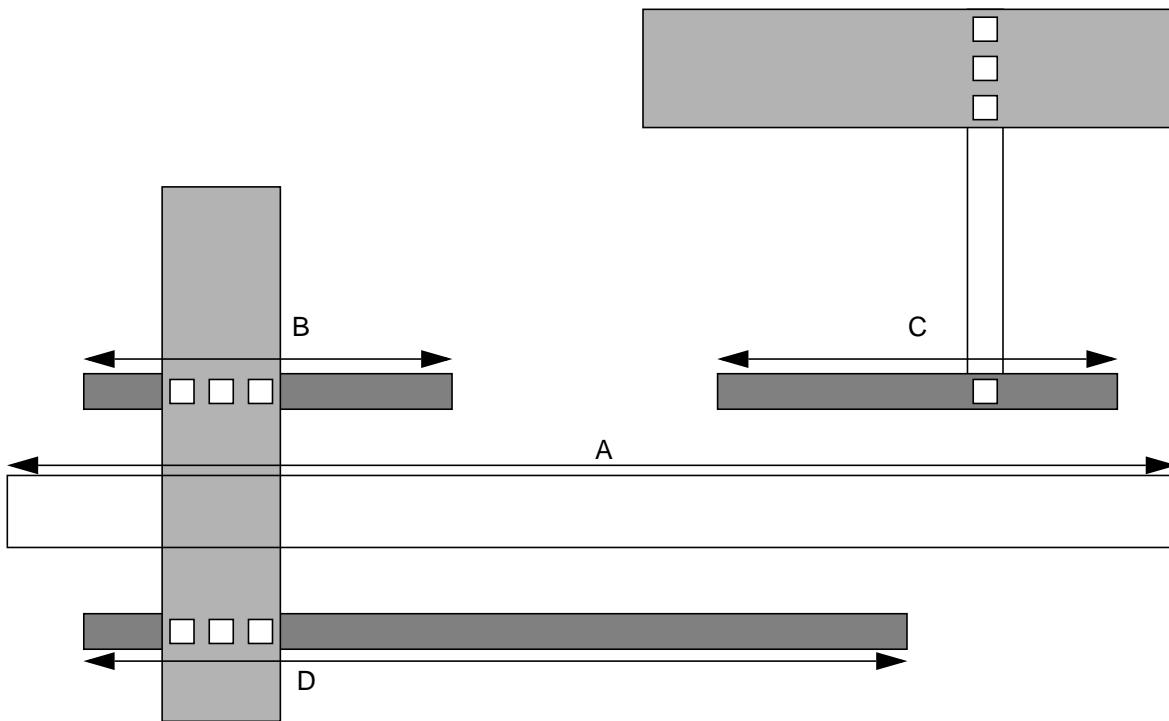
Following is a section of a report:

```
-----
Name      : Shielded net name
Length    : Shielded net length
Shield    : Total length of shielding wire
ratio     : Average shielding ratio
-----
Name      Length   Shield  Ratio    Layer:  Length   Shield  Ratio
-----
netA:      211.5   378.3  0.894
                  METAL2:      5.0      2.2      0.222
                  METAL3:    107.4    180.1      0.839
                  METAL4:     99.1    196.0      0.989
average:   211.5   378.3  0.894
                  METAL2:      5.0      2.2      0.222
                  METAL3:    107.4    180.1      0.839
                  METAL4:     99.1    196.0      0.989
```

## Encounter User Guide

### Using the NanoRoute Router

To help understand the report, see the following figure, which shows a section of `netA`:



In the figure,

- A Represents the shielded net.
- B, C, and D Represent shielding wires.

The software calculates the shielding ratio by using the following formula:

$$\text{Shielding Ratio} = \frac{B + C + D}{A \times 2}$$

## Routing Wide Wires

The NanoRoute router automatically tapers wide wires when connecting to pins, including input/output pins of standard cells, macro cells, and block output pins. The tapered portion of a wire uses the minimum-width wire (the default width).

If you do not want tapering at the output pins, specify the following parameter:

```
setNanoRouteMode -drouteNoTaperOnOutputPin true
```

## Using Non-Default Rules

By default, the NanoRoute router treats non-default rule spacing as a soft option; that is, when routing resources are available, it honors the non-default rule. If the area is too congested, and resources are not available, the router might not honor the rule.

If you enable signal-integrity driven routing, the router attempts to minimize overall coupling capacitance in the design. If you enable timing-driven routing, the router also favors critical nets when choosing spacing.

You can use up to 254 nondefault rules. Nondefault rules do not necessarily decrease the routing speed. Routing speed does decrease, however, due to the following factors:

- The ratio of non-default rule wires to default rule wires increases.
- The amount of space between wires increases.
- The number of additional nondefault vias increases, due to the nondefault rules.

In congested areas, the router might violate nondefault spacing rules in order to avoid design-rule violations and complete the routing. Its flexibility with regard to nondefault spacing decreases the overall wirelength and benefits timing and signal integrity because it allows some shorter nets to be more easily tolerated near adjacent nets without causing violations.

**Note:** You can force the router to honor the nondefault rules by specifying the following option:

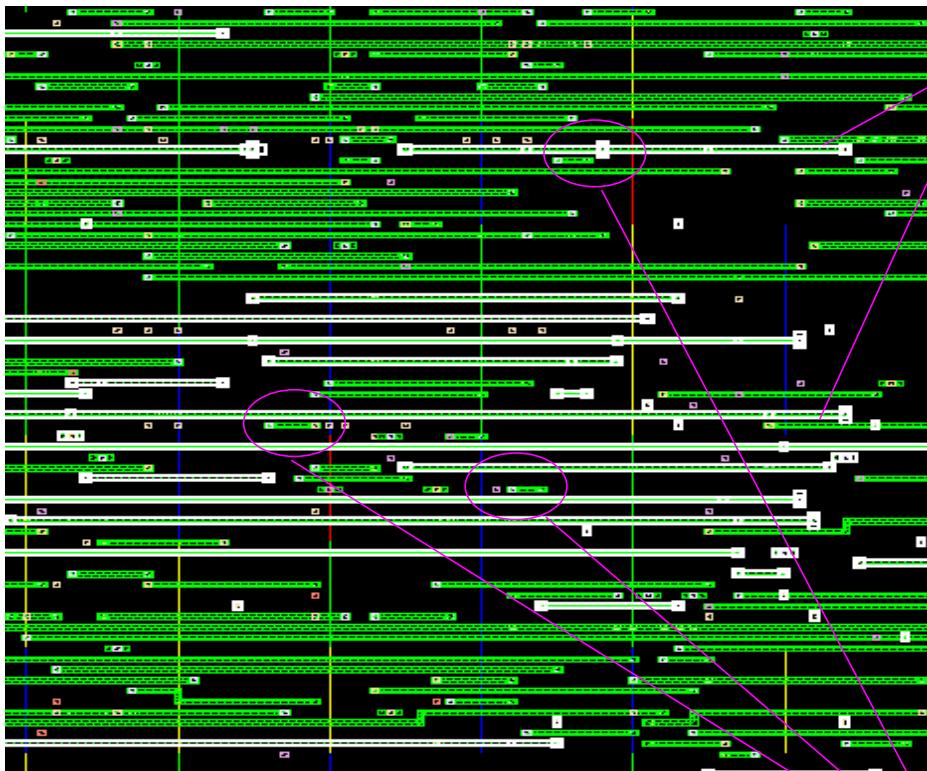
```
setNanoRouteMode -routeStrictlyHonorNonDefaultRule true
```

[Figure 20-9 on page 752](#) illustrates nondefault spacing (“soft spacing”) routing.

## Encounter User Guide

### Using the NanoRoute Router

Figure 20-9



With soft spacing, the router does not need to detour around these nets.

With soft spacing, some of the shorter nets that are close to other nets are tolerated.

## Repairing Process Antenna Violations

The NanoRoute router can repair process antenna violations concurrently with DRC violations during the search-and-repair step. During the postroute optimization step, when there are no more DRC violations, the router repairs additional process antenna violations. This two-step methodology allows the router to use more aggressive methods to repair the process antenna violations early on and saves CPU time.

During postroute optimization, the router repairs antenna violations by changing layers (also called antenna stapling or layer hopping). It also repairs process antenna violations by inserting diode cells as close as possible to input gates to discharge current, and deleting and rerouting nets with violations.

**Note:** After routing, run the `globalNetConnect` command to ensure connectivity to power and ground pins in antenna cells added during process antenna repair. For information, see [globalNetConnect](#) in the *Encounter Text Command Reference*.

The router supports hierarchical process antenna calculations and repair.

For information on PAE calculations, and the LEF and DEF antenna parameters, see [“Calculating and Fixing Process Antenna Violations”](#) in the *LEF/DEF Language Reference*.

## Repairing Violations on Multiple-Pin Nets

On multiple-pin nets, the router does the following:

- On a two-pin net that has one input pin with antenna information and one output pin without antenna information, the router tries to repair the antenna violation based on input antenna information only.
- On a two-pin net that has one input pin without antenna information and one output pin with antenna information, there is usually no antenna violation on the output pin, so the router skips antenna repair.
- On a two pin-net where the router does not have any antenna information on either pin, the router skips antenna repair.
- On a three-pin net that has two input pins—one with antenna information and one without antenna information—and one output pin without antenna information, the router skips antenna repair.

## Changing Layers

The router automatically shortens wires whose area exceeds the gate/wire area ratio set in the LEF file. This process might not guarantee that it can resolve all antenna violations—if the routing area is congested, process antenna violations can still occur, just as shorts and spacing violations can occur.

## Using Diodes

The router inserts antenna diode cells or uses preplaced diode cells to repair violations. It can swap filler cells with antenna diode cells and fill the gap automatically if an antenna diode cell is not the same size as the filler cell it replaced. A later routing pass does not remove previously placed diodes.

The antenna diode cells must have the same LEF SITE definition as the standard cells. Specify the diode cell name using the *Diode Cell Name* option on the NanoRoute form or the `-routeAntennaCellName` option on the text command line.

## Deleting and Rerouting Nets with Violations

If the design has more than 100 DRC violations, and you are using LEF 5.4 or later, the router deletes and reroutes nets with process antenna violations.

## Repairing Violations on Cut Layers

The NanoRoute router detects antenna violations on cut layers and repairs them by inserting diodes. To repair these violations, you must specify a value in your LEF file for the `ANTENNADIFFAREARATIO` (or `ANTENNACUMDIFFAREARATIO`) for the cut layers. For each cut layer, the value for `ANTENNADIFFAREARATIO` (or `ANTENNACUMDIFFAREARATIO`) must be larger than the value for `ANTENNAAREARATIO` (or `ANTENNACUMAREARATIO`).



*Important*  
If you do not use diodes to repair process antenna violations, the router cannot repair the violations on cut layers.



*Tip*  
To highlight the diodes that the router inserts, use the choose *Edit – Select by Name*. To highlight the diodes, type `*_antenna_*`.

For information on the Select by Name form, see “Edit Menu” in the *Encounter Menu Reference*.

- To specify the diode cells, use `-routeAntennaCellName`.



#### *Tip*

To force the router to do more layer changing and skip diode insertion, specify the following option:

```
setNanoRouteMode -routeInsertAntennaDiode false
```

After the router repairs as many violations as possible by layer changing, reset this option to `true` and repeat process antenna repair.

## Process Antenna Options

Use the following options to repair violations caused by process antennas:

- `setNanoRouteMode` options:
  - `-drouteFixAntenna`
  - `-routeAntennaCellName`
  - `-routeAntennaPinLimit`
  - `-routeDeleteAntennaReroute`
  - `-routeFixTopLayerAntenna`
  - `-routeIgnoreAntennaTopCellPin`
  - `-routeInsertAntennaDiode`
  - `-routeInsertAntennaInVerticalRow`
  - `-routeInsertDiodeForClockNets`
- `setAttribute -nets netName -skip_antenna_fix`

## Examples

- The following commands shows the basic strategy for repairing process antenna violations:

```
setNanoRouteMode -drouteFixAntenna true  
setNanoRouteMode -routeAntennaCellName "ANTENNA"  
setNanoRouteMode -routeInsertAntennaDiode true
```

## Encounter User Guide

### Using the NanoRoute Router

---

```
globalDetailRoute  
globalNetConnect
```

The NanoRoute router runs global and detailed routing. After repairing DRC violations, it repairs as many process antenna violations as it can by layer hopping during postroute optimization. If any process antenna violations remain, the router repairs them by inserting antenna diode cells named `ANTENNA`.

- The following commands repair process antenna violations by inserting diodes and filler cells. The filler cells are specified by the `setFillerMode -core` command. They fill any gaps that is left when a diode replaces a large filler cell.

```
setNanoRouteMode -routeInsertAntennaDiode true  
globalDetailRoute  
globalNetConnect
```

For information on adding filler cells, see `setFillerMode` and `addFiller` in the [“Placement Commands”](#) chapter of the *Encounter Text Command Reference*.

## Using a Design Flow that Includes Astro or Apollo

The NanoRoute router uses the information in the Milkyway technology file to automatically map vias and layers in the design to Astro scheme format. The router maps only the routing data, so it does not map DEF vias or special routing.

To use Astro or Apollo in your design flow, you must have a LEF technology file that contains layer and via descriptions that match one-to-one with the descriptions in the Milkyway technology file.



***Check with the foundry before making any changes to the original files.***

You can create a rule file to map layers or vias that are not mapped automatically.

In native mode, the following commands are used in this flow:

■ generateLef

Converts Milkyway technology file descriptions, Milkyway CLF files, customized LEF technology files from customers, and other older syntax and non-optimal LEF files to a LEF file with target rules and automatically generated vias. Issue this command before routing.

■ loadATFile

Contains the path to the technology file required by Astro. Issue this command after loading the LEF file.

You need to issue this command only once, since the mapping results are persistent.

■ tdfOut

Outputs a routed database based on the mapping results. Issue this command after routing.

If you are running the router standalone mode, issue the following command:

■ pdi export design -aef -rule

Finds the technology file and automatically maps the layers and vias from the LEF file to a format that Astro can read. Optionally, includes user-created rules for additional mapping. Issue this command after routing.

## **Encounter User Guide**

### Using the NanoRoute Router

---

The following command outputs a file named `MyOutputFile`, using a rule file named `tfo.map`:

```
pdi export_design -aef -rule tfo.map MyOutputFile
```

If you do not need any additional mapping, the only line in `tfo.map` is

```
techfile apollo.tf
```

## Troubleshooting

If you have problems with your design, try the following troubleshooting tips:

1. Check the log file for errors and warnings. Correct the problems and continue routing or reroute, as appropriate.

For example, the router might stop routing automatically if it finds too many violations. If the router stops unexpectedly, check the log file for a message that the router has reached the maximum number of violations and set the following setNanoRouteMode parameter to false to continue routing:

`-drouteAutoStop`

2. Verify connectivity and geometry before and after routing and compare results.

You can also use the checkRoute command to verify the connectivity. It is faster than verifyConnectivity, but does not output a report.

3. Save the database after routing and restore it in a new session in the Encounter software.

Saving and restoring the database clears temporary data structures in memory.

4. Issue the defOut command, then defIn, and reroute.

The defOut command saves all routing information in DEF and restores a clean database for routing.

**Encounter User Guide**  
Using the NanoRoute Router

---

---

## Using the Encounter Mixed Signal Router

---

- [Overview](#) on page 762
- [Before You Begin](#) on page 763
- [Results](#) on page 763
- [Specialized Routing Techniques](#) on page 764
  - [Matched Nets](#) on page 764
  - [Differential Pair Nets](#) on page 768
  - [Bus Routes](#) on page 769
  - [Shielded Nets](#) on page 769
- [Using Routing Constraints](#) on page 773
- [Constraint File Format](#) on page 773
- [Generic Constraints](#) on page 774
- [Specialized Constraints and Keyword Descriptions](#) on page 776
  - [NETS](#) on page 776
  - [MATCH](#) on page 778
  - [DIFFPAIR](#) on page 780
  - [BUS](#) on page 782
  - [SHIELDING](#) on page 784
- [Creating a Constraint File](#) on page 797
- [Editing a Constraint File](#) on page 805
- [Loading a Constraint File](#) on page 802
- [Sample Constraint File](#) on page 807

## Overview

You can use the Encounter software to design and implement the following styles of mixed-signal ICs:

- Analog-on-top (AOT)

In AOT-style designs, the chip is mostly analog, but has some digital blocks. AOT designs use Virtuoso® software as the cockpit and Encounter® software to integrate the digital blocks.

See the Virtuoso documentation for descriptions of the Virtuoso commands, GUI, and use model.

- Digital-on-top (DOT)

In DOT-style designs, the chip is mostly digital, but has some analog blocks. DOT designs use Encounter software as the cockpit and Virtuoso software to integrate the analog blocks.

This document provides information on using mixed signal constraints in the DOT flow.

### Related Topics

- Application Notes on SourceLink®

- [Solution Flow Name: Encounter-Virtuoso Mixed Signal Floorplanning and Physical Implementation Flow—Big designs with large number of blocks](#)
  - [Solution Flow Name: Virtuoso-Encounter Mixed Signal Physical Design Flow—Sea of Standard Cells Methodology](#)

- *Encounter Text Command Reference*

- [“Mixed Signal Commands” chapter](#)

- *Encounter Menu Reference*

- [“Route Menu” \(Mixed Signal Router\)](#)
  - [“Tools Menu” \(Mixed Signal Constraint Editor\)](#)
  - [“Verify Menu” \(Verify Mixed Signal Constraints\)](#)

## Using the Mixed Signal Router

Use the Encounter mixed signal router to preroute critical signal nets in a mixed signal design. After routing, the nets are saved as special nets so the NanoRoute® router does not delete or alter them.

In addition to honoring the LEF rules, the router performs the following types of specialized routing by observing design rules specified in a routing constraint file:

- Matched net routing
- Differential pair routing
- Bus routing
- Shielded routing, including coaxial shielded routing, differential pair shielded routing, and bus shielded routing for bus.

The constraint file can also include design rules for the following net attributes—the router honors the rules as long as they do not override the rules specified in the LEF file:

- Minimum number of via cuts
- Width and maximum width
- Spacing
- Routing layers
- Minimum and maximum resistance
- Minimum and maximum capacitance
- Tapering

## Before You Begin

Before routing, the design must be fully placed so the router can take congestion from placement into account.

## Results

After routing, the database is ready for verification with the mixed signal verifier and for routing the remainder of the signal nets with the NanoRoute router.

## Specialized Routing Techniques

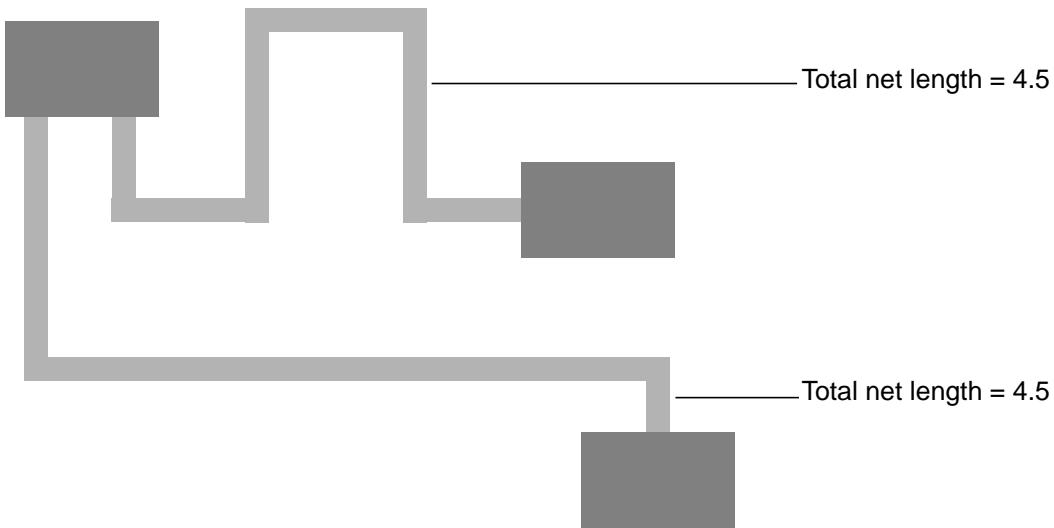
### Matched Nets

Matched nets are constrained by resistance, but not by topology. Matched nets are not limited to two nets. You can specify a tolerance value for the difference in resistance of the nets, so the router attempts to match the resistance of the nets, but does not create a violation unless the difference in resistance is greater than the tolerance value.

For information on router resistance calculations, see “[Minimum and maximum resistance](#)” on page 775.

For the syntax of matched nets constraints, see [MATCH](#) on page 778

The following figure shows a pair of matched nets.



### Controlling the Width of Serpentine Coils in Matched-Net Routing

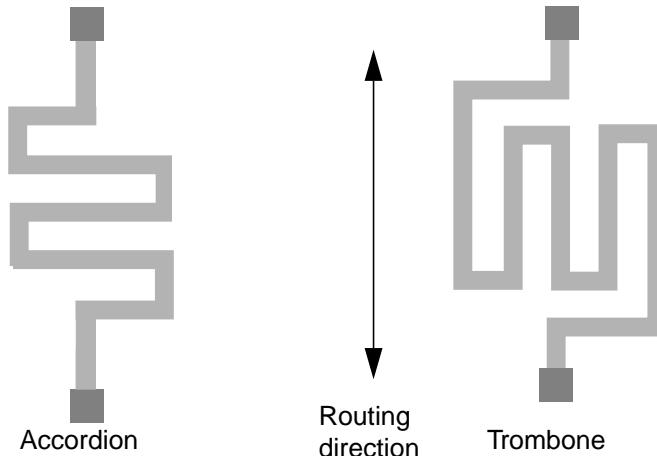
In order to match the resistance of wires on which matched-net constraints are applied, the router might need to lengthen short wires and create jogs that cause the wires to take the shape of a serpentine coil. If these jogs are too wide, they can block routing resources and increase coupling.

- If the coils are orthogonal to the routing, the router creates an accordion shape.
- If the coils are parallel to the routing, the router creates a trombone shape.

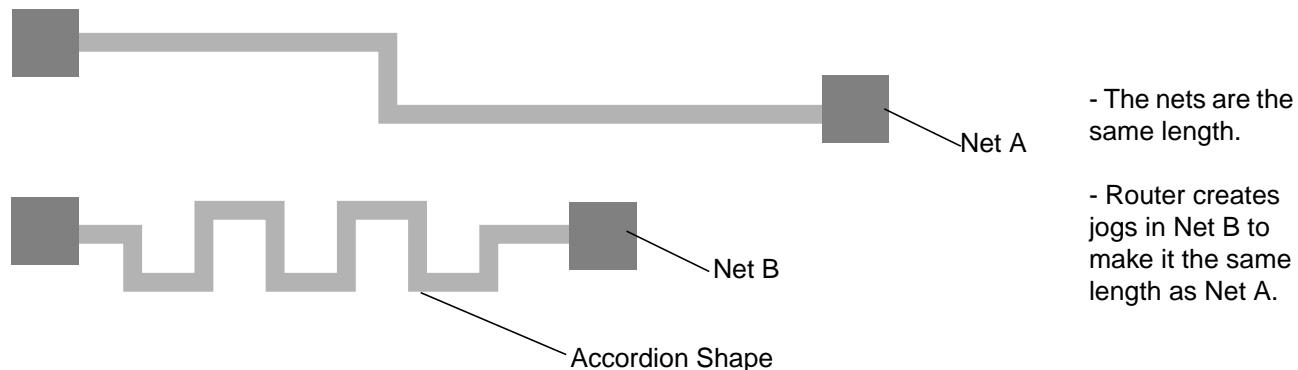
## Encounter User Guide

### Using the Encounter Mixed Signal Router

The following figure shows two coils: an accordion and a trombone.



The following figure shows matched nets with an accordion:



You can control accordion and trombone shapes by specifying values for the width of the coil and the spacing between the wires in the coil.

#### ***Controlling the Coil Width***

To control the width of the coil, use the following variable before you run the mixed signal router:

```
msMatchMaxCoilWidth integer
```

The software multiplies minimum width of the wire that creates the coil by the integer specified by this variable. The result is the maximum distance from the center line of the coil to an outside edge of the coil.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

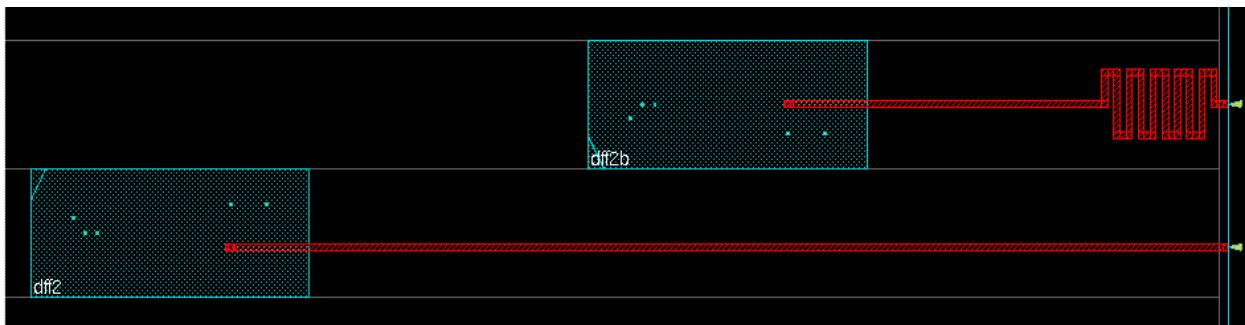
---

If you specify 0, or do not use this variable, the maximum width of the accordion shape is not constrained. In general, Cadence recommends a value of 5 or 6 for this variable.

For example, if you type the following command, the maximum distance from the center of the coil to an outside edge is five times the minimum width of the wire:

```
set msMatchMaxCoilWidth 5
```

The following figure shows a pair of matched nets: The top net has an accordion shape. The distance from the centerline of the accordion to the centerline of a wire segment that forms the accordion's outside edge is five times the wire width.



### **Controlling the Coil Spacing**

To control the spacing between the coil wires, use the following variable before you run the mixed signal router:

```
msMatchMinCoilSpacing float
```

The software routes the wires so that the spacing between the wires in the coil is at least the specified float value. This variable allows you to use a float value to give the software more flexibility when controlling the spacing.

For example, if you type the following command, the minimum distance between the wires is 1.5 times the wire width:

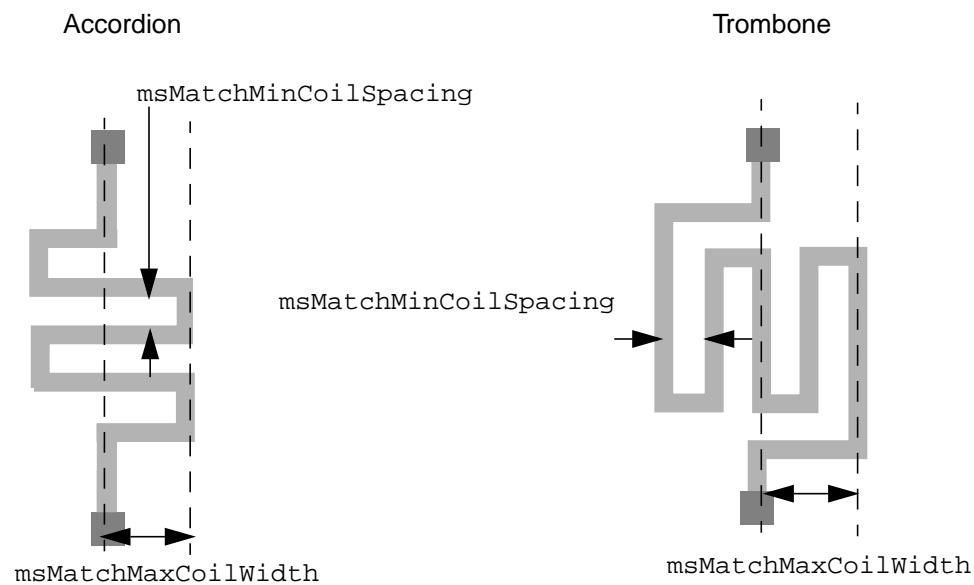
```
set msMatchMinCoilSpacing 1.5
```

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

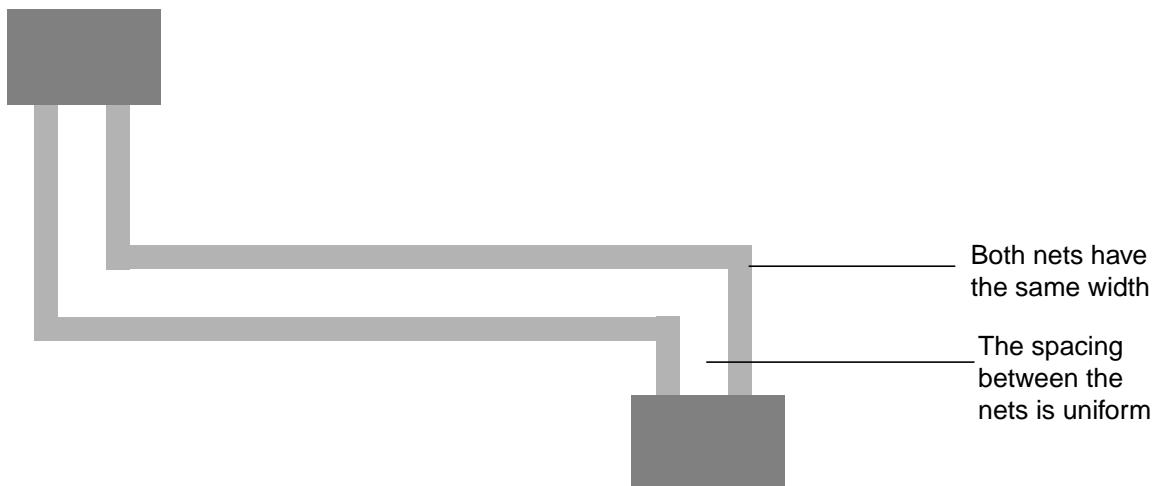
The following figure shows how the `msMatchMaxCoilWidth` and `msMatchMinCoilSpacing` variables control the matching style:



## Differential Pair Nets

Differential pair nets are symmetrical nets that are constrained by width and spacing. You can specify a threshold value for the spacing so the router attempts to meet the spacing, but does not create a violation unless the spacing varies by more than the threshold value.

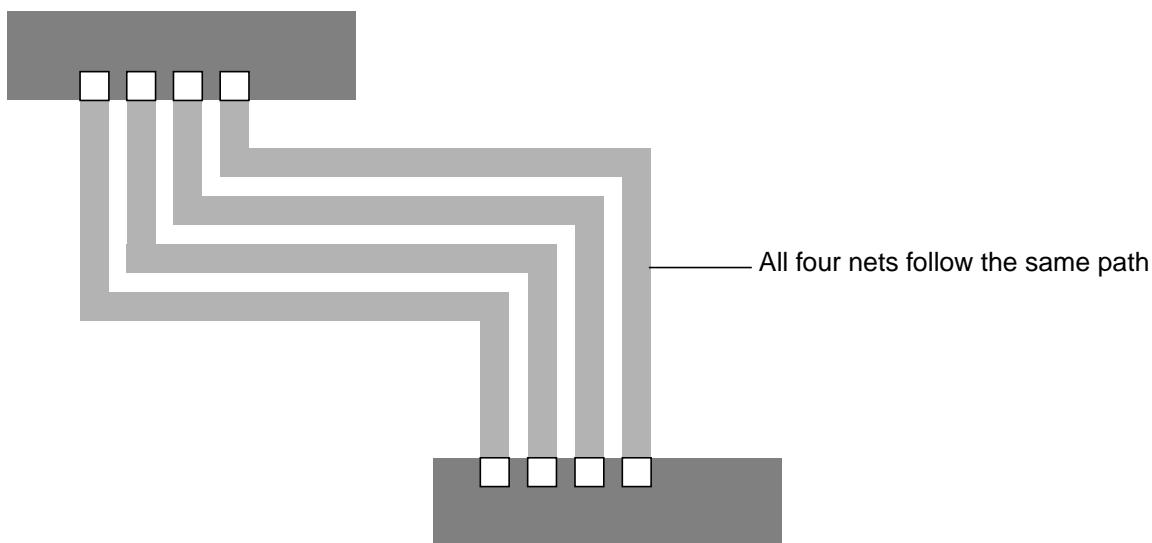
The following figure shows differential pair nets.



## Bus Routes

Bus routes are bundles of nets that follow the same pattern from the source to the target. The router can route a maximum of 512 nets in a bus. If it cannot route the entire bus following the same pattern, it routes the section of the bus that it can route, and issues a warning that gives the reason it cannot finish routing the bus. You can break down the remaining sections of the bus into progressively smaller sections until the router can route the entire bus. For example, if the router is trying to route a 256-net bus, but can route only 64 nets, it routes the first 64 nets and issues a warning. You can then break down the rest of the bus into smaller bundles and rerun the router.

The following figure shows a routed bus with four nets.



## Shielded Nets

Shielded nets are noise-sensitive signal nets that are protected by special nets that are routed nearby.

Shielded routing has the following characteristics:

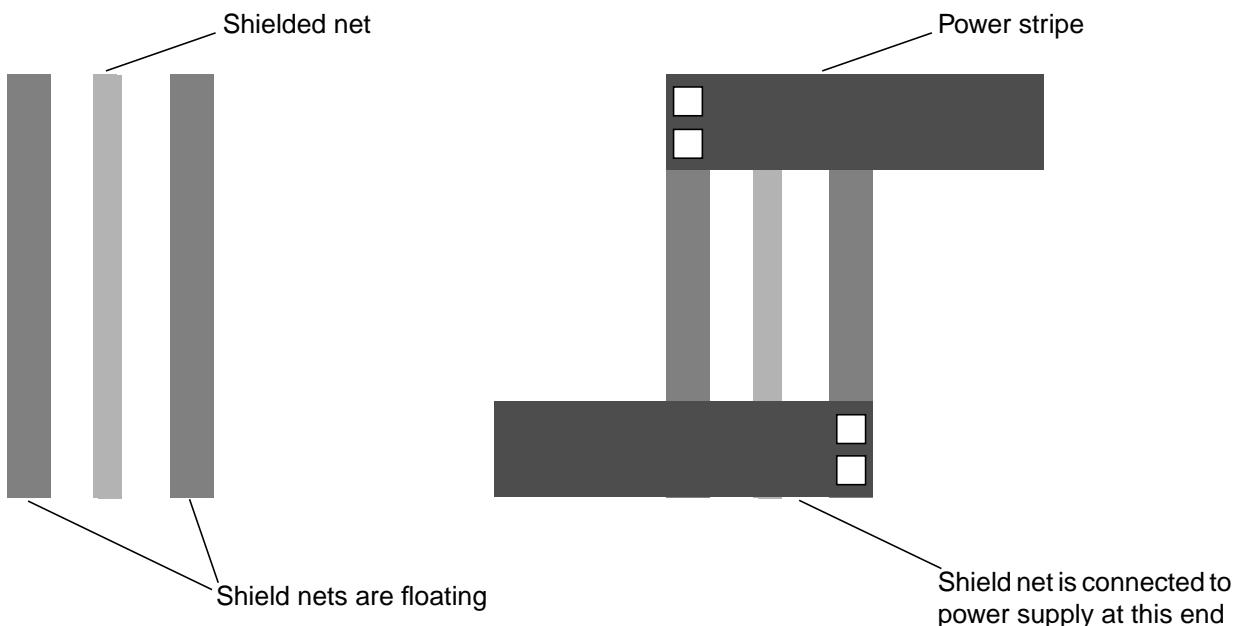
- The noise-sensitive net (the *shielded net*) is shielded by a special net (the *shield net*) on its right and left sides.
- A noise-sensitive net can also be shielded on the top and bottom (the next higher and lower routing layers). This type of shielding is called coaxial shielding.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

- The shield net can be unconnected to the main power or ground supply lines (this is called a floating shield) or it can be connected. If it is connected, the connection can be at any point along the shield.
- The router can shield differential pair nets.
- By default, parallel shielded nets share a shield net.

The following figures show a net segment that is shielded on the right and left sides. In the figure on the left, the shield nets are floating. In the figure on the right, the shield nets are connected to the power supply.



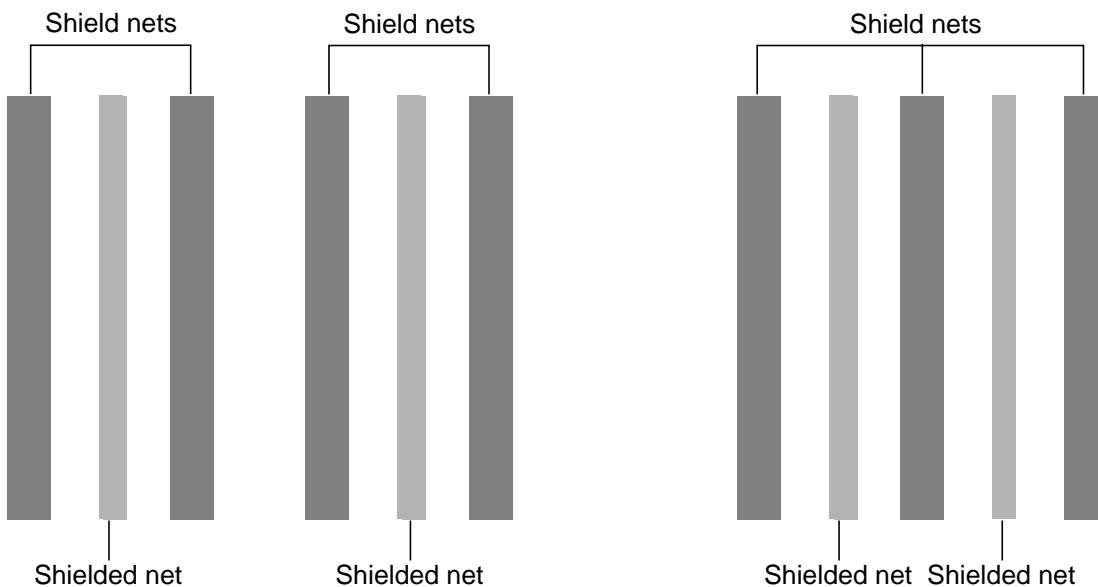
## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

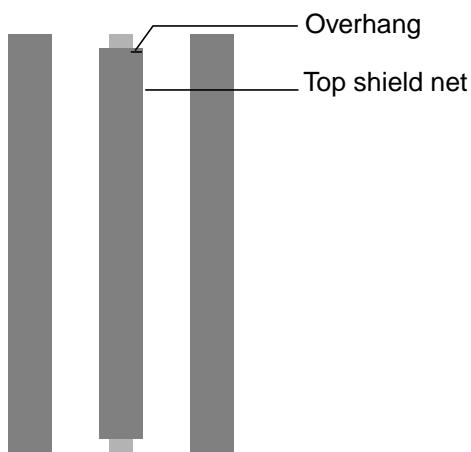
#### Shared Shield Nets

By default, the router allows parallel nets to share shield nets. The following figures show segments of two nets that are shielded on the right and left sides. In the figure on the left, each shielded net has its own shield nets. In the figure on the right, the shielded nets share the center shield net.



#### Coaxial Shield Nets

The following figure shows a segment of a net that is shielded on the right, left, top, and bottom sides. The shield nets are wider than the net that is shielded. The portion of the top and bottom shield net that is wider than the shielded net is called the overhang.



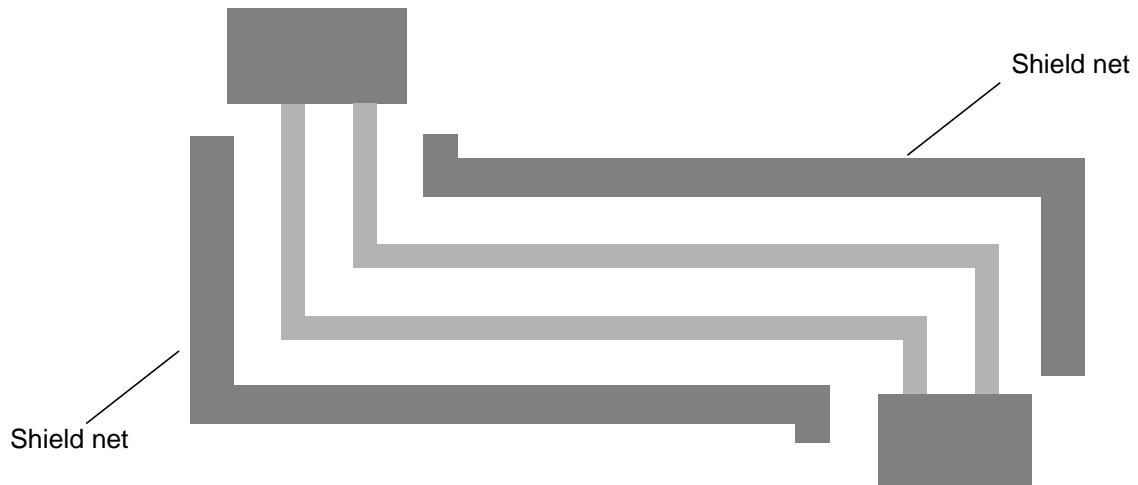
## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

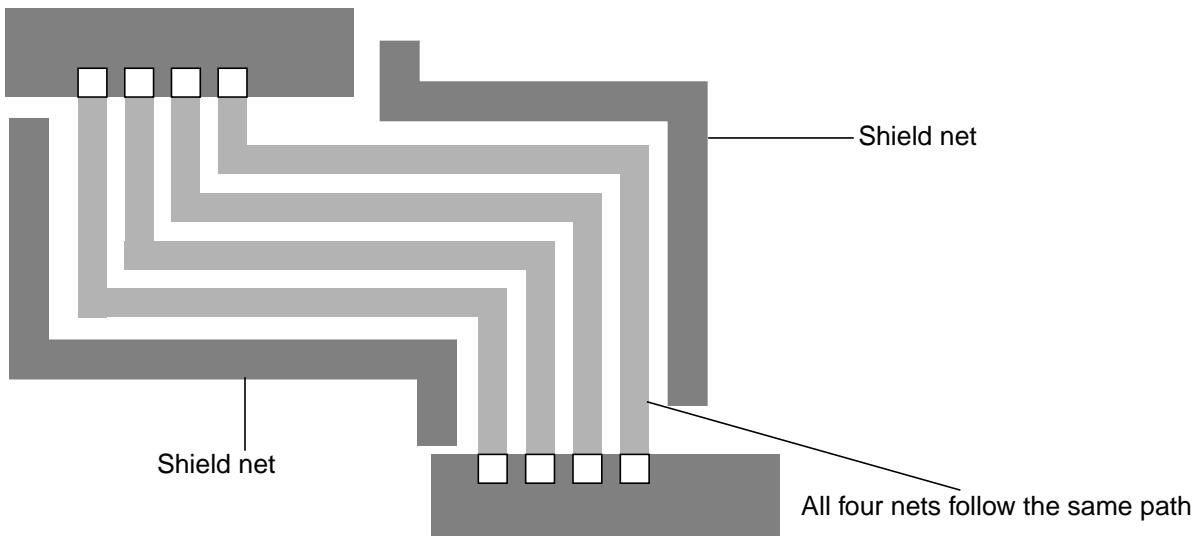
#### Shielded Differential Pair Nets

The following figure shows shielded differential pair nets. Differential shielded pair nets can be shielded on the outside right and left sides.



#### Shielded Bus Nets

The following figure shows shielded bus nets. Bus nets can be shielded on the outer left and right sides.



## Using Routing Constraints

The router honors the LEF rules for the routing layers. It honors the constraints in the routing constraint file only if they do not conflict with the rules specified in the LEF file for the design. If there is a conflict, the router generates a message and uses the LEF rule.

You can create a constraint file or edit an existing constraint file by using the Mixed Signal Constraint Editor in the Encounter GUI or by using a text editor.

For more information, see [“Creating a Constraint File” on page 797](#)

## Constraint File Format

The constraint file is an ASCII file with the following format:

- The file does not have a header.
- Constraint parameters and keywords are specified in uppercase letters.
- There is no punctuation at the end of the lines.
- Comments begin with a number sign (#).
- The number of constraints is not limited.
- Constraints may be generic or specialized. Generic constraints can be listed separately from specialized constraints, or they can be included as part of a specialized constraint. For more information, see [“Generic Constraints” on page 774](#).
- Each specialized constraint is one of the following types: NETS, MATCH, DIFFPAIR, SHIELD, BUS. For more information, see [“Specialized Constraints and Keyword Descriptions” on page 776](#).
- Each specialized constraint has the following format:

```
CONSTRAINT_TYPE
  KEYWORD value
  KEYWORD value
  KEYWORD value
  ...
  constrained_net_name constrained_net_name ...
END CONSTRAINT_TYPE
```

- Specialized constraints may be listed in any order, but in practice they are grouped by type, that is, all the DIFFPAIR constraints are listed, then all the SHIELD constraints, and so on.

## Generic Constraints

The routing constraint file can include generic constraints that are used alone or as part of the constraints that control specialized routing. If a generic constraint is used alone, it is included at the top of the constraint file and applies to all specialized constraints unless it is denoted in the specialized constraint.

For example, if many differential pair constraints limit the distance between two differential pair nets to 0.5 microns, the constraint file can include the following generic constraint before the rest of the differential pair constraints:

```
PAIRGAP 0.5
```

This constraint applies to any differential pair constraints that follow that do not include a PAIRGAP keyword.

In the following example, nets A and B will have a PAIRGAP of 0.5, but nets C and D will have a PAIRGAP of 0.8:

```
PAIRGAP 0.5
DIFFPAIR
    WIDTH 0.2
    MINCUT 2
    DTMF_INST/A DTMF_INST/B
END DIFFPAIR

DIFFPAIR
    WIDTH 0.2
    MINCUT 2
    MINRES 200
    MAXRES 500
    MAXCAP 600
    PAIRGAP 0.8
    DTMF_INST/C DTMF_INST/D
END DIFFPAIR
```

The constraint file can include generic constraints based on any of the specialized constraint keywords, plus the following:

- Minimum number of via cuts
- Minimum width
- Spacing
- Routing layers
- Tapering

## Encounter User Guide

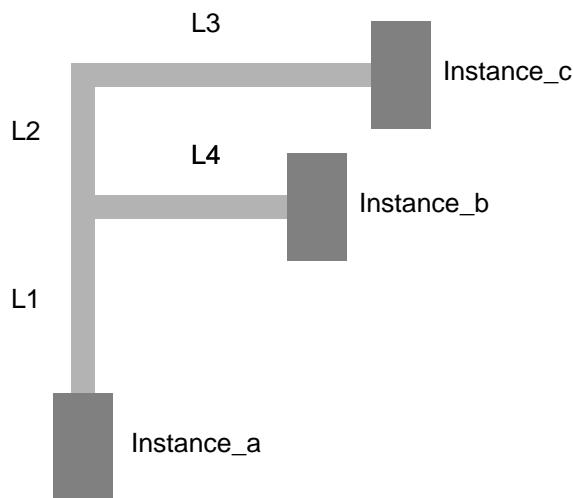
### Using the Encounter Mixed Signal Router

#### ■ Minimum and maximum resistance

The router estimates the effective resistance of a net and compares it to the minimum and maximum resistance in the constraint file. The resistance is measured in ohms.

The router determines the path with the highest resistance, and uses that path as the effective resistance.

The figure shows a multiple-pin net with four segments: L4, L1, L2, L3, and L4.



Before it routes the nets, the router estimates the effective resistance using the following formula:

Resistance = (*average resistivity of all metal layers*) x ((L1 + L2 + L3 + L4) / (*average width of all metal layers*))

After routing, the router calculates the effective resistance using the following formula:

Resistance =  
((Resistivity of L1)\*(L1/Width of L1)) +  
((Resistivity of L2)\*(L2/Width of L2)) +  
((Resistivity of L3)\*(L3/Width of L3)) +  
((Resistivity of L4)\*(L4/Width of L4))

If the pin on Instance\_b is an output pin, valid paths are L1 + L2 + L3 and L1 + L4.

If the pin on Instance\_b is a bidirectional pin, valid paths are L1 + L2 + L3, L1 + L4, and L4 + L2 + L3.

#### ■ Minimum and maximum capacitance

The Encounter software calculates capacitance using the following formula:

Total capacitance =  
area\_capacitance + fringe\_capacitance + coupling\_capacitance

## Specialized Constraints and Keyword Descriptions

The constraint file can contain the following specialized constraints:

- [NETS](#) on page 776
- [MATCH](#) on page 778
- [DIFFPAIR](#) on page 780
- [BUS](#) on page 782
- [SHIELDING](#) on page 784

**Note:** Unless otherwise noted, the default value for each keyword in the constraints is the value in the LEF file.

### NETS

Defines a set of rules for constraining the width, resistance, capacitance, spacing, and number of via cuts of a net or group of nets.

```
NETS
  WIDTH minWidth
  MAXRES maxResistance
  MAXCAP maxCapacitance
  MINCAP minCapacitance
  MINRES minResistance
  MINCUT minCuts
  TAPERING numberTaperSteps taperEndWidth
  ROUTELAYERS {bottom_layer_num:top_layer_num | list_of_layer_numbers ...}
  SPACING spacing
    {signal_net_name | list_of_signal_nets}
END NETS
```

MAXCAP *maxCapacitance*

Specifies the maximum capacitance allowed on the constrained nets, in picofarads.

MAXRES *maxResistance*

Specifies the maximum resistance allowed on the constrained nets, in ohms.

MINCAP *minCapacitance*

Specifies the minimum capacitance allowed on the constrained nets, in picofarads.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

**MINCUT** *minCuts*

Specifies the minimum number of via cuts allowed on the constrained net.

**MINRES** *minResistance*

Specifies the minimum resistance allowed on the constrained nets, in ohms.

**ROUTELAYERS** {*bottom\_layer\_num*:*top\_layer\_num* | *list\_of\_layer\_numbers*}

Specifies the layers on which the constrained nets can be routed.

**SPACING** *spacing*

Specifies the distance between the constrained nets and other nets, in microns.

**TAPERING** [[*numberTaperSteps*] [*taperEndWidth*]]

Specifies that the router should reduce the width of wide wires when they connect to a pin and, optionally

- The width of the wire when it connects
- The number of steps the router takes to reduce the wire width. The maximum number of steps supported is 3.

If you do not specify any parameters, the router tapers the wires to the width of the pin in one step. If you specify only one parameter, the router assumes it is the number of taper steps.

**Note:** The width of a wire on a taper step cannot be more than three times that of the width of the wire on a neighboring step. If you specify values that do not meet this requirement, the router adjusts the values to increase the number of steps. If it cannot meet the requirement by increasing the number of steps, the router issues a warning and does not taper the wire.

**WIDTH** *minWidth*

Specifies the minimum width of the constrained nets, in microns.

*signal\_net\_name* | *list\_of\_signal\_nets*

Specifies a constrained net or a list of constrained nets.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

## MATCH

Defines a set of rules for constraining nets by resistance. Contains a tolerance value, by which resistance of the nets may vary without causing a violation. Optionally, contains constraints on width, capacitance, number of via cuts, routing layers, and spacing.

```
MATCH
  WIDTH minWidth
  TOLERANCE value
  MAXRES maxResistance
  MAXCAP maxCapacitance
  MINRES minResistance
  MINCAP minCapacitance
  MINCUT minCuts
  TAPERING numberTaperSteps taperEndWidth
  SPACING spacing
  MATCHSTYLE {ACCORDION | TROMBONE}
  ROUTELAYERS {bottom_layer_num:top_layer_num | list_of_layer_numbers ...}
    list_of_signal_nets
END MATCH
```

**MATCHSTYLE (ACCORDION | TROMBONE)**

Specifies whether the software generates coils orthogonal or parallel to the router.

*Default:* ACCORDION

Specify one of the following values:

**ACCORDION**

Generates coils along the direction that is orthogonal to the routing.

**TROMBONE**

Generates coils along the direction that is parallel to the routing.

**MAXCAP *maxCapacitance***

Specifies the maximum capacitance allowed on the constrained nets, in picofarads.

**MAXRES *maxResistance***

Specifies the maximum resistance allowed on the constrained nets, in ohms.

**MINCUT *minCuts***

Specifies the minimum number of via cuts allowed on the constrained net.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

MINRES *minResistance*

Specifies the minimum resistance allowed on the constrained nets, in ohms.

ROUTELAYERS {*bottom\_layer\_num*:*top\_layer\_num* |  
*list\_of\_layer\_numbers*}

Specifies the layers on which the constrained nets can be routed.

SPACING *spacing*

Specifies the distance between the constrained nets and other nets, in microns.

TAPERING [[*numberTaperSteps*] [*taperEndWidth*]]

Specifies that the router should reduce the width of wide wires when they connect to a pin and, optionally

- The width of the wire when it connects
- The number of steps the router takes to reduce the wire width. The maximum number of steps supported is 3.

If you do not specify any parameters, the router tapers the wires to the width of the pin in one step. If you specify only one parameter, the router assumes it is the number of taper steps.

**Note:** The width of a wire on a taper step cannot be more than three times that of the width of the wire on a neighboring step. If you specify values that do not meet this requirement, the router adjusts the values to increase the number of steps. If it cannot meet the requirement by increasing the number of steps, the router issues a warning and does not taper the wire.

TOLERANCE *value*

Specifies a percentage value for the upper limit of the allowable difference in resistance of the constrained nets.

*Default:* 20

For more information, see “[Matched Nets](#)” on page 764.

WIDTH *minWidth*

Specifies the minimum width of the constrained nets, in microns.

*list\_of\_signal\_nets*

Specifies the list of constrained nets.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

## DIFFPAIR

Defines a set of rules for constraining symmetrical nets by width and spacing. Contains a threshold value by which the spacing of the nets may vary without causing a violation. Optionally, contains constraints on maximum width, resistance, capacitance, routing layers, minimum number of cuts, and distance from other nets.

```
DIFFPAIR
  WIDTH minWidth
  PAIRGAP spacing
  THRESHOLD value
  MAXRES maxResistance
  MAXCAP maxCap
  MINRES minResistance
  MINCUT minCuts
  TAPERING numberTaperSteps taperEndWidth
  ROUTELAYERS {bottom_layer_num:top_layer_num | list_of_layer_numbers ...}
  SPACING spacing
    {signal_net_1 | signal_net_2}
END DIFFPAIR
```

**MAXCAP** *maxCapacitance*

Specifies the maximum capacitance allowed on the constrained nets, in picofarads.

**MAXRES** *maxResistance*

Specifies the maximum resistance allowed on the constrained nets, in ohms.

**MINCAP** *minCapacitance*

Specifies the minimum capacitance allowed on the constrained nets, in picofarads.

**MINCUT** *minCuts*

Specifies the minimum number of via cuts allowed on the constrained net.

**MINRES** *minResistance*

Specifies the minimum resistance allowed on the constrained nets, in ohms.

**PAIRGAP** *spacing*

Specifies the distance between the two differential pair nets, in microns.

**ROUTELAYERS** {*bottom\_layer\_num*:*top\_layer\_num* | *list\_of\_layer\_numbers*}

Specifies the layers on which the constrained nets can be routed.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

**SPACING *spacing***      Specifies the distance between the constrained nets and other nets, in microns.

**TAPERING [ [ *numberTaperSteps* ] [ *taperEndWidth* ] ]**

Specifies that the router should reduce the width of wide wires when they connect to a pin and, optionally

- The width of the wire when it connects
- The number of steps the router takes to reduce the wire width. The maximum number of steps supported is 3.

If you do not specify any parameters, the router tapers the wires to the width of the pin in one step. If you specify only one parameter, the router assumes it is the number of taper steps.

**Note:** The width of a wire on a taper step cannot be more than three times that of the width of the wire on a neighboring step. If you specify values that do not meet this requirement, the router adjusts the values to increase the number of steps. If it cannot meet the requirement by increasing the number of steps, the router issues a warning and does not taper the wire.

**THRESHOLD *value***

Specifies a relative tolerance value for the difference between the longest and the shortest net. Therefore, the router attempts to match the length of nets in the differential pair nets, unless the difference varies by more than the threshold.

The threshold is calculated by using the following formula:

$$Threshold = \frac{L2 - L1}{L1}$$

where L2 is the length of the longer net and L1 is the length of shorter net.

For example, if the constrained nets are 24 microns and 20 microns, the threshold is calculated as  $(24-20)/20 = 0.2$  or 20%

**WIDTH *minWidth***

Specifies the width of constrained nets, measured in microns.

***signal\_net\_1* | *signal\_net\_2***

Specifies the constrained nets.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

## BUS

Defines a set of rules for routing bundles of nets that follow the same pattern from the source to the target.

```
BUS
  WIDTH minWidth
  BUSGAP spacing
  SAMEAYER {0 | 1}
  MAXRES maxResistance
  MAXCAP maxCapacitance
  MINCAP minCapacitance
  MINRES minResistance
  MINCUT minCuts
  TAPERING numberTaperSteps taperEndWidth
  SPACING spacing
    list_of_signal_nets
END BUS
```

**BUSGAP *width***              Specifies the spacing between the wires in the bus, in microns.

**MAXCAP *maxCapacitance***

Specifies the maximum capacitance allowed on the constrained nets, in picofarads.

**MAXRES *maxResistance***

Specifies the maximum resistance allowed on the constrained nets, in ohms.

**MINCAP *minCapacitance***

Specifies the minimum capacitance allowed on the constrained nets, in picofarads.

**MINCUT *minCuts***

Specifies the minimum number of via cuts allowed on the constrained nets.

**MINRES *minResistance***

Specifies the minimum resistance allowed on the constrained nets, in ohms.

**SAMEAYER {0 | 1}**

Specifies whether all the nets in the bus must be routed on the same layer (0) or can be routed on different layers (1).

*Default:* 0

**SPACING *spacing***

Specifies the distance between the constrained nets and other nets, in microns.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

**TAPERING** [ [*numberTaperSteps*] [*taperEndWidth*] ]

Specifies that the router should reduce the width of wide wires when they connect to a pin and, optionally

- The width of the wire when it connects
- The number of steps the router takes to reduce the wire width. The maximum number of steps supported is 3.

If you do not specify any parameters, the router tapers the wires to the width of the pin in one step. If you specify only one parameter, the router assumes it is the number of taper steps.

**Note:** The width of a wire on a taper step cannot be more than three times that of the width of the wire on a neighboring step. If you specify values that do not meet this requirement, the router adjusts the values to increase the number of steps. If it cannot meet the requirement by increasing the number of steps, the router issues a warning and does not taper the wire.

**WIDTH** *width*

Specifies the minimum width of the wires in the bus, in microns.

*list\_of\_signal\_nets*

Specifies the nets in the bus. List the nets in descending or ascending order. You can use the following syntax:

- To specify bus bits *a[0]* through *a[15]*, type *a[0:15]*.
- To specify bus bits *a[0]* through *a[15]* in multiples of 3, type *a[0], a[3], a[6], a[9], a[11], a[14]*.

## SHIELDING

Defines a set of rules for protecting noise-sensitive signal nets by routing special nets nearby.

### Simple Shielding

Shielding on right and left sides uses the following syntax:

```
SHIELDING
  CONNECTSUPPLY {FLOAT | ANYPOINT}
  MAXCAP maxCapacitance
  MAXRES maxResistance
  MINCAP minCapacitance
  MINCUT minCuts
  MINRES minResistance
  NOSHARESHIELD {0 | 1}
  ROUTELAYERS {bottom_layer_num:top_layer_num | list_of_layer_numbers}
  SPACING spacing
  SHIELDGAP spacing
  SHIELDNET special_net_name
  SHIELDWIDTH width
  TAPERING numberTaperSteps taperEndWidth
  WIDTH minWidth
  {signal_net_name | list_of_signal_nets}
END SHIELDING
```

CONNECTSUPPLY {FLOAT | ANYPOINT}

Describes where the router connects the shield nets to the power supply.

Specify one of the following methods:

FLOAT	Does not connect the shield—leaves it floating.
-------	---

ANYPOINT	Connects the shield to the supply at any point along the wire.
----------	--

MAXCAP *maxCapacitance*

Specifies the maximum capacitance allowed on the constrained nets, in picofarads.

MAXRES *maxResistance*

Specifies the maximum resistance allowed on the constrained nets, in ohms.

MINCAP *minCapacitance*

Specifies the minimum capacitance allowed on the constrained nets, in picofarads.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

**MINCUT** *minCuts*

Specifies the minimum number of via cuts allowed on the constrained net.

**MINRES** *minRes*

Specifies the minimum resistance allowed on the constrained nets, in ohms.

**NOSHARESHIELD** { 0 | 1 }

Specifies whether shields are shared. By default, the router shares shields when possible.

Specify 0 if you do not want the router to share shield nets.

**Note:** Cadence recommends you set the NOSHARESHIELD attribute by using the Mixed Signal Router form. For information, see [Mixed Signal Router](#) in the *Encounter Menu Command Reference*.

**ROUTELAYERS** {*bottom\_layer\_num*:*top\_layer\_num* | *list\_of\_layer\_numbers*}

Specifies the layers on which the constrained nets can be routed.

**SPACING** *spacing*

Specifies the distance between the constrained nets and other nets, in microns.

**SHIELDGAP** *spacing*

Specifies the distance between the shielded net (the signal net) and the shield nets (special nets), in microns.

**SHIELDNET** *special\_net\_name*

Specifies a special net to use as a shield net.

**SHIELDWIDTH** *width*

Specifies the width of the shield nets, in microns.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

**TAPERING** [ [*numberTaperSteps*] [*taperEndWidth*] ]

Specifies that the router should reduce the width of wide wires when they connect to a pin and, optionally

- The width of the wire when it connects
- The number of steps the router takes to reduce the wire width. The maximum number of steps supported is 3.

If you do not specify any parameters, the router tapers the wires to the width of the pin in one step. If you specify only one parameter, the router assumes it is the number of taper steps.

**Note:** The width of a wire on a taper step cannot be more than three times that of the width of the wire on a neighboring step. If you specify values that do not meet this requirement, the router adjusts the values to increase the number of steps. If it cannot meet the requirement by increasing the number of steps, the router issues a warning and does not taper the wire.

**WIDTH** *minWidth*

Specifies the minimum width of the constrained nets, in microns.

*signal\_net\_name* | *list\_of\_signal\_nets*

Specifies a constrained net or a list of constrained nets.

## Coaxial Shielding

Shielding on the right, left, top and bottom sides uses the following syntax:

```
SHIELDING
COAXSHIELDVIAGAP viaGap
CONNECTCOAXSHIELDS {0|1}
CONNECTSUPPLY {FLOAT | ANYPOINT}
MAXCAP maxCapacitance
MAXRES maxResistance
MINCAP minCapacitance
MINCUT minCuts
MINRES minResistance
OVERHANG width
ROUTELAYERS {bottom_layer_num:top_layer_num | list_of_layer_numbers}
SHIELDGAP spacing
SHIELDLAYERS integer
SHIELDNET special_net_name
SHIELDWIDTH width
SPACING spacing
TAPERING numberTaperSteps taperEndWidth
WIDTH minWidth
```

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

```
{signal_net_name | list_of_signal_nets}  
END SHIELDING
```

COAXSHIELDVIAGAP *viaGap*

Specifies the gap between the via cuts that connect the top or bottom shield to the side shield.

CONNECTCOAXSHIELDS {0 | 1}

Specifies whether the top or bottom shield should connect to the side shields.

*Default:* 0

CONNECTSUPPLY {FLOAT | ANYPOINT}

Describes where the router connects the shield nets to the power supply.

Specify one of the following methods:

FLOAT	Does not connect the shield—leaves it floating.
-------	---

ANYPOINT	Connects the shield to the supply at any point along the wire.
----------	--

MAXCAP *maxCapacitance*

Specifies the maximum capacitance allowed on the constrained nets, in picofarads.

MAXRES *maxResistance*

Specifies the maximum resistance allowed on the constrained nets, in ohms.

MINCAP *minCapacitance*

Specifies the minimum capacitance allowed on the constrained nets, in picofarads.

MINCUT *minCuts*

Specifies the minimum number of via cuts allowed on the constrained net.

MINRES *minRes*

Specifies the minimum resistance allowed on the constrained nets, in ohms.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

OVERHANG <i>width</i>	Specifies one-half of the difference between the width of the signal net and the width of the bottom and top shield nets.														
ROUTELAYERS { <i>bottom_layer</i> : <i>top_layer</i>   <i>layer_number</i> <i>layer_number</i> ... }	Specifies the layers on which the constrained nets can be routed.														
SHIELDGAP <i>spacing</i>	Specifies the distance between the shielded net (the signal net) and the left and right shield nets (special nets), in microns.														
SHIELDLAYERS <i>integer</i>	<p>Indicates which sides of the noise-sensitive net are shielded by special nets. For coaxial shielding, specify 7.</p> <p>Specify one of the following values:</p> <table><tr><td>1</td><td>Shield on the right and left sides only</td></tr><tr><td>2</td><td>Shield on the bottom only</td></tr><tr><td>3</td><td>Shield on the bottom, right, and left sides</td></tr><tr><td>4</td><td>Shield on the top only</td></tr><tr><td>5</td><td>Shield on the top, right, and left sides only</td></tr><tr><td>6</td><td>Shield on the top and bottom only</td></tr><tr><td>7</td><td>Shield on the top, bottom, right, and left sides</td></tr></table>	1	Shield on the right and left sides only	2	Shield on the bottom only	3	Shield on the bottom, right, and left sides	4	Shield on the top only	5	Shield on the top, right, and left sides only	6	Shield on the top and bottom only	7	Shield on the top, bottom, right, and left sides
1	Shield on the right and left sides only														
2	Shield on the bottom only														
3	Shield on the bottom, right, and left sides														
4	Shield on the top only														
5	Shield on the top, right, and left sides only														
6	Shield on the top and bottom only														
7	Shield on the top, bottom, right, and left sides														
SHIELDNET <i>special_net_name</i>	Specifies a special net to use as a shield net.														
SHIELDWIDTH <i>width</i>	Specifies the width of the right and left shield nets, in microns.														
SPACING <i>spacing</i>	Specifies the distance between the constrained nets and other nets, in microns.														

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

TAPERING [ [ *numberTaperSteps* ] [ *taperEndWidth* ] ]

Specifies that the router should reduce the width of wide wires when they connect to a pin and, optionally

- The width of the wire when it connects
- The number of steps the router takes to reduce the wire width. The maximum number of steps supported is 3.

If you do not specify any parameters, the router tapers the wires to the width of the pin in one step. If you specify only one parameter, the router assumes it is the number of taper steps.

**Note:** The width of a wire on a taper step cannot be more than three times that of the width of the wire on a neighboring step. If you specify values that do not meet this requirement, the router adjusts the values to increase the number of steps. If it cannot meet the requirement by increasing the number of steps, the router issues a warning and does not taper the wire.

WIDTH *minWidth* Specifies the minimum width of the constrained (signal) nets, in microns.

*signal\_net\_name* | *list\_of\_signal\_nets*

Specifies a constrained nets to shield.

## Differential Pair Shielding

Differential pair shielding uses the following syntax:

```
DIFFPAIR
COAXSHIELDVIAGAP viaGap
CONNECTCOAXSHIELDS {0|1}
CONNECTSUPPLY {FLOAT | ANYPOINT}
MAXCAP maxCapacitance
MAXRES maxResistance
MINCAP minCapacitance
MINCUT integer
MINRES minResistance
OVERHANG width
PAIRGAP spacing
ROUTELAYERS {bottom_layer_num:top_layer_num | list_of_layer_numbers}
SHIELDGAP spacing
SHIELDLAYERS integer
SHIELDNET special_net_name
SHIELDWIDTH width
SPACING spacing
TAPERING numberTaperSteps taperEndWidth
THRESHOLD value
WIDTH width
```

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

```
{signal_net_1 | signal_net_2}  
END DIFFPAIR
```



In this constraint, the SHIELDGAP must be defined on the line immediately preceding the SHIELDWIDTH definition.

COAXSHIELDVIAGAP *viaGap*

Specifies the gap between the via cuts that connect the top or bottom shield to the side shield.

CONNECTCOAXSHIELDS {0 | 1}

Specifies whether the top or bottom shield should connect to the side shields.

CONNECTSUPPLY {FLOAT | ANYPOINT}

Describes where the router connects the shield nets to the power supply.

Specify one of the following methods:

FLOAT	Does not connect the shield—leaves it floating.
-------	---

ANYPOINT	Connects the shield to the supply at any point along the wire.
----------	--

MAXCAP *maxCapacitance*

Specifies the maximum capacitance allowed on the constrained nets, in picofarads.

MAXRES *maxResistance*

Specifies the maximum resistance allowed on the constrained nets, in ohms.

MINCAP *minCapacitance*

Specifies the minimum capacitance allowed on the constrained nets, in picofarads.

MINCUT *minCuts*

Specifies the minimum number of via cuts allowed on the constrained net.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

**MINRES** *minRes*

Specifies the minimum resistance allowed on the constrained nets, in ohms.

**OVERHANG** *width*

Specifies one-half of the difference between the width of the signal net and the width of the bottom and top shield nets. For an illustration of the overhang, see the figure for [“Coaxial Shield Nets”](#) on page 771.

**PAIRGAP** *spacing*

Specifies the distance between the two differential pair nets, in microns.

**ROUTELAYERS** {*bottom\_layer*:*top\_layer* | *layer\_number* *layer\_number* ...}

Specifies the layers on which the constrained nets can be routed.

**SHIELDGAP** *spacing*

Specifies the distance between the shielded net (the signal net) and the left and right shield nets (special nets), in microns.

**SHIELDLAYERS** *integer*

Indicates which sides of the noise-sensitive net are shielded by special nets. For coaxial shielding, specify 7.

Specify one of the following values:

- |       |   |
|-------|---|
| 1     | Shield on the right and left sides only   |
| 2     | Shield on the bottom only   |
| 3     | Shield on the bottom, right, and left sides   |
| 4     | Shield on the top only  |
| 5     | Shield on the top, right, and left sides only   |
| 6     | Shield on the top and bottom only   |
| 7     | Shield on the top, bottom, right, and left sides  |
| 8/9   | Shield on the right and left side with<br>interleave shielding between differential pair<br>nets      |
| 10/11 | Shield on bottom, right, and left side with<br>interleave shielding between differential pair<br>nets |

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

12/13

Shield on top, right, and left side with  
interleave shielding between differential pair  
nets

14/15

Shield on top, bottom, right, and left side with  
interleave shielding between differential pair  
nets

**SHIELDNET** *special\_net\_name*

Specifies a special net to use as a shield net.

**SHIELDWIDTH** *width*

Specifies the width of the right and left shield nets, in microns.

**SPACING** *spacing*

Specifies the distance between the constrained nets and other  
nets, in microns.

**TAPERING** [ [ *numberTaperSteps* ] [ *taperEndWidth* ] ]

Specifies that the router should reduce the width of wide wires  
when they connect to a pin and, optionally

- The width of the wire when it connects

The number of steps the router takes to reduce the wire width.  
The maximum number of steps supported is 3.

If you do not specify any parameters, the router tapers the wires  
to the width of the pin in one step. If you specify only one  
parameter, the router assumes it is the number of taper steps.

**Note:** The width of a wire on a taper step cannot be more than  
three times that of the width of the wire on a neighboring step. If  
you specify values that do not meet this requirement, the router  
adjusts the values to increase the number of steps. If it cannot  
meet the requirement by increasing the number of steps, the  
router issues a warning and does not taper the wire.

**THRESHOLD** *value*

Specifies a value for the spacing between constrained nets  
(PAIRGAP), so the router attempts to meet the specified  
spacing but does not create a violation unless the spacing  
varies by more than the threshold.

*Default:* 20

**WIDTH** *minWidth*

Specifies the minimum width of the constrained (signal) nets, in  
microns.

*signal\_net\_1* | *signal\_net\_2*

Specifies a constrained net to shield.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

#### Bus Shielding

Bus shielding uses the following syntax (the shielding statements are included in the **BUS** constraint and are shown in bold type below):

```
BUS
  BUSGAP spacing
  COAXSHIELDVIAGAP viaGap
  CONNECTCOAXSHIELDS {0|1}
  CONNECTSUPPLY {FLOAT | ANYPOINT}
  MAXCAP maxCapacitance
  MAXRES maxResistance
  MINCAP minCapacitance
  MINCUT integer
  MINRES minResistance
  OVERHANG width
  SAMELAYER {0|1}
  SHIELDGAP spacing
  SHIELDLAYERS integer
  SHIELDNET special_net_name
  SHIELDWIDTH width
  SPACING spacing
  TAPERING numberTaperSteps taperEndWidth
  WIDTH minWidth
    list_of_signal_nets
END BUS
```

**BUSGAP** *spacing*      Specifies the spacing between the wires in the bus, in microns.

**COAXSHIELDVIAGAP** *viaGap*  
Specifies the gap between the via cuts that connect the top or bottom shield to the side shield.

**CONNECTCOAXSHIELDS** {0|1}  
Specifies whether the top or bottom shield should connect to the side shields.

**CONNECTSUPPLY** {FLOAT | ANYPOINT}  
Describes where the router connects the shield nets to the power supply.

Specify one of the following methods:

FLOAT      Does not connect the shield—leaves it floating.

ANYPOINT      Connects the shield to the supply at any point along the wire.

**MAXCAP** *maxCapacitance*

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

Specifies the maximum capacitance allowed on the constrained nets, in picofarads.

**MAXRES** *maxResistance*

Specifies the maximum resistance allowed on the constrained nets, in ohms.

**MINCAP** *minCapacitance*

Specifies the minimum capacitance allowed on the constrained nets, in picofarads.

**MINCUT** *minCuts*

Specifies the minimum number of via cuts allowed on the constrained net.

**MINRES** *minRes*

Specifies the minimum resistance allowed on the constrained nets, in ohms.

**OVERHANG** *width*

Specifies one-half of the difference between the width of the signal net and the width of the bottom and top shield nets. For an illustration of the overhang, see the figure for [“Coaxial Shield Nets”](#) on page 771.

**SAMELAYER** {0 | 1}

Specifies whether all the nets in the bus must be routed on the same layer (0) or can be routed on different layers (1).

*Default:* 0

**SHIELDGAP** *spacing*

Specifies the distance between the shielded net (the signal net) and the left and right shield nets (special nets), in microns.

**SHIELDLAYERS** *integer*

Indicates which sides of the noise-sensitive net are shielded by special nets. For coaxial shielding, specify 7.

Specify one of the following values:

- |   |   |
|---|---|
| 1 | Shield on the right and left sides only       |
| 2 | Shield on the bottom only                     |
| 3 | Shield on the bottom, right, and left sides   |
| 4 | Shield on the top only                        |
| 5 | Shield on the top, right, and left sides only |
| 6 | Shield on the top and bottom only             |

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

7	Shield on the top, bottom, right, and left sides
8/9	Shield on the right and left side with interleave shielding between bus nets
10/11	Shield on bottom, right, and left side with interleave shielding between bus nets
12/13	Shield on top, right, and left side with interleave shielding between bus nets
14/15	Shield on top, bottom, right, and left side with interleave shielding between bus nets

**SHIELDNET *special\_net\_name***

Specifies the nets in the bus to shield.

**SHIELDWIDTH *width*** Specifies the width of the right and left shield nets, in microns.

**SPACING *spacing*** Specifies the distance between the constrained nets and other nets, in microns.

**TAPERING [ [*numberTaperSteps*] [*taperEndWidth*] ]**

Specifies that the router should reduce the width of wide wires when they connect to a pin and, optionally

- The width of the wire when it connects

The number of steps the router takes to reduce the wire width. The maximum number of steps supported is 3.

If you do not specify any parameters, the router tapers the wires to the width of the pin in one step. If you specify only one parameter, the router assumes it is the number of taper steps.

**Note:** The width of a wire on a taper step cannot be more than three times that of the width of the wire on a neighboring step. If you specify values that do not meet this requirement, the router adjusts the values to increase the number of steps. If it cannot meet the requirement by increasing the number of steps, the router issues a warning and does not taper the wire.

**WIDTH *minWidth***

Specifies the minimum width of the constrained (signal) nets, in microns.

## **Encounter User Guide**

### Using the Encounter Mixed Signal Router

---

#### *list\_of\_signal\_nets*

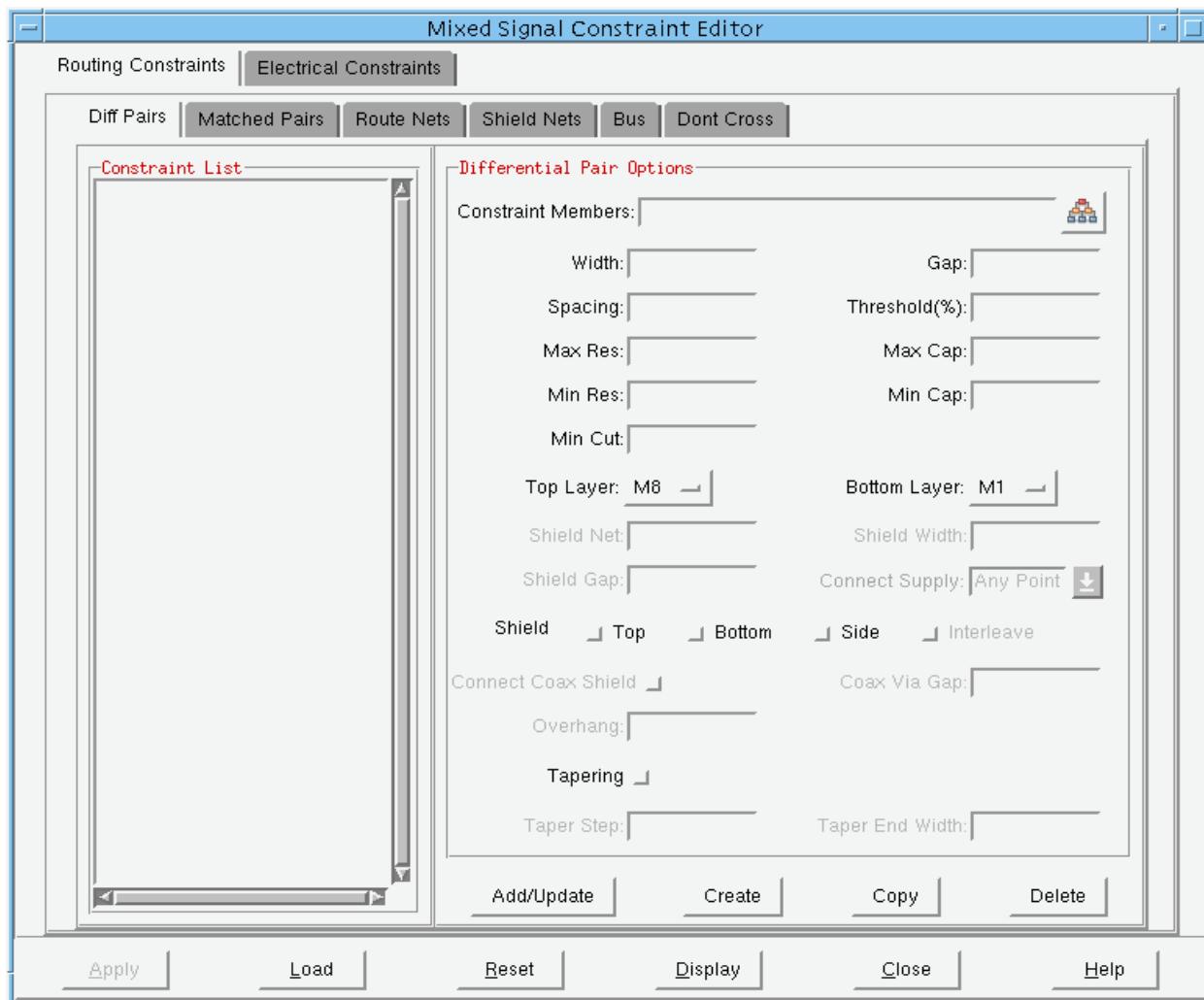
Specifies the nets in the bus to shield. List the nets in descending or ascending order. You can use the following syntax:

- To specify bus bits  $a[0]$  through  $a[15]$ , type  $a[0:15]$ .
- To specify bus bits  $a[0]$  through  $a[15]$  in multiples of 3, type  $a[0], a[3], a[6], a[9], a[11], a[14]$ .

## Creating a Constraint File

### Using the Mixed Signal Constraint Editor

- On the main Encounter menu, select *Tools – Mixed Signal – Constraints*.  
The Mixed Signal Constraint Editor form is displayed.



### Selecting Nets to Add to a Constraint

1. Select the tab for the type of constraint you want to create: *Diff Pairs*, *Matched Pairs*, *Route Nets*, or *Bus*. For information on the types of constraints, see “[Specialized Constraints and Keyword Descriptions](#)” on page 776.

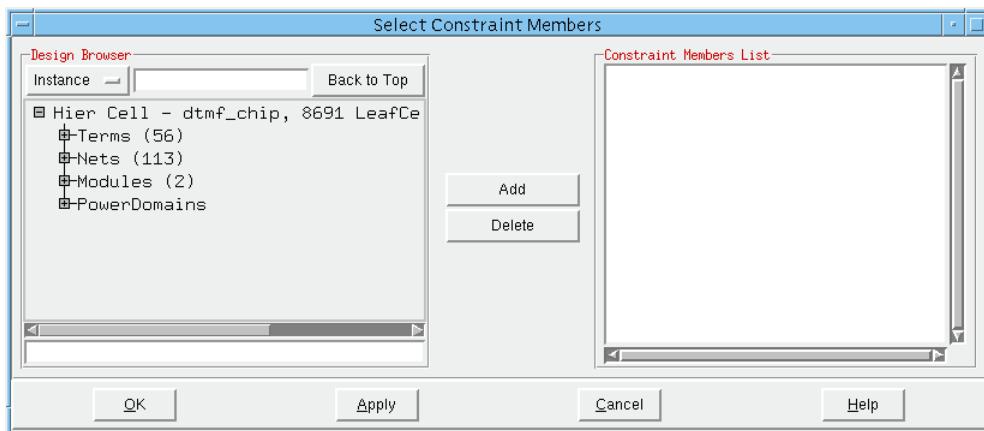
## Encounter User Guide

### Using the Encounter Mixed Signal Router

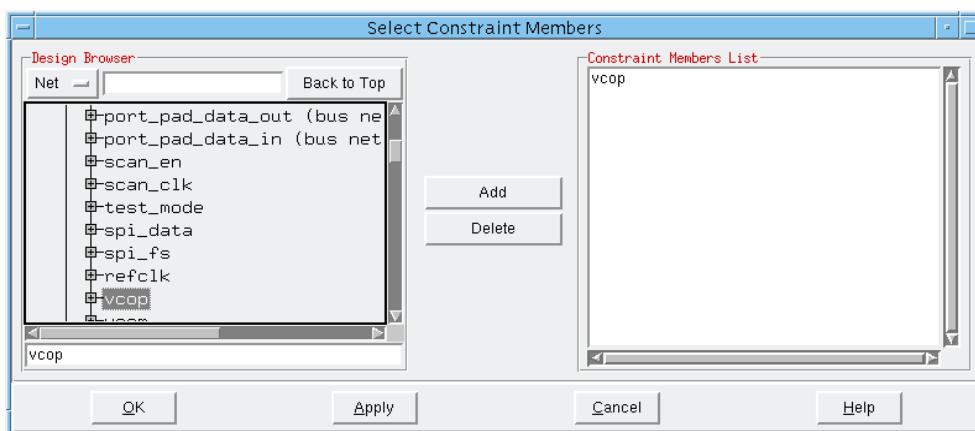
2. Click the hierarchy icon.



The Select Constraint Members form is displayed.



3. Select *Net* from the drop-down list.
4. Traverse the net hierarchy until you see the name of the net (or nets) you want to constrain.
5. Select a net and click *Add*. The net is added to the *Constraint Members List*.

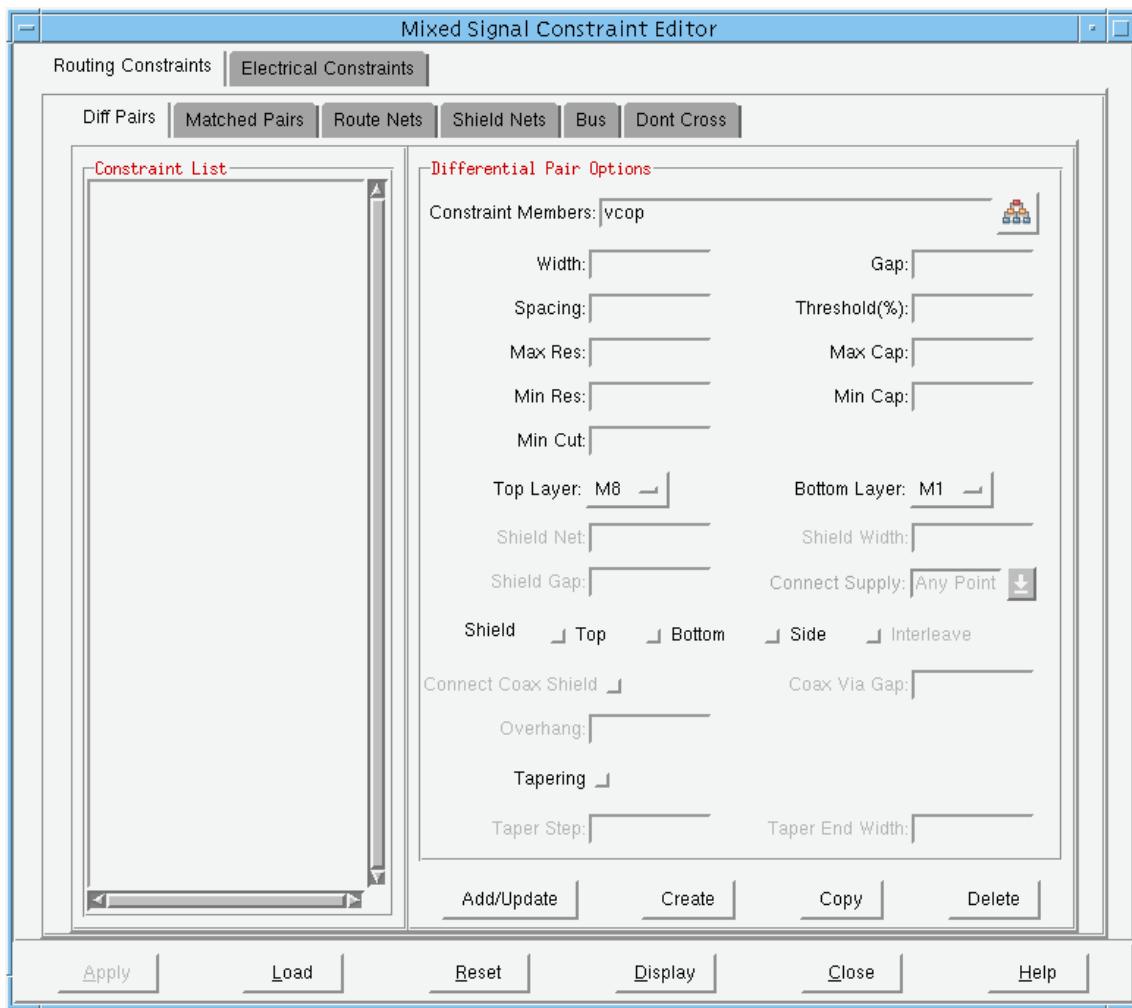


6. Repeat step 5 until you have added all the nets for the constraint to the *Constraint Members List*.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

7. Click **Apply**. The nets are added to the *Constraint Members* text entry box on the Mixed Signal Constraint Editor form.



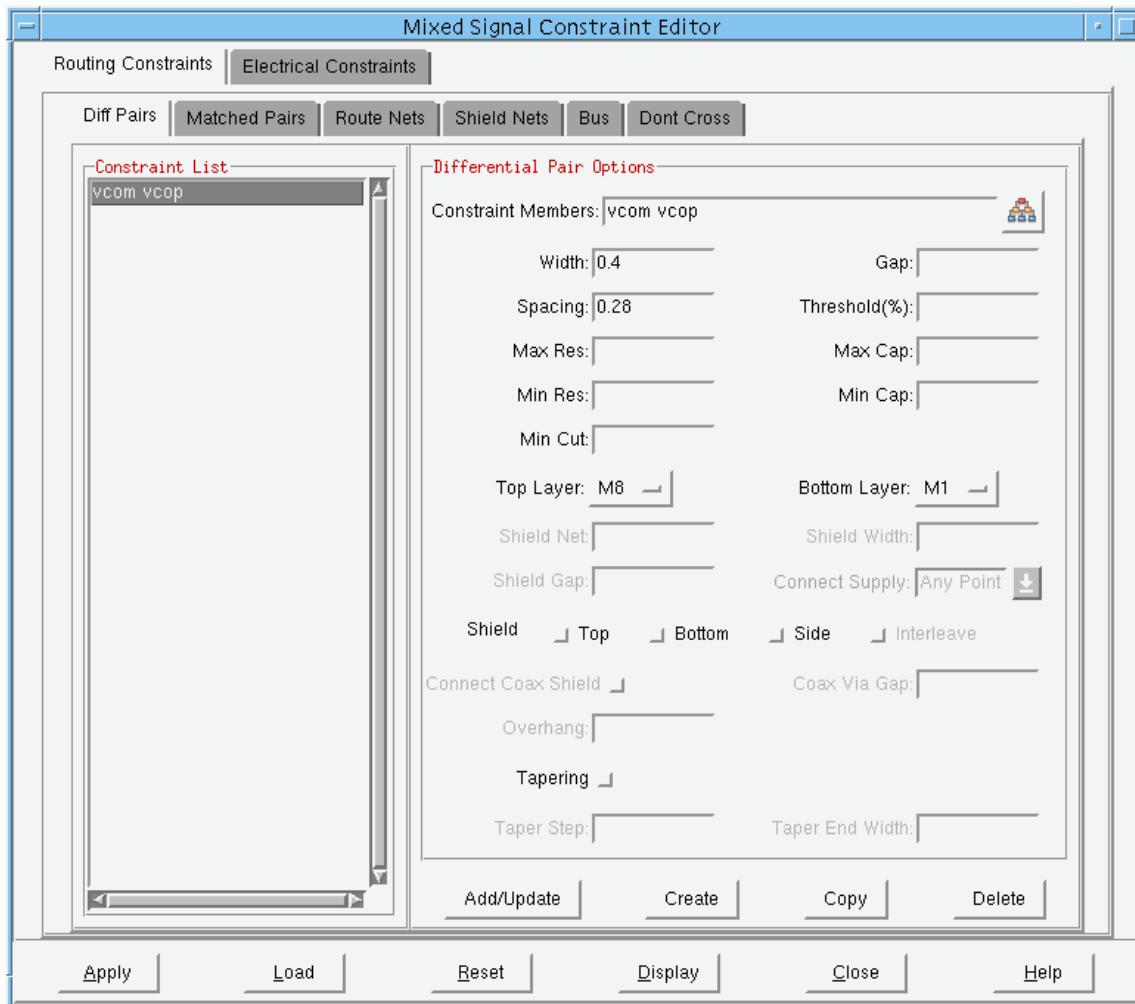
#### Entering Values for Constraint Parameters

1. Enter values for the appropriate parameters on the Mixed Signal Constraint Editor form. For example, to constrain the net width to 0.4 microns and the spacing to 0.28 microns, type .4 in the *Width* text entry box and .28 in the *Spacing* text entry box.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

2. Click *Add/Update*. The constraint is added to the *Constraint List* on the Mixed Signal Editor form.



3. Repeat the steps in “[Selecting Nets to Add to a Constraint](#)” on page 797 and “[Entering Values for Constraint Parameters](#)” on page 799 for each constraint.
4. After creating all the constraints and adding them to the Constraint List, click the *Display* button on the Mixed Signal Constraint Editor form to see the Display All form. The form lists the constraints, including all the parameter values, on one page. You cannot make any changes to the constraints from the *Display All* form, but you can see and compare values to make sure they are correct.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

The following figure shows the left-hand portion of the Display All form.

Display All									
Constraint Type	Constraint Member	Aggressor Net	Victim Nets	Width	Gap	Spacing	Overhang	Threshold(%)	Max Resistance
diffPair	vcom vcop	--	--	0.4	0.0	0.28	--	0.0	0.0

### Saving the Constraint to the Constraint File

At this point the constraints are not yet saved in a constraint file. To save the file, complete the following step:

- On the Mixed Signal Constraint Editor form, click the *Apply* button.

The software saves the constraint file with the name  
*topLevelCellName.msrouternumber.const*.

Every time you click *Apply*, the software increments *number*.

### Using a Text Editor

Start a text editor and create a constraint file using the syntax described in [“Specialized Constraints and Keyword Descriptions”](#) on page 776.

## Loading a Constraint File

### Using the Mixed Signal Constraint Editor Form

1. On the main Encounter menu, select *Tools – Mixed Signal – Constraints*.  
The Mixed Signal Constraint Editor form is displayed.
2. Click *Load*.  
The Load MS Constraint File form is displayed. Constraint files have a `.const` extension.
3. On the Load MS Constraint form, select the appropriate constraint file.
4. Click *Open*.  
The software populates the appropriate fields on the Mixed Signal Constraint Editor form with information from the constraint file you selected.

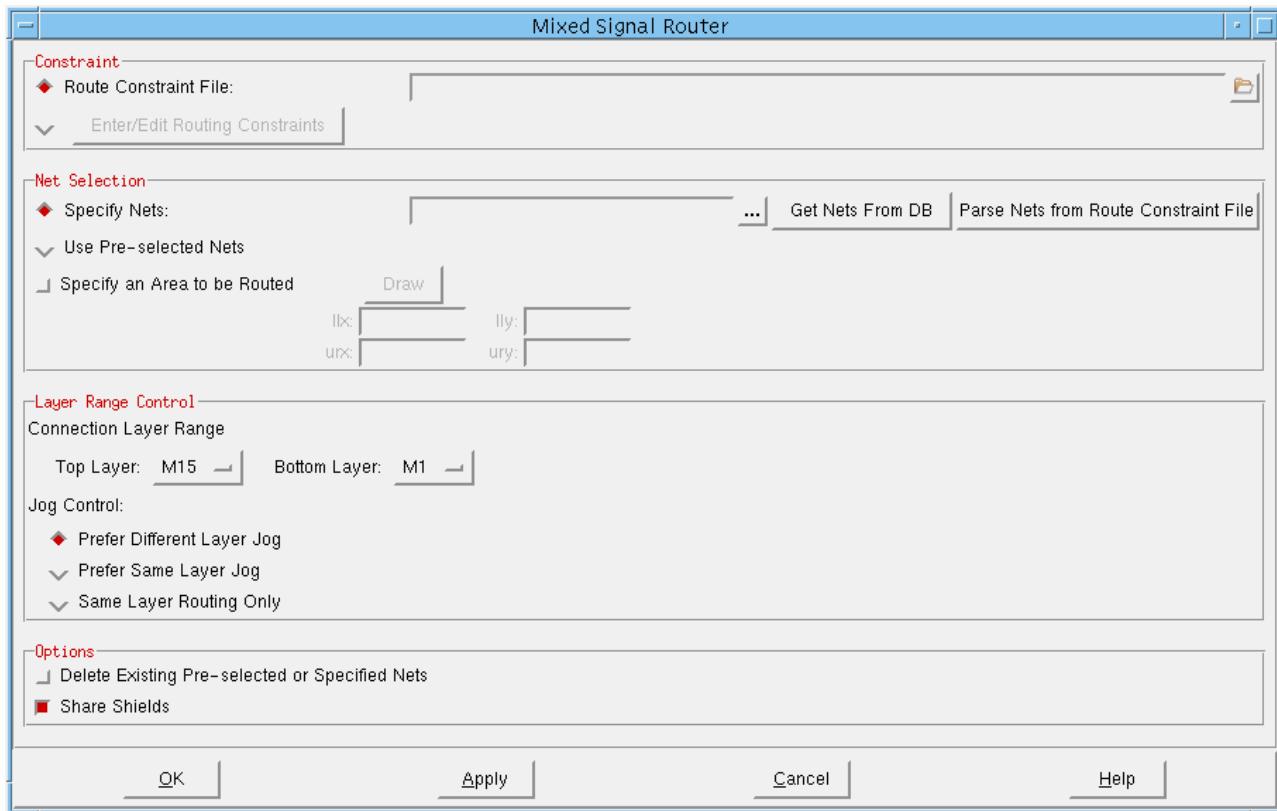
### Using the Mixed Signal Router Form

1. On the main Encounter menu, select *Route – Mixed Signal – Route*.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

The Mixed Signal Router form is displayed.



2. Select the *Route Constraint File* option.

3. Click the folder icon.

The Mixed Signal Router Constraint File form is displayed. Constraint files have a .const extension.

4. On the Mixed Signal Router Constraint File form, select the appropriate constraint file.

5. Click *Open*.

The software populates the *Route Constraint File* text entry field on the Mixed Signal Router form with the constraint file you selected.

Complete one of the following steps:

- Click *Enter/Edit Routing Constraints*.

The Mixed Signal Constraint Editor form is displayed, and the appropriate fields are populated with the information from the constraint file. You can now edit the constraints.

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

- Click *Parse Nets from Route Constraint File*.

The constraints are saved to the database when they are parsed. The nets are listed in the *Specify Nets* text entry box. You can now select and edit the list of nets to route.

## Using the **routeMixedSignal** Command

Type the following command:

```
routeMixedSignal -constraintFile fileName
```

For more information, see [routeMixedSignal](#) in the *Encounter Text Command Reference*.

## Editing a Constraint File

### Using the Mixed Signal Constraint Editor

1. Load the constraint file using one of the methods described in [“Loading a Constraint File”](#) on page 802.
2. Select *Enter/Edit Routing Constraints* on the Mixed Signal Router form.  
The Mixed Signal Constraint Editor form is displayed, with the constraints listed in the *Constraint List* and constrained nets displayed in the *Constraint Member* text entry box.
3. Select the appropriate tab for the type of constraint to edit: *Diff Pairs*, *Matched Pairs*, *Route Nets*, *Shield Nets*, or *Bus*.

### Editing an Existing Constraint

1. Select a constraint from the *Constraint List*.

The names of the nets in the constraint are displayed in the *Constraint Member List*, and the constraint parameter values are displayed for each option.

You can change any of the parameter values of the selected constraint, you can copy the constraint and use it as the basis for a new constraint, or you can delete it.

- To change the value for a constraint option, highlight the option value, delete it, type in a new value, and click *Add/Update*.
- To copy a constraint and use it as the basis for a new constraint, highlight a constraint on the *Constraint List*, click *Copy*, delete the net names in the *Constraint Members* text entry box, and select other nets to constrain.
- To delete a constraint, highlight it on the *Constraint List* and click *Delete*.

2. When you finish editing the constraints, click *Apply*.

### Creating a Constraint to Add to the Constraint File

1. Click the *Create* button to clear the fields on the Mixed Signal Constraint Editor form.
2. Follow the steps in [“Selecting Nets to Add to a Constraint”](#) on page 797, [“Entering Values for Constraint Parameters”](#) on page 799, and [“Saving the Constraint to the Constraint File”](#) on page 801.

## Using a Text Editor

Open the constraint file in a text editor and edit it, following the syntax in “[Constraint File Format](#)” on page 773. Save the file with a .const extension.

## Sample Constraint File

```
DIFFPAIR
  WIDTH 0.200000
  PAIRGAP 0.100000
  THRESHOLD 20
  MAXRES 400.000000
  MAXCAP 500.000000
  MINRES 2.000000
  MINCUT 1
  SHIELDNET VSS
  SHIELDWIDTH 0.200000
  SHIELDGAP 0.100000
  CONNECTSUPPLY FLOAT
  SHIELDLAYERS 1
    DTMF_INST/digsynca DTMF_INST/digsyncd

END DIFFPAIR

DIFFPAIR
  WIDTH 0.200000
  PAIRGAP 0.100000
  THRESHOLD 20
  MAXRES 400.000000
  MAXCAP 500.000000
  MINRES 2.000000
  MINCUT 1
    DTMF_INST/digsyncc DTMF_INST/digsyncb

END DIFFPAIR

MATCH
  WIDTH 0.200000
  TOLERANCE 20
  MINCUT 1
    DTMF_INST/digsync1 DTMF_INST/digsync2 DTMF_INST/digsync3 DTMF_INST/digsync4
END MATCH

MATCH
  WIDTH 0.200000
  TOLERANCE 20
  MINCUT 1
    DTMF_INST/digsync3 DTMF_INST/digsync4
END MATCH

SHIELDING
  SHIELDWIDTH 0.800000
  MINCUT 1
  CONNECTSUPPLY ANYPOINT
  SHIELDLAYERS 7
  SHIELDNET AVSS
    analogN4I analogN3I analogN2I analogN1I
END SHIELDING

SHIELDING
  SHIELDWIDTH 1.000000
  MINCUT 1
  CONNECTSUPPLY ANYPOINT
  SHIELDLAYERS 7
  SHIELDNET AVSS
```

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

```
analogE4I analogE3I analogE2I analogE1I
END SHIELDING

SHIELDING
  SHIELDWIDTH 1.200000
  MINCUT 1
  CONNECTSUPPLY ANYPOINT
  SHIELDLAYERS 7
  SHIELDNET AVSS
    analogW4I analogW3I analogW2I analogW1I
END SHIELDING

SHIELDING
  SHIELDWIDTH 0.500000
  MINCUT 1
  CONNECTSUPPLY FLOAT
  SHIELDLAYERS 7
  SHIELDNET AVSS
    analogS4I analogS3I analogS2I analogS1I
END SHIELDING

BUS
  WIDTH 0.460000
  BUSGAP 0.460000
  SPACING 0.460000
  MINCUT 1
    DTMF_INST/dxout[14] DTMF_INST/dxout[0] DTMF_INST/dxout[1] DTMF_INST/dxout[2]
    DTMF_INST/dxout[3] DTMF_INST/dxout[4] DTMF_INST/dxout[5] DTMF_INST/dxout[6]
    DTMF_INST/dxout[7] DTMF_INST/dxout[8] DTMF_INST/dxout[9] DTMF_INST/dxout[10]
    DTMF_INST/dxout[11] DTMF_INST/dxout[12] DTMF_INST/dxout[13] DTMF_INST/dxout[15]
END BUS

BUS
  WIDTH 0.460000
  BUSGAP 0.460000
  SPACING 0.460000
  MINCUT 1
    DTMF_INST/dout[15] DTMF_INST/dout[14] DTMF_INST/dout[13] DTMF_INST/dout[12]
    DTMF_INST/dout[11] DTMF_INST/dout[10] DTMF_INST/dout[9] DTMF_INST/dout[8]
    DTMF_INST/dout[7] DTMF_INST/dout[6] DTMF_INST/dout[5] DTMF_INST/dout[4] DTMF_INST/
    dout[3] DTMF_INST/dout[2] DTMF_INST/dout[1] DTMF_INST/dout[0]
END BUS

BUS
  WIDTH 0.500000
  BUSGAP 0.500000
  SPACING 0.600000
  MINCUT 1
    DTMF_INST/a[7] DTMF_INST/a[6] DTMF_INST/a[5] DTMF_INST/a[4] DTMF_INST/a[3]
    DTMF_INST/a[2] DTMF_INST/a[1] DTMF_INST/a[0]
END BUS

BUS
  WIDTH 0.500000
  BUSGAP 0.500000
  SPACING 0.600000
  MINCUT 1
    DTMF_INST/cntrlyo[7] DTMF_INST/cntrlyo[6] DTMF_INST/cntrlyo[5] DTMF_INST/
    cntrlyo[4] DTMF_INST/cntrlyo[3] DTMF_INST/cntrlyo[2] DTMF_INST/cntrlyo[1]
    DTMF_INST/cntrlyo[0]
END BUS
```

## Encounter User Guide

### Using the Encounter Mixed Signal Router

---

```
WIDTH 0.5
BUSGAP 0.5
SPACING 0.6
MINCUT 1
SHIELDNET gnd
SHIELDWIDTH 0.5
SHIELDGAP 0.2
CONNECTSUPPLY ANYPOINT
SHIELDLAYER 1
    DTMF_INST/cntrlxo[7]      DTMF_INST/cntrlxo[6]      DTMF_INST/cntrlxo[5]
    DTMF_INST/cntrlxo[4]      DTMF_INST/cntrlxo[3]      DTMF_INST/cntrlxo[2]
    DTMF_INST/cntrlxo[1]      DTMF_INST/cntrlxo[0]
END BUS

BUS
WIDTH 0.460000
BUSGAP 0.460000
SPACING 0.460000
MINCUT 1
    DTMF_INST/cntrlxo[15]  DTMF_INST/cntrlxo[14]  DTMF_INST/cntrlxo[13]  DTMF_INST/
cntrlxo[12]  DTMF_INST/cntrlxo[11]  DTMF_INST/cntrlxo[10]  DTMF_INST/cntrlxo[9]
DTMF_INST/cntrlxo[8]  DTMF_INST/cntrlxo[7]  DTMF_INST/cntrlxo[6]  DTMF_INST/
cntrlxo[5]  DTMF_INST/cntrlxo[4]  DTMF_INST/cntrlxo[3]  DTMF_INST/cntrlxo[2]
DTMF_INST/cntrlxo[1]  DTMF_INST/cntrlxo[0]
END BUS
```

**Encounter User Guide**  
Using the Encounter Mixed Signal Router

---

---

## Optimizing Metal Density

---

- [Overview](#) on page 812
- [Before You Begin](#) on page 813
- [After You Complete Adding Via and Metal Fill](#) on page 813
- [Metal Fill Features](#) on page 814
- [Specifying Metal Fill Parameters](#) on page 820
- [Recommendations for Adding Timing-Aware Metal Fill](#) on page 821
- [Adding Metal Fill Over Macros](#) on page 824
- [Recommendations for Power Strapping Mode](#) on page 825
- [Adding Via Fill](#) on page 826
- [Trimming Metal Fill](#) on page 827
- [Verifying Metal Density](#) on page 827
- [Adding Metal Fill Using the GUI](#) on page 829

## Overview

The dielectric layers in chip designs often vary in thickness due to the different patterns of metal on successive metal layers. The thickness variations reduce yield and impact chip performance. To minimize the variations, you can add inactive metal segments, called metal fill, to open areas of the design. The metal fill makes the topology of the metal layers more uniform, which reduces the variations in metal density.

The additional metal increases cross-coupling capacitance, however, so it is important to balance the decrease in thickness variations with the increase in capacitance.

- To simplify the estimation of cross-coupling capacitance added by metal fill, the software adds metal fill in a staggered pattern. For more information, see “[Metal Fill Features](#)” on page 814.
- To minimize cross-coupling capacitance within layers, the software adds metal fill in timing-aware mode. For more information, see “[Timing-Aware Metal Fill](#)” on page 819.

In addition to adding metal fill to reduce thickness variations in metal layers, the software can also add cuts to meet minimum cut density requirements. The added cuts are modelled as via fill. For more information, see “[Adding Via Fill](#)” on page 826.

The chip manufacturer usually specifies a target metal density percentage for the metal layers and a range of acceptable minimum and maximum metal density. The metal fill commands help you achieve metal density within the acceptable range and the via fill commands help you meet the cut density requirements.

The software uses parameters specified in the LEF file or the fill commands to analyze the density and determine the size and position of the fill. It divides the design into windows and adds metal or cuts to open areas in each window until the metal and cut density meet the density requirements.

You can add fill to one or more layers at both the chip and block level.

Add via fill before you add metal fill.

If you perform additional routing after inserting fill, you can trim away fill that causes DRC violations.

### Related Topics

To see where metal fill fits into the design flow, see [Route the Design and Run Postroute Optimization and Analyze SI, Run Post-SI Optimizatin and Physical Verification and Generate GDSII Stream File](#) in the *Encounter Flat Implementation Flow Guide*.

## Before You Begin

- Complete detailed routing.
- Add via fill before adding metal fill.

To make sure metal fill is viewable, select the following options in the main Encounter window:

- Floorplan or Physical view
- *SNet* visibility toggle
- *Metal Fill* visibility toggle

To see the *Metal Fill* visibility toggle, move the slider bar under the *All Colors* button to the left.

For more information on setting object visibility, see “[The Main Window](#)” chapter in the *Encounter Menu Reference*.

## Adding Metal Fill in Multiple-CPU Processing Mode

You can add metal fill to the design in multi-threading mode by running the following command before adding the metal fill:

- `setMultiCpuUsage`

For more information on this and other multiple-CPU commands, see the “[Multiple-CPU Processing](#)” commands chapter in the *Encounter Text Command Reference*.

Alternately, fill in the appropriate parameters on the [Design – Multiple CPU Processing](#) form. (You can also access this form by clicking the *Set Multiple CPU* button on the *Route – Add Metal Fill* form.)

For more information, see [Accelerating the Design Process by Using Multiple CPU Processing](#).

## After You Complete Adding Via and Metal Fill

After adding via and metal fill, extract parasitics and run timing and signal-integrity analysis.

## Metal Fill Features

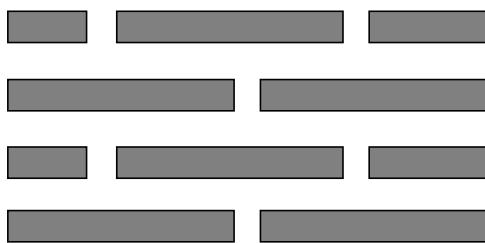
Metal fill has the following features:

- It can be square or rectangular shaped.
- It can be added in a staggered or non-staggered pattern.
- It can be connected to power or ground (tied-off) or left unconnected (floating).
- It can be added in timing aware or non-timing aware mode.
- It can be part of the power and ground structure.

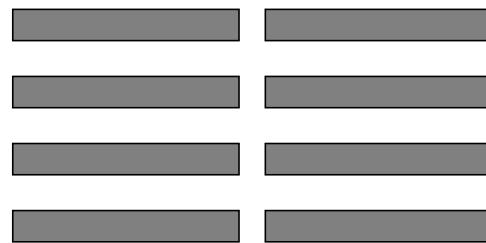
### Staggered Metal Fill Pattern

Staggering metal fill spreads out the effects of cross-coupling capacitance because the staggered pattern ensures that the metal fill does not line up on adjacent layers. The staggered pattern is most effective on lightly congested layers. By default, the software adds metal fill that is staggered in the preferred routing direction and not staggered in the non-preferred direction. The following figures show staggered and non-staggered patterns for both rectangular and square metal fill.

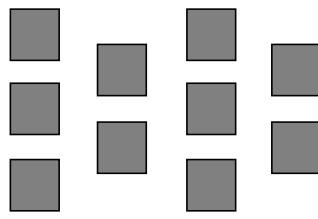
Staggered rectangular metal fill



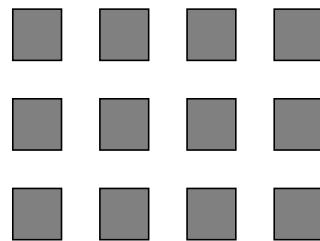
Non-staggered rectangular metal fill



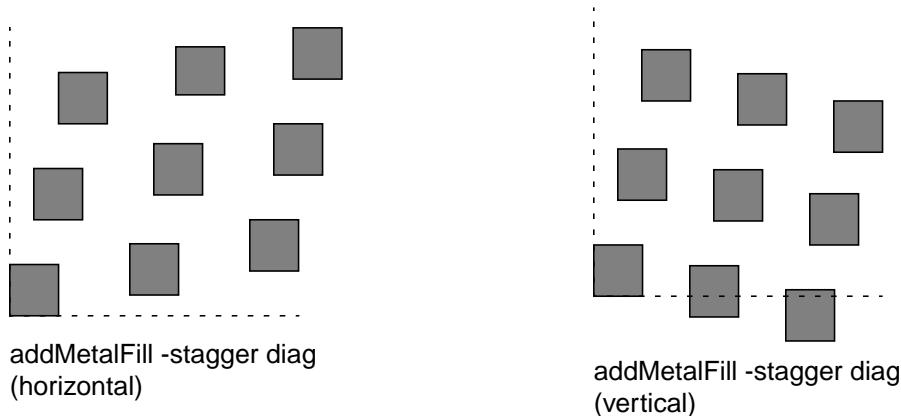
Staggered square metal fill



Non-staggered square metal fill



Metal fill that is staggered in both directions can also be added. This type of metal fill has a diagonal pattern. It is most apparent when it is added to the upper layers where there is not a lot of routing. The following figures show metal fill that is staggered diagonally:



## Connected and Floating Metal Fill

Metal fill segments can be connected (tied-off) to power or ground shapes on adjacent routing layers or left unconnected (floating). When it ties off metal fill, the software creates vias that fit within the area where the metal fill segment overlaps with a power or ground shape on an adjacent routing layer. It does not create vias that are larger than the overlapped area, or “cross-vias,” in which the via layer is contained within the same layer as the metal fill segment.

By default, the software creates both connected and floating metal fill. It is difficult to tie off all metal fill, so usually some shapes are left floating. You can minimize the number of floating shapes by including the following parameters when you run the `setMetalFill` command:

```
-removeFloatingFill  
-nets netNameList
```

If you remove the floating metal fill, however, it is more difficult to reach the preferred density requirements. In addition, floating metal fill has the following advantages over tied-off metal fill:

1. Lower cross coupling capacitance, especially if you specify short metal fill segments (long metal fill segments act like they are really tied off).
2. Easier to trim when there are violations. You can trim floating metal fill that causes DRC violations with the `trimMetalFill` command. If you add tied-off metal fill, however, you must delete it manually to avoid problems with vias.

When tied-off metal fill is trimmed, the vias cause the following problems:

## **Encounter User Guide**

### Optimizing Metal Density

---

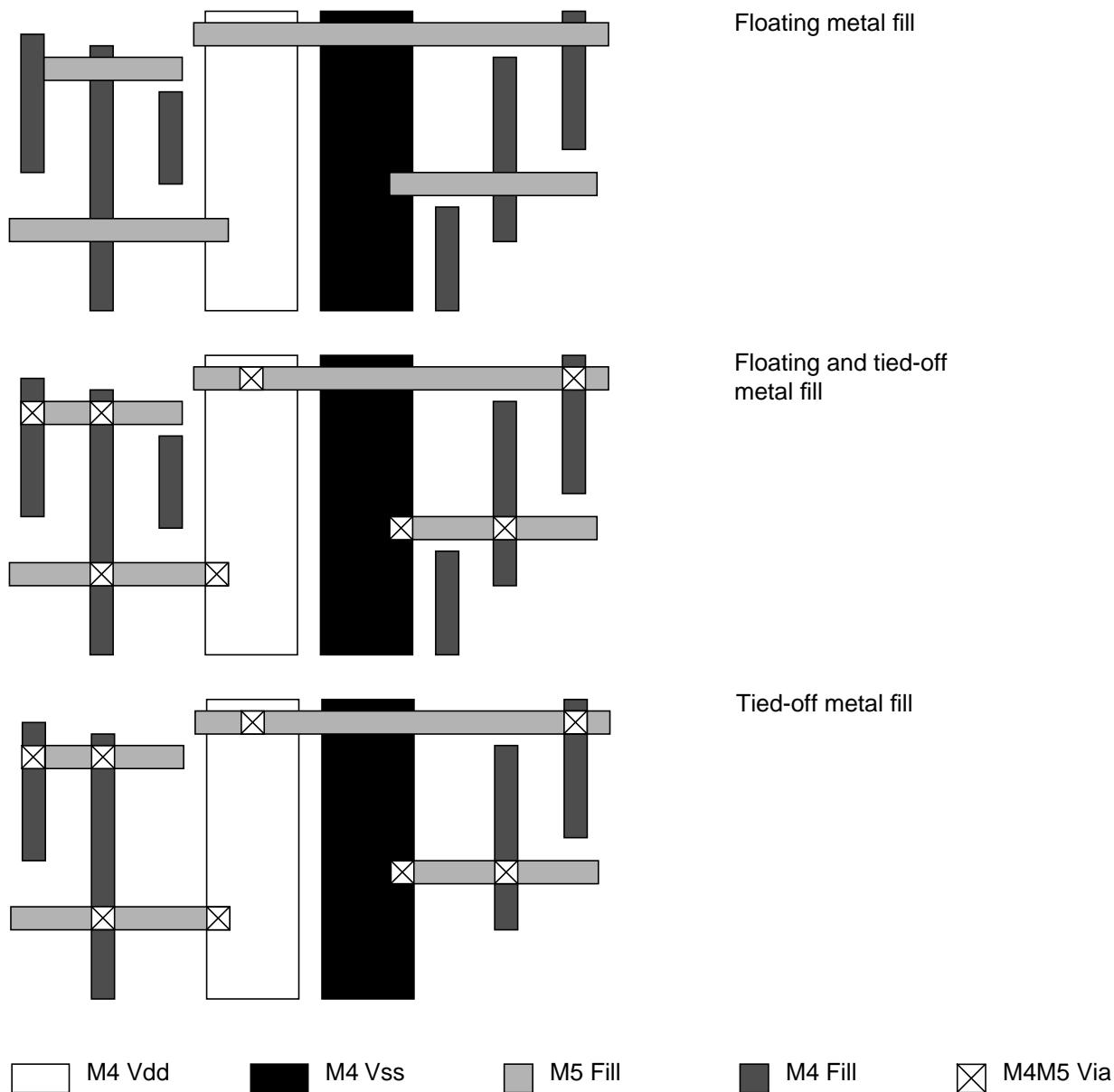
- ❑ If the vias are not deleted, they cause shorts to new wires.
- ❑ If the vias are deleted, the following problems might occur:
  - An isolated piece of previously tied-off metal fill might be left after trimming.
  - If the new routing was added during an ECO in which some layers were frozen, the change might affect a layer that should have been left frozen.

For more information, see the figures that follow and “[Trimming Metal Fill](#)” on page 827.

## Encounter User Guide

### Optimizing Metal Density

The following figures show a section of a design with metal fill. In the first figure, all the metal fill is floating. In the second figure, some of the metal fill is floating and some is tied off. In the third figure, all of the metal fill is tied off.

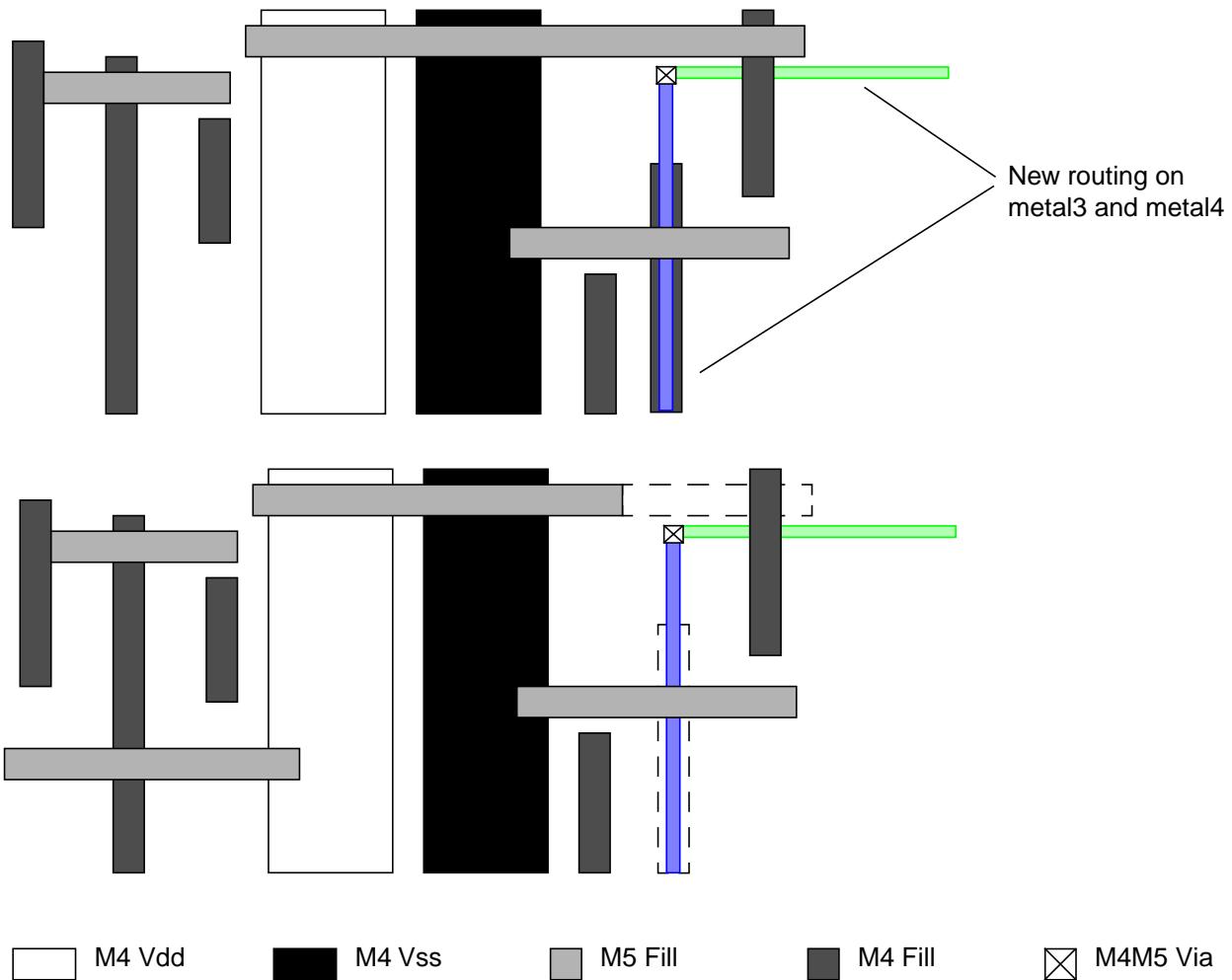


## Encounter User Guide

### Optimizing Metal Density

The following figures show the same design after an ECO, in which routing was added on *metal3* and *metal4*.

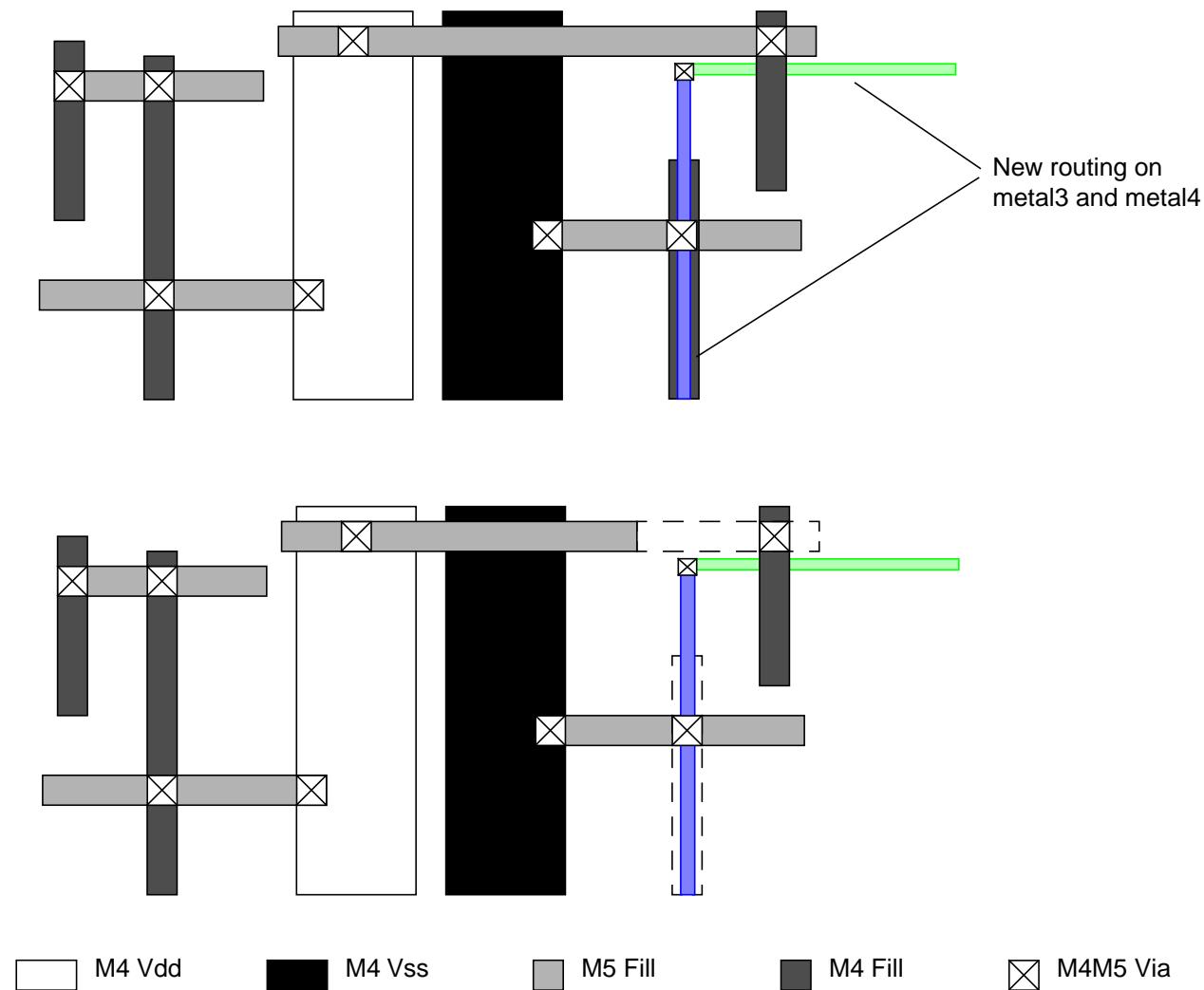
These figures show what happens when you use floating metal fill. The first figure shows the design with the added routing. The second figure shows the design after the metal fill is trimmed. The dotted lines show where the metal fill was trimmed.



## Encounter User Guide

### Optimizing Metal Density

These figures show what happens when you use tied-off metal fill. The first figure shows the design with the added routing. The second figure shows the design after the metal fill is trimmed. The dotted lines show where the metal fill was trimmed.



### Timing-Aware Metal Fill

When it adds timing-aware metal fill, the Encounter software avoids adding the fill near clock and signal nets and adds more fill near power and ground nets.

The software assigns a high cost to adding metal fill near clock nets, a moderate cost to adding it near signal nets, and zero cost to adding it near power and ground nets. It adds the fill, based on the cost, to achieve the preferred metal density with the least effect on timing.

The software adds timing-aware metal fill by default.

- To add non-timing aware metal fill, type the following command:  
`addMetalFill -timingAware off`
- To use the Encounter common timing engine (CTE) for static timing analysis (STA), type the following command:

```
addMetalfill -timingAware sta -slackThreshold value
```

If the `buildTimingGraph` command has already run, the software adjusts the costs as a function of the slack (nets with the worst slack have the highest cost). For more information, see “[Timing Analysis](#)” on page 899.

When you run the software in STA mode, it assigns costs to four categories of nets:

- Clock nets are assigned the highest cost.
- Signal nets are assigned a moderate cost.
- Non-critical signal nets are assigned a small cost.
- Power and ground nets (nets marked `+ USE POWER` or `+ USE GROUND` in the DEF file) are assigned 0 cost.

## Specifying Metal Fill Parameters

Some of the metal fill parameters can be specified in the Layer (Routing) section of the LEF file. All of the parameters can be specified by the Encounter metal fill commands or forms. The parameters that can be specified in the LEF file are listed in the table below.

If a parameter is specified in the LEF file, use the specified value. If a parameter is not specified, check the chip manufacturer’s DRC manual for the correct metal fill (or dummy fill) values and specify them manually with the command or form.



***If not specified properly, metal fill can cause DRC violations and increase capacitance unnecessarily. Parameters specified by the metal fill commands override parameters specified in the LEF file only if they are more restrictive than the LEF parameters.***

## Encounter User Guide

### Optimizing Metal Density

The following table lists the metal fill parameters that can be specified in the LEF file and corresponding Encounter metal fill parameters:

Description	LEF Statements	setMetalFill Parameter	Setup Metal Fill Parameter
Minimum distance between a segment of metal fill and another type of object in the design, such as a signal wire	FILLACTIVEESPACING	-activeSpacing	<i>Active Spacing</i>
Minimum density allowed in the design	MINIMUMDENSITY	-minDensity	<i>Metal Density % Min</i>
Maximum density allowed in the design	MAXIMUMDENSITY	-maxDensity	<i>Metal Density % Max</i>
Area the Encounter software uses to examine metal density	DENSITYCHECKWINDOW	-windowSize	<i>Step Size X</i> <i>Step Size Y</i>
Distance the window moves for each metal fill iteration	DENSITYCHECKSTEP	-windowStep	<i>Window Size X</i> <i>Window Size Y</i>

The Encounter software maintains the values specified for these parameters until you reset them or you restart the software.

For more information on LEF, see the [\*LEF/DEF Language Reference\*](#).

## Recommendations for Adding Timing-Aware Metal Fill

Follow these recommendations for metal fill parameters that specify the preferred density, metal fill shape, the space between the metal fill segments, and whether to use metal fill that is connected to special wiring. These parameters are not specified in the LEF file.

- Specify a preferred metal density between 25 percent and 40 percent.

Metal density within this range minimizes the density variation in design windows as well as the impact on added capacitance. The reduced variation improves correlation with early RC estimates, that is, it gives you faster timing convergence, and increases yield.

Determining the appropriate metal density is a process of balancing the decrease in density variation with the increase in capacitance: A density of 35 percent minimizes variation and increases the capacitance a moderate amount, a density of 25 percent adds less capacitance but does not decrease the variation quite as much.

■ Insert rectangular metal fill segments rather than square metal fill segments.

You can achieve the preferred metal density with fewer pieces of rectangular metal fill than with square metal fill. Adding rectangular segments reduces the number of flashes on the reticle, minimizes the density variation across the design windows, and approaches the preferred metal density in more windows.

The following dimensions for rectangular metal fill segments work with most 90 nm and 130 nm process rules:

- Length: 1  $\mu\text{m}$  to 10  $\mu\text{m}$
- Width: Use the width specified in the chip manufacturer's DRC manual for the minimum value. Use two to three times that value for the maximum width.

For example, you can specify the following dimensions:

- 0.4  $\mu\text{m}$  to 1.0  $\mu\text{m}$  for thin layers
- 0.8  $\mu\text{m}$  to 2.0  $\mu\text{m}$  for thick layers

Alternatively, for lower capacitance at the expense of more density variation, reduce the maximum width to the same value as the minimum width.

■ Follow the chip manufacturer's DRC manual for the spacing between metal fill shapes. This is called the gap spacing. The gap spacing is generally one to three times the minimum metal fill width.

The following dimensions for gap spacing work with most 90 nm and 130 nm process rules:

- 0.4  $\mu\text{m}$  for thin layers
- 0.8  $\mu\text{m}$  for thick layers

Alternatively, for lower capacitance at the expense of more density variation, use values like 0.8  $\mu\text{m}$  for thin layers and 1.6  $\mu\text{m}$  for thick layers.

■ Add metal fill to all metal layers or run the `verifyMetalDensity` command to determine where metal fill is needed.

■ Use metal fill that is not connected to special wiring.

Unconnected (floating) metal fill adds less capacitance to the design and is easier for postroute and postmask changes to handle than connected (tied-off) metal fill.

Alternatively, you can use tied-off metal fill whenever possible and floating metal fill when tied-off metal fill cannot be created. Either method is more likely to meet the preferred-metal density requirements than using tied-off metal fill throughout the design.

## Timing-Aware Examples

The following examples specify conservative values for a 90 nm or 130 nm eight-layer design where metal layers 1 through 6 are thin metal and metal layers 7 and 8 are thick metal.

The following command sets values for the active spacing, window size, window step, minimum density, and maximum density for all eight layers:

```
setMetalFill -layer "1 2 3 4 5 6 7 8" -activeSpacing 0.6 \
    -windowSize 100 -windowStep 100 \
    -minDensity 20 -maxDensity 70
```

The following command sets values for the gap spacing, preferred density, minimum and maximum width, and minimum and maximum length for the thin-metal layers:

```
setMetalFill -layer "1 2 3 4 5 6"
    -preferredDensity 35 -gapSpacing 0.4 \
    -minWidth 0.4 -maxWidth 1.0 \
    -minLength 1.0 -maxLength 10.0
```

The following command sets values for the gap spacing, preferred density, minimum and maximum width, and minimum and maximum length for the thick-metal layers:

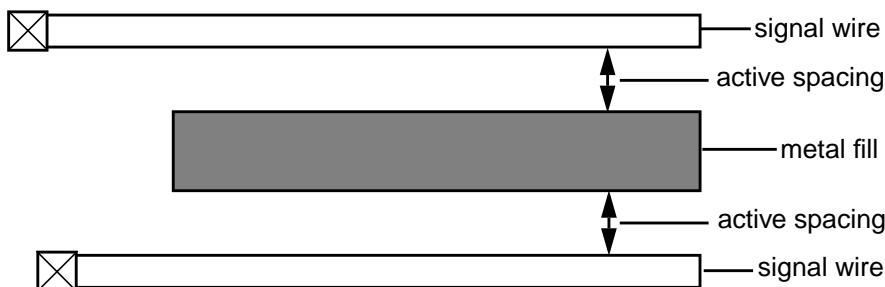
```
setMetalFill -layer "7 8"
    -preferredDensity 35 -gapSpacing 0.8 \
    -minWidth 0.8 -maxWidth 2.0 \
    -minLength 1.0 -maxLength 10.0
```

The following command adds metal fill to all eight layers:

```
addMetalFill -layer "1 2 3 4 5 6 7 8"
```

## Specifying the Active Spacing Value

The space between metal fill and nonmetal-fill geometries is called the *active spacing*, as shown in the following figure.



The Encounter software uses the `FILLACTIVEESPACING` value in the LEF file for the active spacing. If `FILLACTIVEESPACING` is not specified, you can set it manually by using one of the following methods:

- Specifying a value for `setMetalFill -activeSpacing` on the text command line

**Note:** The `setMetalFill -activeSpacing` settings are used for creating regular `FILLWIRE` shapes. For `FILLWIREOPC` shapes, design rules are used.
- Specifying a value for *Active Spacing* on the Setup Metal Fill form

If no value is specified in the LEF file, and you do not specify one manually, the software uses 0.6 microns for thin layers (less than 0.24 microns) and 0.8 microns for thick layers as the default active spacing value.

The default active spacing value is usually large enough that you can avoid using Optimal Proximity Correction (OPC) for the metal fill shapes. In addition, the default active spacing minimizes the increase in cross-coupling capacitance caused by the metal fill, which in turn reduces the additional timing delay.

As you increase the active spacing value, you reduce the space available for metal fill. A large increase—for example, 2  $\mu\text{m}$  to 3  $\mu\text{m}$  for a 90 nm or 130 nm process—might prevent you from meeting the minimum density rule for some windows.

## Adding Metal Fill Over Macros

For adding metal fill to macros, the added metal fill is based on the recalculated metal density for that metal layer. If the added fill is less than the preferred metal density (the default is 35

percent), the software tries to add metal fill shapes on that metal layer to meet the preferred density goal. Otherwise, it does not add any metal fill shapes.

For better metal density accuracy, use the LEF DENSITY table—a list of rectangles with metal density numbers. These rectangles cover the entire macro bounding box for that metal layer. The rectangles are defined in the MACRO section of the LEF file and are honored by the addMetalFill and verifyMetalDensity commands. To force addMetalFill to place correct metal fill shapes for a layer, place obstructions on the layer to block areas where metal fill should not be placed.

The DENSITY rectangles on a layer should not overlap and should cover the entire area of the macro. Choose the size of the rectangles based on the uniformity of the density of the block. If the density is uniform, a single rectangle can be used. If the density is not uniform, the size of the rectangles can be specified to be 10 to 20 percent of the density window size, so that any error due to non-uniform density inside each rectangle area is small.

For example, if the metal density rule is for a 100 m x 100 m window, the density rectangles can be 10 x10 m squares. Any non-uniformity will have little impact on the density calculation accuracy.

If two adjacent rectangles have the same or similar density, they can be merged into one larger rectangle, with one average density value. The choice between accuracy and abstraction is left to the abstract generator.

The DENSITY table syntax is:

```
[DENSITY
  {LAYER layerName ;
   {RECT x1 y1 x2 y2 densityValue ;} ...
  } ...
END] ...
```

For more information on LEF MACRO DENSITY, see the [Macro](#) section of the “LEF Syntax” chapter in the *Encounter LEF/DEF Language Reference*.

## Recommendations for Power Strapping Mode

In power strapping mode, the software makes mesh connections to power and ground bus wiring, instead of the tree-type connections used in regular connected mode. This configuration allows the metal fill shapes to carry current as part of the power and ground structure. Power strapping uses the maximum possible number of cuts in vias, based on the intersection area between layers, instead of using the minimum-cut based connections used in regular connected mode.

To get the best results in power strapping mode, follow these recommendations:

- Use longer maximum lengths (at least 100 µm).

Longer lengths increase the number of times a single metal fill shape intersects with the existing power/ground mesh.

- Use higher values for preferred density (40 percent to 50 percent).

Higher preferred density increases the number of metal fill segments retained as candidates for power strapping.

- Use wider metal fill

## Adding Via Fill

When it adds via fill, the Encounter software does the following:

- Attempts to add vias that meet cut density requirements
- Uses metal width and spacing values specified by the `setMetalFill` command (or Set Metal Fill form) to determine size and allowed placement locations
- Adds either tied-off or floating vias until meeting the preferred cut density
  - In tied-off vias, either the top or bottom layer is connected to power or ground.
  - Floating vias are not connected to power or ground.



### *Important*

To get the best results from via fill, add it before adding metal fill. You can minimize the need for via fill by inserting multiple-cut vias with the NanoRoute router prior to adding via fill. For information, see [setNanoRouteMode](#).

In the recommended flow, the software adds via fill to free space prior to adding other metal fill shapes. It does not connect via fill to metal fill.

Use the fill commands in the following order:

1. `setViaFill`
2. `setMetalFill`
3. `addViaFill`
4. `addMetalFill`

## Trimming Metal Fill

The automatic routers, including the NanoRoute® router, ignore metal fill (`FILLWIRE` and `FILLWIREOPC`) shapes and might create routes that cause shorts or DRC violations.

To remove the shorts and violations, complete the following steps:

- To remove floating metal fill that causes shorts or violations, run the following command:

```
trimMetalFill [-deleteViols] [-ignoreSpecialNets]
```

This command repairs violations caused by the metal fill shapes. If the metal density drops below the target after trimming the metal fill, re-run the `addMetalFill` command.

The `trimMetalFill` command trims metal and via fill shapes based on the following spacing rules:

- Between `FILLWIRE` and `FILLWIREOPC` shapes, the active spacing value, or minimum spacing, based on DRC rules, whichever is larger, is required.
- Between `FILLWIRE` shapes, the gap spacing value, or minimum spacing, whichever is larger, is required.
- Between `FILLWIREOPC` and active shapes, minimum spacing is required.
- Between `FILLWIREOPC` shapes, minimum spacing is required.

For more information, see [trimMetalFill](#).

- To remove connected metal fill, complete the following steps:

- a. Delete all metal fill by running the following command:

```
deleteMetalFill -shapes {FILLWIRE FILLWIREOPC}
```

For more information, see [deleteMetalFill](#).

- b. Insert new metal fill.

For information on `FILLWIRE` and `FILLWIREOPC`, see [Shape](#) in the “DEF Syntax” chapter of the *LEF/DEF Language Reference*. (`FILLWIREOPC` is not supported by LEF 5.6.)

## Verifying Metal Density

After adding or trimming metal fill, use the Verify Metal Density and Verify Geometry features to verify that the metal fill has been added correctly.

## **Encounter User Guide**

### Optimizing Metal Density

---

For more information, see the “[Verify Commands](#)” chapter of the *Encounter Text Command Reference*.

## Adding Metal Fill Using the GUI

1. Determine the minimum and maximum size for metal fill shapes for each layer, then set these values on the *Size & Spacing* page of the Setup Metal Fill form.
  - If you are using rectangular metal fill, use the *Rectangle Length* and *Metal Fill Width* values.
  - If you are using square metal fill, use the *Metal Fill Width* and *Square Decrement* values.
2. Determine the spacing around metal fill shapes for each layer, then set the value on the *Size & Spacing* page of the Setup Metal Fill form. You must set two types of spacing values:
  - Spacing between a metal fill shape and an active metal shape. An active metal shape can be a signal wire, a power wire, a cell, a pin, or any other structure that is not classified as a fillwire.
  - Spacing between a metal fill shape and another metal fill shape.
3. Determine the minimum, maximum, preferred, and external metal density for each layer, then set these values on the *Window & Density* page of the Setup Metal Fill form.
4. Use the Verify Metal Density form to create a *Verify Density* report.
5. Locate an area in the design for which metal density is too low, then select that area on the Add Metal Fill form.
6. Determine whether you want metal fill to be square or rectangular, then choose the appropriate value on the Add Metal Fill form.
7. Click *OK* or *Apply* on the Add Metal Fill form to add metal fill shapes to the area that you specified.

## **Encounter User Guide**

### Optimizing Metal Density

---

---

## Timing Budgeting

---

- [Overview](#) on page 832
- [Is My Design Ready for Budgeting?](#) on page 834
- [Deriving Timing Budgets](#) on page 835
  - [Budgeting Using the GUI](#) on page 835
  - [Budgeting Using Text Commands](#) on page 835
  - [Top-Level Budgets Derived by Using Active Logic View](#) on page 836
  - [Deriving Preliminary Budgets in Early Design Phase](#) on page 837
- [Budgeting Output Files for MMMC Designs](#) on page 839
- [Constraints Adjustment](#) on page 842
- [Analyzing Timing Budgets](#) on page 844
  - [Resolving Conflicts with Path-Based Exceptions](#) on page 844
  - [Budgeting Clock Latency in Propagated Mode](#) on page 847
- [Calculating Timing Budgets](#) on page 849
- [Customizing Budget Generation](#) on page 852
- [Verifying Timing Budgets](#) on page 854
- [Reading the Justify Budget Report](#) on page 855
- [Constraints Support in Budgeting](#) on page 860
- [Warning Report](#) on page 863

## Overview

In hierarchical design flows, chip-level timing constraints must be mapped correctly to corresponding block-level constraints. The Encounter® software does this automatically to produce predictable timing convergence.

The Encounter software apportions budgets to blocks using a path-based method, which might not have a direct relationship to the size of the blocks themselves. Encounter supports two ways to perform timing budgeting in hierarchical designs:

- Without Trial IPO

Timing optimization is not run before generating budgets at the port boundaries.

- With Boundary Trial IPO (the default)

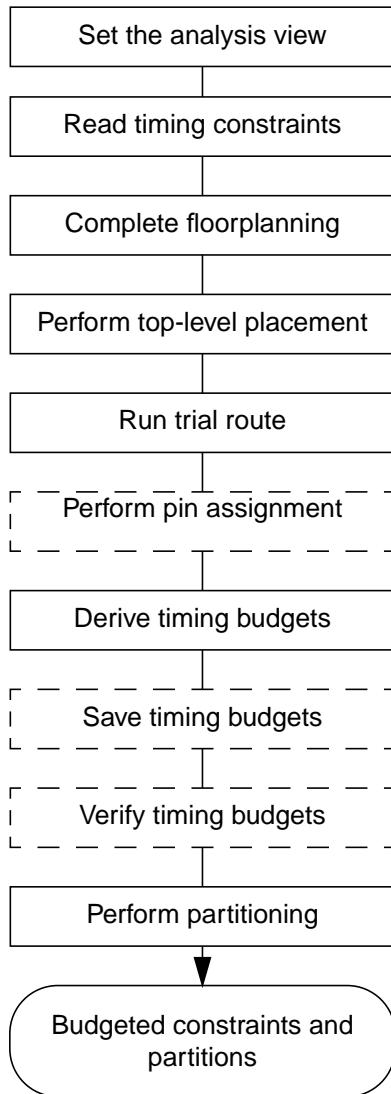
Timing optimization fixes are done for the top level nets and the interface nets.

MMMC mode is active if you have specified `set analysis view`. In MMC mode, timing budgeting is run per-view, and generates view files for partition implementation and top-level implementation (not chip assembly).

## Encounter User Guide

### Timing Budgeting

The following flowchart shows how timing budgeting is performed within the overall design flow.



**Note:** You can set the analysis view at any time before performing timing analysis, but could also be done before reading the constraint file.

## Is My Design Ready for Budgeting?

In order to close timing on the hierarchical level, you must be able to close timing on the flat design first. If the fully flat placement of a design (based on the partition fences, pin placement, and so on) does not meet timing before partitions are committed, then it is unlikely that timing will close after the partitions are committed and the budgets generated.

To help you decide whether the design is ready for budgeting, run virtual IPO on the flat design. This builds a timing graph, which allows you to examine the design for failing inter-partition paths. When you find these paths, you can use the information to determine why the problems occur and how you can fix them. In a hierarchical implementation, you might need to iterate top-level floorplanning and virtual IPO several times before creating a floorplan that can meet timing.

When you are convinced that the timing will close at this stage of the design process, you can then run timing budgeting.

## Deriving Timing Budgets

You can generate timing budget constraint files for each top-level partition using either the partition graphical user interface (GUI) or the various text commands.

### Budgeting Using the GUI

To generate the constraints files using the GUI, complete the following steps:

1. Read in the timing constraint file during design import.
2. Complete the floorplan for the design to partition.  
The more complete the floorplanning, the better the timing budgeting results.
3. Run top-level placement and preferably trial route, choosing medium effort for both.
4. (Optional) Use the Assign Partition Pins form (*Partition – Assign Pins*) to assign the partition pins.
5. Derive timing budgets. Choose *Derive Timing Budget* from the *Partition* menu, and specify the options you need on the Derive Timing Budget form.
6. (Optional) Save timing budgets. Select *Design – Save – Save Timing Budget*.
7. Partition the design. Select *Partition – Partition*.

Use the Save Partition form (*Design – Save – Partition*) to save the partitions to their directories. The directories are called *topCellName* for the top-level and *partitionName* for the partitions.

For each partition, this procedure creates a timing constraint file named *partitionName.constr.pt* for PrimeTime format. These files are located in each partition directory. The library model files, *partitionName.lib*, are created in the top-level directory. The constraints file *top\_level.top.constr* is created for the top level.

The result is budgeted constraints and partitions.

### Budgeting Using Text Commands

To generate the constraints files using the text commands, complete the following steps:

1. Read the timing constraint file during design import.
2. Complete the floorplan for the design to partition.

The more complete the floorplanning, the better the timing budgeting results.

3. Run top-level placement and preferably trial route.
4. Perform pin assignment.
5. Derive the timing budgets using the `deriveTimingBudget` command with the `-ptn` parameter.
6. (Optional) Save the derived budgets using the `saveTimingBudget` command with the `-ptn` parameter.
7. Partition the design.

The result is budgeted constraints and partitions.

## Top-Level Budgets Derived by Using Active Logic View

The Encounter software provides a top-level interface timing analysis flow to perform partitioning and budgeting on a trimmed-down version of the timing graph: a virtual partition. This flow saves memory usage and provides faster run time on large designs. The results of the flow are comparable to results of partitioning and budgeting with the full timing graph.

To perform partitioning and budgeting using top-level timing analysis, complete the following steps:

1. Load the hierarchical design into the database. Specify the partition information in the database.

```
source init.enc
```

2. Run placement.

```
setPlaceMode -fp true -ignoreScan true  
placeDesign
```

3. Run Trial Route and perform pin assignment.

```
trialRoute  
assignPtnPin
```

4. Run Trial Route to honor assigned pins.

```
trialRoute -honorPin
```

5. Use the `createActiveLogicView` command to identify interface logic for all the partitions.

```
createActiveLogicView -type flatTop
```

This command marks the interface logic in all the partitions in the chip-level design. When you build the timing graph after this step, the software masks the core logic inside the partitions and ignores it to reduce the run time and memory usage.

**6. Derive timing budgets.**

```
deriveTimingBudget -trialIPO
```

The command rebuilds the timing graph as part of its operation.

**7. (Optional) Evaluate the budgeting results.**

```
justifyBudget -pin pinABC Partition1
```

**8. Save the timing budgets.**

```
saveTimingBudget
```

**9. Clear the virtual partition marking after you save the timing budgets.**

```
clearActiveLogicView
```

## **Deriving Preliminary Budgets in Early Design Phase**

The Encounter software provides a flow for deriving timing budgets in the early stages of the design to obtain a preliminary estimate of the budgets. The software uses the net delay and net load that you specify using the [setBudgetingMode](#) command. To perform the preliminary budgeting, you use the -preliminary parameter of the [deriveTimingBudget](#) command. The software does not use trialIPO operations for calculating budgets for this flow.

The software uses the top-level net delays that you specify using the -topLevelDelayPerLen and -topLevelMinDelayPerNet parameters of the [setBudgetingMode](#) command. The software calculates the total delay value using the value of the -topLevelDelayPerLen parameter and the total length of the net. If the total delay value that the software calculates is less than the -topLevelMinDelayPerNet parameter, the software uses the value of the -topLevelMinDelayPerNet parameter. Otherwise the software uses the value of the -topLevelDelayPerLen parameter. The software assumes the block-level delays are zero.

The software uses the lump capacitance that you specify using the [setBudgetingMode](#) -overrideNetCap command for both top-level and block-level nets. The software uses this value to calculate the cell delay.

The software assumes all the net delay is on the top-level and does not perform boundary net adjustments. Therefore, when you run the [justifyBudget](#) command after generating the preliminary budgets, the Adjustment by Net Delay value is zero in the budgeting report.

## Encounter User Guide

### Timing Budgeting

---

The software proportions the budget according to the calculated values. If you do not use the `-preliminary` parameter in the `deriveTimingBudget` command, the software adds three extra buffer delays to the delay of the positively slacked path that has positive slack. If you use the `-preliminary` parameter, the software distributes the positive slack equally between source block, destination block, and top-level. Therefore, the value of following fields in the budgeting report is zero:

- Virtual buffering adjustment
- External buffering adjustment

For negative slack, the software uses the `-freezeTopLevelNegPathOnly` parameter of the `deriveTimingBudget` Command.

To perform preliminary budgeting, complete the following steps:

1. Load the hierarchical design in the database. Specify the partition information in the database.

```
source init.ENC
```

2. Run placement.

```
setPlaceMode -fp -ignoreScan  
trialRoute -noDetour -floorplanMode  
placeDesign
```

**Note:** Steps 1 and 2 are optional. You can source a routed design instead.

3. Set the delay values to be used during budgeting.

```
setBudgetingMode -topLevelDelayPerLen value  
-topLevelMinDelayPerNet value  
-overrideNetCap value
```

4. Derive timing budgets.

```
deriveTimingBudget -ptn partitionName -preliminary
```

5. Verify the budgets.

```
justifyBudget -short -pins pinList partitionname
```

6. Save the budgets.

```
saveTimingBudget -dir dirName -pt -rptNegSlackOnPorts value
```

## Budgeting Output Files for MMMC Designs

In MMMC mode, timing budgets are derived automatically for per view. Use the following command:

```
deriveTimingBudget -justify
```

This command derives the budgets for all ports of the instances or partitions specified by -inst or -ptn. Use `deriveTimingBudget -justify` instead `justifyBudget` to generate a report for per view/per partition.

Files are generated per view. The files are generated with the following structure:

```
budget_justify/Partition1/Partition1_view1.justify  
budget_justify/Partition1/Partition1_view2.justify  
budget_justify/Partition1/Partition1_view3.justify  
budget_justify/Partition2/Partition2_view1.justify  
budget_justify/Partition2/Partition2_view2.justify
```

In cases of setup and hold timing budgeting:

```
budget_justify/Partition1/Partition1_view1_setup.justify  
budget_justify/Partition1/Partition1_view1_hold.justify  
budget_justify/Partition1/Partition1_view2_setup.justify  
budget_justify/Partition1/Partition1_view2_hold.justify
```

A view is a combination of a mode (.sdc files) and a corner (libraries). The delay of the timing arcs through the partition ports may be different in terms of the constraints files (modes) and the libraries (corners). This means that one mode at the chip level becomes two or more modes at the partition/ top level. In this case, modes must be cloned. Similarly, one corner becomes two different corners at the top level because the timing model of a partition is not the same for the two different views. In this case, corners are cloned.

- Corner cloning affects only top-level data
- Mode cloning affects partition and top-level data

## Corner Cloning

The following example shows budgeting output for views having the same corner (C1) and different modes (M1 and M2).

```
View 1 (M1 C1)  
View 2 (M2 C1)
```

The output data from `deriveTimingBudget` are for the top level are as follows:

```
Top_view1.constr  
Top_view2.constr  
Ptn_view1.lib
```

## Encounter User Guide

### Timing Budgeting

---

Ptn\_view2.lib

The output data are for the block level are as follows:

Ptn\_view1.constr.pt  
Ptn\_view2.constr.pt

At the top level, for the corners, the view definition file contains the following:

View 1: Corner1 (pointing to corner 1 + Ptn\_view1.lib)  
View 2: Corner1\_budgeting1 (pointing to corner 1 + Ptn\_view2.lib)

Here, the corner is cloned.

## Mode Cloning

The following example shows budgeting output for two views with the same mode (M1), but different corners (C1 and C2):

View 1 (M1 C1)  
View 2 (M1 C2)

The output data from deriveTimingBudget are for the top level are as follows:

Top\_view1.constr  
Top\_view2.constr  
Ptn\_view1.lib  
Ptn\_view2.lib

The output data are for the block level are as follows:

Ptn\_view1.constr.pt  
Ptn\_view2.constr.pt

At the top level, for modes, the view definition file contains the following:

View 1: Mode 1 (pointing to the Top\_view1.constr)  
View 2: Model\_budgeting1 (pointing to the Top\_view2.constr)

In this case, the modes are cloned.

## Setup and View Handling for MMMC Designs

Views can be created for both setup as well as hold analysis. For example, the following command can set the same view for both setup and hold analysis:

```
set_analysis_view -setup [view] -hold [view]
```

The budgeting output constraint file contains setup as well as hold time constraints, but the .lib files are written separately for setup and hold.

The view definition file libraries for setup and hold are specified as follows:

```
create_library_set -name name_max -timing [... Ptn_max.lib]  
create_library_set -name name_min -timing [... Ptn_min.lib]  
create_delay_corner -name corner-name  
-late_library_set name_max  
-early_library-set name_min
```

## Constraints Adjustment

The timing budgeting process produces a `.lib` file for a partition that will be used during top-level implementation, and a SDC file for partition implementation. When top-level and block-level implementations are run in parallel, the timing model and the SDC files must match in order for chip assembly to succeed.

To ensure that the files match, timing budgeting makes adjustments for the following constraints:

- Capacitance

For each partition input pin, the tool produces the following output:

- ❑ In the `.lib` file, a specification of the pin's capacitance.
  - ❑ In the SDC constraint file, a `set_max_capacitance` constraint.

If a `max_capacitance` constraint in the SDC file is greater than the capacitance specified in the `.lib` file, this could lead to timing violations during the reassembly. The partition optimization might change the load of a partition input pin to a value such that the buffer, chosen at the top level with respect to the small capacitance specified in the `.lib` file, would not be able to drive the load.

The correlation adjustment done by budgeting ensures that the `pin_capacitance` specification in the `.lib` file and the `set_max_capacitance` constraint in the partition SDC to be very nearly the same.

- Transition

For each partition output pin, the tool produces the following output:

- ❑ In the `.lib` file, a lookup table describing the pin transitions with respect to load.
  - ❑ In the SCD constraint file, a `set_max_transition` constraint.

If the `.lib` lookup table indicates a range of transition values that are all less than the `set_max_transition` value used to constrain partition implementation, it could be possible for you to perform top-level implementation assuming that the transition will be, for example, 500 ps, while the partition implementation can pass with a transition of 1 ns on the same port. This situation could result in problems after reassembly.

The correlation adjustment done by budgeting ensures that the `set_max_transition` constraint in the partition SDC is within the lookup table in the partition `.lib`.

- Load and `max_cap`

For each partition output pin, the tool produces the following output:

## Encounter User Guide

### Timing Budgeting

---

- ❑ In the .lib file, a max\_capacitance DRV for the pin.
- ❑ In the SDC constraint file, a set\_load constraint.

A max\_capacitance constraint in the .lib greater than the set\_load constraint in the SDC can lead to timing violations during reassembly. The top-level optimization might change the load of the partition output pin to an unrealistic value for the buffer implemented within the partition, and chosen with respect to the small set\_load constraint.

The correlation adjustment done by budgeting ensures that the max\_capacitance in the .lib file and the set\_load constraint in the partition SDC file are nearly the same.

## Analyzing Timing Budgets

To analyze timing budgets, you must first identify all the boundary pins of the partitions. For each partition pin, the Encounter software generates timing constraints in the form of timing `set_input_delay` or `set_output_delay` if the pin is an input or output pin, respectively. The software divides the total available budget among all partitions involved, where their boundary pins constitute part of the path.

A pin might have multiple paths going through it. Multiple paths through the same port are handled by CTE budgeting. In case of multiple paths related to the same clocks and the same number of clock cycles, the tool automatically chooses the best path for deriving the budgets.

## Resolving Conflicts with Path-Based Exceptions

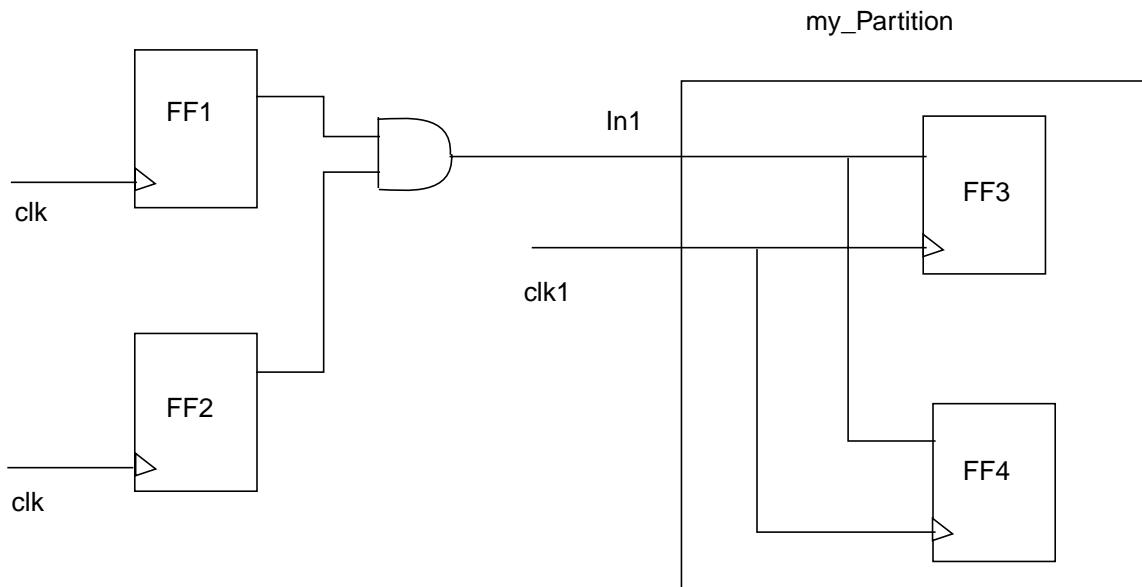
Budgeting generates one input or output delay assertion for each group of paths controlled by the same group of path-based exceptions. For `set_input_delay` generation at a partition port, the path group is the union of paths originating from the same clock phase that is controlled by the same group of exceptions at the partition port. For `set_output_delay` generation, the path group is the union of paths with the same clock phase at the end point that is controlled by the same set of exceptions traversing backward at the partition port.

CTS generates a prototype latency file, which is used during budgeting. CTS estimates the network latency for the top level and partitions. Budgeting then uses these values to generate latency constraints. The following example shows a command flow:

```
setBudgetingMode -localLatency  
estimatePartition -specFile top.ctstch -noAssignClockPin -keepBufTree  
trialRoute  
deriveTimingBudget  
saveTimingBudget -dir budgets
```

## Examples

All of the following examples use the same design:



### Case 1

Two virtual clocks will be created for each same group of path-based exceptions during budgeting.

- Chip-level exceptions:

```
set_multicycle_path 2 -setup -from FF1 -to my_partition/FF3/D
```

- For single cycles path and `clk`:

```
set_input_delay -clock clk_v0 number In1
```

(based on worst path from FF1)

- For multicycle path and `clk`:

```
set_input_delay -clock clk_v1 number In1
```

(based on worst path from FF1)

```
set_multicycle_path 2 -from clk_v1 -through in1 -setup -to FF3/D
```

## Case 2

Two virtual clocks are created for each same group of path-based exceptions.

- Chip-level exceptions:

```
set_multicycle_path 2 -setup -from FF1 -to my_partition/FF4/D  
set_false_path -from FF1 -to my_partition/FF3/D  
set_multicycle_path 2 -setup -from FF2
```

- For multicycle paths from FF1:

```
set_input_delay -clock clk1_v0 number In1  
(based on path from FF1 to my_partition)  
set_multicycle_path 2 -setup -from clk1_v0 -through In1 -to FF4/D
```

- For false path from FF1:

```
set_false_path -from clk1_v0 -through in1 -to FF3/D
```

- For multicycle path from FF2:

```
set_input_delay -clock clk2_v0 number In1  
(based on worst path from FF2)  
set_multicycle_path 2 -from clk2_v0 -through in1 -setup
```

## Case 3

Two virtual clocks are created.

- Chip-level exceptions:

```
set_multicycle_path 3 -setup -from FF1 -to my_partition/FF3/D  
set_multicycle_path 2 -setup -from FF2 -to my_partition/FF4/D
```

- For multicycle 3 path and clk:

```
set_input_delay -clock clk_v0  
(based on worst of paths from FF1)  
set_multicycle_path 3 -from clk_v0 -through in1 -setup -to FF4/D
```

- For multicycle 2 path and clk:

```
set_input_delay -clock clk_v1 number In1  
(Based on worst path form F2)  
set_multicycle_path 2 -from clk_v1 -through in1 -setup -to FF4/D
```

## Case 4

Two virtual clocks are created:

- Chip-level exceptions:

```
set_multicycle_path 2 -setup -from FF1 -to my_partition/FF4/D  
set_false_path -from FF1 -to my_partition/FF3/D  
set_multicycle_path 2 -setup -from FF2
```

- For multicycle 2 path from FF1:

```
set_input_delay -clock clk1_v0 number In1  
(based on path from FF1 to my_partition/FF4)  
Set_multicycle_path 2 -setup -from clk1_v0 -through In1 -to FF4/D
```

- For false path from F1:

```
set_false_path -from clk1_v0 -through in1 -to FF3/D
```

- For multicycle 2 path from FF2:

```
set_input_delay -clock clk2_v0 number In1  
(based on worst of paths from FF2)  
set_multicycle_path 2 -from clk2_v0 -through in1 -setup
```

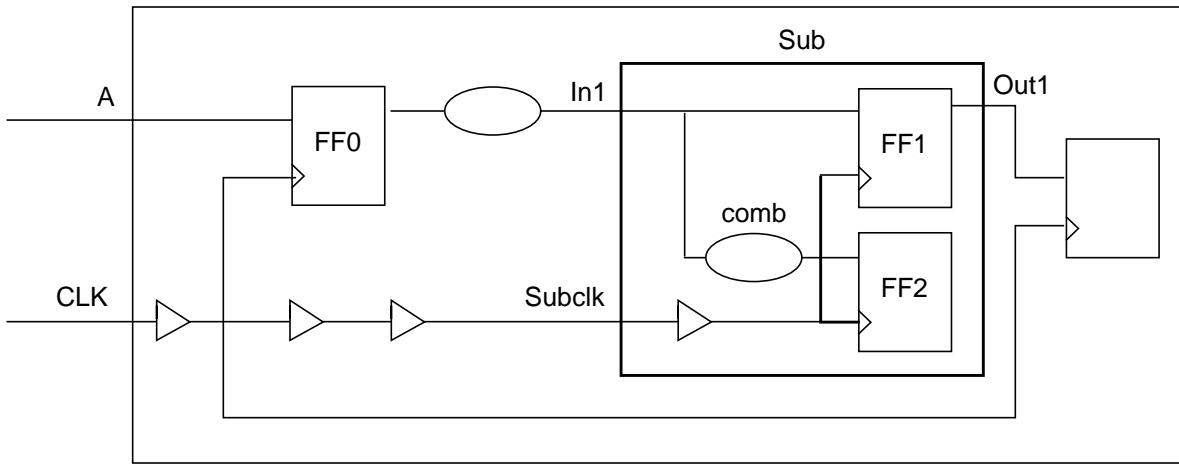
## Budgeting Clock Latency in Propagated Mode

The Encounter software includes clock latency in the constraints generated for clocks in propagated mode. The clock latency is included in the `set_input_delay` and `set_output_delay` constraints. The clock latency is added to `set_input_delay` and subtracted from the `set_output_delay`. This feature is useful when a clock tree is present in your design.

For multiple paths, both source and propagated clock latency is included in the `set_input_delay` and `set_output_delay` constraints. The software adds the `-source_latency_included` and `-network_latency_included` constraints in the `set_input_delay` and `set_output_delay` constraints for all inputs and outputs related to clocks in propagated mode. Consider the following figure.

## Encounter User Guide

### Timing Budgeting



The `deriveTimingBudget` command result in the following constraints for the `Sub` partition:

```
create_clock -clock subclk -waveform...
set_clock_latency -source (top_source + 0.2 + 0.2 + 0.2) subclk
set_input_delay -clock subclk (Input delay + top_source + 0.2)
    -source_latency_included -network_latency_included In1
set_output_delay -clock subclk (output_delay -top_source -0.2)
    -source_latency_included -network_latency_included Out1
```

Where,

*top\_source* = source latency of the clock at the top level

0.2 = delay through each buffer in the clock network

## Calculating Timing Budgets

Encounter proportions timing budget for partitions based on the path segment length, with a slight difference in calculation when the slack on a path is positive or negative.

For paths with negative slack, the proportioning formula for a setup check (max budgeting) is:

$$SD/TD * AT = BB(\text{neg})$$

For paths with negative slack, the proportioning formula for a hold check (min budgeting) is:

$$SD/TD * (AT + HT) = BB(\text{neg})$$

**Note:** If  $AT + HT$  is less than zero, the software does not use the proportioned value. The software uses the timing analysis values for input or output delays.

For paths with positive slack, the proportioning formula for a setup check (max budgeting) is:

$$SD + SD/TD * PS = BB(\text{pos})$$

For paths with positive slack, the proportioning formula for a hold check (min budgeting) is:

$$SD - SD/TD * PS = BB(\text{pos})$$

where:

- $SD$  is the delay through a path segment.
- $TD$  is the total delay of the path.
- $AT$  is the total available time. This could be the number of clock periods for multicycle paths, or the clock period minus the fixed delays.
- $HT$  is the hold time.

**Note:** For max budgeting, hold time is not same as setup time. Setup time is represented as an extra delay for the path. On the other hand, hold time is equivalent to required time, that is the amount of time a signal cannot change.

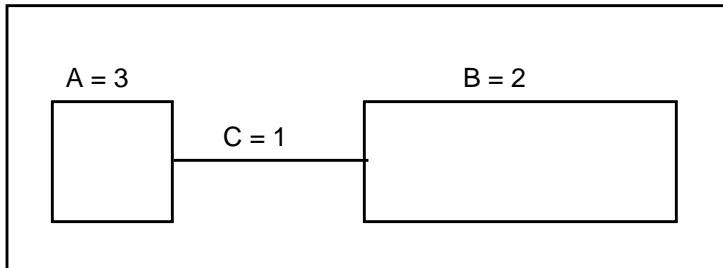
- $BB$  is the baseline budget
- $PS$  is the positive slack

**Note:** For a positively slacked path, budgeting adds virtual buffer delays to the path. The software usually adds three virtual buffer delays. In case of abutted designs, budgeting adds two virtual buffer delays. In case of feedthrough paths, budgeting distributes three buffer delay through all segments of the path.

## Encounter User Guide

### Timing Budgeting

#### Example 23-1 Negatively Slacked Path



In this example, block A is connected to block B via top-level net C. The budget of the top-level net is not fixed. When placed and routed, the path segment through block A needs 3 ns, path segment through block B needs 2 ns, and net C requires 1 ns. The available time to be budgeted is 5 ns.

The software calculates the following values:

$$\text{Budget for block A} = 3/6 * 5 = 2.5 \text{ ns}$$

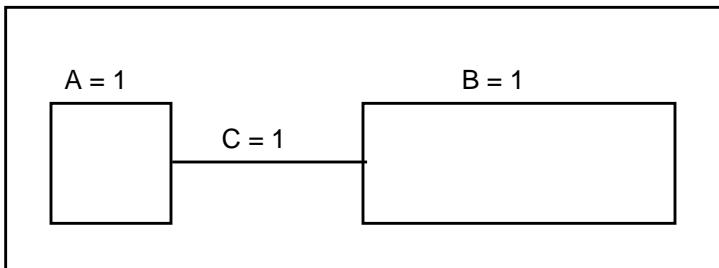
$$\text{Budget for block B} = 2/6 * 5 = 1.67$$

$$\text{Budget for net C} = 0.83$$

$$\text{Output delay at A} = \text{Budget for block B} + \text{Budget for net C}$$

$$\text{Input delay at B} = \text{Budgets for blocks A} + \text{Budget for net C}$$

#### Example 23-2 Positively Slacked Path



In this example, the path segment through blocks A and B, and net C require 1 ns each. The total delay is 3 ns. The total available budget is 5 ns. Therefore, positive slack is 2 ns.

The software calculates the following budget values:

## **Encounter User Guide**

### Timing Budgeting

---

Budget for A, B, and C =  $1 + 1/3 * 2 = 1.66$  ns

## Customizing Budget Generation

You can customize budget generation according to the design stage and timing requirements. To customize budget generation, use the following commands in Encounter:

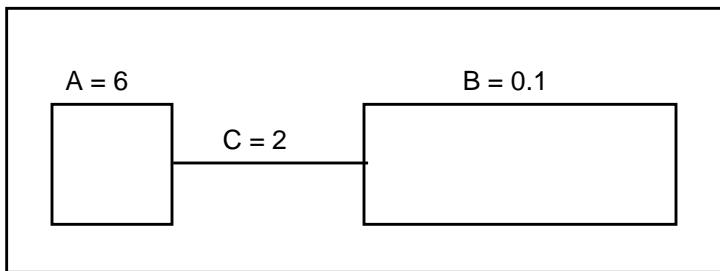
- The `-freezeTopLevel` parameter of the `deriveTimingBudget` command fixes the top-level timing budget and proportions the timing budget for the partitions. The commands consider blocks that are not being budgeted as fixed. If the top-level design has no buffers or glue logic, using the `-freezeTopLevel` parameter might not make much difference in the generated budgets.
- The `deriveTimingBudget [-ignoreDontTouch | -noIgnoreDontTouch]` command is used to consider `don't_touch` blocks. The `-noIgnoreDontTouch` parameter considers `don't_touch` as fixed delay. The `-ignoreDontTouch` parameter does not consider `don't_touch` as fixed delay. The budgeting results change based on whether fixed delay is considered during trial IPO.
- At the top level, you can set the `set_input_delay` and `set_output_delay` constraints on the hierarchical ports (or partition ports). The Encounter software generates budgets for the hierarchical ports based on the set constraints.
- For negative slack paths use the `-snapBudget` parameter of the `saveTimingBudget` command. This parameter reserves a certain percentage of available budget for the top level or partitions. The remainder of the available budget is proportioned to the partitions based on path segment length. This prevents under-budgeting of the top level or partitions.

Before using the `-snapBudget` parameter, run the `setBudgetingMode` command with no parameters to set the threshold budget ratio as the percentage of the overall available budget. The software uses this percentage to reserve budgets for the partitions when you use the `-snapBudget` parameter.

To use the `-snapBudget` parameter, complete the following steps:

- a. `setBudgetingMode 0.05`
- b. `deriveTimingBudget SH17 SH25`
- c. `saveTimingBudget -snapBudget -dir ptn -pt`

### **Example 23-3 Using the saveTimingBudget Command**



In this example, path segment through block A requires 6 ns, path segment through block B requires 0.1 ns, and net C requires 2 ns. The available time to be budgeted is 5 ns.

The software saves the following budgets when you use the `saveTimingBudget` command without the `-snapBudget` parameter:

$$\text{Budget for block B} = 0.1/8.1 * 5 = 0.06$$

$$\text{Budget for block A} = 6/8.1 * 5 = 3.70$$

$$\text{Budget for net C} = 1.24$$

The software saves the following budgets when you use the `setBudgetingMode` command with a ratio of 0.05, and the `saveTimingBudget` command with the `-snapBudget` parameter.

$$\text{Budget for block B} = \text{snapped to } 5\% \text{ of } 5 \text{ ns} = 0.25 \text{ ns}$$

$$\text{The remaining allowed time} = 5 - 0.25 = 4.75 \text{ ns}$$

$$\text{Budget for block A} = 6/8 * 4.75 = 3.56$$

$$\text{Budget for net C} = 2/8 * 4.75 = 1.19$$

- The `setBudgetingMode -topLevel` command specifies the minimum percentage of available allowed time to be set aside for the top. For example, if the clock period is 10ns and this minimum value is set to 0.1, then 10 percent of the clock period is set aside for top-level. The remaining 9 ns will be proportioned between the partitions.

## Verifying Timing Budgets

The Encounter software provides feedback on how the budgets are generated. The feedback is provided in a budget report file. You can verify the timing budgets by analyzing the results in the report file. To verify timing budget values, generate the report file by completing the following steps:

1. Derive budgets for partitions. In the command tool (console) window, type the following:

```
deriveTimingBudget -justify -ptn partitionName
```

The `deriveTimingBudget` command generates the timing models and stores them in the partition directories.

The command creates one justify report per view. The report contains debug data to justify timing budget for each pin and partition.

Views can be created for both setup and hold analysis, so the command generates budgets for setup and/or hold analysis type specified with `set_analysis_view`.

2. (Optional) Obtain a report consisting of headers for the reports generated per view:

```
justifyBudget -short
```

When in MMMC mode, `-short` is the only valid parameter for `justifyBudget`. If you want full reports, use `deriveTimingBudget -justify`.

3. Save the generated budgets by typing the following:

```
saveTimingBudget -ptn partitionName
```

The `saveTimingBudget` command saves time budget files of specified hierarchical instances to a specified directory. If you specify the `-setupHold` parameter in the `deriveTimingBudget` command, the `saveTimingBudget` command saves both setup and hold budgets.

## Reading the Justify Budget Report

You use the `deriveTimingBudget -justify` command to generate a budget report per view containing the debug data to justify the timing budget for a pin. For a negatively slacked path, the software distributes the total available time (in a simple clock period case) proportionally between ports of instances along the path. For a positively slacked path, the software usually adds some buffer delays to the generated delay values (built in positive slack).

The report generated contains the following fields:

- Adjustment for budget available time

Derived as follows:

Path Fixed Delay + Fixed Delay For Feedthrough Blocks - Clock Skew + Value of Constraint for the Receiving Register (HoldTime)

Where, Fixed Delay For Feedthrough Blocks is the two buffer delay distributed between all feedthrough blocks.

- Fixed delay adjustment

Specifies the delay that cannot be modified. The fixed delay adjustment includes: `set_input_delay`, `set_output_delay`, all cell delays for the cells marked as `dont_touch` if `-ignoredonttouch` is not used, delays of top level segments if `-freezeTopLevel` is used, any snapped delays calculated by using `setBudgetingMode -snapFdBudgetTo` or `-snapInputBudgetTo` or `-snapOutputBudgetRatio`. If, during timing analysis, the path segment delay used to generate a budgeting constraint for the port falls below specified threshold value the delay segment is snapped to the specified value and is considered as fixed delay during budget allocation.

- Virtual clock adjustment

Specifies a special adjustment to map the virtual clock into clocks pertaining to partitions. This number is generated when you use the `saveTimingBudget -noVirtualClock` command.

- Top Level Adjustment

Specifies the top-level delay value. The top-level delay value cannot be less than the minimum percentage of total available budget specified using the `-topLevel` parameter of the `setBudgetingMode` command.

- RC Adjustment (RC)

## Encounter User Guide

### Timing Budgeting

---

Specifies the input delays. During timing analysis the input delays are adjusted by the delay due to input port drive cell that was added by budgeting as a `set_drive` command in the generated constraint file. The Adjustment by RC number is subtracted from the delay value in budgeting so that this effect is not counted twice in the budget.

- Adjustment by clock latency

Specifies the clock latency of the driving object.

- Total Delay (`totDel`)

Specifies the total path delay.

- Initial Slack

Initial Slack = (Data Required Time - fixed delay) – (Path segment number1 delay + Path segment number 2 delay).

- Virtual Buffering Adjustment

Specifies the total extra delays added to the positive slacked path. This number is usually three extra buffer delays. In case of abutted designs, the number is two extra buffer delays.

**Note:** In case of feedthrough paths, three buffer delay is distributed through all segments of the path.

- Slack after Virtual Buffering Adjustment (`slack`)

The Encounter software takes out three buffers worth of delay from positive slack to safeguard minimum partition budget. This adjustment is used only for positive slacks.

- External Buffering Adjustment

Specifies the extra delay that is external to partition port. This is usually equivalent to two buffer delays. This is part of the virtual buffering adjustment. This delay is added to the input delay for the input ports and output delay for the output ports.

- Budgeted constraint

Budget = Adjustment for budget available time \* Delay for path outside the partition / Absolute total delay + Adjustment by fixed delay + Adjustment by virtual clock + Adjustment by clock latency - Adjustment by RC + External Buffering Adjustment

- External segment delay

Delay of the path segment outside of the partition.

- This block's segment delay

Delay of the path segment inside of the partition.

- Fixed delay through feedthrough
  - Amount of extra delay allocated to the path feedthrough segments.
- External Segment Fixed Delay from Budget Snap
  - The fixed delay for the path segment external to the partition contributed by using `setBudgetingMode -snapFdBudgetTo` or `-snapInputBudgetTo` or `-snapOutputBudgetRatio`.
- Total External Segment Fixed Delay
  - Fixed delay of the path segment outside of the partition.
- External Segment Extra Delay From Budget Snap
  - The extra delay added to the external path segment when you use `setBudgetingMode -snapOutputBudgetRatio` and `-snapInputBudgetRatio` and if external segment path delay is below user defined threshold.
- Fixed Delay Adjustment
  - The total path fixed delay.
- Clock Skew
  - The path clock skew.

**Note:** The report precision (the number of digits printed after the decimal point) is 3.

## Design Example

```
module Top(in1, clk1, clk2, out);
    input in1;
    input clk1;
    output out;
    input clk2;
    wire c0, c1, c2;

bfx0 buf0(.A(in1), .Z(c0));
    SUB      i_sub1(.sub_in(c0),
                  .sub_clk(clk1),
                  .sub_out(c1));
bfx0 buf1(.A(c1), .Z(c2));
    SUBn     i_sub2(.sub_in(c2),
                  .sub_clk(clk2),
                  .sub_out(out));
endmodule // TOP
```

## Encounter User Guide

### Timing Budgeting

---

```
module SUB (sub_in, sub_clk, sub_out);
    output sub_out;
    input sub_in;
    input sub_clk;
    df1qx1 sub_FF (.D(sub_in), .CP(sub_clk), .Q(sub_out));
endmodule // SUB
module SUBn (sub_in, sub_clk, sub_out);
    output sub_out;
    input sub_in;
    input sub_clk;
    df1qx1 sub_FF (.D(sub_in), .CP(sub_clk), .Q(sub_out));
endmodule // SUBn
```

## SDC Constraints for Design Example

```
current_design Top
create_clock -name clk1 -period 1 -waveform {0 0.5} [get_ports {clk1}]
set_input_delay 0.2 -clock clk1 [get_ports {in1}]
set_multicycle_path 2 -from [get_pins {i_sub1/sub_FF/CP}] -to [get_pins {i_sub2/sub_FF/D}]
create_clock -name clk2 -period 1 -waveform {0 0.5} [get_ports {clk2}]
set_output_delay 0.1 -clock clk2 [get_ports {out}]
```

## Generated Report for Design Example

To validate the budgets with positive slack in the design example, “[Design Example](#)” on page 857, type the following command:

```
justifyBudget -inst i_sub2 -pin sub_in
```

The following report was generated:

```
HInstance: i_sub2
Port: sub_in
Budgeted constraint type: set_input_delay(setup rise)
Virtual Clock: clk1_V0
Initial budget available time + clock skew = 2.000

One Buffer Delay for Adjustment(cell bfx2): 0.198
Fixed Delay for Feedthrough Paths(fixFdThru)= 0.000
External Segment Fixed Delay From Budget Snap(snapExtFixedDel) = 0.000
Total External Segment Fixed Delay(extFixDel) = 0.000
This Block's Segment Fixed Delay from budget snap(snapIntFixedDel) = 0.000
Total This Block's Segment Fixed Delay(intFixDel) = 0.000
External Segment Extra Delay From Budget Snap (snapExtDelExtra) = 0.000
This Block's Extra Delay From Budget Snap (snapIntDelExtra) = 0.000
Path Extra Delay From Budget Snap (snapExtraDel) = (0.000 + 0.000) = 0.000
Fixed Delay on the Path(pathFixDel) = (0.000 + 0.000 + 0.000 + 0.000) = 0.000
Fixed Delay Adjustment(fixtureDel)= 0.000
Clock Skew(clkSkew): 0.000
```

## Encounter User Guide

### Timing Budgeting

---

```

Adjustment for budget available time= -(pathFixDel + fixFdThru - clkSkew +
snapExtraDel)
= -(0.000 + 0.000 - 0.000 + 0.000) = -0.000
Available budget after adjustments(AvailTime)= (2.000 - 0.000) = 2.000

```

```

External Segment Delay(extSegDel): 0.638
This Block's Segment Delay(segDel): 0.273
Total delay(totDel): 0.638 + 0.273 = 0.910
Initial Slack = AvailTime - totDel
Initial Slack = 2.000 - 0.910 = 1.090
Virtual Buffering Adjustment: (3 x 0.198) = 0.594
Slack after Virtual Buffering Adjustment(slack): 1.090 - 0.594 = 0.496

```

```

External Virtual Buffering Adjustment(extVirBuf)= 0.396
Top Level Adjustment(topLev): 0.000
Virtual Clock Adjustment(virClk): 0.000
RC Adjustment(RC): 0.009
Budgeted constraint = extSegDel + slack * extSegDel / totDel + extVirBuf + topLev
+ fixDel + virClk + startClkLat - RC
Budgeted constraint = 0.638 + 0.496 * 0.638 / 0.910 + 0.396 + 0.000 + 0.000
+ 0.000 - 0.009 = 1.372

```

```

Path 1: MET Setup Check with Pin i_sub2/sub_FF/CP
Endpoint: i_sub2/sub_FF/D (^) checked with rising edge of 'clk2'
Beginpoint: i_sub1/sub_FF/Q (^) triggered by rising edge of 'clk1'
Other End Arrival Time 0.000
- Setup 0.269
+ Phase Shift 1.000
+ Cycle Adjustment 1.000
= Required Time 1.731
- Arrival Time 0.641
= Slack Time 1.090
Clock Rise Edge 0.000
= Beginpoint Arrival Time 0.000

```

Instance	Arc	Cell	Delay Time	Arrival Time	Required Slew	Load	Instance Location
<hr/>							
	clk1 ^			0.000	1.090	0.000	0.007
i_sub1/ sub_FF	CP ^ -> Q ^	df1q x1	0.182	0.182	1.272	0.063	0.005 (43.12, 366.80)
buf1	A ^ -> Z ^	bfx0	0.455	0.637	1.727	1.003	0.040 (43.96, 398.16)
i_sub2/ sub_FF	D ^	df1q x1	0.004	0.641	1.731	1.003	0.040 (43.12, 766.64)
<hr/>							

## Constraints Support in Budgeting

- `group_path`

This constraint is supported in timing optimization and timing analysis. In budgeting, it is not pushed-down inside the partition and top-level SDC. This will affect timing budgets, because the constraint affects chip-level timing analysis.

- `create_clock`

If a top-level clock `CK` is inverted, then while generating the budgets for a partition a new negative clock `CK_B%ENC` is created for the partitions connected to the negative clock. For example, if `CK` is defined as:

```
create_clock -name CK -period 7.500 -waveform { 0.000 3.750 } \
[list [get_ports {clk}]]
```

The negative clock is:

```
create_clock -name CK_B%ENC -period 7.500 -waveform { 3.750 7.500 } \
[list [get_ports {losdclk0_rp}]]
```

Where, `losdclk0_rp` is the clock port of the partition.

- `create_generated_clock`

- `set_clock_latency`

The `set_clock_latency` constraint is generated when you use the `setAnalysisMode -skew true` command. The clock latency is not budgeted between the partitions. The `setAnalysisMode -clockPropagation sdcControl`, along with `set_clock_propagation` constraint, do not cause the network delay through the clock tree to be budgeted for the partitions. The same clock latency is assigned to all the partitions if specified in the top-level clock constraints.

- `set_clock_uncertainty`

- `set_input_delay`

- `set_output_delay`

- `set_input_transition`

- `set_load`

- `set_drive`

- `set_driving_cell`

- `set_max_transition`

- `set_max_capacitance`

## Encounter User Guide

### Timing Budgeting

---

- `set_multicycle_path`
- `set_false_path`

Encounter timing analysis requires that the `set_false_path` and `set_multicycle_path` constraints have valid startpoints and endpoints for the `-from` and `-to` options. This corresponds to the requirement of PrimeTime.

Valid startpoints are:

- Input ports
- Input part of bidirectional ports
- Clock pins of sequential cells
- Pins associated with `set_input_delay`
- Pins associated with `set_path_delay -from`

Valid endpoints are:

- Output ports
- Output part of bidirectional ports
- Data pins of sequential cells
- Pins associated with `set_output_delay`
- Pins associated with `set_max_delay -to`

During budgeting, the Encounter software generates valid budgets for partitions based on invalid constraints at the top. For example, if `set_multicycle_path 2 -from SUB/IN1` is set at the top level, it is ignored during timing analysis, because a hierarchical pin is not a valid startpoint for `set_multicycle_path` constraint. However budgeting generates `set_multicycle_path -from IN1` for partition which is valid when the constraints are sourced for the partition because `IN1` is a top-level port for partition and a valid start point.

- `set_case_analysis`
- `set_max_delay`
- `set_min_delay`
- `set_logic_zero`
- `set_logic_one`

## Encounter User Guide

### Timing Budgeting

---

Partition ports could be left unconstrained, which means that there are some ports missing set\_input\_delay or set\_output\_delay constraints in the constraint file. Several factors can cause a partition I/O being unconstrained. For instance, set\_false\_path, set\_case\_analysis, set\_disable\_timing in an input constraint file can effectively cut paths through a port. The Set\_input\_delay constraint at the top-level, without a reference clock is another possibility which can cause some partition ports being unconstrained. Missing timing arcs in cell timing model can also cut timing paths. If a constant signal (1'b0, 1'b1) is assigned to a net leading to a partition port in Verilog®, the constant signal can also cause that port to be left unconstrained.

#### ■ Min and -hold

The following commands are supported:

- ❑ Set\_clock\_latency -min
- ❑ Set\_clock\_transition -min
- ❑ Set\_clock\_uncertainty -hold
- ❑ Set\_drive -min
- ❑ Set\_driving\_cell -min
- ❑ Set\_input\_delay -min
- ❑ Set\_output\_delay -min
- ❑ Set\_false\_path -hold
- ❑ Set\_load -min
- ❑ Set\_min\_delay
- ❑ Set\_multicycle\_path -hold

## Warning Report

The `saveTimingBudget -ptn` command generates a warning report (`partition_or_instance.warn`) for each partition and stores these reports in the partition subdirectories.

### Pin Constraint Values Greater than Available Time

The `.warn` report contains a section entitled “Pin constraint values greater than available time.”

The software checks whether generated `input_delay/output_delay` budgets are less than a maximum allowed time in the clock period for the delay. The maximum allowed time is defined as a delay between active edge of the starting clock and the sampling edge of the sampling clock. This time may vary based on phase shift and multicycle path directives. If `deriveTimingBudget tsConsCheck` is specified, budget checking will use more conservative value for available time:

The tool subtracts `clk2q` delays from available time when checking `output_delay` statements and setup time when checking `input_delay` statements.

The following command reports all partition ports that have slack less than `<value>` in `partition_dir/partition_name/partition_name/constr.warn`:

```
savePartition/saveTimingBudget -rptNegSlackOnPorts value
```

For example:

```
*.warn file:  
...  
/* Start Section: Instance ports with slack < 0.020 */  
/* End Section: Instance ports with slack < 0.020 */
```

### Warning Report Example

The warning report format is as follows:

```
*****  
* Timing constraint sanity check File  
*****  
  
/* Start Section: Pin constraint values greater than available time */  
/* End Section: Pin constraint values greater than available time */  
  
/* Start Section: Dropped chip level exceptions */  
/* End Section: Dropped chip level exceptions */  
  
/* Start Section: Ports without constraints */  
Port: sub1_out1 (setup)  
Port: sub1_out2 (setup)
```

# Encounter User Guide

## Timing Budgeting

---

```
Port: sub1_out3 (setup)
Port: sub1_out4 (setup)
Port: sub1_out1 (hold)
Port: sub1_out2 (hold)
Port: sub1_out3 (hold)
Port: sub1_out4 (hold)

/* End Section: Missing constraints due to constant signals at ports */
/* Start Section: Missing constraints due to false path assignments at ports */
/* End Section: Missing constraints due to false path assignments at ports */
/* Start Section: List of ports w/o constraints with added false_path assertions */
Port: sub1_out1 (setup)
Port: sub1_out2 (setup)
Port: sub1_out3 (setup)
Port: sub1_out4 (setup)
Port: sub1_out1 (hold)
Port: sub1_out2 (hold)
Port: sub1_out3 (hold)
Port: sub1_out4 (hold)

/* End Section: List of ports w/o constraints with added false_path assertions */
/* Start Section: Constraints with merged paths */
Constraint: set_input_delay 3.0552 -clock {vclk1_virtual_setuphold_0} -max -rise -
trail \
[find -ports {sub1_in1}]
Constraint: set_input_delay 2.7617 -clock {vclk1_virtual_setuphold_0} -max -fall -
trail \
[find -ports {sub1_in1}]

Merged Path:
Start_clk vclk1 (F) End_clk vclk1 (F)
Constraint: set_input_delay 0.0227 -clock {vclk1_virtual_setuphold_0} -min -rise -
trail \| [find -ports {sub1_in1}]
Constraint: set_input_delay 0.0018 -clock {vclk1_virtual_setuphold_0} -min -fall -
trail \| [find -ports {sub1_in1}]

Merged Path: |
Start_clk vclk1 (F) End_clk vclk2 (R) Corresponding exception: set_min_delay
5.0000

/* End Section: Constraints with merged paths */
/* Start Section: Toplevel constraints applied to ports */
/* End Section: Toplevel constraints applied to ports */
/* Start Section Unconnected Ports */
/* End Section Unconnected Ports */

/* Start Section: Missing constraints due to constant signals at ports */
Port sub1_in1 set to logical ZERO, 6 inversions
Port sub1_in2 set to logical ONE, 0 inversions
Port sub1_out1 set to logical ONE, 9 inversions
Port sub1_out2 set to logical ZERO, 3 inversions
/* End Section: Missing constraints due to constant signals at ports */
```

---

## **RC Extraction**

---

- [Overview](#) on page 866
- [Before You Begin](#) on page 867
- [Native RC Extraction](#) on page 868
- [CCE RC Extraction](#) on page 870
- [Generating a Capacitance Table](#) on page 874
  - [Inputs for Generating a Capacitance Table](#) on page 874
  - [Capacitance Table Generation Flow](#) on page 875
  - [Generating Capacitance Table With Specified Scaling Factors](#) on page 880
- [Reading a Capacitance Table](#) on page 882
- [Correlating Native Extraction With Sign-Off Extraction](#) on page 883
  - [Correlating SPEF Files Using the Ostrich Utility](#) on page 884
  - [Comparing SPEF Files Using a Perl Script](#) on page 887
  - [Defining the Scaling Factor](#) on page 890
- [Sign-Off Extraction Using QRC](#) on page 891
  - [Inputs for QRC Sign-Off Extraction](#) on page 892

## Overview

You can perform three types of extraction in Encounter:

- Native RC Extraction

Provides quick parasitic extraction (default mode) for design prototyping, or generates more accurate parasitics (detailed mode) for cross-coupling and signal integrity analysis. For more information, see [“Native RC Extraction” on page 868](#).

- Common Cadence Extraction (CCE)

Provides full chip or incremental near sign-off quality extraction support for the timing and SI optimization flow. Performing CCE-based extraction provides the best correlation results with signoff extraction. In addition, it provides the benefit of seamless integration with Encounter.

**Note:** CCE RC Extraction requires an additional QRC XL license. For more information, see [“CCE RC Extraction” on page 870](#).

- Standalone Sign-Off RC Extraction using QRC

Used to obtain sign-off quality detailed parasitic extraction. QRC can also be used with native extraction to generate RC scaling factors. For more information, see [“Sign-Off Extraction Using QRC” on page 891](#).

The following table summarizes the type of extraction used during the design process.

Extraction Type	When
Default	Used during optimization both before and after clock tree synthesis.
Detailed	Used during post-route optimization.
CCE	Used during the timing sign-off and SI optimization flow.
Standalone Sign-Off (QRC)	Used during chip assembly and timing sign-off processes.

## Before You Begin

Before running extraction, you enter the RC scaling factor values in the Timing Defaults page of the Design Import form. Scaling values provide better correlation between the FE estimated parasitics and the signoff extraction results by multiplying the extracted resistance and capacitance. For example, a capacitance scaling factor of 1.1 increases the extracted values by ten percent.

The following requirements are needed for both detailed and sign-off extraction:

- Capacitance table—2-D or 3-D field solver
- Interconnect technology (ICT) input file
  - Typical Case
  - Worst Case
  - Best Case
- RC scaling factors
  - R (resistance)
  - $C_L$  (total lumped capacitance). This is required for default extraction as well.
  - $C_x$  (Cross-coupling capacitance)

## Results

- Native (detailed, detail, and CCE) extraction—Creates binary RC Database (RCDB) and then generates an equivalent SPEF file on demand.
- Standalone QRC Extraction—Creates Standard Parasitics Extraction File (SPEF).

## Specifying Temporary File Locations

You can specify a temporary file location for CCE extraction.

The temporary file location is chosen based on the following order of precedence:

1. If you specify a directory using the `FE_TMPDIR` environment variable, the software uses that directory as the temporary file location.
2. If you specify a directory using the `TMPDIR` environment variable, the software uses that directory as the temporary file location.

3. Saves the files to the current directory (if writable).
4. Saves the files to the /tmp directory.

## Native RC Extraction

There are two modes for native RC extraction:

■ Default

In default mode, the total capacitance for each net is calculated based on the net geometry and the local wire density. The Encounter software does not calculate separate coupling capacitance in this mode. The default mode uses the wire geometries provided by TrialRoute.

**Note:** Default mode is only used for static timing analysis (STA).

■ Detailed

In detailed mode:

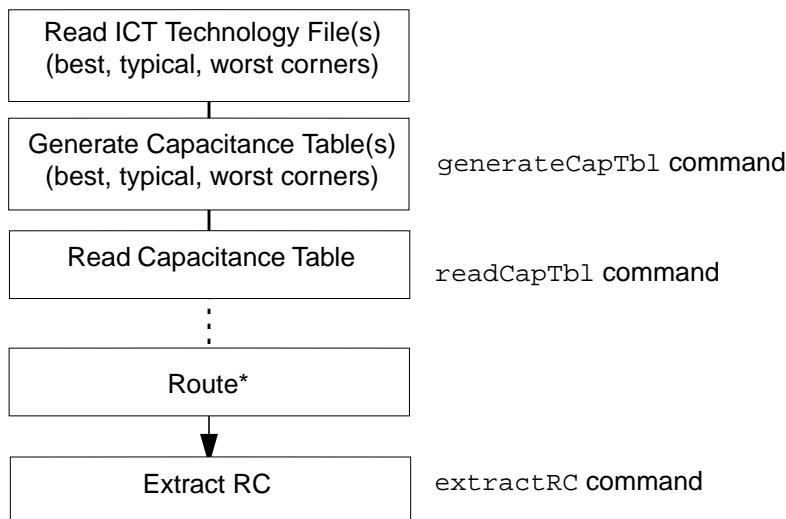
- RC values that are generated can be used for both STA (including cross-coupling) and signal integrity analysis to generate more accurate results for a particular process technology.
- The software calculates the coupling capacitance component for each segment by considering the actual geometries of neighboring nets on the same metal layer and the adjacent metal layer when a full capacitance table is provided during design import.

## Encounter User Guide

### RC Extraction

---

The following figure shows the native extraction flow.



\* Trial Route, WRoute, NanoRoute

To perform native extraction, complete the following steps:

1. Create an IceCaps Technology (ICT) file for each process corner—You can either create an ICT file manually or extract it from a QRC or QX Techfile using the Techgen utility.

**Note:** The Techgen utility is shipped with the EXT release.

For most technologies you can get this file from the technology vendors such as TSMC and UMC. For technologies that are not available through the technology vendors, you enter the fabrication process information into an ASCII-format interconnect technology (ICT) input file. For more information on creating the ICT files, see Appendix A “[Creating the ICT File](#)”, in the *Encounter User Guide*.

**Note:** The `minWidth` and `minSpacing` values should be the same in all three ICT files.

2. Generate a capacitance table using ICT files for each process corner. For more information, see “[Generating a Capacitance Table](#)” on page 874.
3. Read in the capacitance table values. For more information, see “[Reading a Capacitance Table](#)” on page 882.
4. Use the `setExtractRCMode` and `extractRC` command to perform extraction. You can also use the [Specify RC Extraction Mode](#) and [Extract RC](#) GUI forms to perform the extraction.
5. To correlate native extraction results with sign-off extraction, you compare SPEF files from native (detailed) and sign-off extraction to generate the new scaling factors for total capacitance, cross-coupling capacitance, and resistance. Using these scaling factors,

the native extraction results are closer to the sign-off extraction results, while only taking a fraction of the run time required for sign-off extraction. For more information, see [“Correlating Native Extraction With Sign-Off Extraction”](#) on page 883.

## CCE RC Extraction

There are two modes for CCE-based RC extraction:

- Full chip
  - Performs near sign-off quality extraction on the complete design.
- Incremental
  - Performs near sign-off quality incremental extraction on the design.  
Encounter recognizes the changes that have taken place since the last extraction and incrementally extracts the changed region of the design, then stitches the new data with the previously extracted parasitic data.

By default, the incremental mode is enabled.

**Note:** If you prefer to use a faster but slightly less accurate variant of CCE, you can invoke the fastCCE engine. To invoke fastCCE, use the following command:

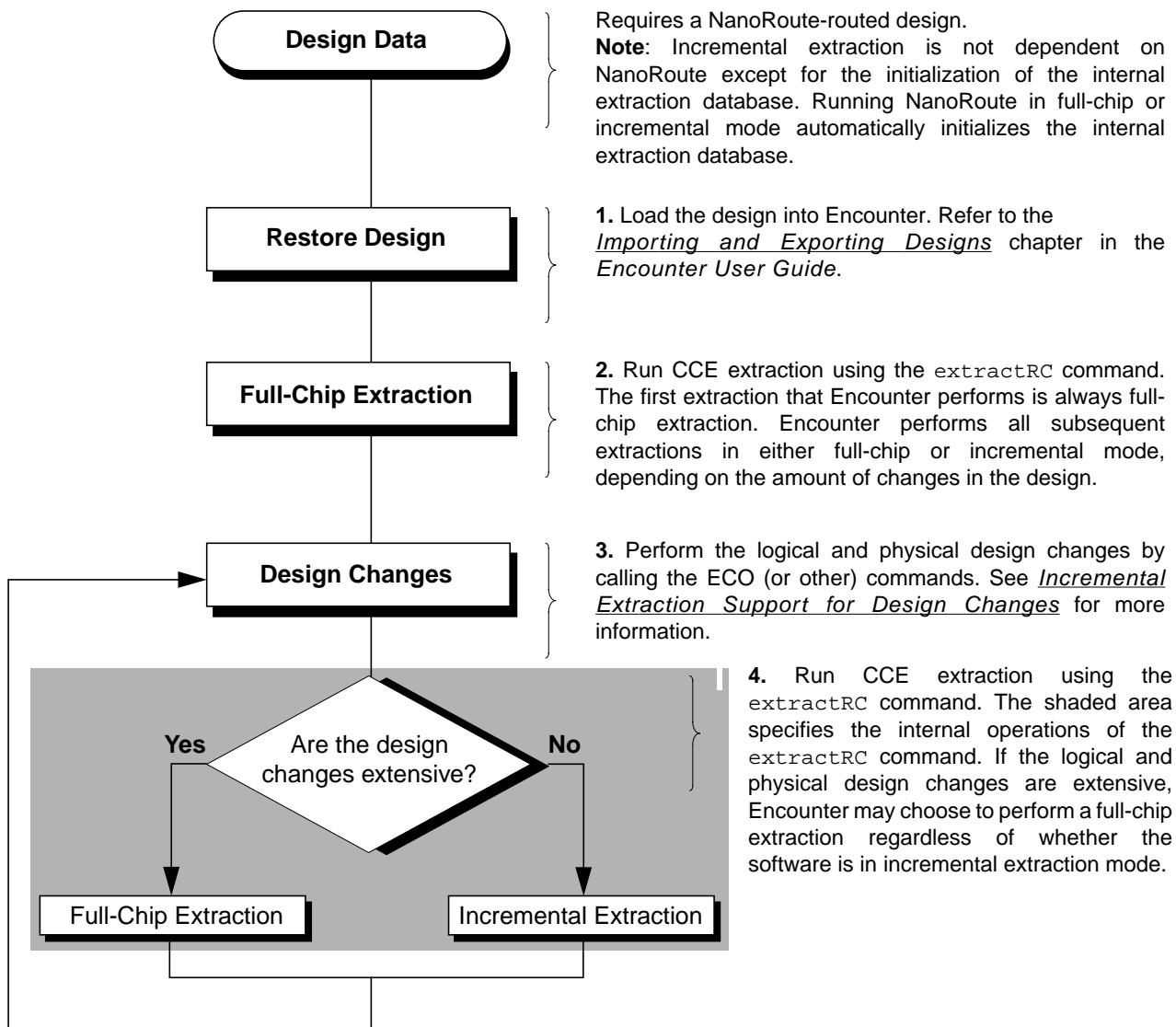
```
setExtractRCMode -engine CCE -effortLevel medium
```

The fastCCE engine is more accurate compared to detailed extraction and does not require a QRC license.

## Encounter User Guide

### RC Extraction

The following figure shows the CCE RC extraction flow.



## Related Topics

- [Flat Implementation Flow](#) chapter in the *Encounter Flat Implementation Flow Guide*
  - [“Analyze SI, Run Post-SI Optimization and Physical Verification and Generate GDSII Stream File”](#)

## Incremental Extraction Support for Design Changes

Encounter supports incremental extraction on design changes made with one (or more) of the following commands:

### ■ Interactive ECO Commands

After making ECO changes, native sign-off extraction recognizes the changes that have taken place since the last extraction. It incrementally extracts the changed region of the design and stitches the new data with the previously extracted parasitic data.

For example:

```
setExtractRCMode -engine CCE
extractRC
ecoAddRepeater -net netName -cell cellName
...
ecoPlace
ecoRoute
extractRC
```

Refer to the *Interactive ECO Commands* chapter in the *Encounter Text Command Reference* for command details.

### ■ Wire Edit Commands

For example:

```
setExtractRCMode -engine CCE
extractRC
editSelect -area areaValue -net netName
editMove y distance
editDelete -net netName
...
extractRC
```

Refer to the *Wire Edit Commands* chapter in the *Encounter Text Command Reference* for command details.

### ■ The optDesign -postRoute -si command.

For example:

```
setQRCTechfile techFileName
setExtractRCMode -engine CCE -incremental true
optDesign -postRoute -si
```

## Related Topics

To see where CCE extraction fits in the design flow, see [Analyze SI, Run Post-SI Optimization and Physical Verification and Generate GDSII Stream File](#) in the *Encounter Flat Implementation Flow Guide*.

## Generating a Capacitance Table

A capacitance table enables more accurate extraction results. It contains three parts:

- **Header:** Contains process information and manufacturing effects. The information in the header is used for resistance extraction and to correct the extracted values for the specified manufacturing effects.
- **Basic Captable Part:** Contains the coefficients used by the default capacitance extraction engine. This part contains the area, fringe, and lateral coupling capacitance coefficients organized per conductor layer for wires with different width and spacing. The basic capturable part is presented in a readable tabular format.
- **Extended Captable Part:** Contains the coefficients used by the detail capacitance extraction engine. This part is much larger compared to the basic capturable part because the coefficients are generated on more complex profiles, which account for geometries on multiple layers. The extended capturable part is an ASCII dump of the binary stored data.

The capacitance coefficients in the capacitance table are calculated by using 2-D and 3-D field solvers on the generated profiles. The 3-D field solver is shipped with the Encounter software and is called Coyote. Each technology requires one capacitance table. To consider process corner variations, you can generate multiple capacitance tables, one for each process corner.



The capacitance table must be generated before running extraction.

If the capacitance table is not defined before extraction, Encounter generates a basic capacitance table using default process parameters and using heuristic equations for calculation. It is strongly recommended to provide a capturable for the technology used in the design to ensure maximum accuracy.

### Inputs for Generating a Capacitance Table

To generate a capacitance table, you need an ICT file and optionally a technology LEF file. The technology LEF file provides the capacitance generated with design specify widths and spacings used in non-default routing rules. In addition, it provides information on the actual spacing between regular wires as defined by the PITCH statement. The use of a LEF file increases the simulation points and consequently reduces the need of the extractor to use interpolation. The LEF file is used for the extended capturable part only.

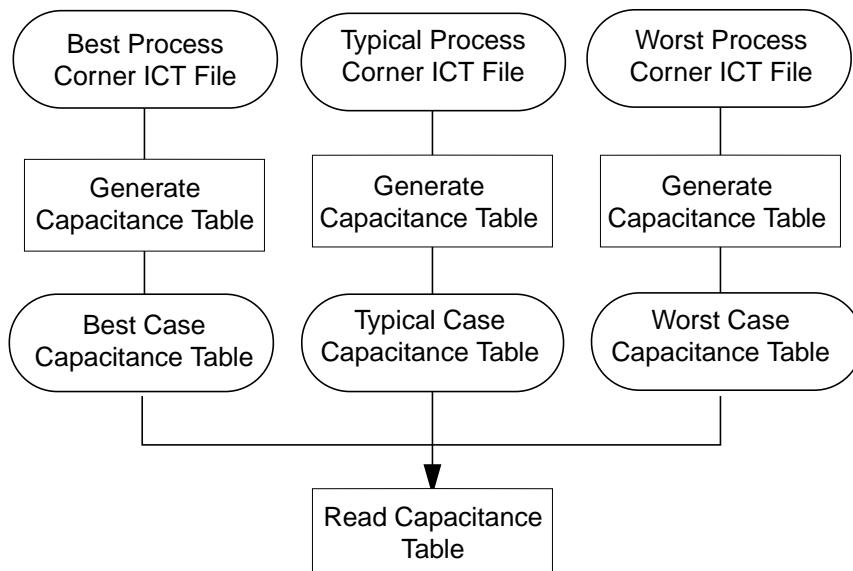
Fabrication process information in the ICT file can consist of the following:

- The minimum spacing and minimum width of the conductors as specified in the design rules for the conductor layers.
- The thicknesses of the conductor layers.
- The heights of the conductor layers above the substrate (measuring height from the field) or as a delta from a previously defined lower-level conductor layer.
- The resistivities of the conductor layers. The ICT file can contain a constant sheet resistance value, a width-dependent sheet resistance vector, or a resistivity (rho) table.
- The interlayer planar dielectric constant, its height above the substrate (measuring height above the field), and its thickness.
- The names of the top conductor layer of a via, the bottom conductor layer of the via, and the contact resistance of the via with their associated cut resistance.

For more information on the syntax of the ICT file, see Appendix A [“Creating the ICT File”](#).

## Capacitance Table Generation Flow

The following figure shows the flow for generating the three process corner capacitance tables.



**Note:** The best, typical, and worst corner designations are naming conventions used to differentiate the three separate corners (capacitance tables). You can define the contents of the ICT files according to your design requirements. You can also use the scaling factors to

convert the typical corner cap table to a three corner cap table using scaling factors. For more information, see “[Generating Capacitance Table With Specified Scaling Factors](#)” on page 880.

To generate a capacitance table, perform the following steps:

1. Generate an ICT file for each process corner. For an example ICT file, see Appendix A “[Creating the ICT File](#)”, in the *Encounter User Guide*.
2. Generate a capacitance table for each ICT file. Use the `generateCapTbl` command within Encounter or the `generateCapTbl` standalone executable.

**Note:** Generating a capacitance table is CPU-intensive and can take a full day to run for newer technologies. The `generateCapTbl` standalone executable which can be found in the `bin` directory of your Encounter hierarchy runs independent of Encounter; it has the same syntax as the `generateCapTbl` command.

#### *Important*

You can now use the [multi-cpu processing](#) commands to generate capacitance table in parallel mode when you use the `generateCapTbl` command within Encounter. This functionality is not available for standalone capacitance table generation.

For example, use the following set of commands to generate a capacitance table file in parallel mode:

```
setDistributeHost -rsh -add { host1 host2 host3 }
setMultiCpuUsage -numHosts 3
generateCapTbl -ict sample.ict -output sample.capTbl
```

**Note:** This functionality does not include multi- and super-threading operations. Therefore, the following options of the `setMultiCpuUsage` command are not supported:

- `-numThreads`
- `-superThreadsNumHosts`
- `-superThreadsNumThreads`

## Capacitance Table Examples

The extended capacitance table is based on a 3-D field solver which is used by native extraction and NanoRoute.

The extended capacitance table is generated by either the Cadence field solver (Coyote) or QuickCap. QuickCap requires a separate license from Random Logic Corporation. The

## Encounter User Guide

### RC Extraction

---

following sample files show portions of extended capacitance tables for Coyote and QuickCap.

#### **Example 24-1 Capacitance Table**

```
PROCESS_VARIATION ...
LAYER M1
    MinWidth          0.09000
    MinSpace          0.09000
#    Height           0.54000
    Thickness         0.20260
    TopWidth          0.12100
    BottomWidth       0.09300
    WidthDev          0.00000
    ThermalC1         2.65000e-03
    ThermalC2         -2.64100e-07
    WireEdgeEnlargement
        WeeWidths      0.107 0.127 0.152 0.197 0.287 0.377 0.467 0.557 0.647 0.917
1.017 2.017 3.017 4.517 7.517 12.017
        WeeSpacings     0.073 0.093 0.118 0.163 0.253 0.343 0.433 0.523 0.613 0.883
0.983 1.483 1.983 2.483 2.983 4.983
        WeeAdjustments   -0.001 -0.003 -0.003 -0.009 -0.009 -0.01 -0.01 -0.011 -
0.016 -0.018 -0.018 -0.019 -0.019 -0.019 -0.019
                           0.003 -0.002 -0.003 -0.009 -0.009 -0.01 -0.01 -0.011 -
0.016 -0.018 -0.018 -0.019 -0.019 -0.019 -0.019
                           0.008 0.003 0 -0.009 -0.009 -0.01 -0.01 -0.011 -0.016 -
0.018 -0.018 -0.019 -0.019 -0.019 -0.019
...
                           0.027 0.019 0.011 -0.009 -0.009 -0.01 -0.01 -0.011 -0.016
-0.018 -0.018 -0.019 -0.019 -0.019 -0.019
Rho
    RhoWidths        0.09 0.11 0.135 0.18 0.27 0.36 0.45 0.54 0.63 0.9 1 2 3
4.5 7.5 12
    RhoSpacings      0.09 0.11 0.135 0.18 0.27 0.36 0.45 0.54 0.63 0.9 1 1.5 2
2.5 3 5
    RhoValues        0.0301 0.0288 0.0272 0.0257 0.0236 0.0225 0.0218 0.0216
0.0216 0.0216 0.0216 0.0216 0.0216 0.0215 0.0215
                           0.0294 0.0286 0.0272 0.0257 0.0235 0.0224 0.0218 0.0216
0.0216 0.0216 0.0216 0.0216 0.0216 0.0215 0.0215
                           0.0286 0.0279 0.0269 0.0257 0.0235 0.0224 0.0218 0.0216
0.0215 0.0216 0.0216 0.0216 0.0215 0.0215 0.0215
...
                           0.0264 0.0262 0.0259 0.0257 0.0235 0.0224 0.0218 0.0216
0.0215 0.0215 0.0215 0.0215 0.0214 0.0213 0.0213
```

## Encounter User Guide

### RC Extraction

---

```
WireThicknessRatio
  WtrMinThicknessRatio      0.914688
  WtrMaxThicknessRatio      1.03875
  WtrTileWidth              100 100
  WtrStepperWindowWidth     50 50
  WtrMaxSpacing              5
  WtrWidthRanges            0.09 0.18 12
  WtrDensityPolynomialOrder 4
  WtrWidthPolynomialOrder    4
  WtrPolynomialCoefficients
{
  0 7180.07 -3744.08 625.29 -33.3498
  0 -8418.26 4361.73 -720.647 37.5707
  0 3011.8 -1560.96 257.063 -13.1704
  0 -369.306 193.11 -31.9649 1.58144
  0 -2.73935 -0.912962 0.476445 0.0054143
}
{
  -0.00355249 0.0134349 -0.377017 2.18449 -1.14899
  0.0086884 0.0669357 0.00360917 -3.62062 1.91715
  -0.0108639 -0.126014 0.84772 1.42329 -0.92238
  0.00784181 0.0469798 -0.580639 0.11264 0.0642999
  -0.00204508 -0.00330229 0.125923 -0.179971 0.100731
}
```

END

LAYER M2

...

...

...

VIA VIA1

```
  TopLayer        M2
  BottomLayer     M1
  ThermalC1       7.81500e-04
  ThermalC2       -2.57400e-06
  Resistance      3.00000
```

## Encounter User Guide

### RC Extraction

---

END

...

END\_PROCESS\_VARIATION

BASIC\_CAP\_TABLE ...

M1

width(um)	space(um)	Ctot(Ff/um)	Cc(Ff/um)	Carea(Ff/um)	Cfrg(Ff/um)
0.090	0.072	0.3549	0.1313	0.0502	0.0123
0.090	0.090	0.3115	0.1248	0.0502	0.0147
0.090	0.270	0.1803	0.0237	0.0502	0.0418
0.090	0.450	0.1728	0.0074	0.0502	0.0541
0.090	0.630	0.1721	0.0023	0.0502	0.0587
0.090	0.810	0.1720	0.0007	0.0502	0.0602
0.090	0.990	0.1720	0.0002	0.0502	0.0607
0.090	1.170	0.1720	0.0001	0.0502	0.0608
0.270	0.072	0.3797	0.1114	0.1267	0.0151
...					
9.000	0.990	4.2967	0.0002	4.2226	0.0369
9.000	1.170	4.2967	0.0001	4.2226	0.0370

M2

width(um)	space(um)	Ctot(Ff/um)	Cc(Ff/um)	Carea(Ff/um)	Cfrg(Ff/um)
0.100	0.080	0.3330	0.1275	0.0458	0.0161
0.100	0.100	0.2659	0.0919	0.0458	0.0182

.....

END\_BASIC\_CAP\_TABLE

EXTENDED\_CAP\_TABLE ...

# SolverExe: coyote

# Solver Type: coyote

1.02	8	8 t	1	
0.5385	0.2046	3.9	0	0
0.017	0	3	2	1
3	0			

....

END\_EXTENDED\_CAP\_TABLE

### **Example 24-2 Rho (Resistivity Table) Included in the Capacitance Table**

Rho

RhoWidths	0.14	0.28	10
RhoSpacings	0.14	0.28	1
RhoValues	0.0351	0.02	0.0176
	0.0451	0.03	0.01
	0.0702	0.04	0.02

### **Example 24-3 Wire Edge Enlargement - Resistance Included in the Capacitance Table**

WireEdgeEnlargementR

WeeWidths	0.12	0.16	0.24
WeeSpacings	0.108	0.17	0.24
WeeAdjustments	0	0	0.002
	0.011	0.007	0.002
	0.022	0.012	0.002

### **Example 24-4 Wire Edge Enlargement - Capacitance Included in the Capacitance Table**

WireEdgeEnlargementC

WeeWidths	0.12	0.16	0.24
WeeSpacings	0.108	0.17	0.24
WeeAdjustments	0.01	0.01	0.02
	0.01	0.07	0.02
	0.02	0.02	0.02

## **Generating Capacitance Table With Specified Scaling Factors**

You can specify scaling factors to convert a specific corner capacitance table into another capacitance table for a different process corner. You use the `generateCapTbl` command to input a capacitance table and to specify the scaling factors. You use the following parameters of the `generateCapTbl` command:

- `-incaptable fileName`  
Specifies the name of an existing capacitance table in ASCII format.
- `-cap totalCapFactor`  
Specifies the capacitance scaling factor.
- `-xcap crossCouplingFactor`  
Specifies the cross-coupling capacitance scaling factor.

## **Encounter User Guide**

### RC Extraction

---

- *-res resistanceFactor*  
Specifies the resistance scaling factor.

## Reading a Capacitance Table

To consider process corner variations, you can read multiple capacitance tables, one for each process corner. In the three-corner flow, you have a choice of reading:

- Up to three capacitance tables one for each best, typical, and worst corners.
- Two capacitance tables, one for the best-case and one for the worst-case corner.
- Single capacitance table, which the software uses for all analysis modes.

There are two ways to read in the capacitance table:

- Use the following command in the Encounter configuration file:

```
set rda_Input(ui_captbl_file) "xxx.capTbl"
```

Or,

```
set rda_Input (ui_captbl_file) "-typical fileName1 -best fileName2 -worst  
fileName3"
```

- Use the readCapTable command to read the capacitance table after loading the design and before running extraction.

```
readCapTable  
{-typical fileName1 -best fileName2 -worst fileName3 | fileName}
```

Examine the command log for any possible error messages. The number of metal layers specified in the ICT and the LEF file (if used) at the time of capacitance table generation must match or be higher than the actual number of layers used in your design (current LEF/DEF).

The capacitance table contains standard names for metal layers (M1, M2...), not the names used in the LEF file.

**Note:** You must read the capacitance table before specifying the extraction mode.

In multi-mode multi-corner (MMMC) mode, you must read in one capacitance table per RC corner analysis point. For more information, see the Configuring the Setup for Multi-Mode Multi-Corner Analysis section of the *Performing Multi-Mode Multi-Corner Timing Analysis and Optimization* chapter.

## Correlating Native Extraction With Sign-Off Extraction

The Encounter software accommodates an extraction flow that uses process-dependent scaling factors to generate extraction values that are close to the sign-off extraction values. With these scaling factors, the results generated by the native extraction correlate to the results of sign-off extraction. The run time for the native extraction flow is much less than for sign-off extraction.

Complete the following steps to generate the RC scaling factor to correlate native extraction results with sign-off extraction.

1. From the routed DEF, generate a SPEF file using the `runQRC` command. For more information on generating a SPEF file, see “[Sign-Off Extraction Using QRC](#)” on page 891.
2. Generate a capacitance table file using the ICT process file(s). For more information, see “[Generating a Capacitance Table](#)” on page 874.
3. Read in the generated capacitance table. For more information, see “[Reading a Capacitance Table](#)” on page 882.
4. Generate a SPEF file using the Timing `extractRC` and `rcOut` commands.

For default mode, extraction should be run in the preroute design stage and not on the final routed design. This way the scaling factors to improve correlation will also take into account the difference between trial routes and final routes.

For detail mode, extraction should be run on the same routed design that was used for the signoff extraction SPEF file generation.

5. Compare the SPEF file from native extraction with the SPEF file from sign-off extraction using the Ostrich parasitics correlation utility. Use the correlation utility to generate RC factors (scaling factors) for total capacitance, cross-coupling capacitance, and resistance. For more information, see “[Correlating SPEF Files Using the Ostrich Utility](#)” on page 884.

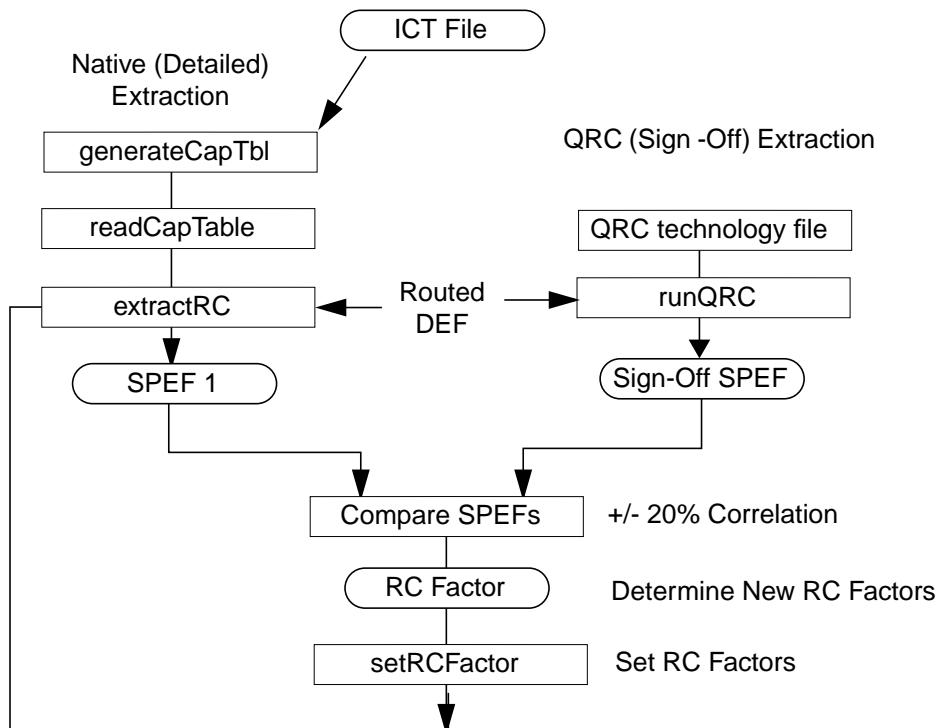
The extracted values are accurate if the total capacitance scaling factor has a deviation that is within 20 percent of 1.0 (that is, 0.80 to 1.20).

You can also use the Perl script `spefCapCmp.pl` to compare the SPEF files, and generate scaling factor for total capacitance. For more information, see “[Comparing SPEF Files Using a Perl Script](#)” on page 887.

6. Specify these scaling factors using `setRCFactor` before future runs of native extraction. For more information, see “[Defining the Scaling Factor](#)” on page 890.

7. Rerun the `extractRC` command to generate a new SPEF file. This file contains capacitance and resistance values that correlate to the values in the QRC sign-off SPEF file.

The following figure shows the flow for generating RC scaling factors.



## Correlating SPEF Files Using the Ostrich Utility

Use Ostrich to correlate the SPEF files generated using native (detailed) extraction and sign-off extraction. Ostrich is a standalone utility in Encounter. Ostrich generates the scaling factors after correlating the SPEF files. You can then set the scaling factors for the next extraction cycle.

### Related Topics

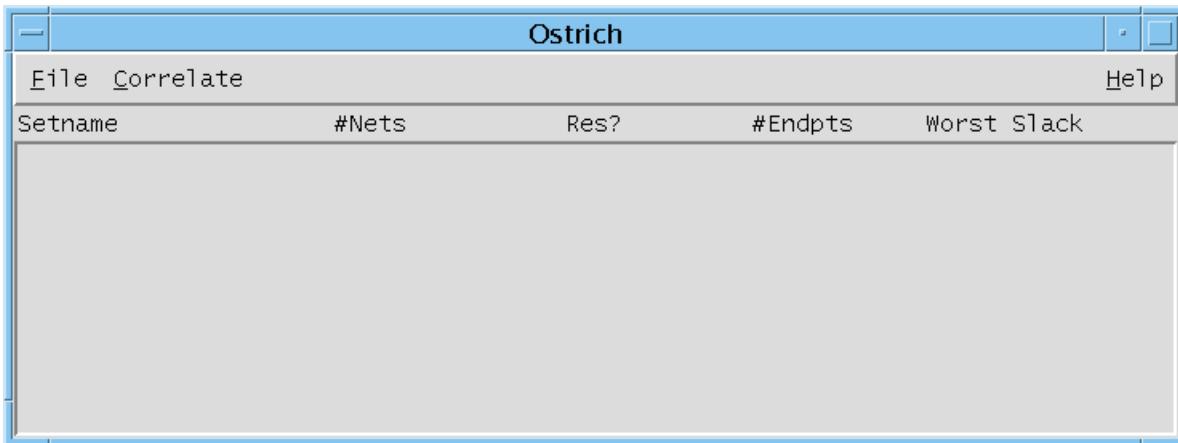
- [Flat Implementation Flow](#) chapter in the *Encounter Flat Implementation Flow Guide*
  - [“Results”](#)

To correlate the SPEF files, complete the following steps:

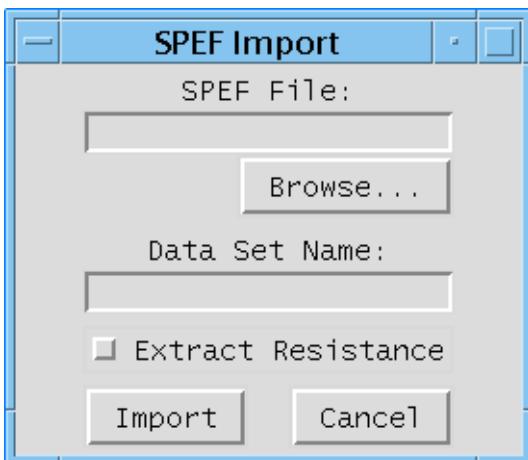
## Encounter User Guide

### RC Extraction

1. Type `ostrich` at the Encounter prompt. This opens the main Ostrich window.



2. In the Ostrich window, click on *File - Import - SPEF*. This opens the SPEF Import form.



3. In the SPEF Import form, specify the name of the sign-off SPEF file and a name in the *Data Set Name* field. Next, click on the *Import* button to add the SPEF values in the Ostrich window. To correlate the resistance values, select the *Extract Resistance* option.
4. Similarly, import the native extraction SPEF file using the SPEF Import form.

## Encounter User Guide

### RC Extraction

5. Click on *Correlate - Build Plot* option in the Ostrich Window. This opens the Build Correlation Plot window.

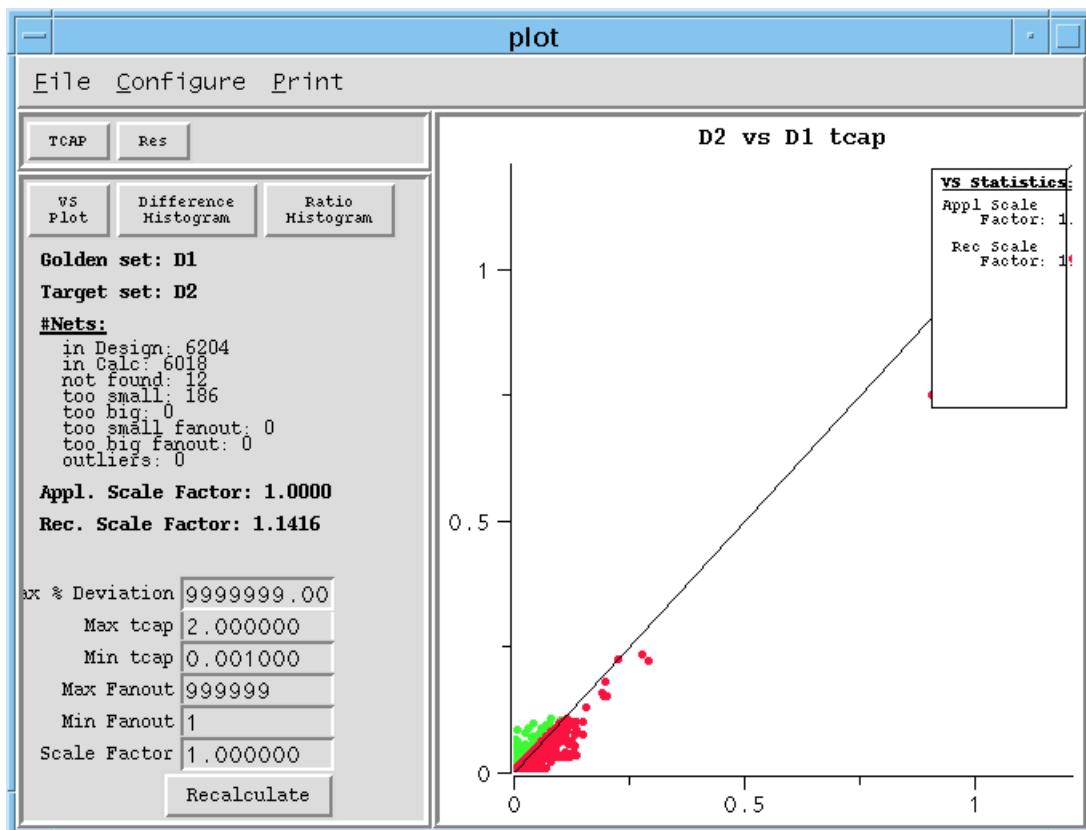


6. Select the *Golden Setname* and *Target Setname* corresponding to the sign-off SPEF, and native extraction SPEF respectively.
7. Select *Build TCAP plots*, *Build RES plots*, and *Build XCAP plots* options. Click on the *Build* button.

## Encounter User Guide

### RC Extraction

- Click on the Correlate - draw Plot - plot option in the Ostrich window. This opens the plot window.



The plot window displays the suggested scale factor.

## Comparing SPEF Files Using a Perl Script

Use the Perl script `spefCapCmp.pl` to compare the SPEF files generated by the `runQRC` and `extractRC` commands. You specify a range of total capacitance on a net to select the nets for comparison. This script resides in the `$ENCOUNTER/bin` directory. The Perl script generates the RC scaling factors for total capacitance.

If the capacitance scaling is outside the range of +/- 20% (0.8-1.2), you need to reevaluate the flow for possible mistakes during parasitics generation.

To run the Perl script, use the following command at the UNIX prompt:

```
spefCapCmp.pl -ref signoff_file_name.spef -cmp file_name.spef [-minCap value]
[-maxCap value] [-ex openNetFile]
```

**-ref** *signoff\_file\_name.spef*

Specifies the SPEF file generated by the `runQRC` command.

**-cmp** *file\_name.spef*

Specifies the SPEF file generated by the `extractDetailRC` command.

**-minCap** *value*

Specifies the minimum value, in picofarads, of the total capacitance on a net for the comparison script.

*Default:* 0.01

**-maxCap** *value*

Specifies the maximum value, in picofarads, of the total capacitance on a net.

*Default:* 5.0

**-ex** *openNetFile*

Specifies the name of the file that contains open nets to be excluded from the report file.

The Perl script generates the following output:

- Report file `spefCapCmp.rpt`, which contains the scaling factors and statistical data for total capacitance. For more information, see “[Report File Example](#)” on page 889.
- The `tcap.plot` file, which contains coordinates for plotting the comparison values for total capacitance.
- The `res.plot` file, which contains coordinates for plotting the comparison values for resistance.

To display a scatter plot of the comparison values, use the following command:

`xgraph -P -nl file_name.plot`

**Note:** `xgraph` is a public domain utility that you can download from the internet.

## Related Topics

- [Flat Implementation Flow](#) chapter in the *Encounter Flat Implementation Flow Guide*
  - [“Results”](#)

## Encounter User Guide

### RC Extraction

---

#### **Report File Example**

```
#Ref. Cap file:      lambda_aftercrosstalkfix.spf
#Cmp. Cap file:     lambda_detailrc.spf
#-----
#          Total Capacitance Statistics      #
#-----
#Total Capacitance range considered:      0.0100 -    5.0000 [pF]
#Total number of nets:                   418399
#Total number of nets with nonzero lumped Cap: 213095
#Total number of nets discarded (small lumped Cap): 152272.00
#Suggested Capacitance Scale Factor           0.9577 <---- (Ref. = FE(Cmp.)* 0.9577)
#Mean (Cap Scalar):                     1.04
#Total Sum of residual square:          22.89
#Variance:                            0.00
#Standard deviation:                  0.02
#Coefficient of variation:            1.90%
#Normal distribution range (sigma):   1.00 to 1.08
# -----
# Correlation results: #of Nets  %
# -----
# Nets [..., 0.1]:      0      0.00
# Nets [0.1, 0.2]:      0      0.00
# Nets [0.2, 0.3]:      0      0.00
# Nets [0.3, 0.4]:      0      0.00
# Nets [0.4, 0.5]:      0      0.00
# Nets [0.5, 0.6]:      0      0.00
# Nets [0.6, 0.7]:      0      0.00
# Nets [0.7, 0.8]:      0      0.00
# Nets [0.8, 0.9]:     800     0.38
# Nets [0.9, 1.0]:    20004    9.39
# Nets [1.0, 1.1]:    24729   11.60
# Nets [1.1, 1.2]:   10048    4.72
# Nets [1.2, 1.3]:   3224     1.51
# Nets [1.3, 1.4]:   1164     0.55
# Nets [1.4, 1.5]:   454      0.21
# Nets [1.5, 1.6]:   227      0.11
# Nets [1.6, 1.7]:   90       0.04
```

```
# Nets [1.7, 1.8]:      54      0.03
# Nets [1.8, 1.9]:      11      0.01
# Nets [1.9, 2.0]:      10      0.00
# Nets [2.0, ...]:      8       0.00
# Nets discarded : 152272    71.46
```

## Defining the Scaling Factor

You can specify the scaling factors in the following three ways:

- Change the technology file.

You can change the `ScaleFactor` in the technology file. This scaling is used for each technology.

- Store the value in the configuration file by using the following commands:

```
set rda_Input(ui_defcap_scale) "NUMBER"
set rda_Input(ui_detcap_scale) "NUMBER"
set rda_Input(ui_xcap_scale) "NUMBER"
set rda_Input(ui_defres_scale) "NUMBER"
set rda_Input(ui_detres_scale) "NUMBER"
```

- Use the following Tcl command:

```
setRCFactor -defcap scaleFactor -detcap scaleFactor -xcap scaleFactor
             -defres scaleFactor -detres scaleFactor
```

**Note:** When saving the design with the `saveDesign` command, the scale factors that are specified with the `setRCFactor` command are saved in the configuration file, which can be later restored.

For information on the `setRCFactor` command, see [setRCFactor](#) in the *Encounter Text Command Reference*.

The value specified in the Tcl command supersedes the value in the configuration command, which supersedes the value in the technology file. The default value for all scaling factors is 1.0.

## Sign-Off Extraction Using QRC

QRC standalone extraction is accessible through the Encounter software for generating detailed parasitics. You can perform sign-off extraction after running WRoute or NanoRoute.

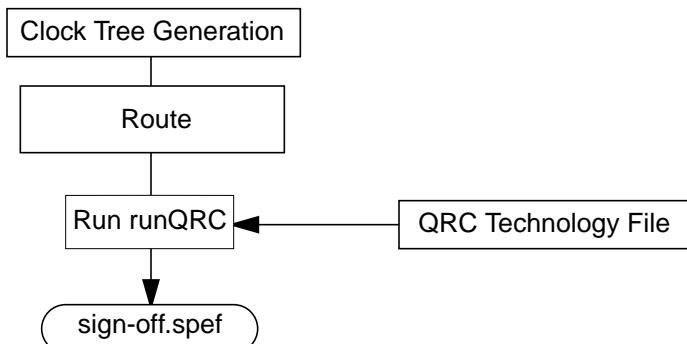
**Note:** QRC standalone extraction requires a separate license.

QRC can also be used with native extraction to generate an RC scaling factor. For more information, see [“Correlating Native Extraction With Sign-Off Extraction”](#) on page 883.

QRC standalone extraction uses a suite of 3-D models that are generated once per process. During extraction, QRC geometrically analyzes each conductor in all three dimensions, generates parameters based on specific 3-D regions, and then passes the parameters to the models for capacitance calculation.

The models use an influence region called a dynamic halo to account for all near-body and multi-level interconnect capacitive effects, including the impact of crossover fringe, corners, and capacitive shading. The 3-D extraction techniques do not rely on pattern-matching techniques, so they do not produce boundary errors. This type of modeling extracts and maintains both lumped and fully distributed net-to-net coupled capacitance.

The following figure shows the sign-off extraction flow.



You can use the [Extraction Setup](#) form located in the Encounter *Timing* menu for sign-off extraction. You can access this form by choosing *Timing – QRC Standalone Extraction* from the Encounter menu.

## Inputs for QRC Sign-Off Extraction

QRC standalone extraction is a DEF-based flow. When you perform sign-off extraction through the Encounter interface, a routed DEF will automatically be created. The following inputs are required before you can start sign-off extraction:

- Design Exchange Format (DEF) file—Contains the design-specific information of a circuit and is a representation of the design at any point during the layout process.
- QRC technology file—Contains the interconnect resistance and capacitance data for generating RC values and wireload models for the design. It also contains the process information for the metal interconnect layers, including metal thickness, metal resistance, and line-to-line capacitance values of metal layers for determining coupling capacitance.
- LEF files—Contains the relevant cell content that includes standard cells and macros from the design import configuration file.
- QRC command file—Contains commands and variables that define the extraction environment (technology filename, library name, and so on), specifies the net(s) to extract and how to extract them, controls the resistance and capacitance extraction, and specifies the extraction outputs.

---

## Calculating Delay

---

- [Overview](#) on page 894
- [Data Preparation](#) on page 895
  - [Operating Conditions](#) on page 895
  - [ECSM Libraries](#) on page 895
- [Delay Calculation Modes and Related Controls](#) on page 896
- [Choosing A Delay Calculation Engine](#) on page 897
- [Running Delay Calculation](#) on page 897

## Overview

You can perform delay calculation at various stages of the design flow to continuously validate your timing. Encounter provides two delay calculator engines that you can use:

- **Encounter<sup>TM</sup>** delay calculator engine
  - Used to provide quick delay calculation for design prototyping, optimization, and general timing analysis.
- **SignalStorm<sup>®</sup>** delay calculator
  - Used to perform final delay calculation and timing analysis. SignalStorm provides better accuracy, can calculate multi-driver nets, and uses more advanced effective current source model (ESCM) modeling.

## Data Preparation

In order to perform delay calculation, you must have a placed design loaded in Encounter.

Delay calculation uses information from the following data files:

- A DSPF-format or SPEF-format netlist that contains the detailed parasitic resistance and capacitance of the interconnect, and the gates that are used to drive this interconnect.
- .lib file (Timing information)
- Timing constraints file
- LEF file

Delay calculation generates the following information:

- Optionally, an SDF file, which provides delay information for instances and RC interconnect.

## Operating Conditions

Encounter does not automatically import operating conditions from the SDC constraints. Therefore, you must ensure that operating conditions are specified before running delay calculation. Use the `setOpCond` commands to specify the minimum and maximum libraries, and operating conditions.

## ECSM Libraries

By default, Encounter supports ECSM-based Liberty (.lib) timing libraries for performing delay calculation.

## Delay Calculation Modes and Related Controls

Delays are calculated differently, depending on the number of terminals (fanouts) a net has and the Elmore time constant. The following table lists the calculation modes and related controls used by the Encounter software for delay calculation.

	<b>Fanouts &gt;= 1,000</b>	<b>1,000 &gt; Fanouts &gt;= 100</b>	<b>100 &gt; Fanouts and Elmore &gt; 5 ps</b>	<b>5 ps &gt; Elmore</b>
<b>Algorithm</b>	Uses the default delay parameters.	Uses simplified delay calculation mode.	Uses full RC delay calculation mode.	Uses simplified delay calculation mode.
<b>Cell Delay</b>	Uses lumped C lookup with a default value of 0.5 pF.  The value is controlled by the <code>setDefaultNetLoad</code> command.	Uses lumped C lookup from total parasitics on the net.	Uses full RC.	Uses lumped C lookup from total parasitics on the net.
<b>Wire Delay</b>	Uses the default value of 1 ns.  The value is controlled by the <code>setDefaultNetDelay</code> command.	Uses Elmore delay.	Uses full RC.	Uses Elmore delay.
<b>Driver Slew Rates</b>	Uses the default value of 0 ps.  The value is controlled by the <code>setInputTransitionDelay</code> command.	Uses lumped C lookup from total parasitics on the net.	Uses full RC.	Uses lumped C lookup from total parasitics on the net.
<b>Interconnect Slew Degradation</b>	None	None	Uses full RC. (Slews are degraded across interconnect.)	None
<b>Controls</b>	Fanout threshold is controlled by the <code>setUseDefaultDelayLimit</code> command. The default value is 1,000 fanouts.	Fanout threshold is controlled by the <code>-elmore</code> option in the <code>buildTimingGraph</code> command. The default value is 100 fanouts.		No control for minimum Elmore threshold.

## Choosing A Delay Calculation Engine

1. Choose *Timing – Specify Analysis Condition – Specify Delay Calculation Mode*.  
The Delay Calculation Mode form appears.
2. Select *FE-DC* to use the Encounter delay calculator, or *SignalStorm* to use the SignalStorm delay calculator.

Alternatively, you can specify the delay calculation engine by issuing the `setDelayCalMode` text command:

- Issue the following command to specify the Encounter delay calculator:  
`setDelayCalMode -engine feDC`
- Issue the following command to specify the SignalStorm delay calculator:  
`setDelayCalMode -engine signalStorm`

**Note:** The default delay calculator is Encounter (`feDC`).

## Running Delay Calculation

The delay calculation engine set using the `-engine` parameter of the `setDelayCalMode` command is called automatically during timing analysis. You can choose to write the delays to a standard delay format (SDF) file using the `write_sdf` command.

For example, the following command saves the results to the SDF file named `TOPCHIP_SP.sdf`:

```
write_sdf TOPCHIP_SP.sdf
```

## **Encounter User Guide**

### Calculating Delay

---

---

## Timing Analysis

---

- [Overview](#) on page 900
- [Timing Analysis Features](#) on page 901
- [Before You Begin](#) on page 902
- [Reading Timing Libraries](#) on page 903
- [Reading Timing Constraints](#) on page 904
- [Timing Analysis Results](#) on page 906
- [Setting Operating Conditions](#) on page 907
- [Calculating Clock Latency](#) on page 908
- [Defining RC Corners](#) on page 909
- [Specifying Timing Analysis Modes](#) on page 911
- [Clock Path Pessimism Removal](#) on page 927
- [Analyzing Timing Problems](#) on page 933

## Overview

The goal of timing analysis is to verify that a design meets timing requirements under a specified set of timing constraints, such as arrival and required times, operating conditions, slew rates, false paths, and path delays. Performing timing analysis lets you determine how fast a design can run without incurring timing violations. You can use the results of timing analysis to fine tune and debug the speed-limiting, critical paths in a design.

You can perform timing analysis using Cadence® and Synopsys constraint formats and timing libraries .lib, and Stamp Models.

## Timing Analysis Features

Timing analysis includes the following features and capabilities:

### Static Timing Analyzer (STA)

- Performs setup time analysis, which analyzes violating paths due to timing
- Performs hold time analysis
- Performs analysis in ideal and propagated mode
- Reports asynchronous violating paths
- Reports violating paths after running pre-clock tree synthesis (CTS) skew

### What-If Timing Analysis

Use what-if timing analysis to modify instance cell timing information to reach top level timing requirements, after which you can manually change the timing model of a standard cell or modify the timing arcs of blackboxes or blackblobs. Once you have defined the initial timing model of the blackboxes or blackblobs, you can modify arc definitions and verify the consequences in timing analysis.

### Wireload Model Generation in Hierarchical and Flat Format

The delay information in the technology library applies to the timing arcs from input ports to output ports of each cell and the corresponding wire delays. The cell delays and the wire delays are expressed as a function of the physical characteristics of the nets in the design, such as wire capacitance and wire resistance. A wireload model uses the fanout count of a net and estimates its capacitance and resistance. The wireload models in the hierarchical format are generated for cells and instances based on external nets (hierarchical view). The wireload models in the flat format are generated for cells and instances based on internal nets (flat view).

### Minimum and Maximum Timing Analysis

To read in libraries with multiple operating conditions for minimum and maximum analysis, you can:

- Define the libraries in the configuration file or in the Encounter GUI
- Specify the `setOpCond` command

- Specify the `setTimingLibrary` command (optional)
- Specify the `setAnalysisMode` command

### Timing Analysis Ideal and Propagated Modes

<b>setAnalysisMode</b>	<b>Clock Propagation</b>	<b>Clock Latency</b>
-skew false	Forced Ideal	No Effect
-skew true    -clockPropagation forcedIdeal	Forced Ideal	SDCs in Effect
-skew true    -clockPropagation sdcControl	*SDCs in Effect	**SDCs in Effect

\* Both `-clockPropagation sdcControl` and `set_propagated_clock` required.

\*\* The closest (`set_clock_latency` or `set_propagated_clock`) assertion to the clock endpoint determines ideal vs. propagated mode.

## Before You Begin

Before running timing analysis, read in the timing libraries, timing constraints, and the netlist.

Optionally, you can also set the following conditions:

- Specify the delay calculation and RC extraction data.  
Use the *Timing* and *Power* pages in the Design Import form to specify these values. For more information, see “[Design Import – Timing](#)” in the *Encounter Menu Reference*.
- Specify the operating conditions to use for timing analysis  
Use the operating conditions to specify process, voltage, and temperature (PVT) values. Operating conditions are defined in the timing library and read into the Encounter session when you import the design. You can use a single set of operating conditions for setup and hold analysis, or you can specify minimum and maximum conditions.  
For more information, see “[Setting Operating Conditions](#)” on page 907.
- Check and report timing libraries by generating the timing library report.
- Check and report cell footprints by generating the cell footprint report.

- Define RC corners for extraction. You can set the RC corner as best, typical, or worst.
- Specify the analysis mode you want to use for timing analysis. There are three types of analysis modes: single, best-case worst-case (BC-WC), and on-chip variation. For more information, see [“Specifying Timing Analysis Modes”](#) on page 911.

## Reading Timing Libraries

The timing library contains the timing models provided by the ASIC or intellectual property (IP) vendors. You can read in the library files while importing the design using the [Design Import](#) menu command.

You can use the Design – Design Import form to specify the timing libraries. If you enter information in only one of the *Max Timing Libraries*, *Min Timing Libraries*, or *Common Timing Libraries* fields, Encounter runs single value analysis: Both setup and hold analysis use the libraries entered in that field. If you provide values for both *Max Timing Libraries* and *Min Timing Libraries*, Encounter uses these libraries for both BcWc (default) or OCV analysis.

If you read in both Min and Max library groups, the software uses Max library delay values for max path analysis and Min library delay values for min path analysis. To change this behavior, you can set the `-max` and `-min` parameters of the `setTimingLibrary` command to the same group.

For example, to run setup and hold analysis using Max Timing Libraries, use the following command:

```
setTimingLibrary -max Max -min Max
```

## Resolving Discrepancies in Timing Libraries

By default, the time and capacitance unit values for the design are set using the values specified in the timing libraries, when the libraries are loaded into the design. If the values in the timing libraries differ, the software generates warning messages and sets the value to internal default setting of 1ns for time units and 1pF for capacitance units.

To change the unit definitions being used by the software, use the [setLibraryUnit](#) command or the [Design Import - Advanced - Timing](#) form after importing the design.

## Reading Timing Constraints

To ensure that your design meets the timing requirements, you must specify what the requirements are by setting the constraints. You can use the timing constraints to set:

- Timing context for constraint assertions.
- Boundary conditions such as input and output delays.
- Slew rates
- Path exceptions such as false paths, path delays and cycle additions.
- Disable certain paths in the design.

## Constraints Quick Reference

The following constraints are supported:

- `add_to_collection`
- `all_clocks`
- `all_inputs`
- `all_outputs`
- `all_registers`
- `create_clock`
- `create_generated_clock`
- `current_design`
- `find`
- `foreach_in_collection`
- `get_cells`
- `get_clocks`
- `get_designs`
- `get_lib_cells`
- `get_lib_pins`
- `get_nets`

## Encounter User Guide

### Timing Analysis

---

- `get_libs`
- `get_pins`
- `get_ports`
- `group_path`
- `remove_from_collection`
- `reset_propagated_clock`
- `set_annotated_check`
- `set_annotated_delay`
- `set_annotated_transition`
- `set_case_analysis`
- `set_clock_gating_check`
- `set_clock_latency`
- `set_clock_transition`
- `set_clock_uncertainty`
- `set_disable_clock_gating_check`
- `set_disable_timing`
- `set_dont_touch`

Use the `set_dont_touch` constraint to constrain nets, instances, and cells. If the `set_dont_touch` constraint in the timing constraints file is overly constrained, running timing optimization does not correct the timing because timing optimization honors the constraint.

- `set_dont_touch_network`
- `set_dont_use`
- `set_drive`
- `set_driving_cell`
- `set_false_path`
- `set_fanout_load`
- `set_input_delay`

- `set_input_transition`
- `set_load`
- `set_logic_one`
- `set_logic_zero`
- `set_max_capacitance`
- `set_max_delay`
- `set_max_fanout`
- `set_max_time_borrow`
- `set_max_transition`
- `set_min_delay`
- `set_min_pulse_width`
- `set_multicycle_path`
- `set_output_delay`
- `set_propagated_clock`
- `set_resistance`
- `source`

## Timing Analysis Results

After performing timing analysis, you can identify all timing violations and slacks. You can also identify all of the critical paths in combinatorial or clocked circuits at both block and chip level.

You can use the `timeDesign` command to generate timing summary reports at different stages in the implementation flow. You can also use reporting commands such as `report_timing` to perform detailed analysis.

## Setting Operating Conditions

Integrated circuits display performance differences depending on the fabrication and environmental process, voltage and temperature (PVT) characteristics. The nominal values for these parameters and the corresponding delay information are contained in the library. Each timing library contains one or more operating conditions. Each operating condition is identified by name and specifies the PVT parameters. This information is used to calculate accurate cell delays from the nominal cell delays and the derating factors.

You can use a single set of operating conditions for setup and hold analysis, or you can specify minimum and maximum conditions.

In Encounter, you can specify the operating conditions using the setOpCond command or the Specify Operating Condition menu command.

Depending on the design requirements, you can use the following parameters of the `setOpCond` command:

- `-library`

Sets the operating condition from a single library. You can define a single operating condition from a library, and the software uses the PVT characteristics associated with that library for every library in the design. When you use this parameter in multiple `setOpCond` commands, the software uses the value specified in the last command. For example, consider the following commands:

```
setOpCond WCCOM -library wcTestLib  
setOpCond BCCOM -library bcTestLib  
setOpCond WCCOM -library wcTestLib1  
setOpCond BCCOM -library bcTestLib1  
setOpCond -library wcTestLib2  
setOpCond -library bcTestLib2
```

In this case, the software uses the last value defined (`bcTestLib2`). Therefore, the operating conditions in the `bcTestLib2` library are used for both setup and hold analysis.

- `-min minOpCond [-minLibrary minLib]`  
`-max maxOpCond [-maxLibrary maxLib]`

Specifies the operating condition used for hold or setup analysis. For example, consider the following command:

```
setOpCond -max worst -maxLibrary wcTestLib -min best -minLibrary bcTestLib
```

In this case, the software uses the `worst` operating condition defined in `wcTestLib` for max path analysis and the `best` operating condition defined in `bcTestLib` for min path analysis.

## Calculating Clock Latency

The Encounter software calculates clock latency based on the following two settings:

- Analysis mode set using the `setAnalysisMode` command.
- The `set_propagated_clock` and `set_clock_latency` constraints values.

Depending on these settings, the clock latency can be equal to either 0.0 or the value of the `set_clock_latency` constraint, or the delay computed by propagation along the clock path.

The Encounter software sets the clock latency for various combinations of analysis mode settings as follows:

- `setAnalysisMode -skew true -clockPropagation sdcControl` (**Default Setting**)
  - Latency is defined by the precedence of `set_propagated_clock` and `set_clock_latency` in the SDC.
  - If both `set_propagated_clock` and `set_clock_latency` are not specified, no clock latency is reported (ideal mode).
- `setAnalysisMode -skew true -clockPropagation forcedIdeal`
  - If `set_clock_latency` command is in the timing constraint file, the clock latency specified in the constraint is used (ideal mode).
  - If `set_clock_latency` is not specified, 0ns clock latency is reported (ideal mode).

**Note:** The `-clockPropagation forcedIdeal` option forces ideal clock mode, even if `set_propagated_clock` is specified in the constraints file.

- `setAnalysisMode -skew false -clockPropagation sdcControl`

Or,

`setAnalysisMode -skew false -clockPropagation forcedIdeal`

- No latency is reported (ideal mode).

**Note:** When you use the `-skew false` parameter, clock latencies are ignored.

## Defining RC Corners

Before performing timing analysis, you can define which process corners (best, worst, or typical) will be used for min and max path analysis.

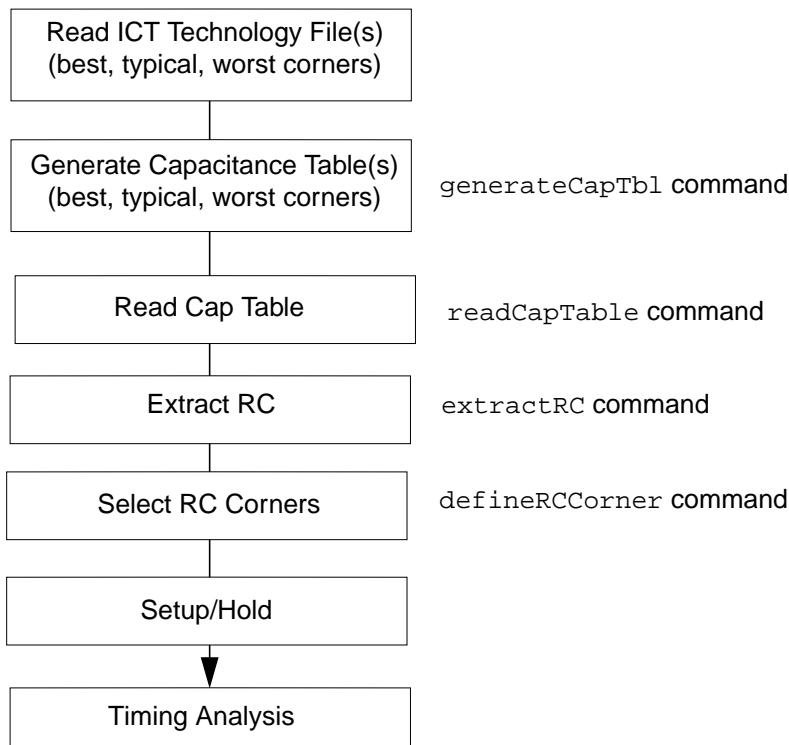
The procedure for defining process corners is as follows:

1. Generate an ICT file for each process corner. For an example ICT file, see Appendix A “[Creating the ICT File](#)”, in the *Encounter User Guide*.
2. Generate three separate capacitance tables. For more information on generating capacitance tables, see “[Capacitance Table Generation Flow](#)”, in the *Encounter User Guide*.
3. Use the `defineRCCorner` command to define the process corner for setup and hold checks. The following example shows the use of the `defineRCCorner` command:

```
defineRCCorner -late {best | typical | worst} temp  
-early {best | typical | worst} temp
```

Best, typical, and worst correspond to the ICT files and the capacitance tables generated with the `genCapTbl` command.

The following figure shows the flow for extracting parasitic and defining RC corners for timing analysis.



## **Encounter User Guide**

### Timing Analysis

---

## Specifying Timing Analysis Modes

The Encounter software provides different timing analysis modes and performs different calculations for setup and hold checks for each mode. The timing analysis modes are divided as follows:

- [Single Timing Analysis Mode](#) on page 913

In single analysis mode, only maximum delay values and -max options of constraints are used for both min and max analysis.

- [Best-Case Worst-Case \(BC-WC\) Timing Analysis Mode](#) on page 917

Minimum delays from BC and max delays from WC should not be used together to evaluate a timing check because the BC and WC operating conditions or corners are vastly different.

In BC-WC analysis mode the software uses the maximum delays for all paths during setup checks and minimum delays for all paths during hold checks.

- [On-Chip Variation \(OCV\) Timing Analysis Mode](#) on page 922

OCV is the small difference in the operating parameter value across the chip. Each timing arc in the design can have an early and a late delay to account for the on-chip process, voltage and temperature variation. These delays are used together in the analysis of each check.

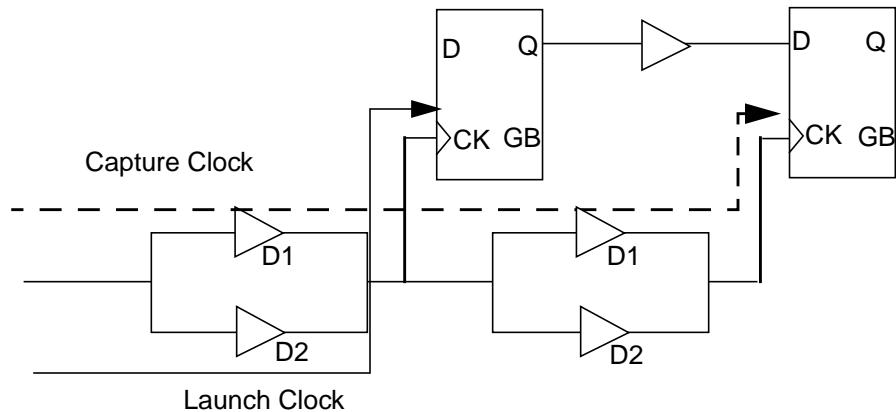
In OCV mode, the software calculates clock and data path delays based on minimum and maximum operating conditions for setup analysis and vice-versa for hold analysis.

### Definition of Early and Late Paths

The timing analysis modes described in this section refer to early and late paths and their usage in slack calculation. The early and late paths are the shortest and the longest paths respectively.

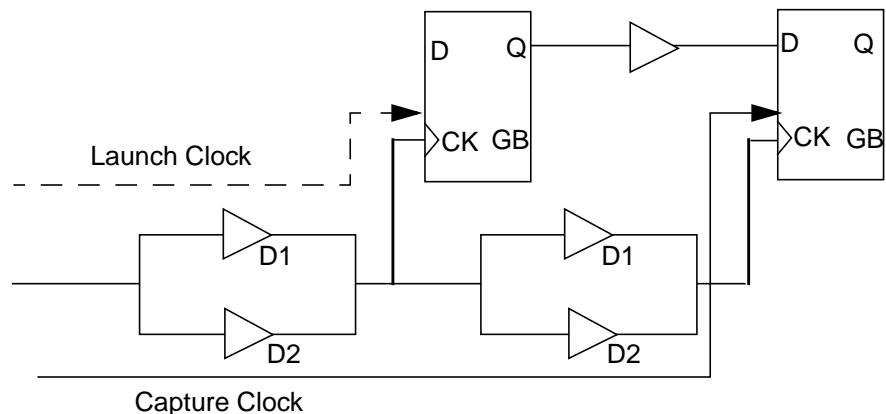
The following figure shows a setup check with late (shown in solid line) launch clock and early capture clock (shown in dotted line).

**Figure 26-1 Setup Check**



The following figure shows hold check with early launch clock (shown in dotted line), and late capture clock (shown in solid line).

**Figure 26-2 Hold Check**



## **Single Timing Analysis Mode**

In this mode, the Encounter software uses a single set of delays (using one library group) based on one set of process, temperature and voltage conditions. In this mode, you specify one operating condition using the `setOpCond` command.

To specify the operating condition with one library group, specify the following command:

```
setOpcond opcondname -library libname
```

To set the timing analysis mode as single, use the `-analysisType single` parameter of the `setAnalysisMode` command.

### **Setup Check in Single Timing Analysis Mode**

For setup check, the software checks the late launch clock and late data paths against early capture clock path.

For zero slack value in a setup check, the following condition should be met:

$$\text{launch clock late path} + \text{data clock late path} \leq \text{capture clock early path} + \text{clock period} - \text{setup}$$

### **Hold Check in Single Timing Analysis Mode**

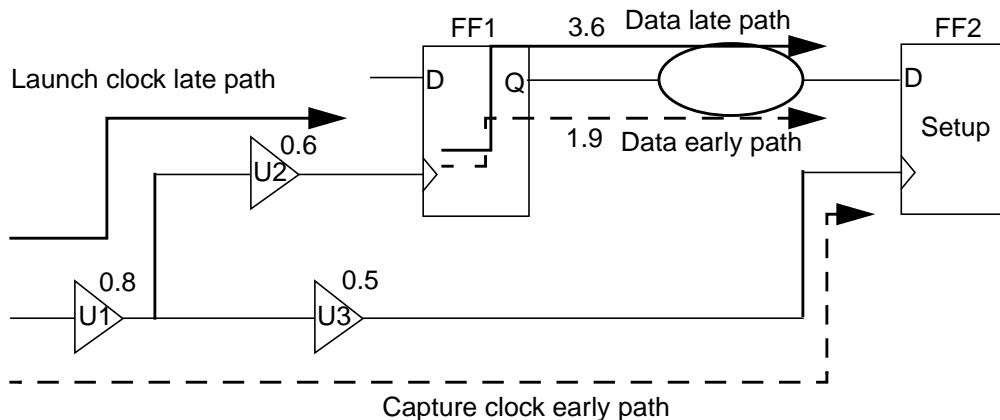
For hold check the software compares the early arriving data against the late arriving clock at the endpoint.

For zero slack value in a hold check, the following condition should be met:

$$\text{launch clock early path} + \text{data clock early path} \geq \text{capture clock late path} + \text{hold}$$

### Example 26-1 Setup Check in Single Timing Analysis Mode

The following figure shows the setup check on the path from FF1 to FF2.



The software uses a library to scale all delays at WC conditions. For setup check, the software considers two paths between the two registers, FF1 and FF2. The software considers only the late path delay to calculate slack during setup check.

The following values are assumed in this example:

Data late path delay = 3.6  
 Data early path delay = 1.9  
 Clock source latency = none  
 Wire delay = 0  
 Clock period = 4  
 Clock mode = Propagated clock mode

The software computes the slack as follows:

$$\text{Launch clock late path delay} = 0.8 + 0.6 = 1.4$$

$$\text{Data late path delay} = 3.6$$

$$\text{Capture clock early path delay} = 0.8 + 0.5 = 1.3$$

$$\text{Setup} = 0.2$$

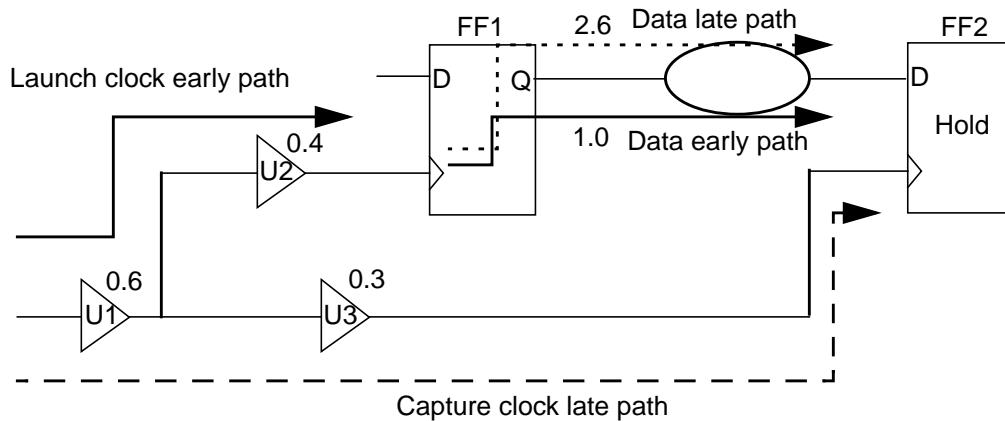
$$\text{Data arrival time} = 1.4 + 3.6 = 5$$

$$\text{Data required time} = 4 + 1.3 - 0.2 = 5.1$$

$$\text{Slack} = 5.1 - 5 = 0.1$$

### Example 26-2 Hold Check in Single Timing Analysis Mode

The following figure shows the hold check on the path from FF1 to FF2.



The software uses a library to scale all delays at BC conditions.

The following values are assumed in this example:

Clock source latency = none  
 Wire delay = 0  
 Clock period = 4  
 Clock Mode = Propagated clock mode

The software computes the slack as follows:

$$\text{Launch clock early path delay} = 0.6 + 0.4 = 1.0$$

$$\text{Data early path delay} = 1.0$$

$$\text{Capture clock late path delay} = 0.6 + 0.3 = 0.9$$

$$\text{Hold} = 0.1$$

$$\text{Data arrival time} = 1 + 1 = 2$$

$$\text{Data Required Time} = 0.1 + 0.9 = 1$$

$$\text{Slack} = 2 - 1 = 1$$

## Performing Timing Analysis in Single Analysis Mode

1. Load in the library file by sourcing the configuration file with the following line:

```
set rda_Input(ui_timelib,max) "${libDir}/stdcell.lib"
```

You can also load the library using the *Design Import* menu command.

2. Set the timing libraries to be used during setup or hold.

```
setTimingLibrary -max Max -min Max
```

**Note:** This step is required if you read in more than one library.

3. Set the operating condition for setup analysis.

```
setOpCond worst -library stdcell
```

You can also use the library defaults by not specifying any `setOpCond` or by specifying `setOpCond {}`.

4. Set the analysis mode to single, setup and propagated clock mode.

```
setAnalysisMode -analysisType single -checkType setup -skew true  
-clockPropagation sdccontrol
```

5. Generate the timing reports for setup.

```
report_timing
```

6. Set the operating condition for hold analysis.

```
setOpCond best -library stdcell
```

7. Set the analysis mode to hold and propagated clock mode.

```
setAnalysisMode -checkType hold -skew true -clockPropagation sdcControl
```

8. Generate the timing reports for hold.

```
report_timing
```

## **Best-Case Worst-Case (BC-WC) Timing Analysis Mode**

In BC-WC timing analysis mode, the Encounter software considers two operating conditions. The software checks both operating conditions in one timing analysis run.

To set the timing analysis mode as BC-WC, use the `-analysisType bcWc` parameter of the `setAnalysisMode` command.

You can use the `set_clock_latency` constraint to set the source latency for a clock in both ideal and propagated mode for setup and hold checks. You can also use the constraint to set the network latency for an ideal clock. The specified source or network latency affects the early and late clock paths for both capture and launch clocks for both min and max operating conditions. The software considers the network latency that you set using the `set_clock_latency -max` or `-min` constraint for ideal clocks only.

### **Setup Check in BC-WC Mode**

For setup check, the software calculates delay values from the Max library group for data arrival time, and network delay of both launch and capture clocks (in propagated mode).

The software scales the delay values using the operating condition that you specify using the `setOpCond -max` command. To define the operating condition, use the following command:

```
setOpCond -max WCopcondname -maxLibrary WClibname
```

The source latency in both ideal and propagated modes for setup checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock late path (max)	<code>set_clock_latency -source -late -max value</code>
Capture clock early path (max)	<code>set_clock_latency -source -early -max value</code>

The network latency in ideal mode for setup checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock late path (max)	<code>set_clock_latency -max value</code>
Capture clock early path (max)	<code>set_clock_latency -max value</code>

## HOLD Check in BC-WC Mode

For HOLD check, the software uses the delay values from the Min library for the data arrival time, and network delay of both launch and capture clocks (in propagated mode). The software scales the delay values using the operating condition that you specify using the `setOpCond -min` command. To specify the operating condition, use the following command:

```
setOpCond -min minopcondname -minLibrray minlibname
```

Clock Path (Operating Condition)	Constraint Used
Launch clock early path (min)	<code>set_clock_latency -source -early -min value</code>
Capture clock late path (min)	<code>set_clock_latency -source -late -min value</code>

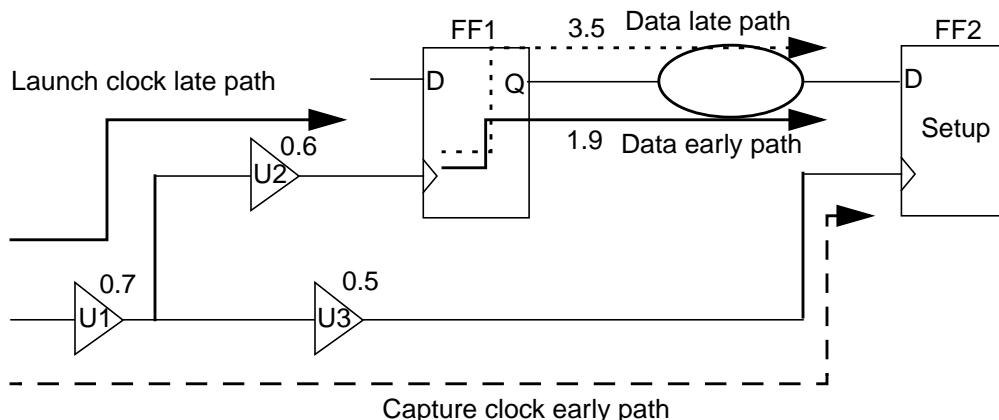
The network latency in ideal mode for hold checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock early path (min)	<code>set_clock_latency -min value</code>
Capture clock late path (min)	<code>set_clock_latency -min value</code>

**Note:** You can also use one library containing two operating conditions in this mode.

## Example 26-3 Setup Check in BC-WC Timing Analysis Mode

The following shows the setup check on the path from FF1 to FF2.



## Encounter User Guide

### Timing Analysis

The software uses the Max library to scale all delays at WC conditions.

The following values are assumed in this example:

Clock source latency = none

Wire delay = 0

Clock period = 4

Clock Mode = Propagated clock mode

The software computes the slack as follows:

Launch clock late path delay =  $0.7 + 0.6 = 1.3$

Data late path delay = 3.5

Capture clock early path delay =  $0.7 + 0.5 = 1.2$

Setup = 0.2

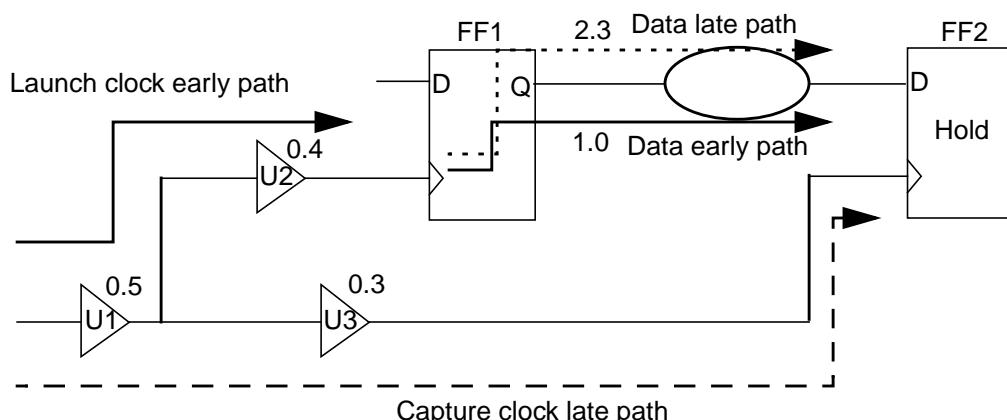
Data arrival time =  $1.3 + 3.5 = 4.8$

Data required time =  $4 + 1.2 - 0.2 = 5$

Slack =  $5 - 4.8 = 0.2$

### Example 26-4 Hold Check in BC-WC Timing Analysis Mode

The following figure shows the hold check on the path from FF1 to FF2.



The software uses the Min library to scale all delays at BC conditions.

## Encounter User Guide

### Timing Analysis

---

The following values are assumed in this example:

Clock source latency = none

Wire delay = 0

Clock period = 4

Clock Mode = Propagated clock mode

The software computes the slack as follows:

Launch clock early path delay =  $0.5 + 0.4 = 0.9$

Data early path delay = 1.0

Capture clock late path delay =  $0.3 + 0.5 = 0.8$

Hold = 0.1

Data arrival time =  $0.9 + 1 = 1.9$

Data required time =  $0.1 + 0.8 = 0.9$

Slack =  $1.9 - 0.9 = 1$

## Performing Timing Analysis in BC-WC Analysis Mode

To perform timing analysis in BC-WC analysis mode, complete the following steps:

1. Read in the min and max libraries by sourcing the configuration file with the following lines:

```
set rda_Input(ui_timelib,max) "${libDir}/stdcellwst.lib"  
set rda_Input(ui_timelib,min) "${libDir}/stdcellbest.lib"
```

You can also load the library using the *Design Import* menu command.

2. Specify BC-WC operating conditions to be used during setup and hold.

```
setOpCond -max wccom-maxLibrary stdcellwst -min bccom -minLibrary stdcellbest
```

You can also use the software defaults by not specifying any operating conditions

3. Set the analysis mode to BC-WC, setup and propagated clock mode.

```
setAnalysisMode -analysisType bcWc -checkType setup -skew true  
-clockPropagation sdcControl
```

4. Generate the timing reports for setup.

```
report_timing
```

## **Encounter User Guide**

### Timing Analysis

---

5. Set the analysis mode to hold and propagated clock mode.

```
setAnalysisMode -checkType hold
```

6. Generate the timing reports for hold.

```
report_timing
```

## On-Chip Variation (OCV) Timing Analysis Mode

### **Related Topics**

To see this step in the design flow, see [Route the Design and Run Postroute Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

### **Setup Check**

In OCV mode, setup check the software uses the timing delay values from the Max library group for the data and the launch clock network delay. The software uses the delay values from the Min library group for the capturing clock network delay assuming that the clocks are set in propagated mode.

**Note:** You can also use one library instead of max and min libraries.

The source latency in both ideal and propagated modes for setup checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock late path (max)	<code>set_clock_latency -source -late -max value</code> Or, <code>set_clock_latency -source -late value</code>
Capture clock early path (min)	<code>set_clock_latency -source -early -min value</code> Or, <code>set_clock_latency -source -early value</code>

The network latency in ideal mode for setup checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock late path	<code>set_clock_latency -max value</code>
Capture clock early path	<code>set_clock_latency -min value</code>

## **Hold Check**

In OCV hold check, the software uses the timing delay values from the Min library for the data arrival time and launch clock network delay. The software uses delay values from the Max library for the capturing clock network delay assuming that the clocks are set in propagated mode.

The source latency in both ideal and propagated modes for hold checks is defined in the constraints used by various clock paths as follows:

Clock Path (Operating Condition)	Constraint Used
Launch clock early path (min)	<code>set_clock_latency -source -early -minvalue</code> Or, <code>set_clock_latency -source -early value</code>
Capture clock late path (max)	<code>set_clock_latency -source -late -maxvalue</code> Or, <code>set_clock_latency -source -late value</code>

The network latency in ideal mode for hold checks is defined in the constraints used by various clock paths as follows:

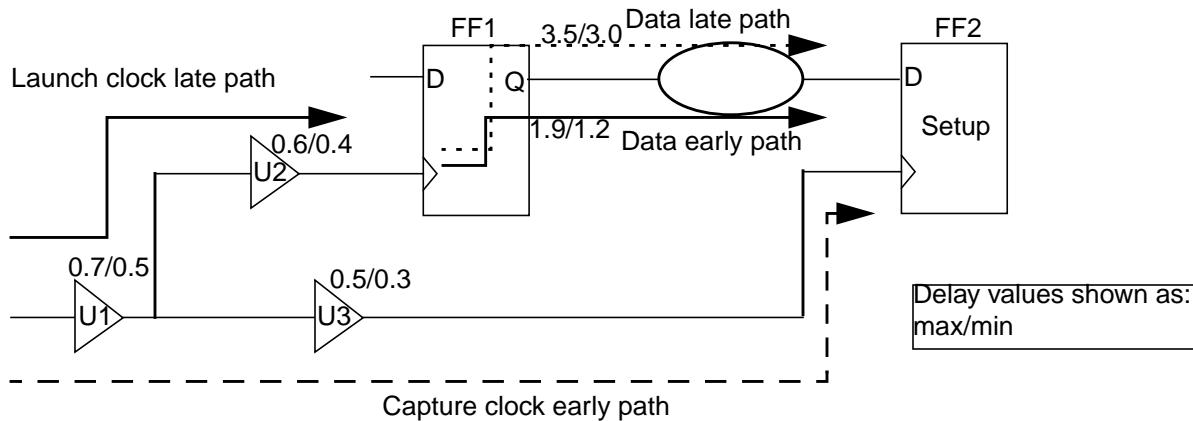
Clock Path (Operating Condition)	Constraint Used
Launch clock early path	<code>set_clock_latency -minvalue</code>
Capture clock late path	<code>set_clock_latency -maxvalue</code>

## **Example 26-5 Setup Check in OCV Timing Analysis Mode**

## Encounter User Guide

### Timing Analysis

The following figure shows the setup check on the path from FF1 to FF2.



The software uses the Max library for all late path delays and Min library for all early path delays.

The following values are assumed in this example:

Clock source latency = none  
Wire delay = 0  
Clock period = 4  
Clock mode = Propagated clock mode

The software computes the slack as follows:

$$\text{Launch clock late path delay (max)} = 0.7 + 0.6 = 1.3$$

$$\text{Data late path delay (max)} = 3.5$$

$$\text{Capture clock early path delay (min)} = 0.5 + 0.3 = 0.8$$

$$\text{Setup} = 0.2$$

$$\text{Data arrival time} = 1.3 + 3.5 = 4.8$$

$$\text{Data required time} = 4 + 0.8 - 0.2 = 4.6$$

$$\text{Slack} = 4.6 - 4.8 = -0.2$$

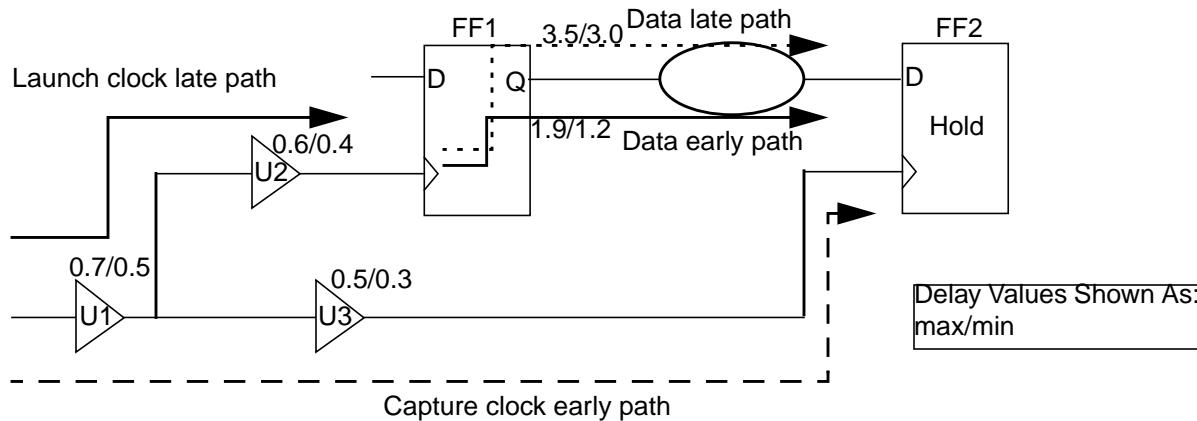
### Example 26-6 Hold Check in OCV Timing Analysis Mode

## Encounter User Guide

### Timing Analysis

---

The following figure shows the hold check on the path from FF1 to FF2.



The software uses the Max library to scale all delays at WC conditions and Min library to scale all delays at BC conditions.

The following values are assumed in this example:

Clock source latency = none  
 Wire delay = 0  
 Clock period = 4  
 Clock mode = Propagated clock mode

The software computes the slack as follows:

$$\text{Launch clock early path delay (min)} = 0.5 + 0.4 = 0.9$$

$$\text{Data early path delay (min)} = 1.2$$

$$\text{Capture clock late path delay (max)} = 0.7 + 0.5 = 1.2$$

$$\text{Hold} = 0.1$$

$$\text{Data arrival time} = 0.9 + 1.2 = 2.1$$

$$\text{Data required time} = 0.1 + 1.2 = 1.3$$

$$\text{Slack} = 2.1 - 1.3 = 0.8$$

## Performing Timing Analysis in OCV Mode with Two Libraries And Operating Conditions

1. Read in the min and max libraries by sourcing the configuration file with the following lines:

```
set rda_Input(ui_timelib,max) "${libDir}/slow.lib"
set rda_Input(ui_timelib,min) "${libDir}/fast.lib"
```

You can also load the library using the *Design Import* menu command.

2. Specify OCV operating conditions to be used during setup and hold.

```
setOpCond -min best -max worst -minLibrary fast -maxLibrary slow
```

3. Set the analysis mode to OCV and propagated clock mode.

```
setAnalysisMode -analysisType onChipVariation -skew true
    -clockPropagation sdcControl
```

4. Generate the timing reports for setup.

```
report_timing -late
```

5. Generate the timing reports for hold.

```
report_timing -early
```

## Using `set_timing_derate` with OCV Analysis Mode

The `set_timing_derate` command affect the following paths in OCV mode:

Violations	Data	Launch Clock	Capture Clock
SETUP	-late	-late	-early
	-data	-clock	-clock
HOLD	-early	-early	-late
	-data	-clock	-clock

## Clock Path Pessimism Removal

Clock Path Pessimism Removal (CPPR) or clock reconvergence pessimism removal (CRPR) is the process of identifying and removing the pessimism introduced in the slack reports for clock paths when the clock paths have a segment in common.

You can introduce early or late delay variations by using the `setAnalysisMode -analysisType onChipVariation` or by using derating factors in BcWc analysis mode. In CPPR mode, the difference between late and early delays is removed for common clock tree segments of the launching and latching devices.

When you use the `setAnalysisMode -analysisType BcWc` and the `set_timing_derate` commands, the software calculates maximum delay value by multiplying the delay by the scale value that you set using `set_timing_derate -late`. Similarly, the software calculates the minimum delay value by multiplying the delay by the scale value that you set using `set_timing_derate -early`.

When you use the `setAnalysisMode -analysisType onChipVariation` command, the software uses the maximum and minimum operating conditions to calculate the minimum and maximum delays. In this case also if you specify one operating condition, the software uses the `set_timing_derate` command.

As shown in Figure 26-3, the setup check at FF2 compares the maximum delay data at the D pin against minimum delay clock at the CLK pin. The maximum delay data at FF2/D consists of a sum of maximum signal delay from FF1/Q to FF2/D, the maximum delay from CLK\_SOURCE to FF1/CLK, and the delay from FF1/CLK to FF1/Q. Similarly, the minimum delay clock arrival time at FF2/CLK is the minimum delay from CLK\_SOURCE to FF2/CLK.

### Related Topics

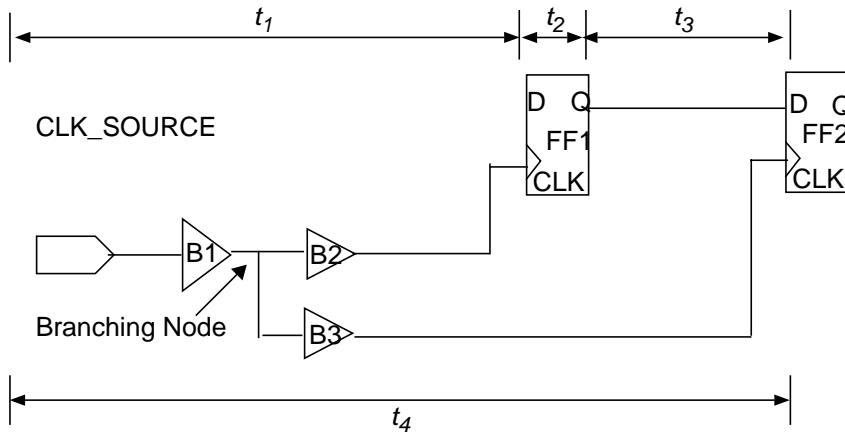
To see this step in the design flow, see [Run CTS and Post-CTS Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

## Encounter User Guide

### Timing Analysis

---

**Figure 26-3 Example Signal Path**



The setup check equation for the example in Figure 26-3 with pessimism is as follows:

$$t_{1max} + t_{2max} + t_{3max} \leq t_{4min} + t_{pa} - t_{su}$$

where,

$t_1$  = Delay value for launch clock late path

$t_2$  = Delay between FF1/CLK and FF1/Q

$t_3$  = Delay between FF1/Q and FF2/D

$t_2 + t_3$  = Delay value for late data path

$t_4$  = Delay value for capture clock early path

$t_{pa}$  = Period adjustment

$t_{su}$  = Setup time

The setup check equation incorrectly implies that the common clock network, B1, can simultaneously use maximum delay for the launch clock late path (clock source to FF1/CLK) and minimum delay for the capture clock early path (clock source to FF2/CLK). You use the CPPR to remove this pessimism.

Setup check equation using CPPR is as follows:

$$t_{1max} + t_{2max} + t_{3max} \leq t_{4min} + t_{pa} - t_{su} + t_{cppr}$$

## Encounter User Guide

### Timing Analysis

---

Where,

$t_{CPPR}$  = Difference in the maximum and minimum delay from the clock source to the branching node

Similarly, hold check equation using CPPR is as follows:

$$t_{1min} + t_{2min} + t_{3min} + t_{cppr} \leq t_{4max} + t_H$$

Where,

$t_1$  = Delay value for launch clock early path

$t_2$  = Delay between FF1/CLK and FF1/Q

$t_3$  = Delay between FF1/Q and FF2/D

$t_2 + t_3$  = Delay value for early data path

$t_4$  = Delay value for capture clock late path

$t_{CPPR}$  = Difference in the maximum and minimum delay from the clock source to the branching node

$t_H$  = Hold time

For example, you use the following `set_timing_derate` command for setup check delays in [Figure 26-3 on page 928](#) in scaled on-chip variation analysis methodology:

```
set_timing_derate -max -late 1 -early 0.9 -clock
```

With the `set_timing_derate` command, if the delay through B1 is 1ns, the software removes the pessimism of 0.1ns. Without CPPR the analysis tool incorrectly assumes that B1 can have a delay of 1 and 0.9. The common path pessimism time ( $t_{CPPR}$ ) is calculated as follows:

$$B1 * Late Derate - B1 * Early Derate = t_{CPPR}$$

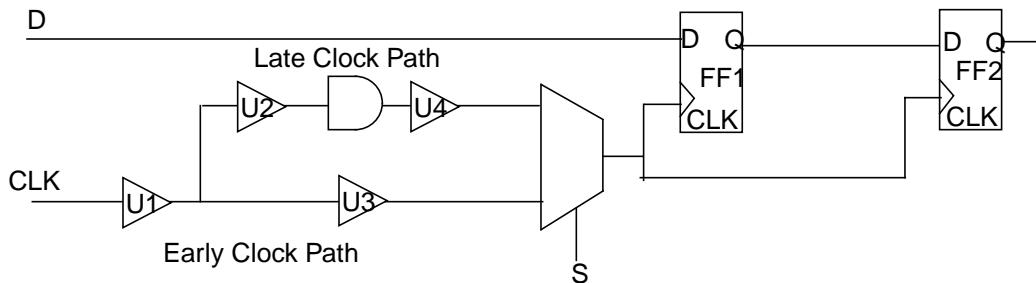
Therefore,

$$t_{CPPR} = 1.0\text{ns} * 1.0 - 1.0\text{ns} * 0.9 = 0.1\text{ns}$$

### CPPR and Reconvergent Logic

If a design contains reconvergent logic on the clock path, the timing analysis software might assume certain pessimism while calculating slack.

The following figure shows a circuit for which timing analysis is done in single analysis mode.



In this case, if `set_case_analysis` is not set at point **S** of the multiplexer, the timing analysis tools assume different delay values for early and late paths. For example, if early path has delay of 0.5ns, and late path has delay of 1ns, a pessimism equal to 0.5ns is introduced in the design.

The above pessimism is not specific to single analysis mode only; it also applies to best-case/worst-case and on chip variation methodology.

Encounter uses a threshold of zero during pessimism removal. That means no pessimism remains in the analysis.

## CPPR Flow

To remove this pessimism, use the `-cppr` parameter in the `setAnalysisMode` command.

In Encounter, the following flow supports the CPPR feature:

1. Load the timing constraints.

```
loadTimingCon top.sdc
```

2. Set the analysis mode to setup, propagated clock and CPPR.

```
setAnalysisMode -checkType setup -skew true
-clockPropagation sdcControl -cppr true
```

3. Set the derating values.

```
set_timing_derate -late 1 -early 0.9 -clock
```

4. Generate timing report. You use the `report_timing` command to remove delay pessimism from paths that have a portion of the clock network in common.

```
report_timing
```

## Encounter User Guide

### Timing Analysis

---

#### Timing Analysis Results Before and After CPPR

The following example shows a timing report generated before CPPR analysis.

```

Path 1: MET Setup Check with Pin reg_2/CK
Endpoint: reg_2/D (v) checked with leading edge of 'CLK1'
Beginpoint: reg_1/Q (v) triggered by leading edge of 'CLK1'
Other End Arrival Time      0.104
- Setup                      0.167
+ Phase Shift                2.000
= Required Time              1.938
- Arrival Time               1.850
= Slack Time                 0.088
Clock Rise Edge              0.000
= Beginpoint Arrival Time    0.000

```

Instance	Arc	Cell	Delay	Arrival Time	Required Time
ck_0	clk ^ A ^ -> Y ^	BUFX2	0.091	0.091	0.088
ck_1	A ^ -> Y ^	BUFX2	0.097	0.188	0.275
ck_2	A ^ -> Y ^	BUFX2	0.094	0.282	0.369
ck_3	A ^ -> Y ^	BUFX2	0.092	0.374	0.462
ck_4	A ^ -> Y ^	CLKAND2X2	0.150	0.524	0.612
reg_1	CK ^ -> Q v	DFFRHQX1	0.288	0.812	0.900
t_1	A ^ -> Y ^	BUFX8	0.111	0.923	1.011
t_2	A ^ -> Y ^	BUFX8	0.092	1.015	1.103
t_3	A ^ -> Y ^	BUFX8	0.092	1.107	1.195
t_4	A ^ -> Y ^	BUFX8	0.092	1.199	1.287
t_5	A ^ -> Y ^	BUFX8	0.092	1.291	1.379
t_6	A ^ -> Y ^	BUFX8	0.092	1.383	1.471
t_7	A ^ -> Y ^	BUFX8	0.092	1.475	1.563
t_8	A ^ -> Y ^	BUFX8	0.092	1.567	1.655
t_9	A ^ -> Y ^	BUFX8	0.092	1.660	1.747
t_10	A ^ -> Y ^	BUFX8	0.088	1.747	1.835
t_11	B ^ -> Y ^	NAND2X1	0.066	1.813	1.901
t_12	A ^ -> Y ^	INVX1	0.037	1.850	1.938

## Encounter User Guide

### Timing Analysis

---

reg_2	D v	DFFRHQX1	0.000	1.850	1.938
-------	-----	----------	-------	-------	-------

The following example shows a timing report generated after CPPR analysis:

```

Path 1: MET Setup Check with Pin reg_2/CK
Endpoint: reg_2/D (v) checked with leading edge of 'CLK1'
Beginpoint: reg_1/Q (v) triggered by leading edge of 'CLK1'
Other End Arrival Time      0.104
- Setup                      0.167
+ Phase Shift                2.000
+ CPPR Adjustment          0.420
= Required Time              2.358
- Arrival Time               1.850
= Slack Time                 0.508
    Clock Rise Edge          0.000
    = Beginpoint Arrival Time 0.000

```

Instance	Arc	Cell	Delay	Arrival Time	Required Time
	clk ^			0.000	0.508
ck_0	A ^ -> Y ^	BUFX2	0.091	0.091	0.598
ck_1	A ^ -> Y ^	BUFX2	0.097	0.188	0.695
ck_2	A ^ -> Y ^	BUFX2	0.094	0.282	0.789
ck_3	A ^ -> Y ^	BUFX2	0.092	0.374	0.882
ck_4	A ^ -> Y ^	CLKAND2X2	0.150	0.524	1.032
reg_1	CK ^ -> Q v	DFFRHQX1	0.288	0.812	1.320
t_1	A ^ -> Y ^	BUFX8	0.111	0.923	1.431
t_2	A ^ -> Y ^	BUFX8	0.092	1.015	1.523
t_3	A ^ -> Y ^	BUFX8	0.092	1.107	1.615
t_4	A ^ -> Y ^	BUFX8	0.092	1.199	1.707
t_5	A ^ -> Y ^	BUFX8	0.092	1.291	1.799
t_6	A ^ -> Y ^	BUFX8	0.092	1.383	1.891
t_7	A ^ -> Y ^	BUFX8	0.092	1.475	1.983
t_8	A ^ -> Y ^	BUFX8	0.092	1.567	2.075
t_9	A ^ -> Y ^	BUFX8	0.092	1.660	2.167
t_10	A ^ -> Y ^	BUFX8	0.088	1.747	2.255

t_11	B ^ -> Y ^	NAND2X1	0.066	1.813	2.321
t_12	A ^ -> Y ^	INVX1	0.037	1.850	2.358
reg_2	D v	DFFRHQX1	0.000	1.850	2.358

## Analyzing Timing Problems

In addition to the detailed timing violation report, the following report commands are helpful in analyzing timing problems:

- check\_timing  
Performs a variety of consistency and completeness checks on the timing constraints specified for a design. Use the `check_timing` command after setting all constraints, but before any timing analysis commands, such as `report_timing`, to verify that the timing environment is complete and self-consistent.
- get\_property  
Retrieves timing information for the specified pin property on the given pin.
- report\_analysis\_coverage  
Provides information about the timing checks in the design.
- report\_annotated\_check  
Reports coverage of annotated timing checks.
- report\_annotations  
Reports SDF design annotations coverage.
- report\_case\_analysis  
Reports ports and pins with `set_case_analysis` constraint.
- report\_cell\_instance\_timing  
Reports instance pin and delay arc timing information.
- report\_clock\_timing  
Generates a clock skew report for the current design.
- report\_clocks  
Reports clock waveform, clock arrival point and clock uncertainty information.

■ [report inactive arcs](#)

Reports all disabled timing arcs and checks.

■ [report path exceptions](#)

Reports design path exceptions such as `set_false_path`, `set_multicycle_path`, `set_max_delay` and `set_min_delay`.

■ [report timing](#)

Generates a timing report that provides information about the various paths in the design. The report typically contains data on the delay through the entire path. The start node and the end node of each path is identified.

■ [reportAnalysisMode](#)

Reports the current setting for building the timing graph. Use `setAnalysisMode` to change the settings.

■ [reportCapViolation](#)

Reports the nets that exceed the maximum capacitance constraints set by the timing library and timing constraints file.

■ [reportClockDomains](#)

Reports the clock domain setting for building the timing graph.

■ [reportTranViolation](#)

Reports the nets that exceed the maximum transition constraints set by the timing constraints file.

## Resolving Buffer-Related Problems

You may encounter some of the following buffer-related problems when running timing analysis:

- The logical cell or buffer equivalence, based on cell functionality not used during timing optimization, can cause timing optimization to ignore timing violations.
- If an incorrect buffer footprint name was entered for the set of buffers to run timing optimization, use the [reportFootPrint](#) command to list the current footprint information.

## Encounter User Guide

### Timing Analysis

---

- If the `in_place_swap_mode:match_footprint` statement is in the timing library, then timing optimization matches up all the cells with same `cell_footprint` name, and logical cell or buffer equivalence will not be used.
- If the `in_place_swap_mode:match_footprint` statement does not exist, then timing optimization derives logical cell equivalence based on matching function:`boolean_eq`.
- If you want the logical cell equivalence based on matching, comment out the `in_place_swap_mode:match_footprint` statement in the timing library.

## **Encounter User Guide**

### Timing Analysis

---

---

## Debugging Timing Results

---

- [Timing Debug Flow](#) on page 939
- [Generating Timing Debug Report](#) on page 940
- [Displaying Violation Report](#) on page 940
- [Analyzing Timing Results](#) on page 941
- [Creating Path Categories](#) on page 947
- [Using Categories to Analyze Timing Results](#) on page 956
- [Editing Table Columns](#) on page 960
- [Cell Coloring](#) on page 962
- [Viewing Schematics](#) on page 964
- [Running Timing Debug with Interface Logic Models](#) on page 964

## Overview

Encounter provides the Global Timing Debug feature for debugging the timing results. The various Timing Debug forms provide easy visual access to the timing reports and debugging tools. Encounter provides different timing debug feature depending on the timing mode that you have selected.

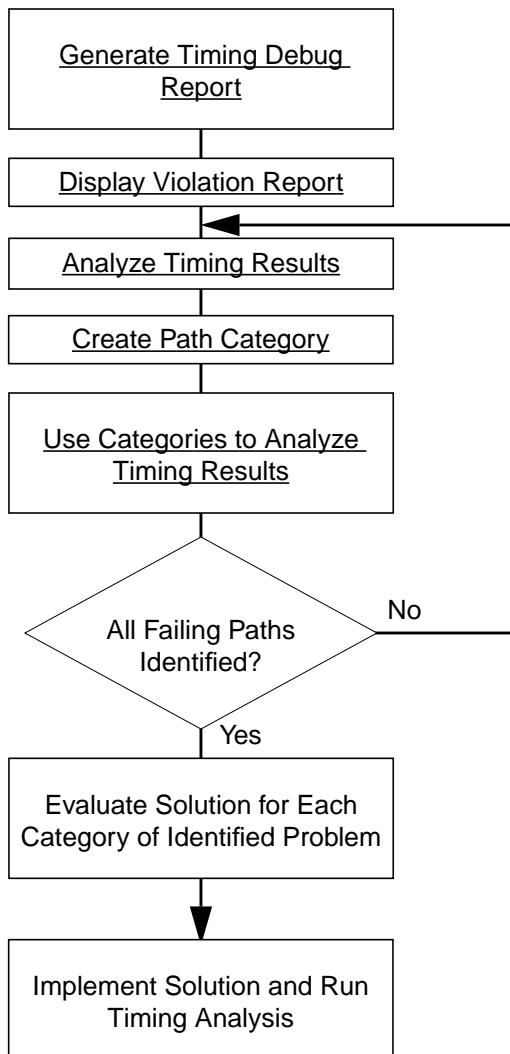
You can group all paths that are failing for the same reason and apply solutions for faster timing closure. You can cross-probe between the timing paths in the timing report and Encounter design display area.

**Note:** If you have a previously saved timing debug report, you can use the timing debug feature even when the design is not loaded in the Encounter session.

## Timing Debug Flow

You can generate a detailed violation report to list the details of all violating paths. You can then use the timing debug capability to visually identify problems with critical paths in this report. After identifying the problems, you group all paths with the same problem under a single category. You can define several categories to capture all problems related to the violating paths before fixing the problems and running timing analysis again.

Following is the flow for debugging timing results.



## Generating Timing Debug Report

Encounter uses a machine readable timing report to display timing debug information. The report is generated in the ASCII format and contains details of all violating paths. By default, the report has .mtarpt extension.

To generate a violation report, use one of the following options:

- Use report\_timing -machine\_readable command
- Use timeDesign -timingDebugReport command
- Use the Generate option in the Display/Generate Timing Report form.

You can also generate text-format report from a machine readable report.

To generate the text report, use the following:

- Use the write\_text\_timing\_report command
- Use the Write Textual Timing Report form

## Displaying Violation Report

To analyze the timing results, you need to load the machine readable timing report in Encounter.

To display the violation report, use one of the following options:

- Specify the file name in the Display/Generate Timing Report form.

**Note:** To select an existing file, deselect the *Generate* option before clicking on the directory icon to the right of the *Timing Report File* field.

By default, the global timing debug engine uses the following command to generate a machine-readable timing analysis report for the GUI display:

```
report_timing -machine_readable -max_points 10000 -max_slack 0.75
```

- Use the load\_timing\_debug\_report command.

**Note:** Use the Append to Current Report option in the Display/Generate Timing Report form to load multiple reports in a single session.

## Analyzing Timing Results

Encounter provides Timing Debug feature to visually analyze timing problems.

You analyze the following data in the Timing Debug form:

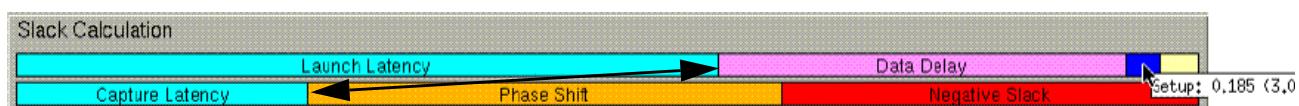
- Visual display of passing and failing paths as a histogram. Failing paths are represented in red and passing paths are represented in green color. The goal of timing debug process is to identify paths that fall in red category.
- Details of the critical paths in the Path list. You identify a critical path in this list for further analyses using the Timing Path Analyzer form.
- Visual display of paths reported in different timing reports. When you load multiple debug reports in a single timing debug session, the paths are displayed in different colors corresponding to the report file they are coming from. You can move the cursor over a path to display the name of the report file.

You analyze the following data in the Timing Path Analyzer form:

- Slack calculation bars for arrival and required times. You can identify clock skew issues, latency balancing or large clock uncertainty issues using these bars.

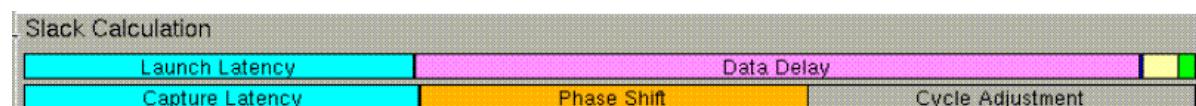
The following examples illustrate the problems that you can identify using the slack calculation bars in the Timing Path Analyzer form.

### Example 27-1



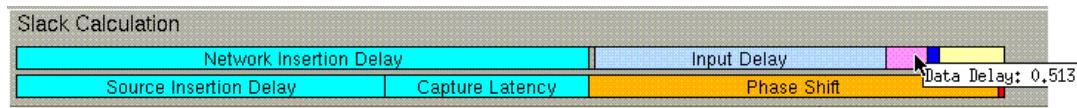
Launch and capture latency components are not aligned. Therefore there can be large clock-latency mismatch in this path.

### Example 27-2



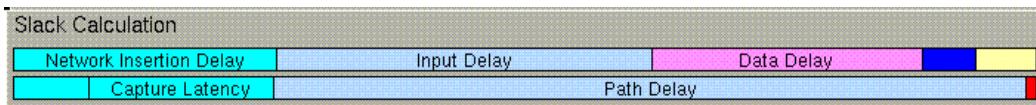
The cycle adjustment bar in the required time indicates presence of multicycle path.

### Example 27-3



Large input delay in an I/O path is represented by the blue bar in the arrival time.

### Example 27-4



Path Delay bar in the required time indicates a `set_max_delay` constraint.

- Path details including launch and capture. This information is provided as tabs in the Timing Path Analyzer form. You can click on a single path to display it in the design display area. You can select each element consecutively to trace the entire path in the design display area. This form has a Status column that indicates the status of the path as follows:

Flag	Description
a	Assign net
b	Blackbox instance
c	Clock net
f	Preplaced instance
i	Ignore net
s	Skip route net
t	“don’t touch” instance
t	“don’t touch” net
x	External net

Flag	Description
a	Assign net
b	Blackbox instance
c	Clock net
f	Preplaced instance
i	Ignore net
s	Skip route net
t	“don’t touch” instance
t	“don’t touch” net
x	External net

- SDC related to the path. The Path SDC tab displays the SDC constraints related to the selected path. The list contains the name of the SDC file, the line number that indicates the position of the constraint in the SDC file, and the constraint definition.

## **Encounter User Guide**

### Debugging Timing Results

---

The commands that can be displayed in the Path SDC tab are:

- `create_clock`
- `create_generated_clock`
- `group_path`
- `set_multicycle_path`
- `set_false_path`
- `set_clock_transition`
- `set_max_delay`
- `set_min_delay`
- `set_max_fanout`
- `set_fanout_load`
- `set_min_capacitance`
- `set_max_capacitance`
- `set_min_transition`
- `set_max_transition`
- `set_input_transition`
- `set_capacitance`
- `set_drive`
- `set_driving_cell`
- `set_logic_one`
- `set_logic_zero`
- `set_dont_use`
- `set_dont_touch`
- `set_case_analysis`
- `set_input_delay`
- `set_output_delay`
- `set_annotated_check`

- set\_clock\_uncertainty
- set\_clock\_latency
- set\_propagated\_clock
- set\_load
- set\_disable\_clock\_gating\_check
- set\_clock\_gating\_check
- set\_max\_time\_borrow

When you create a constraint on the command line, the Path SDC tab interactively displays the result of the additional constraint.

**Note:** MMMC views are not displayed interactively.



You can create a path category directly from SDC constraints in the Path SDC form. When you right-click on a constraint and select *Create Path Category*, the Create Path Category form displays, showing the line number (from the SDC file) and the name of the constraint. Alternatively,

You can also create a path category based on SDC constraint using the `-sdc` parameter of the create\_path\_category command:

```
create_path_category -name category_name -sdc {file_name line_number}
```

where `file_name` is the name of the constraint file and `line_number` is the line number of the SDC constraint in the file

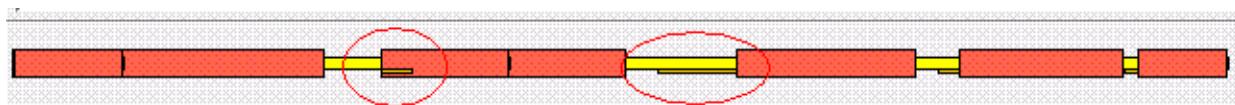
- Schematic display of the path. The Schematics tab displays the gate-level schematic view of the critical path. For more information, see Viewing Schematics on page 964.
- Timing interpretation for the path. This feature provides rule-based path analysis help you discover sources of potential timing problems in a path. By default, the software performs the following checks the following rules:
  - Path structure
    - Transparent Latch in Path
    - Clock Gating
    - Hard Macros
    - HVT Cells

- Buffering List
- Net Fanout
- Level Shifters
- Isolation Cells
- Timing and constraints
  - Large Skew
  - Divider in Clock Path
  - Total SI Delay
  - SI Delay
  - External Delay
- Floorplan
  - Fixed Cells
  - Distance from start to end
  - Distance of repeater chain
  - Detour
  - Multiple power domains
- DRVs
  - Max transitions
  - Max capacitance
  - Max fanout

You can customize the type of timing information reported. The *Edit Timing Interpretation* GUI lets you add, modify, or delete rules you want the tool to check and report.

- Timing bar to analyze delays associated with instances and nets in a path. Use this information to identify issues related to large instance or net delays, repeater chains, paths with large number of buffers, and large macro delays. The small bars

superimposed on net delays or within element delays show incremental (longer or shorter) delays due to noise effects:



- Hierarchical representation of the path in the Hierarchy View field. This representation of the path-delay shows the traversal of a path through the design hierarchy drawn on the time axis. A longer arrow means that there are more instances on its path. Use this information to see the module where the path is spending more time or to identify inter-partition timing problems.

## Viewing Power Domain Information

While debugging critical paths on MSV designs, it is useful to be able to identify power domains and low power cells. The Timing Debug feature displays this information in the following ways:

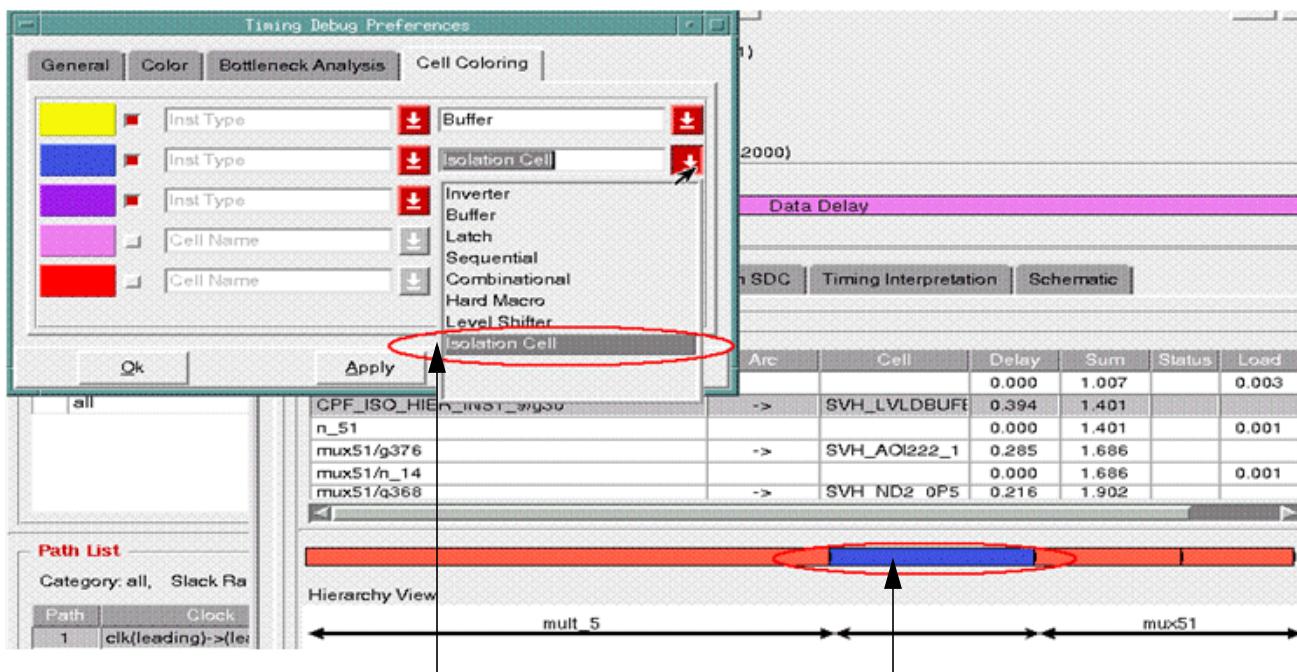
- The level shifters and isolation cells are listed in the Timing Interpretation tab of the Timing Path Analyzer.

Level shifters and Isolation Cells are displayed in the Timing Implementation tab

## Encounter User Guide

### Debugging Timing Results

- The delay bar of the Timing Path Analyzer can display the level shifters and isolation cells. You can also use the Preferences form to specify the colors in which the level shifters and isolation cells are displayed.



The colors for level shifters and isolation cells can be set in the Preferences form....

...and are displayed in the delay bar

## Creating Path Categories

After analyzing the paths in the timing report, you identify problems in various paths. Then you create a group of paths such that all paths in that group have the same timing problem and can be fixed at the same time. In timing debug such a group of paths is called a category. In Encounter, you can either define your own category or use predefined categories to group your paths. The categories that you define are then displayed in the Timing Debug form . The form also displays the paths associated with each category.

## Creating Predefined Categories

There are following predefined categories:

- Basic Path Group

Creates standard path categories according to basic path groups. For more information, see [Path Analysis \(Basic Path Groups\)](#).

- Clock Paths

Creates categories according to launch clock - capture clock combinations. For more information, see [Path Analysis \(Clock Paths\)](#).

- Hierarchical Paths

Creates categories according to the hierarchical characteristics of a path. For more information, see [Path Analysis \(Hierarchical Paths\)](#).

- View

Creates categories according to the view for which the path was generated. For more information, see [Path Analysis \(View\)](#).

- False Paths

Creates a category with paths defined as False paths. For more information see [Path Analysis \(View\)](#).

- Bottleneck

Creates categories based on instances that occur often in critical paths. For more information, see [Timing Debug Preferences – Bottleneck Analysis](#).

- DRV Analysis

Generates or loads a DRV report containing capacitance, transition, or fanout violations. Paths that are affected by the selected DRV types are grouped in a category.

To create or load this report, use the *drv* analysis type on the Path Analysis GUI.

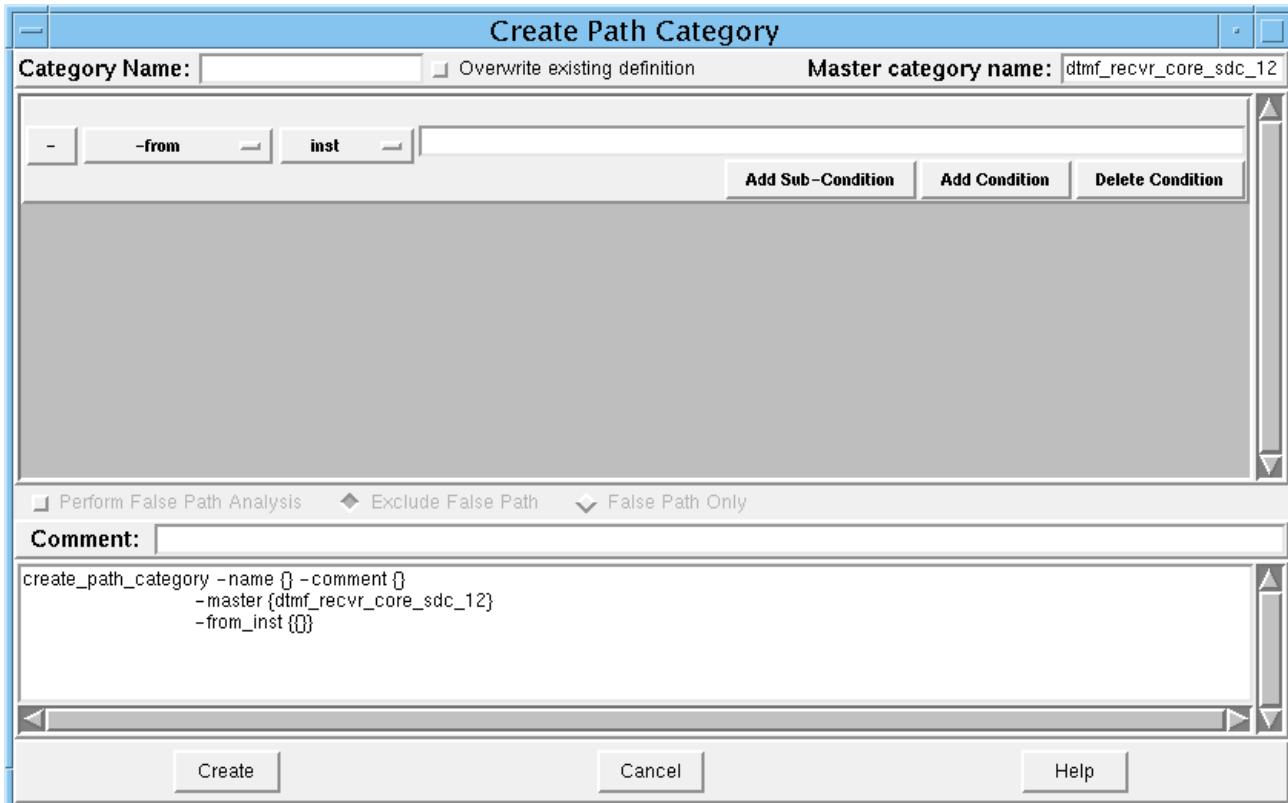
## Creating New Categories

To define a new category, use the [Create Path Category](#) form. The Create Path Category form contains drop-down menus with conditions that you use to define a path category. The

## Encounter User Guide

### Debugging Timing Results

conditions are characteristics that a path must have to be added to the named category. You can define multiple conditions that a path must meet to be added to the category.



The category that you create is added in the Timing Debug form. All paths that meet the conditions set for this category are grouped under the category name. Paths are separated automatically according to MMMC views into different categories, for example:

```
CLOCK1<View_test_mode>
CLOCK2<View_mission_mode>
```

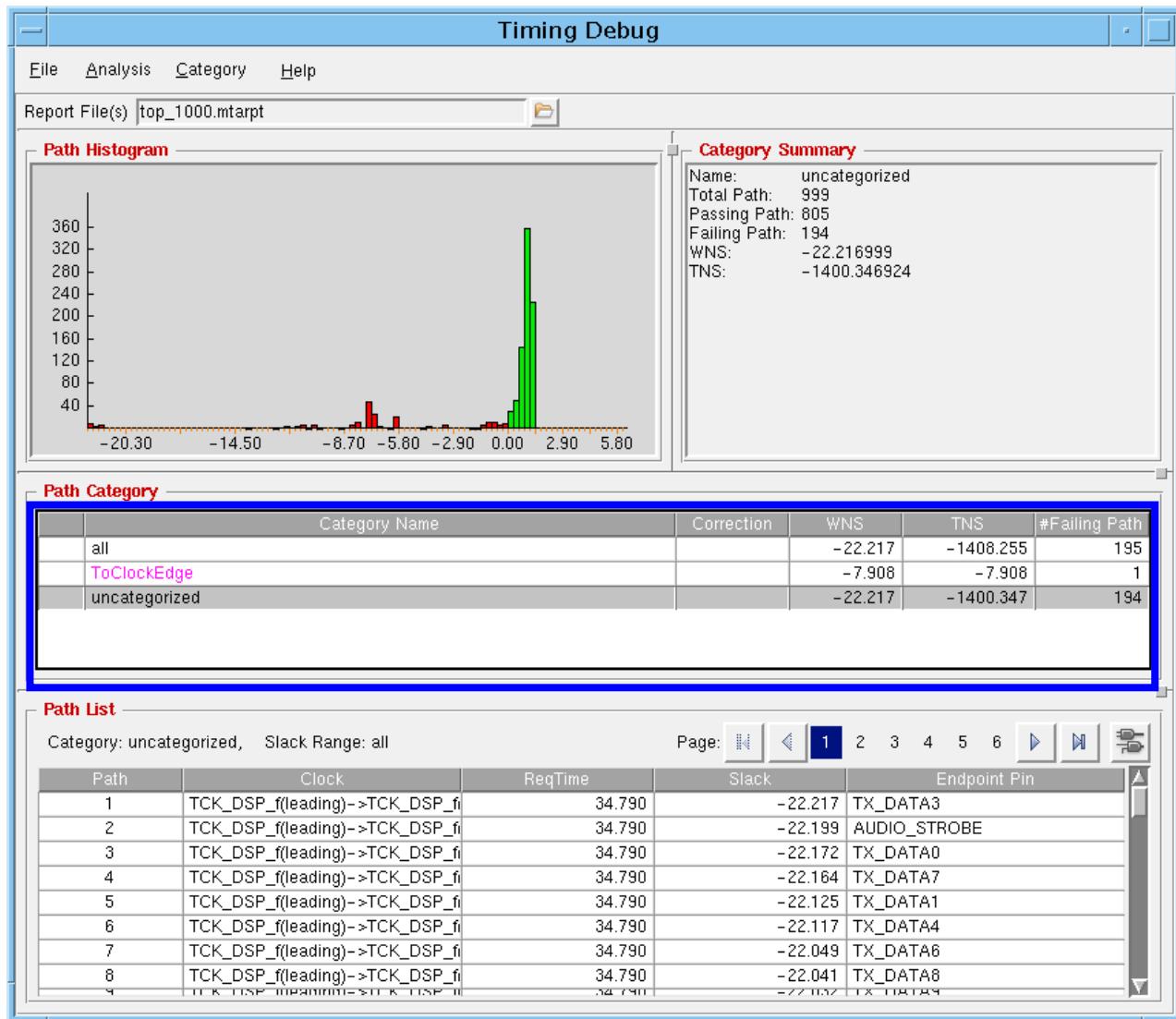
Double-click on the category name in the Timing Debug form to display the list of paths in the *Path List* field.

**Note:** You can add a comment in the *Comment* field to record any notes that you would like to include with the category. The comment appears in the category report file.

## Encounter User Guide

### Debugging Timing Results

Following figure shows categories created by the Create Path Category form.



## Creating Sub-Categories

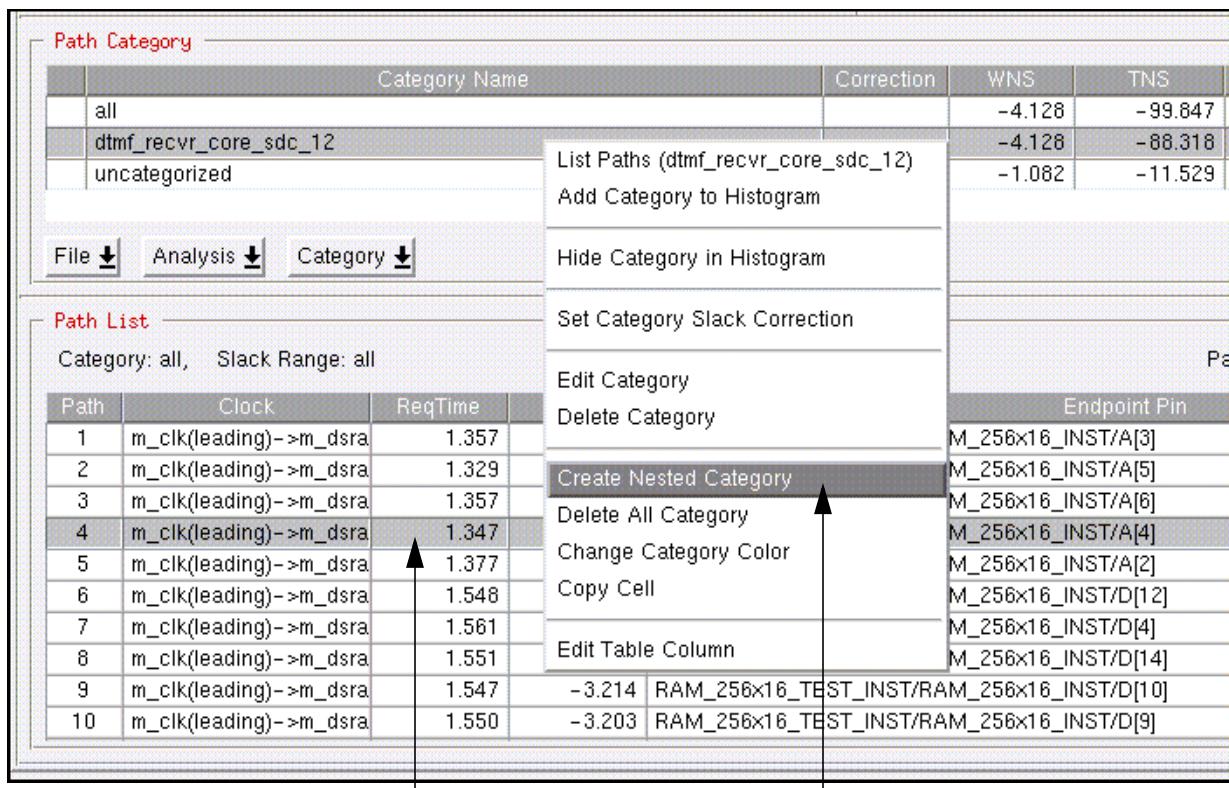
You can create sub-categories based on existing categories. While analyzing a sub-category, global timing debug will traverse the paths in the master category instead of all the paths in the current report.

- [Creating Sub-Categories through the GUI](#) on page 951
- [Creating Sub-Categories through Command Line](#) on page 952

## Creating Sub-Categories through the GUI

In the GUI, you can create a sub category as follows

1. Right-click on a path category, and select *Nested Category*.



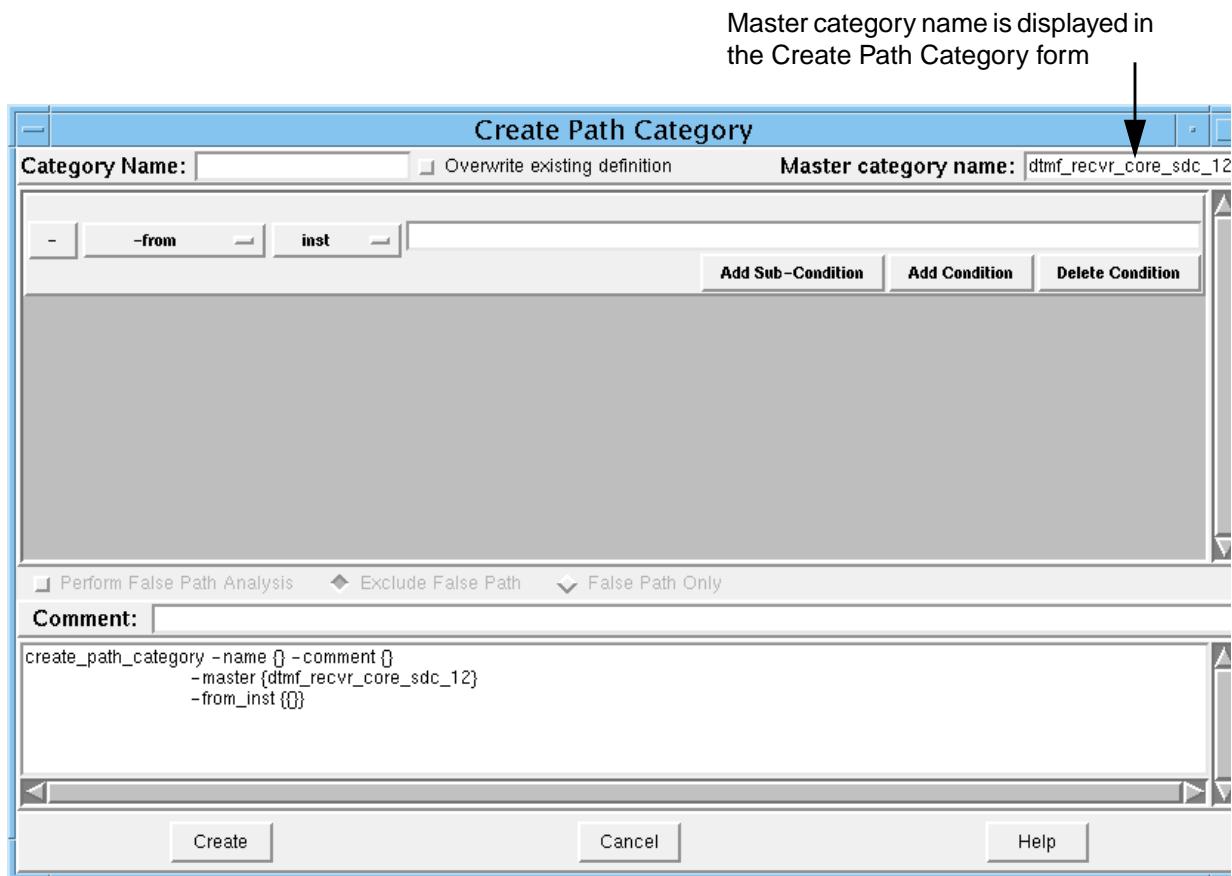
To create a sub-category, right-click on  
the category...

...and then select Create Nested  
Category

## Encounter User Guide

### Debugging Timing Results

The Create Path Category form is displayed.



2. In the *Master category name* field, the name of the category you selected previously is displayed. Create one or more subcategories as explained in [Creating Path Categories](#) on page 947.

**Note:** You can create nested sub-categories, that is, you can further create sub-categories for a sub-category.

You can also use the Category – Create menu command to bring up the Create Path Category form. In the *Master category name* field, type the name of the category for which you want to create the sub-category, and then create one or more subcategories as explained in [Creating Path Categories](#) on page 947.

### Creating Sub-Categories through Command Line

Use the *-master* parameter of the `create_path_category` command to create sub-categories. The category created will be a sub-category of the category name specified with the *-master* parameter.

The following other commands also support sub-categories; to run these commands only on the sub-categories of a particular master category, specify the master category name with the `-master` parameter.

- [analyze paths by basic paths group](#)
- [analyze paths by bottleneck](#)
- [analyze paths by clock domain](#)
- [analyze paths by critical false path](#)
- [analyze paths by drv](#)
- [analyze paths by hierarchy](#)
- [analyze paths by view](#)

**Note:** If the parent category of a sub-category is deleted, the sub-category cannot be edited or changed anymore. However, the sub-category is still displayed in case you want to refer to it.

## Viewing Sub-Categories

The subcategories for a master category are displayed in a hierarchically numbered list below the master category. As an illustration, consider the example shown here:

Path Category		
	Category Name	Correction
	all	
	master_category1	
(1)	nested_category_1a	
(2)	nested_category_2	
(1)	nested_category_1b	
	uncategorized	

In this example:

- `master_category1` is the master category

- nested\_category\_1a and nested\_category\_1b are the sub-categories of master\_category1.  
The prefix (1) is displayed with nested\_category\_1a and nested\_category\_1b.
- nested\_category\_2 is the sub-category of nested\_category\_1a.  
The prefix (2) is shown with nested\_category\_2.

## Hiding path categories

To remove a path category from the histogram display, right-click on a path and select *Hide Category*. The category name in the category list is not hidden, but is marked with an “H” as hidden.

## Reporting Path Categories

To generate a report containing information about path categories, use the following options:

- Use the `write_category_summary` command
- Use the Write Category Report File GUI

The text file contains the following information:

- Category name
- Total number of paths
- Number of passing paths
- Number of failing paths
- Worst negative slack
- Total negative slack
- TNS

Sample report:

Category name	Total path	Passing Path	Failing path	WNS	TNS
------------------	---------------	-----------------	-----------------	-----	-----

## Encounter User Guide

### Debugging Timing Results

---

test_clock<view_test_	3869	768	3101	-5.699	-4802.857
100MHz_1.00V>	Clock Domain Analysis				
@->test_clock<view_test	484	55	429	-2.292	-378.432
_100MHz_1.00V>	Clock Domain Analysis				
my_clk-><view_mission	1	2	11	1.931	-1.931
_166MHz_1.08V>	Clock Domain Analysis				
my_clk_2x<view_mission	156	154	2	-.013	-0.21
_166MHz_1.08V>	Clock Domain Analysis				
cat1875	77	13	64	-3.524	-100.893
	Category of paths that cross i_1875 and start with test_clock - need to change the uncertainty value -advised --by Don				

## Using Categories to Analyze Timing Results

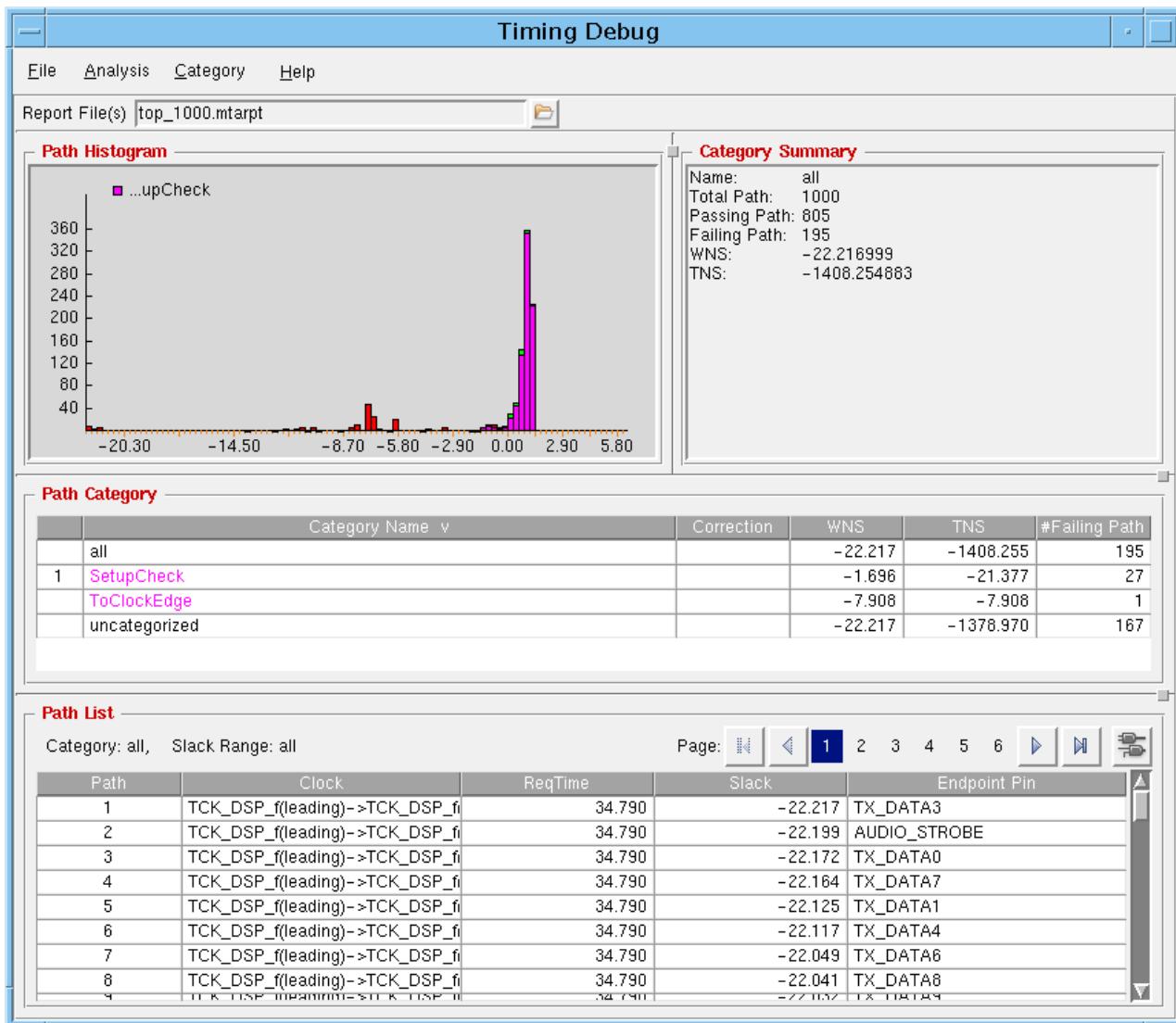
You can use the categories that you create to group the timing paths in the [Timing Debug](#) form. The Timing Debug form displays the category details in the Path Category field. You can perform the following tasks in the Timing debug form to analyze the timing results:

- Double-click on any category to display the details of the paths grouped in that category in the Path List field.
- Right-click on the category name and select Add to Histogram option. The paths related to the selected category are highlighted in a different color in the histogram. This gives you a visual representation of the number of paths that meet the conditions in that category and can possibly have the same timing problem.

## Encounter User Guide

### Debugging Timing Results

For example, in the following figure the *SetupCheck* category was added to the histogram.



Analyzing the resulting the Timing Debug form gives you information for fixing problems related to larger sets of timing paths. After identifying the problems, you can make the required changes such as modify floorplan, script or SDC files and run timing analysis again for further analysis.

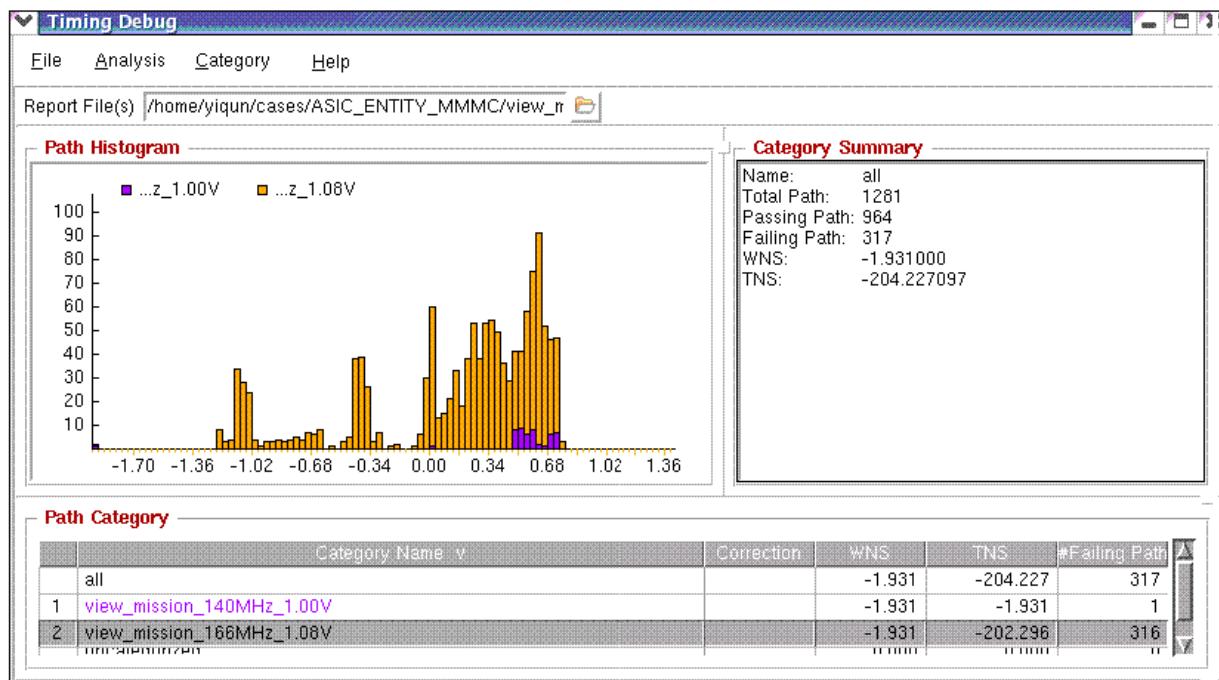
## Analyzing MMMC Categories

Paths are separated automatically according to MMMC views into different categories, for example, the following figure shows two categories based on MMMC views:

## Encounter User Guide

### Debugging Timing Results

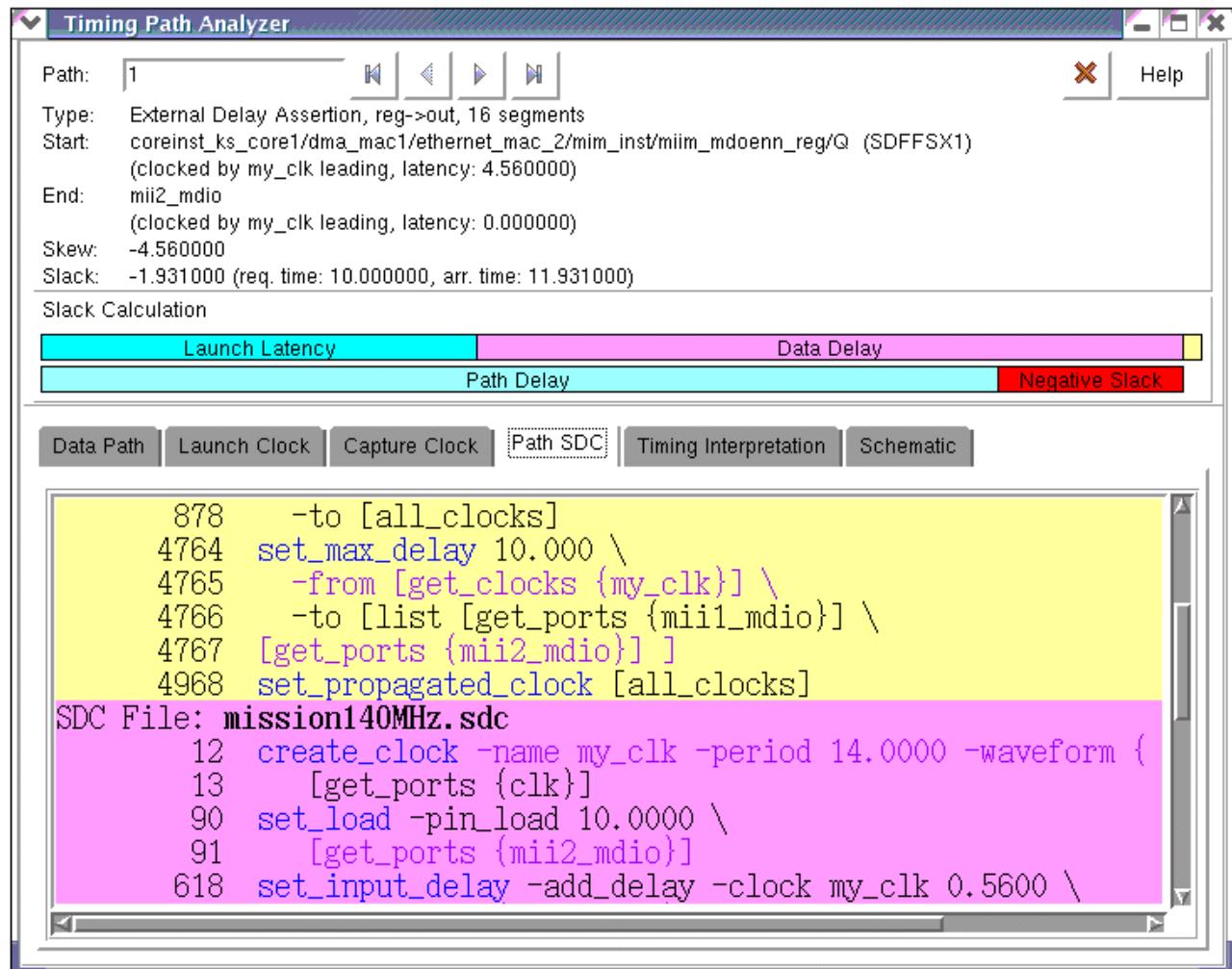
- view\_mission\_140MHz\_1.0V
- view\_mission\_166MHz\_1.8V



1. Right-click on one of view\_mission\_140MHz\_1.0V and choose *List Paths*.
2. Right-click on one the paths and choose Show Timing Path Analyzer.  
The Timing Path Analyzer is displayed.
3. Click on the Path SDC tab to display the SDCs:

## Encounter User Guide

### Debugging Timing Results



Note that the SDCs relative to mode mission\_140MHz that produced the path are highlighted.

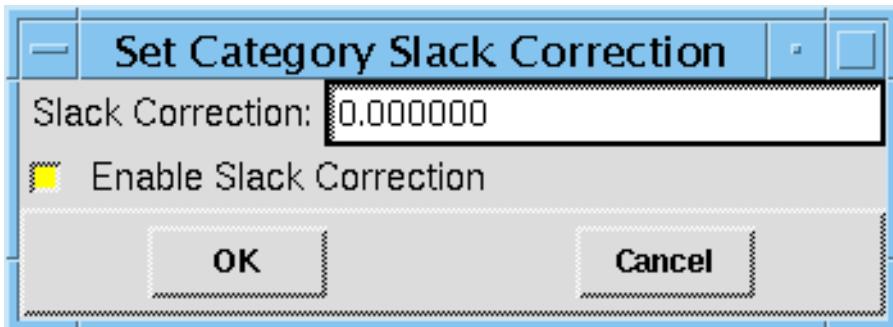
## Manual Slack Correction of Categories

Use the Set Category Slack Correction form to specify the estimated slack correction for the selected category of paths. A slack correction that you apply to a category modifies all the paths in that category. If a path belongs to several categories, all the correction from the categories are added. The worst negative slack and total negative slack values of a category can be affected by the correction applied to another category.

Once you enable the slack correction, the histogram is updated to reflect the slack correction. An asterisk (\*) is added next to the slack value of paths that belong to this category in the *Path List* field in the Timing Debug window. Paths are reordered based on new specified slack. This allows you to filter out the paths that can be fixed and work on the remaining paths.

To access the Set Category Slack Correction form complete the following steps:

1. Choose *Timing – Debug Timing* .
2. Right click on the category name in the *Path Category* field.
3. Choose the *Set Category Slack Correction* option.



To disable the set slack correction value:

1. Right click on the category name in the *Path Category* field.
2. Choose the *Deactive Category Slack correction* option.

## Editing Table Columns

You can customize the dimensions and contents of table columns to suit your needs.

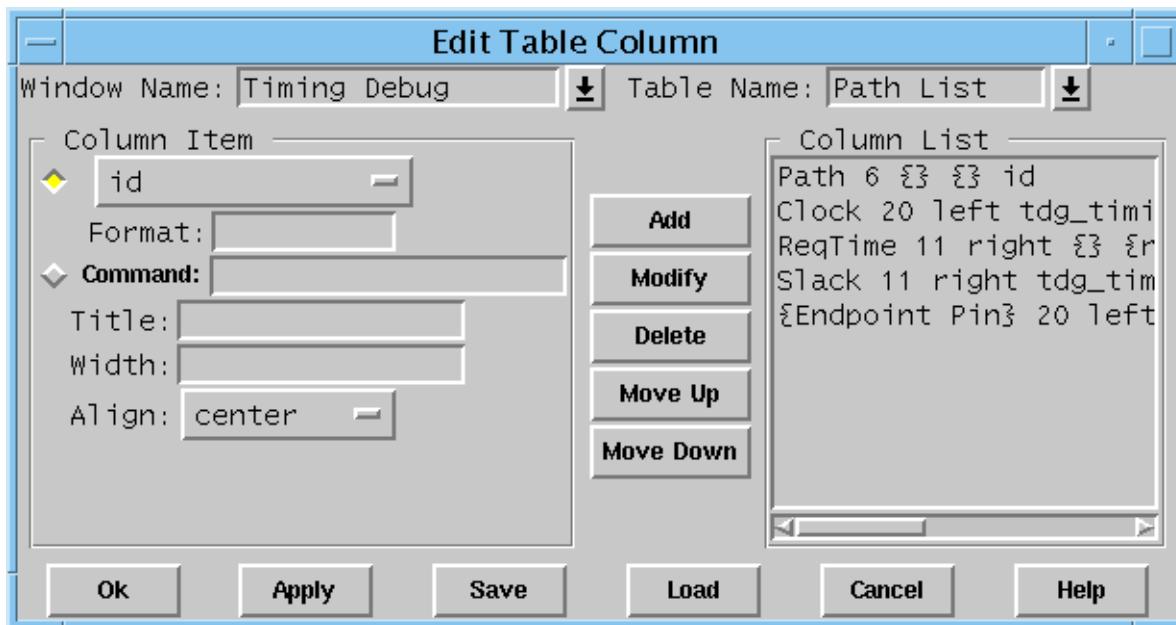
1. Open the *Timing Debug* or *Timing Path Browser* form. , then right-click on a path.  
A drop-down menu is displayed.

## Encounter User Guide

### Debugging Timing Results

#### 2. Select *Edit Table Column*.

The Edit Table Column form is displayed.



3. Choose the timing window that contains the table to want to customize.
4. Choose the table whose columns you want to customize. The selections change according to the timing window you choose.
5. Choose a column item or specify a command.

For commands, specify the procedure you want to use to determine the information you want to include in the column. Source the file containing the procedure before you specify the procedure here.

For example:

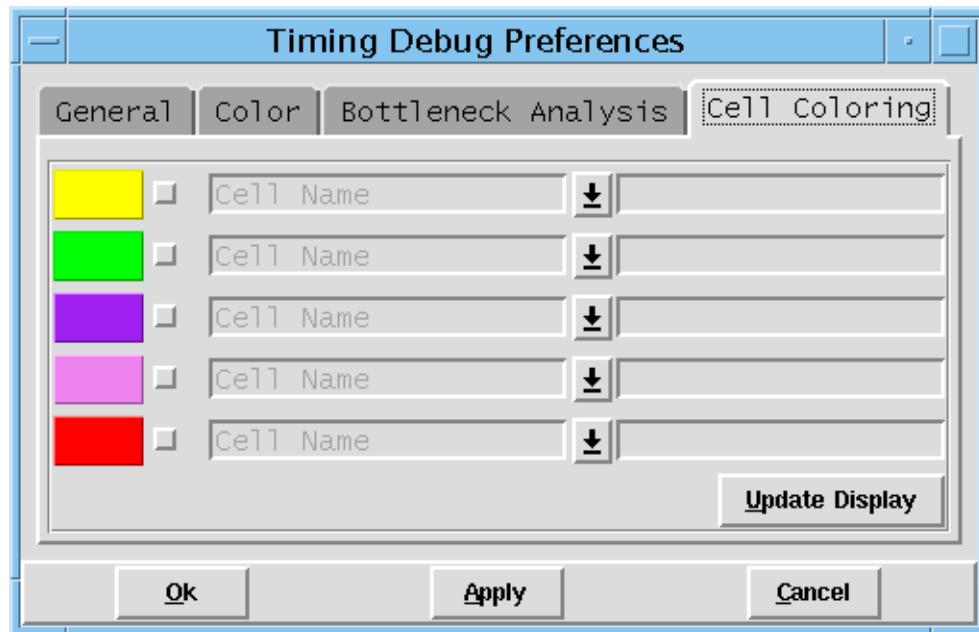
```
# Combine fedge (from edge) and tedge (to edge) #information into a single field
proc my_get_edge {id var} {
    upvar #0 $var p
    if {$p(type) == "inst"} {
        return "$p(fedge) -> $p(tedge)"
    } elseif {$p(type) == "port"} {
        return $p(fedge)
    } else {
        return ""
    }
}
```

6. Build the column list.

- ❑ *Add* adds a column to the column list.
  - ❑ *Modify* let you modify characteristics. Click on a column in the column list. Edit the information, then click *Modify*.
  - ❑ *Delete* removes a column from the column list.
  - ❑ *Move Up* moves a column up in the column list. This effectively moves a column to the left in the table.
  - ❑ *Move Down* moves a column down in the column list. This effectively moves a column to the right table.
7. (Optional) Click *Load*. This opens the GTD (Global Timing Debug) Preferences form. Specify a file name.

## Cell Coloring

Use the Cell Coloring page of the Timing Debug Preferences form to choose colors for specific cells in the delay bar.



When you assign colors, this same colors will be restored when you start a new session.

In the *Cell Name Selection Elements* field for each color, you can choose whether you are providing one of the following:

- Cell name

- Instance
- Procedure that you have defined

The procedure is invoked with the full instance name as the argument. You must source the file containing the procedure before you use this feature.

For example:

```
# Colors when the instance name contains "core/block1"
proc belongs_to_block1 {inst_name} {
    if [regexp {core/block1} $inst_name] {
        return 1
    } else {
        return 0
    }
}
```

## Viewing Schematics

The Critical Path Schematic Viewer displays the gate-level schematic view of the critical path that you select in the Debug Timing - Browser window. To display the Schematic Viewer, click on the schematics icon in the Timing Debug window. You can display additional paths in the Schematic Viewer by using the middle mouse button to drag the path from path list to Schematic Viewer.

You can perform the following tasks in the Module Schematic Viewer:

- View the Gate-level design elements.
- Select an element in the schematic.
  - Click on an object in the schematic to select and highlight it. When you move the cursor to an object, the object type and name of the object appear in the information box.
- Scroll over an object to display the object type and name of the object in the *Object* field.
- Cross-probe between the schematic and design display area.
  - Drag and drop an object from the schematic to the design display area to zoom in and highlight the object.
- Cross-probe between the Schematics window and *Path List* field.
  - Select a path and left-click on the Schematics button above the Path List Table. (This is the button at the far-right side, just above the table).
  - To show multiple paths, select another path, and drag and drop it to the Schematics window.
- Use the menu options provided in the Schematic Viewer. To access the menu options, you can either click on the menu bar or right-click on an object in the schematic. You can use the menu options to perform the following tasks:
  - Manipulate schematic views of fan-in and fan-out cones.
  - Trace connectivity between drivers, objects, and loads.
  - Move between different levels of instance views.

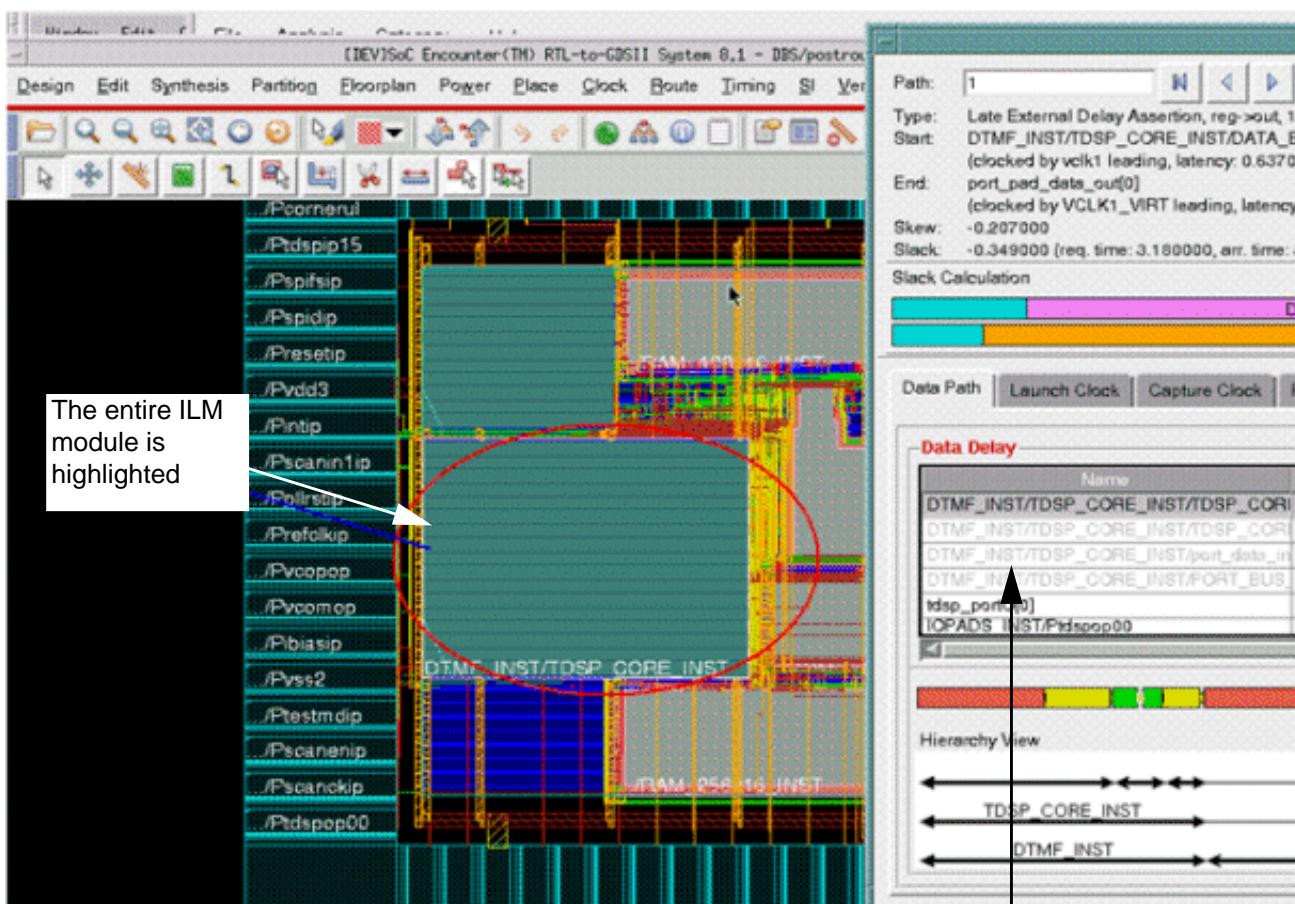
## Running Timing Debug with Interface Logic Models

You can use the timing debug feature with designs containing Interface Logic Models (ILMs).

# **Encounter User Guide**

## Debugging Timing Results

- You must flatten ILMs before creating global timing debug reports. Use the `flattenILM` command first. After loading the report, you can run timing debug in flattened or unflattened mode. The schematic feature is disabled in unflattened mode.
  - The Timing Path Analyzer – Path SDC form displays ILM SDCs rather than the original SDCs.
  - The software highlights the entire ILM module instead of the instances and nets inside the ILM. The instances and the nets inside the ILMs are greyed out in the Timing Path Analyzer – Path SDC form.



Instances and nets inside the ILM module are greyed out in the Timing Path Analyzer – Path SDC form.

## **Encounter User Guide**

### Debugging Timing Results

---

---

## Statistical Static Timing Analysis

---

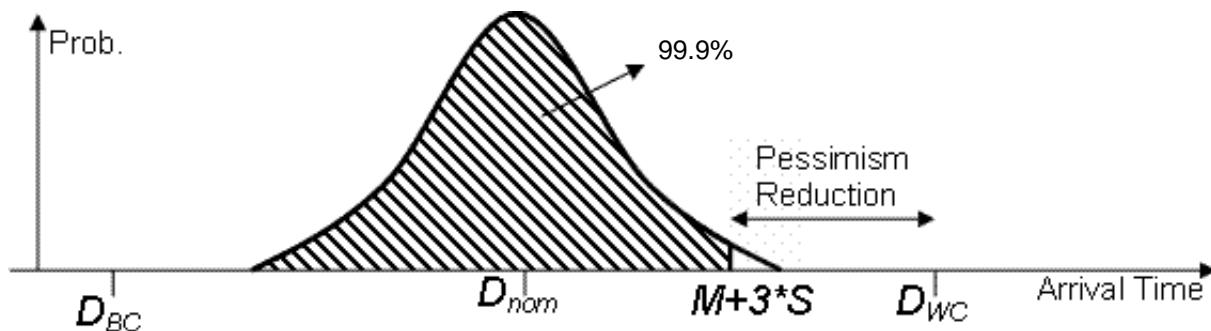
- [SSTA Overview](#) on page 968
- [SSTA Inputs](#) on page 972
- [SSTA Flows](#) on page 978
- [SSTA Outputs](#) on page 981
- [SSTA Correlation With Monte-Carlo Analysis](#) on page 985

## SSTA Overview

A major design challenge in 90nm and below designs are process variations. Process variations account for deviations in the semiconductor fabrication process. Process variations are due to variations in the manufacturing conditions such as temperature, pressure and dopant concentrations. To handle process variations, static timing analysis (STA) provides multi-corner analysis. However due to modelling of large number of process variation corners, STA methodology can be very complex and time consuming. Additionally STA can be very conservative in most cases and can be optimistic in some cases resulting in missed violations. To effectively handle process variation, you can now use Statistical Static Timing Analysis (SSTA).

SSTA represents the slack in terms of probability density function (PDF). PDF accounts for the variability of all process factors being modelled. This methodology targets a specific percentage yield for timing analysis and optimization.

SSTA removes the pessimism present in corner based STA. This helps in removing extra margins in the design and therefore improves design cycle time, and chip area. For example, consider the arrival time of signals on different chips. The arrival time on different chips is different because process variation delays are not constant. The following figure shows arrival time probability density function with different arrival times on x axis, and fractional number of chips on y axis.



In this figure,

$D_{nom}$ : Arrival time at nominal corner

$D_{BC}$ : Arrival time at best case corner

$D_{WC}$ : Arrival times at worst corner

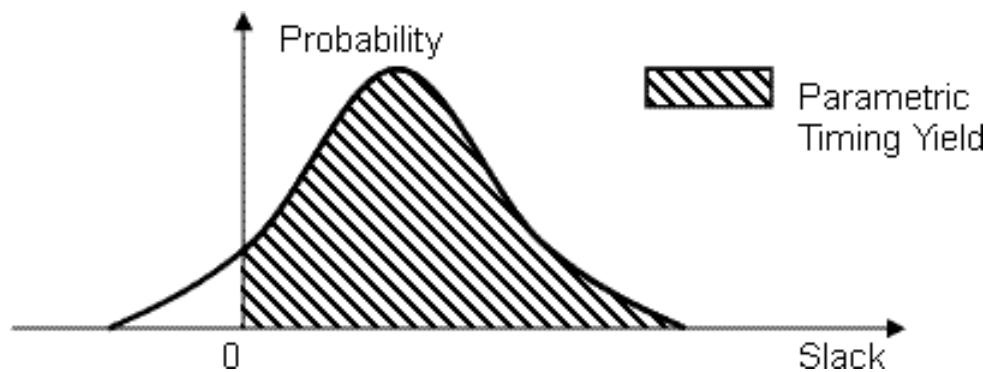
$M + 3*S$ : Worst arrival time for 99.9% yield.  $M$  is the mean value and  $S$  is the standard deviation or sigma value.

## Encounter User Guide

### Statistical Static Timing Analysis

In this example, none of the chips have the arrival time of  $D_{WC}$ . Therefore use of worst corner in STA will result in pessimistic delay analysis. SSTA analysis on the other hand considers complete distribution of arrival times and calculates worst arrival time for 99.9% yield. In this case,  $M+3^*S$  delay is much smaller than worst corner arrival time  $D_{WC}$ .

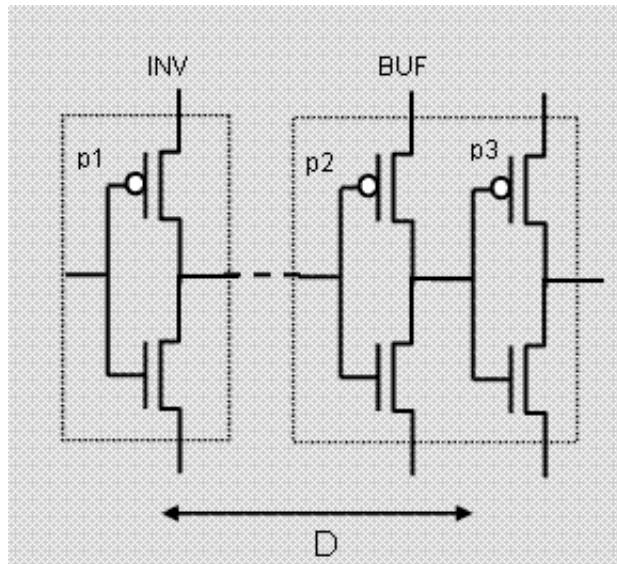
SSTA also provides data to trade-off between performance and yield. In STA, if a violation is reported at worst corner, it is actually violation on the worst chip in a lot. Since STA does not provide any information on the number of chips affected by the violation, you must fix this violation. This impacts cycle time and/or chip area. SSTA, on the other hand, provides the yield number (in terms of fraction of chips that can be affected by this violation) for a given frequency and vice versa. You can use this information to trade-off between yield and frequency. The following figure shows probability density function with slack value on X axis and fractional number of chips on the Y axis.



In this figure, parametric timing yield gives the percentage of chips with positive slack value.

The following figure shows process variations between different cell instances in a design.

**Figure 28-1 Variations between instances in a design**



In this figure,

$p_1$ ,  $p_2$  and  $p_3$  are process parameters for three different transistors

D is Distance between the cell instances

SSTA accounts for the following process variations:

- Global or die-to-die variations

SSTA considers perfect correlation between all devices on the chip. Consider the variations between cell instances shown in Figure 1-1. The following equation shows the correlation between  $p_1$ ,  $p_2$  and  $p_3$  for global or die-to-die variations:

$$\text{Correlation}(p_1, p_2) = \text{Correlation}(p_1, p_3) = \text{Correlation}(p_2, p_3) = 1$$

- Random variations

SSTA considers the process variation of all transistors on the chip to be uncorrelated to each other. Consider the variations between cell instances shown in Figure 1-1. The following equation shows the correlation between  $p_1$ ,  $p_2$  and  $p_3$  for random variations:

$$\text{Correlation}(p_1, p_2) = \text{Correlation}(p_1, p_3) = \text{Correlation}(p_2, p_3) = 0$$

- Spatial variations

SSTA considers the process variations inside the cell to be perfectly correlated. It also considers process variation correlation between transistors in different instances of the

## Encounter User Guide

### Statistical Static Timing Analysis

---

cells as function of distance between instances. Consider the variations between cell instances shown in Figure 1-1. The following equation shows the correlation between  $p_1$ ,  $p_2$  and  $p_3$  for spatial variations:

$$\text{Correlation } (p_2, p_3) = 1 \quad \text{Correlation } (p_1, p_2) = \text{Correlation } (p_1, p_3) = f(D)$$

Where,  $f(D)$  is a user defined function. The value of function decreases as  $D$  increases.

Therefore, the software models the process parameter as follows:

$$P = P_{nom} + P_{global} + P_{sys} + P_{random}$$

Where,

$P$  = Process parameter

$P_{nom}$  = Nominal process variation

$P_{global}$  = Global process variation

$P_{sys}$  = Systemic or spatial process variation

$P_{random}$  = Random process variation

## SSTA Inputs

SSTA requires specific input files containing sensitivities and distribution besides the inputs that you need to perform STA, such as a netlist and timing constraints.

The inputs required for SSTA are as follows:

- [Libraries with sensitivities](#) on page 973
- [Statistical Parameter Distribution Format \(SPDF\) File](#) on page 975
- [Sensitivity-Based SPEF \(S-SPEF\) File](#) on page 977
- Timing Constraints

The timing constraints supported for SSTA are the same as those supported for STA. For a list of supported timing constraints, see the [Timing Constraints Commands](#) chapter in *Encounter text Command Reference*.

- Netlist

## Libraries with sensitivities

SSTA requires S-ECSM libraries. These libraries provide sensitivities of delay, slew, and timing checks (such as setup and hold) to various cell process parameters. The libraries also contain sensitivities of voltage v/s time waveform. The S-ECSM libraries contain sensitivities for both global and random variations. Spatial variations are modelled as global parameters at the library-level.

You can use Encounter Library Characterizer (ELC) or other third party library characterizers to generate S-ECSM libraries. To generate S-ECSM libraries, the following inputs are provided to ELC:

- Spice models: Contains process parameters
- Spice circuits: Contains extracted spice netlist of library cells
- Configuration file: Contains variation of process parameters

For random variations, you can optionally combine the parameters in the library file. The following is a sample of S-ECSM file with sensitivity tables for global parameters, p1 and p4, and combined random parameter (ecsm\_random):

```
ecsm_timing_sensitivity() {
    ecssm_parameter_type : p1 ;
    ecssm_parameter_variation : 2.9 ;
    fall_transition(delay_template_8x7) { ... }
    cell_fall(delay_template_8x7) { ... }
    rise_transition(delay_template_8x7) { ... }
    cell_rise(delay_template_8x7) { ... }
}
ecsm_timing_sensitivity() {
    ecssm_parameter_type : p2;
    ecssm_parameter_variation : 2 ;
    ...
}
ecsm_timing_sensitivity() {
    ecssm_parameter_type : p3;
    ecssm_parameter_variation : 1.7 ;
    ...
}
ecsm_timing_sensitivity() {
    ecssm_parameter_type : p4;
    ecssm_parameter_variation : 1.2 ;
    ...
}
ecsm_timing_sensitivity() {
    ecssm_parameter_type : ecssm_random;
    ecssm_parameter_variation : 1 ;
    ...
}
```

**Encounter User Guide**  
Statistical Static Timing Analysis

---

For more information on library characterization, see *Encounter Library Characterization User Guide*.

## Statistical Parameter Distribution Format (SPDF) File

An SPDF file contains statistics of variation of process parameters. You can specify global, random and spatial variations in this file. You can create this file using the data provided by the foundry.

A sample of SPDF file is as follows:

```
{Standard_Parameter_Format
  {SPDF_Version "1.0"}
  {PROGRAM "SSTA"}
  {PROGRAM_VERSION "1.0"}
  {Units
    {VOLTAGESCALE 100 mV}
    {LENGTHSCALE nm}
    {TIMESCALE 1 ns}
    {Unit_Distance 1 um}
  }
  {PARAMETER
    {A1, Cell,
      {D2D Data}
      {WID Data}
      {Random Data}
    }
  }
  {PARAMETER
    {M1T, Interconnect,
      {D2D Data}
      {WID Data}
    }
  }
}
```

The following sections provide information on the syntax used to specify various variations in the SPDF file.

### Specifying Global or Die-to-Die Variations in SPDF File

To specify the global or die-to-die variations, use the following syntax in the SPDF file:

```
{PARAMETER {Parameter, Cell/Interconnect
  {D2D
    {Gaussian, (Mean, Sigma)}
  }
}}
```

### Specifying Random Variations in SPDF File

To specify the random variations, use the following syntax in the SPDF file:

```
{PARAMETER {Parameter, Cell/Interconnect
  {Random}}
```

```

        {Gaussian, (Mean, Sigma)}
    }
}

```

## Specifying Spatial Variations in SPDF File

To specify the spatial or with-in-die (WID) variation, use the following syntax in the SPDF file:

```

{PARAMETER {Parameter, Cell
    {WID
        {Spatial_Function}
        {Gaussian}
        {Mean_Polynomial, ( μ₀, q1, q2, q3, q4)}
        {Sigma_Polynomial, ( σ₀, r1, r2, r3, r4)}
        {Exponential_Spatial_Correlation, FCD, (k₀, k₁)}
    }
}}

```

Where,

$\mu_0$  : mean value

$\sigma_0$  : sigma or standard deviation.

$k_0, k_1$ : Constant factors that indicates the inverse relationship between correlation and separation of two devices.

FCD (Full-Correlated Distance): Size of grid for spatial correlation variation.

The values  $q_n, r_n, k_0$  and  $k_1$  are supplied by the foundry.

The mean and variation of process parameters can be modeled as the location of the device.

$$\mu_n = \mu_0 + q_{1n}x + q_{2n}x^2 + q_{3n}y + q_{4n}y^2$$

$$\sigma_n = \sigma_0 + r_{1n}x + r_{2n}x^2 + r_{3n}y + r_{4n}y^2$$

Therefore if mean and sigma values are location independent, you can set  $q_n$  and  $r_n$  to 0. However the values for  $k_0$  and  $k_1$  should always be specified.

Spatial Correlation between two devices is modeled as function of distance.

$$\rho_n = \langle 1 + k_0 \rangle e^{k_1 d} - k_0$$

## Sensitivity-Based SPEF (S-SPEF) File

The software requires an S-SPEF file to perform sensitivity-based extraction. Sensitivity-based extraction captures variations in interconnects. In sensitivity based extraction, in addition to nominal values of resistances and capacitances, the software also extracts the sensitivities of R and C to different interconnect process parameters.

Sensitivity-based extraction uses the variations in the following parameters:

- Metal width
- Metal thickness
- Dielectric layer thickness
- Resistivity
- Via
- Dielectric constant

The process of generating S-SPEF file is as follows:

1. Prepare a sensitivity techfile using TechGen. For more information, see *QRC TechGen Reference Manual*.
2. Set the variable to set sensitivity-based extraction in QRC command file and run extraction. For more information on running standalone QRC, see *QRC Extraction User Manual*.

## Loading the S-SPEF File

To load the S-SPEF file for SSTA, complete the following step:

```
spefIn -sspef SSPEF_file
```

**Note:** When you use the S-SPEF file, set the `-useNetSens` parameter in the `setDelayCalMode` command for delay calculation to consider interconnect variations.

## SSTA Flows

You can perform block-based or path-based SSTA. By default the software runs block-based analysis.

- Block-based SSTA

In block-based SSTA, the software analyzes the entire design statistically. The software uses the timing graph in leveled manner and propagates the arrival times from input ports to all end-points. The reports generated in block-based SSTA are specific to end-points.

- Path-based SSTA

In path-based analysis, the software first identifies the potential critical paths using static analysis and then runs statistical analysis on the selected paths. Path-based SSTA uses path-based slew propagation, which makes path-based analysis more realistic (by removing pessimism in worst slew propagation). The reports generated in block-based SSTA are similar to those generated using STA.

## Running Block-Based SSTA

To run block-based SSTA, complete the following steps:

1. Set the analysis mode to statistical.
2. Load the design.
3. Read the SPEF file. Optionally read in the S-SPEF file to include the interconnect parasitic information.
4. Read the SPDF file. For more information on SPDF file, see [Statistical Parameter Distribution Format \(SPDF\) File](#) on page 975.
5. Generate the timing report for top n endpoints.

### Example 28-1 SOC Command File for Running Block-Based SSTA

```
setAnalysisMode -timingEngine statistical
loadConfig configFile
spefIn spefFile
read_spdf spdfFile
setDelayCalMode -engine signalStorm
report_timing -nworst n > rptFile
report_timing -summary > summaryRptFile
```

## Running Path-Based SSTA

In path-based analysis, the software first identifies the potential critical paths using static analysis and then runs statistical analysis on the selected paths. You can then generate path-based timing reports for your design.

To run path-based SSTA, complete the following steps:

1. Set the analysis mode to statistical.
2. Load the design.
3. Read the SPEF file.
4. Generate the timing report using the `-retiming ssta` parameter in the [report timing](#) command.

### Example 28-2 SOC Command File for Running Path-Based SSTA

```
setAnalysisMode -timingEngine statistical
loadConfig configFile
spefIn spefFile
read_spdf spdfFile
setDelayCalMode -engine signalStorm
report_timing -retime ssta -max_path num
```

## SSTA Outputs

You can generate timing reports for block-based or path-based SSTA.

### Block-Based SSTA Report

In a block-based SSTA report, the software reports data for all end points. An endpoint is an input to a register (flip-flop, or latch) or a primary output. [Example 28-3](#) on page 982 shows a sample block-based report. By default the following data is generated for block-based analysis:

- Path-specific information such as slack, arrival time and required time. This information is generated by default in the report.
- Process parameter sensitivities such as cell sensitivities

**Note:** In block-based flow, the arrival times and required times are reported for end points and not for paths. To report values for a path, use the `-from`, `-from_rise`, `-from_fall`, `-through`, `-through_rise`, or `-through_fall` parameters in the [report\\_timing](#) command.

Optionally, you can generate reports for the following data:

- Joint slack information for all endpoints. [Example 28-4](#) on page 982 shows a joint PDF report. To generate joint PDF information, use the `-ssta_jpdf` parameter in the `report_timing` command.
- Criticality of nodes. To report criticality of nodes, use the `-ssta_crticality` parameter in the `report_timing` command. [Example 28-5](#) on page 983 shows a report for criticality of nodes.

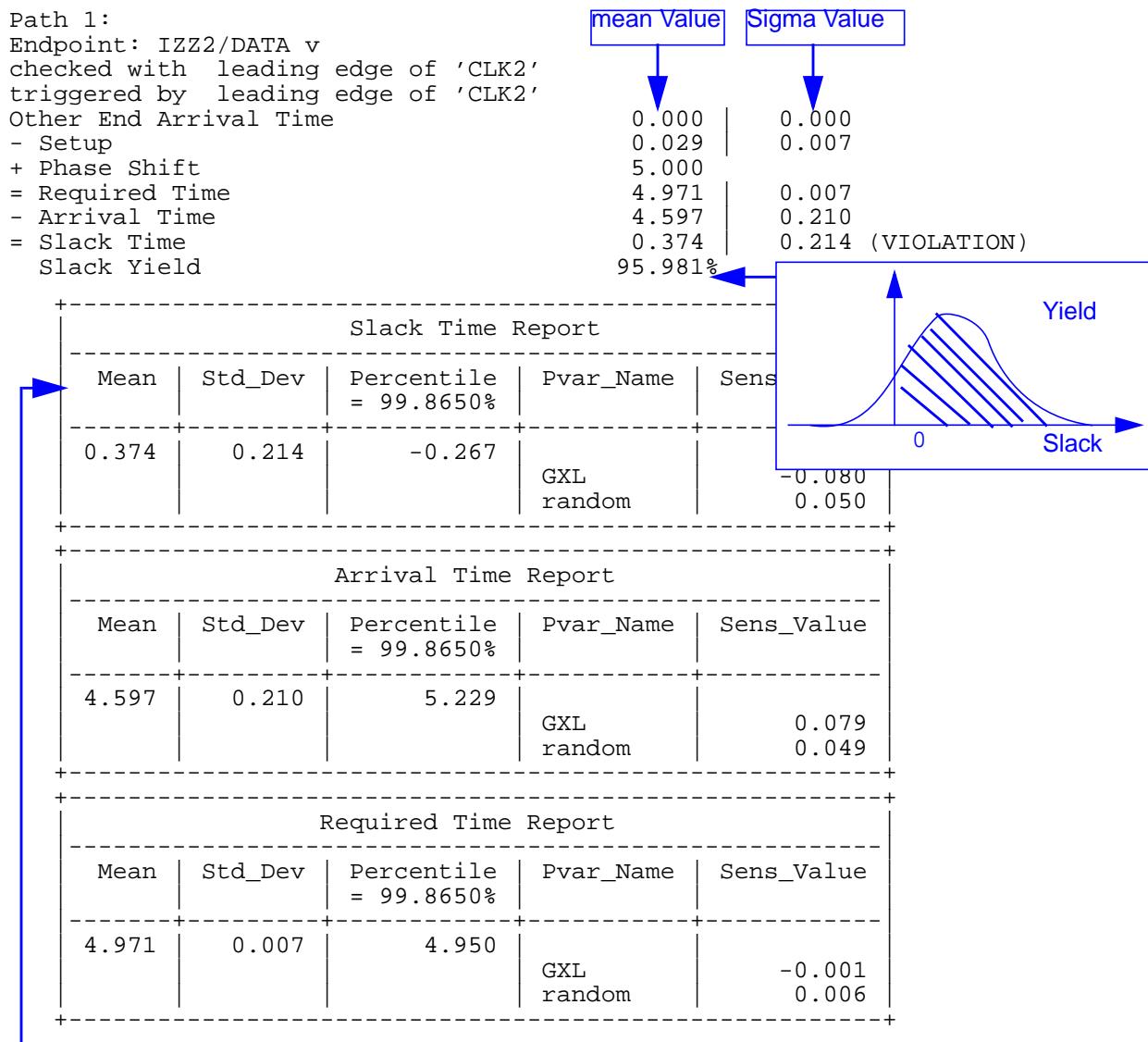
Some other `report_timing` command parameters that you can use for specific information in a block-based SSTA report are:

- `-max_paths num_paths`
- `-nworst value`
- `-max_points value`
- `-early | -late`
- `-rise | -fall`
- `-sort_slack_by {ssta_yield | sstaViolation | ssta_NSigma | ssta_path_criticality} -max_paths 100`

## Encounter User Guide

### Statistical Static Timing Analysis

#### Example 28-3 Block-based SSTA Report



Mean: Mean of path slack

Std\_Dev: Standard deviation of path slack

Percentile: 3 sigma value

Pvar\_Name: Process parameters. Here random is with-in-die parameters.

Sens\_Value: Sensitivity of path slack to process parameters

#### Example 28-4 Joint PDF Report

SSTA Sigma Multiplier (N) (option -ssta\_sigma\_multiplier) = -3.000

+-----+  
| Joint Distribution of Slacks |  
+-----+

# Encounter User Guide

## Statistical Static Timing Analysis

---

Mean	Std_Dev	Mean+N*Std_Dev	Timing_Yield	Pvar_Name	Sens_Value
0.373	0.212	-0.263	96.078%	XL random	-0.080 0.044

### Example 28-5 Criticality of Nodes

Criticalities				
Pin Name	Level	Criticality	Slack(Mean-3*Std_Dev)	
XPECTOP/MEM_TOP/PMEM/i_328314071/E	35	1.000	1.396	
XPECTOP/MEM_TOP/PMEM/i_187112208/Z	34	1.000	1.396	
XPECTOP/MEM_TOP/i_139340/B	5	1.000	1.459	
inscan_mux_1/inscan_mux/Z	24	1.000	1.077	
XPECTOP/MEM_TOP/PMEM/i_187112208/B	33	0.999	1.396	
XPECTOP/MEM_TOP/PMEM/i_328913954/Z	32	0.999	1.396	
XPECTOP/MEM_TOP/i_19328/B	3	0.998	1.472	
XPECTOP/MEM_TOP/i_29329/B	3	0.998	1.472	
XPECTOP/MEM_TOP/i_49331/B	3	0.998	1.472	
XPECTOP/MEM_TOP/i_1114/A	3	0.998	1.472	
XPECTOP/MEM_TOP/i_014599/Z	2	0.998	1.472	
XPECTOP/MEM_TOP/PMEM/dout_reg_16/D	39	0.985	1.596	
XPECTOP/MEM_TOP/PMEM/i_2672/Z	38	0.985	1.596	
XPECTOP/PROC_TOP/CONTROLLER/i_1187/A	9	0.851	1.473	

### Path-Based SSTA Report

Path-based SSTA report contains the minimum statistical distribution slack for all selected paths (JPDF). Each path is reported based on the worst slack. In path-based report, the paths are sorted based on their criticality probability.

The following example command generates path-based report with specified format:

```
report_timing -retime ssta -format {instance cell arc retime_delay retime_slew
retime_delay_sensitivity arrival}
```

### Path-Based SSTA Report

```
Path 1: VIOLATED Setup Check with Pin IZZ2/CLK
Endpoint: IZZ2/DATA (v) checked with leading edge of 'CLK2'
Beginpoint: IN1      (v) triggered by leading edge of 'CLK2'
Other End Arrival Time    0.000 | 0.000
- Setup                  0.029 | 0.007
+ Phase Shift             5.000
= Required Time           4.971 | 0.007
- Arrival Time            4.526 | 0.217
= Slack Time              0.445 | 0.220
                           Clock Rise Edge          0.000
```

# Encounter User Guide

## Statistical Static Timing Analysis

---

```
+ Input Delay          0.000
= Beginpoint Arrival Time 0.000
```

Instance	Cell	Arc	Retime Delay	Retime Slew	Retime Delay Sensitivity	Arrival Time
IA1	IVX1L	IN1 v	0.000	0.100		0.000
IA1	IVX1L	A v -> YB ^	0.353	0.463	XL 0.006 random 0.014	0.000 0.353
...						
IA11	IVX1L	A v -> YB ^	0.000	0.043	XL 0.000 XL 0.001 random 0.002	4.436
IA11	IVX1L	A v -> YB ^	0.054	0.045	random 0.002	4.489
IA12	IVX1L	A ^ -> YB v	0.000	0.045	XL 0.000 XL 0.001 random 0.002	4.489
IA12	IVX1L	A ^ -> YB v	0.037	0.024	XL 0.001 random 0.002	4.526
					random 0.002	
IZZ2	DFX1L		0.000	0.024	XL 0.000	4.526

### Slack Time Report

Mean	Std_Dev	Percentile = 99.8650%	Pvar_Name	Sens_Value
0.445	0.220	-0.216	XL random	-0.080 0.069

### Arrival Time Report

Mean	Std_Dev	Percentile = 99.8650%	Pvar_Name	Sens_Value
4.526	0.217	5.177	XL random	0.079 0.069

### Required Time Report

Mean	Std_Dev	Percentile = 99.8650%	Pvar_Name	Sens_Value
4.971	0.007	4.950	XL random	-0.001 0.006

Path 2: VIOLATED Setup Check with Pin IZZ2/CLK  
 Endpoint: IZZ2/DATA (v) checked with leading edge of 'CLK2'  
 Beginpoint: IN4 (v) triggered by leading edge of 'CLK2'  
 Other End Arrival Time 0.000 | 0.000

## SSTA Correlation With Monte-Carlo Analysis

To ensure the accuracy of your results, you can correlate the SSTA results with the Monte-Carlo Analysis results. To perform Monte-Carlo analysis, complete the following steps:

1. Generate N sample of process parameters using their distributions.
2. Run Spice for each set of parameters.
3. Calculate delay histogram from N-samples.

Monte-Carlo analysis can be performed in a circuit simulator (e.g. spice, spectre), where each delay is calculated using spice simulation.

**Encounter User Guide**  
Statistical Static Timing Analysis

---

---

## Extracting Timing Models

---

- [ETM Overview](#) on page 988
- [ETM Inputs](#) on page 992
- [Guidelines for Generating ETMs](#) on page 993
- [ETM Generation Flow](#) on page 995
- [ETM Outputs](#) on page 1000

## ETM Overview

Encounter provides a mechanism to generate extracted timing models (ETM) in a hierarchical design flow. An ETM is the timing context derived for a digital circuit, which can be used in static timing analysis (STA).

Use of ETM in STA provides the following advantages:

- Reduces the memory requirements and improves the run time.
- Provides a mechanism to hide the proprietary implementation details of the block from a third party.
- Enables you to share the extracted timing models for blocks with other designers working on different parts of the design at the same time.
- Provides faster convergence of timing results for large design databases by providing the actual timing context of the lower-level module without requiring the software to analyze the complete logic of the module with each run.

During ETM generation, the software does not use boundary conditions, such as input transitions, output loads, input delays, output delays or clock periods. Therefore, you do not need to extract the models again if the boundary conditions change at a later stage in development. The software does consider the operating conditions, wire load models, annotated delays or loads, and RC data on internal nets defined for the design during ETM generation. Therefore, you do need to extract the models again if any of these elements change during a design phase.

An accurate ETM preserves the worst-case behavior of the original circuit. You can use ETM to reproduce any timing violation that occurs in the original circuit. An ETM accurately represents critical paths that change with respect to a change in input slew value. In addition, an ETM preserves the self-loop timing checks such as minimum period and minimum pulse width checks.

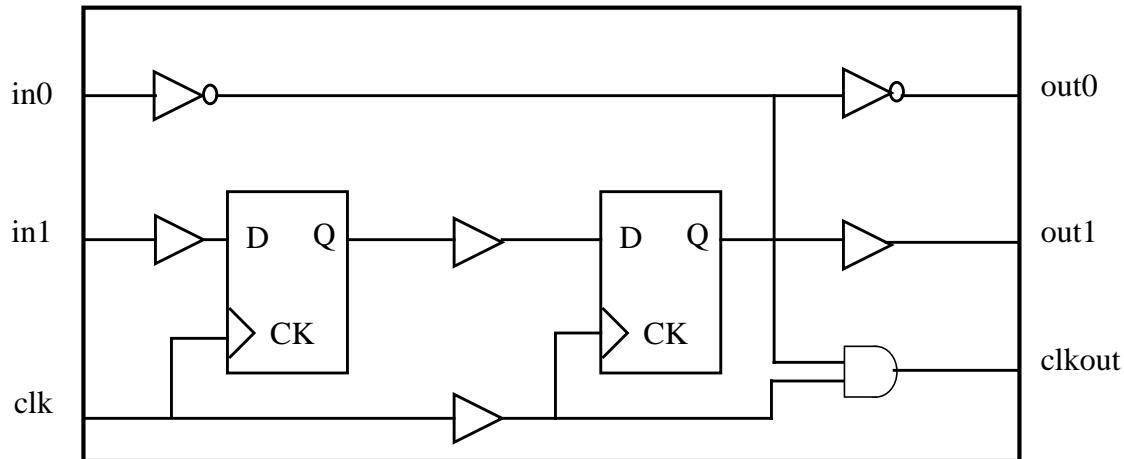
An ETM provides a timing representation by creating timing arcs for the following interface paths:

- Input to the register
- Input to output
- Register to output

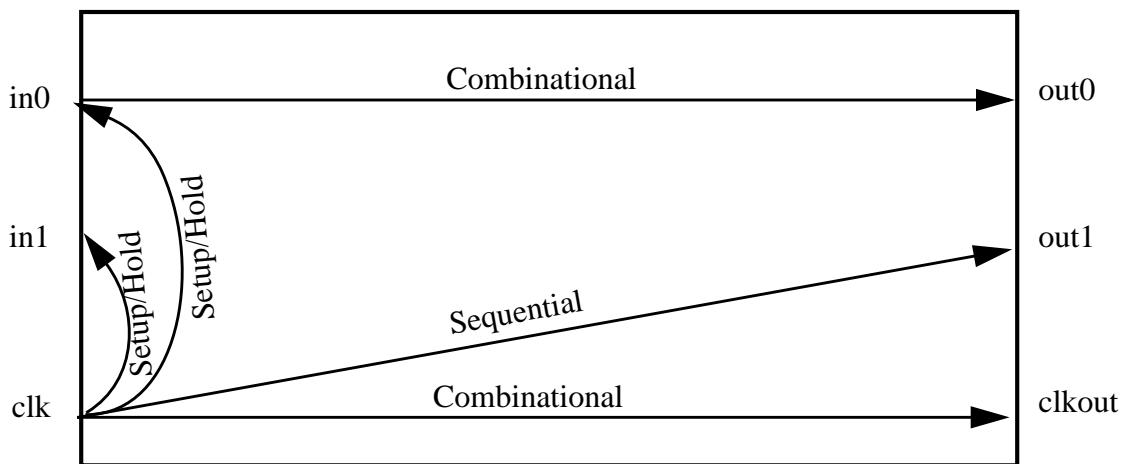
**Note:** Register to register paths are ignored because they do not affect the interface path's timing.

For example, consider the digital circuit shown in [Figure 29-1](#) on page 989.

**Figure 29-1 Gate-Level Netlist Representation of a Digital Circuit**



The ETM representation of gate-level netlist shown in the figure above is as follows:



## Using ETMs in Different Timing Analysis Modes

Use of ETM models for setup or hold analysis depends on the mode of timing analysis - Single, On-chip variation (OCV), or best-case worst-case (BcWc).

In single mode, the software uses late and early paths from a single corner for setup and hold analysis. The timing model that you generate in single mode has late and early paths from a single corner. Therefore, the timing model that you generate for setup is the same as that generated for hold analysis, and you can use the same model for setup or hold analysis.

In OCV mode, the software uses late paths from max corner and early paths from min corner for setup and hold analysis to generate the timing model. Therefore, the timing model that you generate for setup is the same as that generated for hold analysis, and you can use the same model for setup or hold analysis.

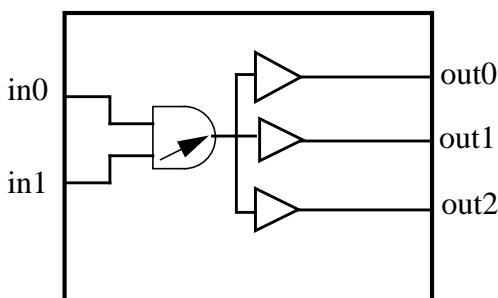
In BcWc mode, the software uses late and early paths from max corner for setup analysis and late and early paths from min corner for hold analysis. The model generated during setup analysis can not be used for hold analysis and vice versa because the software performs setup and hold analysis in different process corners.

## Limitation of Timing Models

Timing models have the following limitations:

- An ETM does not preserve logical behavior of the pins. Therefore they do not contain any conditional arcs.
- Slew propagation from side paths may be inaccurate. The software assumes a single value for the input slew of ports during the extraction process. The software then propagates the input slews and calculates delays and uses these delays for the associated timing arc in the extracted model

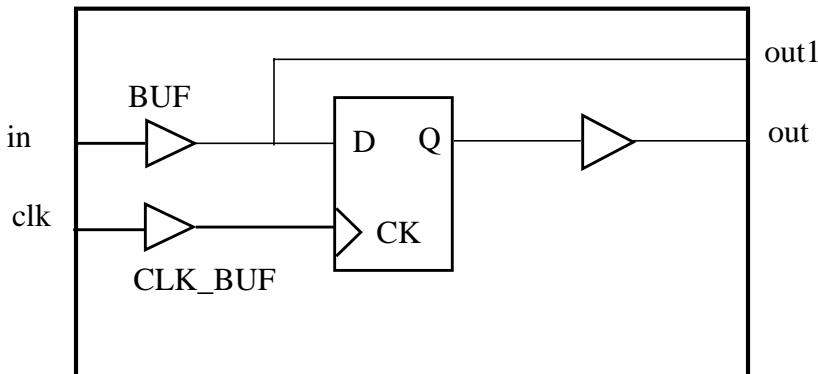
For example, consider the circuit in the figure below. The software assumes that the input slews at both inputs of the AND-gate have the same value. Therefore the software propagates the worst slew value from B->Y transition. However, at the top-level, input slews can be different. Therefore the usage of worst case value from B->Y transition is not correct.



- The check value for the arc is extracted based on the current load. Therefore the output load dependent check paths do not have the load dependency in the extracted model.

For example, consider the circuit in the figure below. In this circuit, the slew at point D depends on the loading at port out. Therefore the check arc value depends on the out

loading. Extracted model do not support the 3D arcs for setup checks. Therefore in this case, the check arc value is calculated using the current loading at out.



- The software does not preserve the three state enable or disable arcs in an ETM. The software evaluates the three state enable or disable expressions and three state arcs with transitions to and from the high impedance state Z (0->Z, 1->Z, Z->0, or Z->1) are transformed to combinational arcs with transitions 0->1, 1->0, 1->0, or 0->1, respectively.

## ETM Inputs

Generation of ETM in Encounter requires the same inputs that you need to perform STA. To generate ETM, load the following data in Encounter

- Design netlist
- Timing libraries
- Timing constraints
- RC data of the nets

## Guidelines for Generating ETMs

You can use several parameters in the `do_extract_model` command and a timing global to define the characteristics of the ETM.

- Use the `-input_slew`, `-clock_slew` and `-output_load` parameters to specify slews at inputs and load at the output pins. If you do not specify these parameters, the software determines the characterization points from the interface elements of the design.

The software characterizes all the check arcs for the slew points as follows

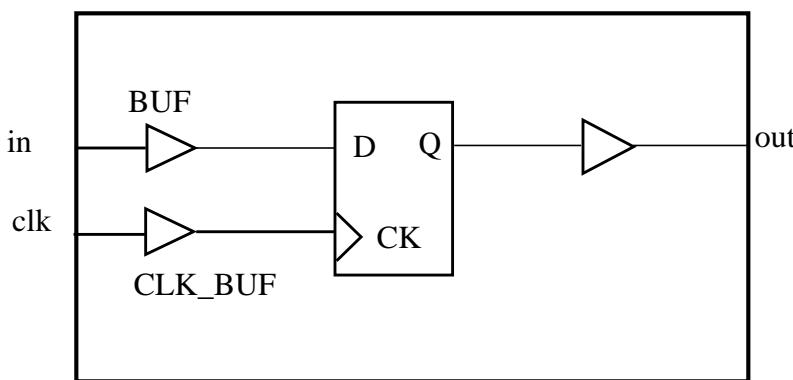
- ❑ Uses the reference slew points from the slew index of the first element after the port to which related pin is connected.
- ❑ Uses the signal slew points from the slew index of the first element after the port to which constrained pin is connected.

The software characterizes all the sequential arcs as follows:

- ❑ Uses the input slew from the slew index of the first element just after the clock port.
- ❑ Uses the output load from the load index of the last element just before the output port.

For example, [Figure 29-1](#) on page 993 shows a circuit used for generating ETM.

**Figure 29-2 Circuit for model extraction**



For this circuit, the library file has the characterization points as follows:

```
Cell (BUF)
{
    ...
    timing ()
```

```
index_1 ("0.0500, 1.4000, 4.5000");
index_2 ("1.0500, 6.5000, 10.0000");
...
}
Cell (CLK_BUF)
{
...
timing ()
index_1 ("1.0500, 2.4000, 3.5000");
index_2 ("0.0500, 4.5000, 5.0000");
...
}
```

If you extract the model, without specifying the `-input_slew`, `-clock_slew` and `-output_load` parameters, the generated library file has the following characterization point for check arc between `clk` and `in`.

```
index_1 ("0 0.0500, 1.4000, 4.5000");
index_1 ("0 1.0500, 2.4000, 3.5000");
```

Similarly the library file has the following characterization points for sequential arcs between `clk` and `out`:

```
index_1 ("0 1.0500, 2.4000, 3.5000");
index_2 ("0 1.0500, 6.5000, 10.0000");
```

- Use the `-resolution` and `-tolerance` parameters to set the run time and accuracy of model extraction. Specifying greater resolution and tolerance results in less accurate models but improves the run time. The higher the tolerance, the faster is the extraction time.

For example, you specify three characterization points for input slews and three characterization points for load values. In this case, the software uses a 3x3 table of slew and load indexes in the library file. However, if you also specify a range of resolution and tolerance, the software uses these values to simplify the table of slew and load indexes. If the middle slew and load values can be interpolated from the boundary values within the range of tolerance and resolution specified, then the software simplifies the 3x3 table to a 2x2 table. Therefore the software characterizes lesser number of slew and load values, thereby improving the run time of model extraction at the cost of accuracy.

- Use the `timing extract model slew propagation mode` global to specify the type of slew propagation to use for generating ETM. You can specify worst slew propagation or path-based slew propagation.

In worst slew propagation mode, the software propagates the worst slew of all the incoming arcs at a converging point for extracting the arcs. This is the recommended mode.

In path based slew propagation mode, the software propagates the actual slew for the path elements for extracting the arcs. This is the default mode.

## ETM Generation Flow

Encounter supports two types of model extraction, blackbox and greybox models.

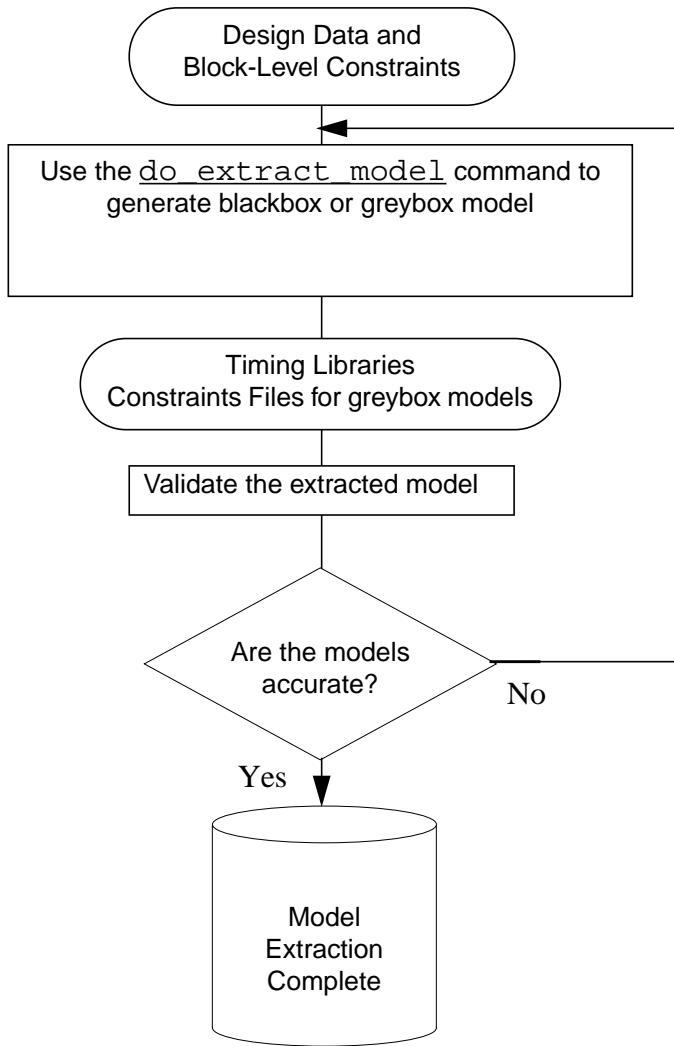
Both blackbox and greybox models preserve the worst-case behavior of the original circuit. The grey box models in addition retain internal pins in the extracted model. This enables better handling of constraints and latches.

Graybox models provide visibility into the internal pins in the circuit. Creating internal pins allows for accurate modeling of latch timing borrowing behavior and path exception handling. For greybox models, the software also generates a constraint file along with the extracted model. This constraints file is loaded incrementally when you use the extracted models at the top-level.

Graybox models have the following advantages:

- Supports arbitrary assertions.
- Guarantees that the model size is smaller than original netlist.
- The graybox model is clock-context independent as it does not adjust any multicycle path in the model as well it preserve the latch structure. That is, a greybox model is valid for clock waveforms that are different from the ones used to build the model.
- In greybox modeling, you do not need to re-extract the model if the path exceptions change. You can modify the generated constraint file instead. The greybox flow proves useful in the early stage of design where path exceptions are constantly changing.

The following figure shows the flow for generating ETMs.



The following procedure shows how to perform model extraction for black-box models:

1. Load the design data including the timing library on which you want to perform model extraction.
2. Specify the type of slew propagation to use for generating ETM. The worst slew propagation mode is recommended:  
`set_global timing_extract_model_slew_propagation_mode worst_slew`
3. Perform model extraction:

```

do_extract_model extracted.lib \
-lib_name extracted_model \
-cell_name extracted_cell \

```

```
-tolerance 0.0 \
-verilog_shell_file top.v \
-verilog_shell_module test_top
write_model_timing -type arc netlist.rpt
```

The above steps will generate the following outputs:

- Extracted timing library (`extracted.lib`)
- Verilog wrapper (`top.v`), which will be used to instantiate the extracted model
- Interface timing characteristics of the original design (`netlist.rpt`)

## Validating the Generated Model

You can validate the timing models for their accuracy and coverage. An accurate ETM preserves the worst-case behavior of the original circuit. Encounter provides automated mechanism for validating the extracted models.

The following commands are used for model validation:

- `write_model_timing`

Writes the interface timing characteristics of the design in the specified timing model report file. The report file contains the following sections:

- ❑ Worst-Case Arc/Slack: Contains details about the extracted arcs and the slack or delay across them depending on the argument specified with the `-type` option.
- ❑ Transition Time: Contains information about transition time at ports.
- ❑ Capacitance: Contains the capacitance values for the ports.
- ❑ Design Rules: Contains the design rules applied to the ports.

- `compare_model_timing`

Compares two reports that contain interface timing characteristics generated by the `write_model_timing` command.

This command compares the interface timing characteristics report generated by using the `write_model_timing` command on the original netlist with the report generated by using the `write_model_timing` command on the model extracted by the `do_extract_model` command for the same design.

To validate the extracted model, complete the following steps:

1. Ensure that the interface timing characteristics report file was generated by using the original gate-level netlist and constraints using the `write_model_timing` command.
2. Load the output files generated during model extraction, which include:
  - Extracted timing model (`extracted.lib`)
  - Verilog wrapper (`top.v`)
  - Design constraints (needed for greybox models)

**Note:** In case of greybox models, load the additional constraints file generated during model extraction by the `do_extract_model` command.

3. Use the `write_model_timing` command on the extracted model (timing library) to generate a model report.

```
write_model_timing -type arc model.rpt
```

4. Use the `compare_model_timing` command to compare reports generated during model extraction and the report generated in step 4:

```
compare_model_timing  
-ref netlist.rpt \  
-compare model.rpt \  
-outFile diff.rpt \  
-percent_tolerance 2 \  
-absolute_tolerance 0.003
```

## Reducing the Size of GreyBox Models

The software uses heuristic techniques to reduce the model size by further preserving some additional pins in a greybox model. To reduce the model size, use the `-gain` parameter in the `do_extract_model` command. The `-gain` parameters retains some existing internal pins in the circuit. These pins are called anchor points. Anchor points, if removed, might lead to an increase in model size. The software uses the numbers of input or output delay arcs to estimate the size of the model. The software assumes that all the delay arcs make equal contribution to the final model size. To select an anchor point, the software aims to minimize the number of delay arcs. Another criteria for selecting anchor point is the fact that the delay arcs are characterized with respect to a minimum number of input slew values. Therefore the software processes the incoming delay arc on each pin before removing them.

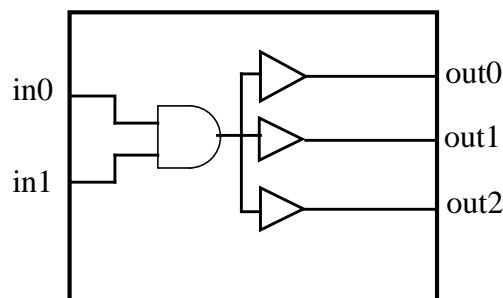
The gain value is calculated as follows:

$$\text{Gain} = (\text{Number of incoming delay arcs} * \text{Number of outgoing delay arcs}) - (\text{Number of incoming delay arcs with anchor point} + \text{Number of outgoing delay arcs with anchor point})$$

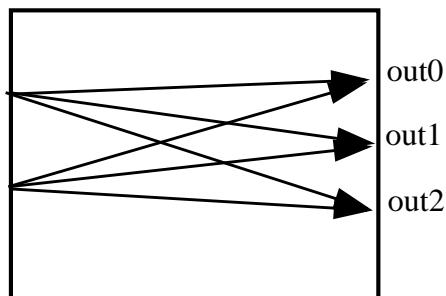
For example, consider the circuit shown in [Figure 29-3](#) on page 999. If the software does not preserve an anchor point, the resulting 6 delay arcs are as shown in [Figure 29-4](#) on page 999.

However if the software considers the AND gate output pin as an anchor point and creates an internal pin, the total number of arcs required is 5 as shown in [Figure 29-5 on page 999](#). There are 2 incoming arcs and 3 outgoing arcs at AND-gate output pin. Therefore the gain in this case is 1. Any pin that has gain equal or greater than value that you specify using the `-gain` parameter is considered by the software for creating an anchor point.

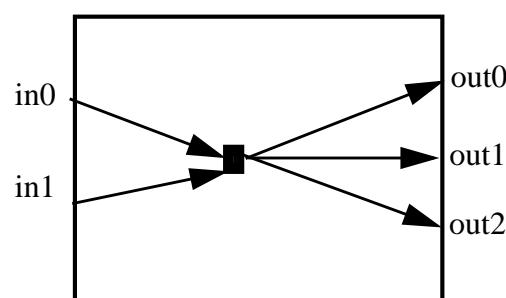
**Figure 29-3 Circuit for generating ETM**



**Figure 29-4 Extracted Model Without Internal Pin as Anchor Point**



**Figure 29-5 Extracted Model With One Anchor Point**



## ETM Outputs

ETM flow generates the following output:

- [Timing Library File](#) on page 1000
- [Timing Constraints Files](#) on page 1008

## Timing Library File

The timing model is generated as a .lib file. The following sections provide information on how these scenarios are handled in .lib file:

- [Boundary Nets](#)
- [Internal Nets](#)
- [Timing Paths](#)
- [Minimum Pulse Width and Minimum Period](#)
- [Path Exceptions](#)
- [Constants](#)
- [Gating Checks](#)
- [Annotated Delays and Slews](#)
- [Design Rules](#)
- [Generated Clocks](#)

### Boundary Nets

Boundary nets are the nets that are directly connected to the input or output ports of the block. Therefore the RC data for the boundary nets changes if the external context of the block changes at any point in the design phase. To generate context independent models, you can exclude the spef or dspef data for boundary nets from delay calculation. You can then stitch a separate file for spef or dspef data for boundary nets when you instantiate the ETM.

The software, by default, considers the RC data while extracting the timing model. ETS does not write out a spef file for boundary nets that can be used at top level. To use RC data with timing models, create the top level spef file for boundary nets or use the default behavior to consider the boundary nets RC data for model extraction.

## Internal Nets

Internal nets are the nets that drive or are driven by an internal instance pin of the block. Internal nets are context independent because they are only connected to internal instance of the block. Therefore the software uses the RC data defined for the internal nets for delay calculation and adds this information to the extracted paths. If you do not define any RC data for internal nets, then the software uses the wire load models to calculate the delay. If you use the set\_load or set\_resistance command to annotate a load or resistance, then the annotated value overrides the annotated spf or the wire load model.

If you use the set\_annotation\_delay command or SDF annotation to annotate the delay on internal nets, the software uses the annotated delay instead of the calculated delay.

## Timing Paths

The following timing paths are included in an ETM:

- [Input to Register Paths](#)
- [Register to Output Paths](#)
- [Register to Output Paths](#)

**Note:** Register to register paths are ignored because they do not affect the interface path's timing.

### Input to Register Paths

Input to register paths are the paths from an input port to a register. In an ETM, the input to register paths are represented by equivalent setup or hold checks. These checks contain the calculated delay from the input port to the register, the setup or hold value of the library cell and the delay from a clock source to the clock pin of the register. The delay for the setup or hold checks is the function of the transition on the input port and the transition at the clock source

Setup check delay = delay (input to register) + delay (setup value of register) - delay (clock source to clock pin)

Hold check delay = delay (input to register) - delay (hold value of register) - delay (clock source to clock pin)

If multiple clocks reach a register, then the software extracts separate setup or hold arc for each clock source.

## Register to Output Paths

The register to output paths are the path from a register to an output port. These paths are a combination of a trigger arc of starting register and the combinational delay from sink of trigger arc to the output port. Therefore the software extracts an equivalent trigger or sequential arc for register to output paths.

The delay for these arcs is a function of the slew at the clock source and the capacitance at the output port. The delay of the arc is calculated as follows:

Sequential arc delay = delay (clock source to clock pin of register) + delay (register clock pin to out port)

The software generates two arcs representing longest and shortest path because the generated ETM can be used for both max and min analysis. The software generates different types of arcs for different valid clock edges such as rising\_edge or falling\_edge.

## Input to Output Paths

The input to output paths are the paths from an input port to an output port. These are combinational paths. Therefore the software generates equivalent combinational arc for these paths.

The delay for these arcs is a function of the slew at the input port and the capacitance at the output port. The delay for the arc is calculated as follows:

Combinational arc delay = delay (delay of all elements in the path)

The software generates two combinational arcs representing longest and shortest path because the generated ETM can be used for both max and min analysis. In case a path does not exist for a particular transition (rise or fall), the software generates half unate arcs such as `combinational_rise` or `combinational_fall`. The timing sense for the arc depends on the function of worst (early or late) paths.

## Minimum Pulse Width and Minimum Period

During timing model extraction the software transfers the minimum pulse width and minimum period constraints defined at the clock pin of the registers to the clock source pins. There might be several types of registers in the fanout of a clock source. Therefore, while transferring the minimum pulse width to the clock source, the software uses the worst minimum pulse width constraint value present on the fanout registers.

The software considers the delay and slew propagation differences of the clock network for rise and fall transitions while extracting the minimum pulse width. This is needed because some networks might have a significant difference in the rise and fall delays. So not considering these differences will cause differences in the minimum pulse width violations.

## Path Exceptions

False paths can only be modelled in greybox models. Blackbox models do not support extraction of false paths. In a greybox model, the software extracts the pins with false path or multicycle path assertions as internal pins. The software generates a constraint file for all internal pins that are extracted. You can then use this constraints file with the generated model to perform analysis.

**Note:** The software ignores the set\_max\_delay or set\_min\_delay constraints during model extraction. For an output port, the software extracts worst delay paths between two ports.

## Constants

The software propagates the case analysis and the constants on the netlist while extracting the model. The software does not consider the conditional arcs or paths that were disabled due to constants. If the constant value changes, you need to re-generate the timing model.

## Gating Checks

Clock gating checks are modeled as setup and hold checks between a clock pin and its enabling signal pin. Depending on the type of clock gating situation, setup and hold checks are inferred as follows:

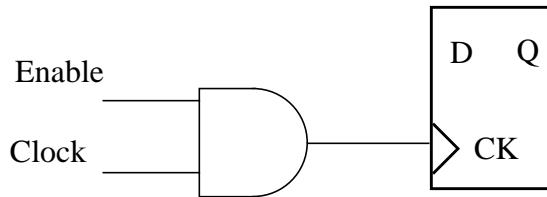
- A clock-gating setup check is inferred with respect to the edge of the clock signal that changes the state of the clock gate from a controlling state to a non-controlling state.
- A clock gating hold check is inferred with respect to the edge of the clock that changes from a non-controlling state to a controlling state.

### Simple Clock Gating with AND Gate

For a simple AND gate, the setup checks are done with respect to the rising edge of the clock signal, and hold checks are done with respect to the falling edge of the clock signal.

Figure 29-6 on page 1004 shows a simple clock gating with AND gate.

**Figure 29-6 Simple Clock Gating with AND Gate**



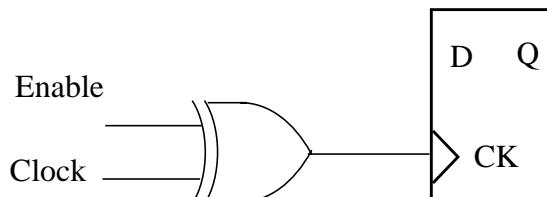
After extracting the clock gating check arc, the signal downstream the gate output is not propagated to the clock pin of the registers.

### No Clock Gating Logic

No setup or hold checks are inferred for clock gates with non-unate arcs, such as multiplexers and XOR gates because a clock signal cannot control the clock gate output.

[Figure 29-7 on page 1004](#) shows an example where there is no clock gating logic:

**Figure 29-7 No Clock Gating Logic**



### Annotated Delays and Slews

Model extraction uses the back annotated delay, slews, and the load information. This information is reflected in the extracted model.

#### ■ Annotated Delays

The annotated delays calculated using the SDF data or the `set_annotationed_delay` command override the delay value of the arc, which was calculated using the library or RC data. The output slew of the arc, however, is calculated using the library or RC data. In case of incremental delays, the delta delay is added to the calculated arc delay.

- Annotated Slews

Slews annotated using the `set_annotated_transition` command override the slew value calculated using the library or RC data. As a result, the downstream delay and slews are calculated using the annotated slew.

- Annotated Load

The annotated load overrides the pin capacitance defined in the library, and is used for delay calculation.

## Design Rules

Model extraction uses the design rules defined in the timing library. The worst (lowest value for `max` design rules and highest value for `min` design rules) values among all the fanout pins is used for the input ports and the worst values among all the fanin pins is used for the output ports.

If the design rule limits are defined at the pin and library level, the design rule is chosen from the pin level because the pin-level information overrides the cell-level information, which in turn overrides the library-level information. However, the design rules defined in the SDC file have the highest priority and override the library values.

**Note:** The annotated delays are generated with a particular context and remain true for that context only. Therefore, annotating the delays or slews makes the model context dependent. To create a context independent model, it is recommended not to annotate the SDF.

## Generated Clocks

Generated clocks defined in a hierarchical block are preserved in the generated ETM model. The following rules are applied for modeling generated clocks in extracted timing models:

- The pin on which the generated clock is specified is preserved as an internal pin in the timing model.
- The name of this internal pin is changed to the generated clock name.
- A `generated_clock` construct is written out, which contains the definition of the `create_generated_clock` constraint that created the generated clock in the original netlist.

For example, if the original netlist contained the following generated clock statement:

```
create_generated_clock -name gclk -source [get_ports clk] -divide_by 2 [get_pins  
buf1/A]
```

During extraction, the pin `buf1/A` will be preserved as an internal pin in the library as `gclk`. The extracted model will be generated as follows:

```
generated_clock (gclk) {  
    master_pin: clk;  
    divided_by: 2;  
    clock_pin: "gclk";  
}  
pin (gclk) {  
    clock: true;  
    direction: internal;  
    .....  
    .....  
}
```

## Handling Multiple Clocks on Same Pin

When multiple generated clocks are defined on a pin in the original netlist, the pin is preserved as an internal pin in the extracted model. The name of the pin is assigned based on the generated clock that was last defined on that pin in the original netlist. In the extracted model, there will be one `generated_clock` construct for each of the two clocks on that pin.

Consider the following example, which shows the constraints used in the original netlist to create generated clocks on the pin `buf1/A`:

```
create_generated_clock -name gclk1 -source [get_ports clk] -add -master_clock CLK  
-divide_by 2 [get_pins buf1/A]  
create_generated_clock -name gclk2 -source [get_ports clk] -add -master_clock CLK  
-divide_by 2 [get_pins buf1/A]
```

During extraction, the pin `buf1/A` will be preserved as an internal pin in the library with a name `gclk2`. The extracted model will be generated as follows:

```
generated_clock (gclk1) {  
    master_pin: clk;  
    divided_by: 2;  
    clock_pin: "gclk2";  
}  
generated_clock (gclk2) {  
    master_pin: clk;  
    divided_by: 2;  
    clock_pin: "gclk2";  
}  
pin (gclk2) {  
    clock: true;
```

```
direction: internal ;
.....
.....
}
```

## Handling Generated Clock on Multiple Pins

When a generated clock is defined on multiple pins, all those pins are preserved as internal pins in the extracted model. Pin names of these internal pins are derived as follows:

- The generated clock name is used as a prefix
- All the pins on which the generated clock is specified are assigned a number on an incremental basis
- The generated clock name and the assigned number are merged using the underscore “\_” character. For example, if the clock name is `clk`, the corresponding clock pins will be named as `clk_1`, `clk_2`, and so on.

Consider the following example, which shows a netlist with clock definitions on multiple pins:

```
create_generated_clock -name gclk1 -source [get_ports clk] -add -master_clock CLK
-divide_by 2 [get_pins {buf1/A buf2/A}]
```

During extraction, the pin `buf1/A` will be preserved as an internal pin in the library as `gclk2`. The extracted model will be generated as follows:

```
generated_clock (gclk1) {
    master_pin: clk;
    divided_by: 2;
    clock_pin: "gclk1 gclk_1 ";
}
pin (gclk1) {
    clock: true;
    direction: internal;
}

pin (gclk1_1) {
    clock: true;
    direction: internal;
}
..
```

## Timing Constraints Files

For greybox models, along with the .lib file, the software generates two constraint files containing a subset of original constraints. The constraint files contain constraints that are required for:

- Standalone model validation. This constraint file contains boundary environment such input transitions and output loads, along with other constraints.
- Top level stitching with the extracted timing model. This is the top-level constraints file.

The software extracts the following constraints in the two constraints files:

- [set\\_false\\_path and set\\_multicycle\\_path constraints](#) on page 1008
- [set\\_disable\\_timing and set\\_case\\_analysis](#) on page 1008
- [create\\_clock and create\\_generated\\_clock](#) on page 1009
- [set\\_input\\_delay and set\\_output\\_delay](#) on page 1009
- [Design Rules](#) on page 1009
- [set\\_load, set\\_resistance and set\\_annotated\\_transition](#) on page 1009
- [set\\_annotated\\_delay and set\\_annotated\\_check](#) on page 1010
- [set\\_input\\_transition and set\\_driving\\_cell](#) on page 1010

### **set\_false\_path and set\_multicycle\_path constraints**

The arcs that you set using the `set_false_path` or `set_multicycle_path` constraints between ports or clocks are preserved as is and written out in the extracted constraints file. The software does not change port names or clock names.

The arcs that you set using the `set_false_path` or `set_multicycle_path` constraints from, through or to internal pins are written out with modified pin names. The software prefixes the pin names with tcl variables. You can use the tcl variables to change the instance names.

### **set\_disable\_timing and set\_case\_analysis**

The `set_disable_timing` and `set_case_analysis` constraints are not written out in the constraints file because the software does not extract the arc that was disabled using the `set_disable_timing` or `set_case_analysis` constraints.

## **create\_clock and create\_generated\_clock**

The software extracts the `create_clock` constraints that you have defined for pins or ports. The software prefixes the internal pin names with tcl variables. You can use the tcl variables to change the instance names.

The software generates the `create_generated_clocks` constraint as part of the `generated_clock` constraint.

## **set\_input\_delay and set\_output\_delay**

The software writes out the `set_input_delay` and `set_output_delay` constraints that you define for ports as is. The constraints that you define for internal pins are not required for timing models and therefore are not written out.

The `set_input_delay` and `set_output_delay` constraints are not written out for top-level constraint file. This is because for an input port the input delay is the delay of path from top level register or port to input port. Similarly for an output port the delay is relative to top-level register or port.

## **Design Rules**

The software does not write out any design rule constraints for the timing model. This is because, design rules are defined in the .lib files.

## **set\_load, set\_resistance and set\_annotated\_transition**

The software uses the `set_load`, `set_resistance` and `set_annotated_transition` constraints defined on the internal pins for delay calculation during extraction.

The software writes out the `set_load`, `set_resistance` and `set_annotated_transition` constraints defined on the port as is to the constraint file.

The `set_load`, `set_resistance` and `set_annotated_transition` constraints are not written out for top-level constraint file. This is because the software uses these values from the top-level environment.

## **set\_annotated\_delay and set\_annotated\_check**

The software uses the `set_annotated_delay` and `set_annotated_check` constraints for delay calculation during model extraction. Therefore the constraints are not written out to the constraint files.

## **set\_input\_transition and set\_driving\_cell**

The software writes the `set_input_transition` and `set_driving_cell` constraints as is to the constraint file.

The `set_input_transition` and `set_driving_cell` constraints are not written out for top-level constraint file.

---

## Optimizing Timing

---

- [Overview](#) on page 1012
- [Before You Begin](#) on page 1012
- [Results](#) on page 1013
- [Interrupting Timing Optimization](#) on page 1015
- [Performing Optimization Before Clock Tree Synthesis](#) on page 1016
- [Performing Post-CTS Optimization](#) on page 1020
- [Performing Postroute Optimization](#) on page 1023
- [Optimizing Power During optDesign](#) on page 1029
- [Using Useful Skew](#) on page 1030
- [Using Active Logic View for Chip-Level Interface Circuit Timing Closure](#) on page 1033
- [Optimizing Timing in On-Chip Variation Analysis Mode](#) on page 1033
- [Using Conformal Constraint Designer During Timing Optimization](#) on page 1037
- [Optimizing Timing Using a Rule File](#) on page 1040
- [Optimizing Timing When the Constraint File Includes the set\\_case\\_analysis Constraint](#) on page 1040
- [Using the Footprintless Flow](#) on page 1040
- [Using Cell Footprints](#) on page 1042
- [Viewing Added Buffers, Instances, and Nets](#) on page 1042
- [Default Naming Conventions](#) on page 1042

## Overview

Optimize timing after running trial route and extracting RCs, after clock tree synthesis (CTS), and after routing. The goals of timing optimization are to correct design rule violations (DRVs) and signal integrity (SI) violations and meet timing. Timing optimization includes the following operations, depending on the design stage:

- Adding buffers
- Resizing gates
- Restructuring the netlist
- Remapping logic
- Swapping pins
- Deleting buffers
- Moving instances
- Applying useful skew

Use the `setOptMode` command (or the **Design – Mode Setup – Optimization** form) to specify global timing optimization parameters. Use the `optDesign` super command (or the **Optimization** form) to optimize timing.

## Before You Begin

Before you optimize timing for the first time, complete the following steps:

1. Reserve placement space of more than five percent of the targeted final design utilization so that there is room to add buffers and remap the network to meet timing requirements.
2. If Assign statements exist in the Verilog® netlist, use one of the following procedures, which are equivalent, to enable optimization to work on Assign nets:
  - ❑ Specify `setDoAssign` on before loading the design data.
  - ❑ Specify `set rda_Input(assign_buffer) {1}` in the configuration file.
3. Specify default and detailed extraction scale factors by using the following commands:
  - ❑ `generateRCFactor`
  - ❑ `setRCFactor`

For more information, see the “[RC Extraction Commands](#)” chapter in the *Encounter Text Command Reference*.

4. Use one of the following methods to set input transitions for the high fanout nets for delay calculation:

- Set the input transitions in the configuration file.
- Run the `setInputTransitionDelay` command.

For more information, see [`setInputTransitionDelay`](#) in the “Delay Calculation Commands” chapter of the Encounter Text Command Reference.

5. Create and load footprints. (Optional)

You are not required to specify footprints. For more information, see “[Using the Footprintless Flow](#)” on page 1040.

## Results

After optimizing timing, the software appends the log file with the following information:

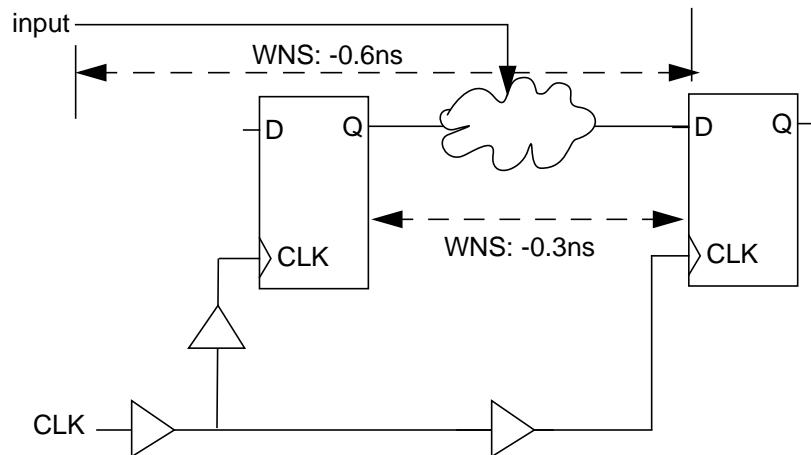
- Worst negative slack, total negative slack (TNS), and the number of failing (violating) paths. The software also reports hold violations if you specify the `-hold` parameter in post-CTS or postroute mode. It writes the values to the log file and writes reports to the working directory.

**Note:** The overall TNS and number of failing paths of a design might not be equal to the total of the TNS and failing paths of the individual path groups. This is because the TNS and number of failing paths are based on the end-point of the path and are not path based.

## Encounter User Guide

### Optimizing Timing

For example, the following figure has a register with two paths, one from a primary input with a slack of -0.6ns and other from another register with a slack of -0.3ns.



In this case the overall TNS will be -0.6ns with 1 violating path (end point-based). But the individual reg2reg TNS will be -0.3ns with 1 violating path and in2reg with a TNS of -0.6ns with 1 violating path. Therefore, the sum total of individual path group TNS is not the same as overall TNS.

- Number of `max_tran`, `max_cap`, and `max_fanout` violations
- Utilization (density)

If you specify path groups, the software produces a slack file and `tarpt` report for them. If you do not specify path groups, the software produces the following four violation reports:

- Register-to-register
- Input-to-register
- Register-to-output
- Input-to-output

The reports contain information about the following violations for the top 50 critical paths:

- Setup violations
- Hold violations
- DRVs (maximum capacitance, maximum transition, and maximum fanout violations)

The software generates the reports and saves them in the file specified by `optDesign -outDir` (or in the `timingReports` directory if `-outDir` is not specified).

## Encounter User Guide

### Optimizing Timing

The filenames are

- *designName\_preCTS\_pathGroup.tarpt*
- *designName\_postCTS\_pathGroup.tarpt*
- *designName\_postRoute\_pathGroup.tarpt*

The summary report has the following format:

optDesign Final Summary							
Setup mode	all	reg2reg	in2reg	reg2out	in2out	clkgate	
WNS (ns):	-0.491	-0.491	N/A	6.868	N/A	N/A	
TNS (ns):	-866.856	-866.856	N/A	0.000	N/A	N/A	
Violating Paths:	5273	5273	N/A	0	N/A	N/A	
All Paths:	69372	69343	N/A	29	N/A	N/A	

DRVs	Real		Total
	Nr Nets(terms)	Worst Vio	Nr Nets (terms)
max_cap	70(70)	-0.004	70(70)
max_tran	0(0)	.000	0(0)
max_fanout	0(0)	0	0(0)

Density: 53.455%

Routing Overflow: 0.00% H and 0.00% V

For more information on timing reports, see [timeDesign](#), in the “Timing Analysis (Common Timing Engine) Commands” chapter of the *Encounter Text Command Reference*.

## Interrupting Timing Optimization

To stop timing optimization use the Control-C key combination. On pressing Control-C, the Encounter software exits at a legal location and outputs the database in a “reasonable” state; that is, the database will be in a state that is useful for debugging purposes only, and not one that you should save and continue to use in the design flow.

For more information, see [Interrupting the Software](#) in the “Getting Started” chapter.

## Performing Optimization Before Clock Tree Synthesis

- [Correcting Violations in Pre-CTS Mode for the First Time](#) on page 1016
- [Performing Rapid Timing Optimization for Design Prototyping](#) on page 1017
- [Using Additional Pre-CTS Timing Optimization Parameters](#) on page 1017
- [Performing Incremental Pre-CTS Optimization](#) on page 1018
- [Changing Default Settings in Pre-CTS Mode](#) on page 1019

### Correcting Violations in Pre-CTS Mode for the First Time

- Before optimizing timing in pre-CTS mode, you must break all timing loops by disabling arcs in the constraint file. If you do not disable the arcs, the software cannot make a valid comparison of WNS between two different runs since it might not break the loops at the same point each time.

Use the following command:

```
set_disable_timing
```

For more information, see [set\\_disable\\_timing](#), in the “Timing Constraint Commands” chapter of the *Encounter Text Command Reference*.

- Use the following command to optimize timing:

```
optDesign -preCTS
```

- To repair DRVs only, use the following command:

```
optDesign -preCTS -drv
```

**Note:** By default, the `optDesign` does not correct fanout violations. To repair fanout violations, run the following command before `optDesign`, starting from the first call of `optDesign -preCTS` up to the last call of `optDesign -postRoute`:

```
setOptMode -fixFanoutLoad true
```

### Related Topics

To see where this step fits in the design flow, see [Place the Design and Run Pre-CTS Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

For more information, see

- [optDesign](#), in the “Timing Optimization Commands” chapter of the *Encounter Text Command Reference*

## Performing Rapid Timing Optimization for Design Prototyping

To optimize timing using low-effort mode for design prototyping, use the following commands:

```
setOptMode -effort low  
optDesign -preCTS
```

In low-effort mode, `optDesign` resizes gates and performs global buffer insertion, but does not restructure the netlist or repair DRVs.

## Using Additional Pre-CTS Timing Optimization Parameters

You can use the following `optDesign` features separately or in combination.

- To run optimization with useful skew, use the following commands:  

```
setOptMode -usefulSkew true  
optDesign -preCTS
```
- To run optimization on specific path groups, use the following commands:

```
clearClockDomains  
setClockDomains -fromType sourcePoint -toType destinationPoint  
optDesign -preCTS
```

The following source and destination points are supported for clock domains:

- input (source point only)
- output (destination point only)
- register
- all

**Note:** If you are using the Common Timing Engine (CTE), you cannot use `setClockDomains` parameters other than `-fromType` and `-toType`.

For example, to run optimization on register-to-register paths, use the following commands:

```
clearClockDomains  
setClockDomains -fromType register -toType register  
optDesign -preCTS
```

**Note:** If you specify a path group, `optDesign` preserves the path group setting. In the previous example, subsequent commands affect the register-to-register path group only. To set a new clock domain, specify `clearClockDomains`, followed by the definition of the clock domain you want to select.

For example, to reset the path group to input-to-register, use the following commands after you run `optDesign`:

```
clearClockDomains  
setClockDomains -fromType input -toType register
```

To specify all path groups, use the following commands:

```
clearClockDomains  
setClockDomains -all
```

See [setClockDomains](#) in the Timing Analysis (Common Timing Engine) Commands chapter of the *Encounter Text Command Reference* for a list of supported source and destination points for clock domains.

- To disable area reclaiming, use the following commands (`optDesign` reclaims area by default):

```
setOptMode -reclaimArea false  
optDesign -preCTS
```

## Performing Incremental Pre-CTS Optimization

Optimize timing incrementally to optimize setup times and area on critical paths. You can use the following features separately or together.

- To run incremental setup-only optimization, use the following command:  
`optDesign -preCTS -incr`
- To run incremental optimization with useful skew, use the following commands:  

```
setOptMode -usefulSkew true  
optDesign -preCTS -incr
```
- To run incremental optimization on specific path groups, use the following commands:

```
clearClockDomains  
setClockDomains -fromType sourcePoint -toType destinationPoint  
optDesign -preCTS -incr
```

For a list of supported source and destination points, see “[Using Additional Post-CTS Timing Optimization Parameters](#)” on page 1021.

**Note:** If you are using CTE, you cannot use the `setClockDomains` parameters other than `-fromType` and `-toType`.

For example, to run incremental optimization on register-to-register paths, use the following commands:

```
clearClockDomains  
setClockDomains -fromType register -toType register  
optDesign -preCTS -incr
```

- To disable area reclaiming, use the following commands (`optDesign` reclaims area by default):

## Changing Default Settings in Pre-CTS Mode

You can change or add parameters for the following commands that optDesign runs automatically:

[setAnalysisMode](#)

optDesign sets `-clkSrcPath false` and `-clockPropagation forcedIdeal` by default. You cannot override these values. You can add other parameters.

[setClockDomains](#)

optDesign uses the parameters you specify. The default is all path groups.

[setExtractRCMode](#)

optDesign sets the extraction mode to `default`. You cannot change this mode. Ensure that you set the appropriate extraction scale factor.

[setOptMode](#)

optDesign sets the following parameters:

■ `-drcMargin`

If you use `setOptMode -drcMargin`, the value you specify is added to a dynamically calculated, internal margin. For example, if you set a margin of `0.2` (20 percent), this multiplies the `max_cap` and `max_tran` SDC constraints by 0.8. The margin can be positive or negative. If you set a margin of `-0.2`, this multiplies the `max_cap` and `max_tran` SDC constraints by 1.20. optDesign writes the margin value to the log file.

■ `-holdTargetSlack`

If you use `setOptMode -holdTargetSlack`, the value you specify is added to a dynamically calculated, internal margin. optDesign writes the hold target slack value to the log file.

■ `-setupTargetSlack`

If you use `setOptMode -setupTargetSlack`, the value you specify is added to a dynamically calculated, internal margin. The default `-setupTargetSlack` value is 0. optDesign writes the setup target slack value to the log file.

[setTrialRouteMode](#)

You can add parameters, but you cannot override the default settings. optDesign sets the `-handlePreroute true` parameter.

## Performing Post-CTS Optimization

- [Correcting Violations in Post-CTS Mode on page 1020](#)
- [Performing Incremental Post-CTS Optimization on page 1021](#)
- [Changing Default Settings in Post-CTS Mode on page 1022](#)

### Correcting Violations in Post-CTS Mode

- To optimize timing after the clock tree is built, use the following commands:

```
optDesign -postCTS
```

optDesign in postCTS fixes DRVs, reclaims area, and fixes setup violations.

**Note:** By default, the optDesign does not correct fanout violations. To repair fanout violations, run the following command before optDesign, starting from the first call of optDesign -preCTS up to the last call of optDesign -postRoute:

```
setOptMode -fixFanoutLoad true
```

- To repair setup and hold violations, use the following commands:

```
optDesign -postCTS  
optDesign -postCTS -hold
```

- To repair design rule violations only, use the following command:

```
optDesign -postCTS -drv
```

- To repair hold violations only, use the following command:

```
optDesign -postCTS -hold
```

### Skipping Path Groups or Clock Domains During Hold Fixing

You can instruct the Encounter software to exclude path groups or clock domains from hold fixing by using the [setOptMode -ignorePathGroupsForHold](#) command.

This feature is useful, for example, when you want to fix only those hold violations that are in the core of the design and skip violations on I/O paths. This can be achieved by applying the following in the clockDomains support mode:

```
setOptMode -ignorePathGroupsForHold {in2reg reg2out in2out}
```

### Related Topics

To see this step in the design flow, see [Run CTS and Post-CTS Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

## Using Additional Post-CTS Timing Optimization Parameters

- To run optimization on specific path groups, use the following commands:

```
clearClockDomains  
setClockDomains -fromType sourcePoint -toType destinationPoint  
optDesign -postCTS
```

For a list of supported source and destination points, see “[Correcting Violations in Pre-CTS Mode for the First Time](#)” on page 1016.

**Note:** If you are using the CTE engine, you cannot use `setClockDomains` parameters other than `-fromType` and `-toType`.

For example, to run optimization on register-to-register paths, use the following commands:

```
clearClockDomains  
setClockDomains -fromType register -toType register  
optDesign -postCTS
```

- To take advantage of useful skew when optimizing timing in post-CTS mode, use the following commands:

```
setOptMode -usefulSkew true  
optDesign -postCTS
```

If you have already performed detailed routing on the clock tree, the Encounter software performs global and detailed ECO routing automatically using the NanoRoute® router in post-CTS useful skew mode. If you do not want `optDesign` to do this, specify the `-noECORoute` parameter as follows:

```
setOptMode -usefulSkew true  
optDesign -postCTS -noECORoute
```

If you specify `-noECORoute` before running optimization, the Encounter software performs trial routing to estimate clock delays.

- To run post-CTS optimization if your design has a clock mesh, use the following commands:

```
setOptMode -usefulSkew false  
optDesign -postCTS
```

## Performing Incremental Post-CTS Optimization

- To run optimization on specific path groups, use the following commands:

```
clearClockDomains  
setClockDomains -fromType sourcePoint -toType destinationPoint  
optDesign -postCTS -incr
```

For a list of supported source and destination points, see “[Using Additional Pre-CTS Timing Optimization Parameters](#)” on page 1017.

## Encounter User Guide

### Optimizing Timing

---

**Note:** If you are using the CTE engine, you cannot use `setClockDomains` parameters other than `-fromType` and `-toType`.

For example, to run incremental optimization on register-to-register paths, use the following commands:

```
clearClockDomains  
setClockDomains -fromType register -toType register  
optDesign -postCTS -incr
```

- To optimize setup time incrementally and reduce area, use the following commands:

```
setOptMode -reclaimArea true  
optDesign -postCTS -incr
```

- To take advantage of useful skew when optimizing timing in incremental post-CTS mode, use the following commands:

```
setOptMode -usefulSkew true  
optDesign -postCTS -incr
```

If you have already performed detail routing on the clock tree, the software performs global and detailed ECO routing automatically using the NanoRoute router in post-CTS useful skew mode. If you do not want the software to do this, specify the `-noECORoute` parameter, as follows:

```
setOptMode -usefulSkew true  
optDesign -postCTS -noECORoute -incr
```

If you specify `-noECORoute`, `optDesign` performs trial routing to estimate clock delays.

- To run incremental post-CTS optimization if your design has a clock mesh, use the following commands:

```
setOptMode -usefulSkew false  
optDesign -postCTS -incr
```

## Changing Default Settings in Post-CTS Mode

You can change or add parameters for the following commands and parameters that `-optDesign` runs automatically:

<u><a href="#">setAnalysisMode</a></u>	<code>optDesign</code> sets <code>-clockPropagation</code> <code>autoDetectClockTree</code> and <code>-clkSrcPath</code> <code>true</code> by default: You cannot override these values. You can add other parameters.
<u><a href="#">setClockDomains</a></u>	<code>optDesign</code> uses the parameters you specify. The default is all path groups.

## Encounter User Guide

### Optimizing Timing

---

[setExtractRCMode](#) optDesign sets the extraction mode to default. You cannot change this mode. Ensure that you set the appropriate extraction scale factor.

[setOptMode](#) optDesign sets the following parameters:

- [-drcMargin](#)

If you use `setOptMode -drcMargin`, this value is added to a dynamically calculated, internal margin. For example, if you set a margin of `0 . 2` (20 percent), this multiplies the `max_cap` and `max_tran` SDC constraints by 0.8. The margin can be positive or negative. If you set a margin of `-0 . 2`, this multiplies the `max_cap` and `max_tran` SDC constraints by 1.20. optDesign writes the margin value to the log file.

- [-holdTargetSlack](#)

If you use `setOptMode -holdTargetSlack`, this value is added to a dynamically calculated, internal margin. optDesign writes the hold target slack value to the log file.

- [-setupTargetSlack](#)

If you use `setOptMode -setupTargetSlack`, this value is added to a dynamically calculated, internal margin. The default `-setupTargetSlack` value is 0. optDesign writes the setup target slack value to the log file.

You can override other parameters.

[setTrialRouteMode](#) You can add parameters, but never override the default settings. optDesign sets the `-handlePreRoute true` parameter.

## Performing Postroute Optimization

- [About Postroute Optimization on page 1024](#)
- [Correcting Violations in Postroute Mode on page 1025](#)
- [Correcting Signal Integrity Violations on page 1027](#)

## About Postroute Optimization

In postroute mode, the Encounter software corrects setup violations and design rule violations unless you specify otherwise. It first operates on all path groups, then on register-to-register paths only. The software performs incremental RC and delay calculation, and runs the NanoRoute router to perform ECO routing.

If filler cell definitions were provided during design import, `optDesign` removes or adds them as needed, following the information given by `setFillerMode`. For more information on this command, see [setFillerMode](#) in the “Placement Commands” chapter of the *Encounter Text Command Reference*.

There should be very few timing violations that need correction. The primary sources of these violations include the following:

- Inaccurate prediction of the routing topology during pre-route optimization due to congestion-based detour routing
- Minor correlation issues between default and detailed RC extraction.
- Incremental delays due to parasitics coupling



Because the violations at this stage are due to inaccurate modeling of the final route topology and the attendant parasitics, it is critical not to introduce additional topology changes beyond those needed to correct the existing violations.

Making unnecessary changes to the routing at this point can lead to a scenario where fixing one violation leads to the creation of others. This cascading effect creates a situation where it becomes impossible to close on a final timing solution with no DRVs.

One of the strengths of postroute optimization is its ability to simultaneously cut a wire and insert buffers, create the new RC graph at the corresponding point, and modify the graph to estimate the new parasitics for the cut wire without re-doing extraction.

To take even more advantage of this feature, you can provide an external SPEF generated by a sign-off extraction tool for improved convergence. If you do, you must provide a full SPEF (reduced SPEF does not work) and one of the following conditions must be met:

- The SPEF must be generated with node locations using the `starN` format. The `runQRC` command has been enhanced to output this format using the `-spefOutput starN` parameter. For example: `runQRC -spefOutput starN`.

or

- The resistance values in the LEF file must match those in the technology file used by signoff extraction to generate the SPEF, which enables the Encounter extraction engine to match the routes with the SPEF RC graph.

## Related Topics

To see this step in the design flow, see [Route the Design and Run Postroute Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

## Correcting Violations in Postroute Mode

- To optimize timing after detailed routing, use the following command:

```
optDesign -postRoute
```

`optDesign` corrects DRVs and setup violations, as it does in pre-CTS and post-CTS modes.

`optDesign` also performs an additional register-to-register optimization if the worst negative slack does not occur on a register-to-register path. The command cuts wires during buffer insertion and resizing. If you do not provide a SPEF file, `optDesign` simultaneously cuts the RC graph at the corresponding point to estimate RCs on the cut wires.

In postroute mode, the software refines placement, then runs the NanoRoute router in ECO mode to reroute affected wires. The software extracts RCs (except if you provide a SPEF file) and reports final timing results. After postroute optimization is complete, run a signoff extractor and compare the results with those generated by the Encounter software.

**Note:** By default, the `optDesign` does not correct fanout violations. To repair fanout violations, run the following command before `optDesign`, starting from the first call of `optDesign -preCTS` up to the last call of `optDesign -postRoute`:

```
setOptMode -fixFanoutLoad true
```

- To correct hold violations only, specify the following commands:

```
optDesign -postRoute -hold
```

The command first does a setup analysis to store the setup slack on all paths. Then it analyzes hold violations and tries to repair hold violations without degrading setup. It does not move any cells that are already in the netlist. If inserting delay cells would degrade setup, it does not insert the cells. A small amount of setup degradation might occur if some nets must be rerouted.

## Encounter User Guide

### Optimizing Timing

---

Hold repair does not degrade the setup worst slack to less than the original value or the `setupTargetSlack` value. You can override the `setupTargetSlack` value by specifying `setOptMode -setupTargetSlack` before you run `optDesign`. By default, hold repair is allowed to degrade the setup total negative slack. Therefore, to disable this feature, set the following:

```
setOptMode -fixHoldAllowSetupTnsDegrade false
```

- To correct setup and hold violations, use the following commands:

```
optDesign -postRoute  
optDesign -postRoute -hold
```

The setup violations must be fixed first. Hold repair ensures that there is no setup slack degradation.

- To take clock reconvergence pessimism removal (CRPR) into consideration when running timing optimization, use the `setAnalysisMode` command before you run `optDesign`. For example:

```
setTimingDerate -max -clock -early 0.8 -late 1.2  
setTimingDerate -min -clock -early 0.8 -late 1.2  
setAnalysisMode -cppr both  
optDesign -postRoute
```

- To run postroute timing optimization on designs containing Interface Logic Models (ILMs), use the following command:

```
optDesign -postRoute -ilm
```

This command flattens ILMs, optimizes timing, then unflattens the ILMs.

- To run postroute setup optimization based on an external SPEF file, use the following commands:

```
spefIn SPEF_file_name  
optDesign -postRoute
```

- To run postroute hold optimization based on an external SPEF file, use the following commands:

```
spefIn SPEF_file_name  
optDesign -postRoute -hold
```

- To run postroute hold optimization based on two external SPEF files (one generated in best case and one in worst case), use the following commands:

```
defineRCCorner -earlySpef best_case_SPEF_file_name  
-lateSpef worst_case_SPEF_file_name  
optDesign -postRoute -hold
```

- To run postroute setup optimization based on a SDF file, use the following commands:

```
setOptMode -setupSdfFile setup_SDF_file_name  
optDesign -postRoute -useSDF
```

- To run postroute hold optimization based on SDF files, use the following commands:

## Encounter User Guide

### Optimizing Timing

---

```
setOptMode -setupSdfFile setup_SDF_file_name  
          -holdSdfFile hold_SDF_file_name  
optDesign -postRoute -hold -useSDF
```

**Note:** You must specify a setup SDF file when doing hold fixing.

**Note:** You can instruct the Encounter software to exclude path groups or clock domains from hold fixing. For more information, see [Skipping Path Groups or Clock Domains During Hold Fixing](#) on page 1020.

#### ■ In MMMC environment:

- ❑ To run postroute setup or hold optimization based on external SPEF files for a design with four active RC corners (two for setup and two for hold), type the following:

```
spefIn -rc_corner cornerMax1 rcMax1.spef  
spefIn -rc_corner cornerMax2 rcMax2.spef  
spefIn -rc_corner cornerMin1 rcMin1.spef  
spefIn -rc_corner cornerMin2 rcMin2.spef  
  
optDesign postRoute
```

or

```
optDesign postRoute hold
```

or

```
optDesign postRoute si
```

**Note:** You must provide SPEF information for each active `rc_corner` (setup and hold). If one corner does not have a SPEF, the tool will run RC extraction for each corner again.

- ❑ To run postroute setup optimization based on external SDF files for a design with two active setup views, type the following:

```
read_sdf -view viewMax1 sdfMax1.sdf  
read_sdf -view viewMax2 sdfMax2.sdf  
optDesign -postRoute -useSDF
```

- ❑ To run postroute hold optimization based on external SDF files for a design with two active setup views and two active hold views, type the following:

```
read_sdf view viewMax1 sdfMax1.sdf  
read_sdf view viewMax2 sdfMax2.sdf  
read_sdf view viewMin1 sdfMin1.sdf  
read_sdf view viewMin2 sdfMin2.sdf  
optDesign postRoute hold -useSDF
```

## Correcting Signal Integrity Violations

#### ■ To correct signal integrity violations when optimizing timing in postroute mode, use the following command:

```
optDesign -postRoute -si
```

## Encounter User Guide

### Optimizing Timing

---

In addition to the timing violations caused by inaccurate route topology modeling, the parasitic cross coupling of neighboring nets can cause the following problems that need to be addressed in high speed designs:

- ❑ An increase or decrease in incremental delay on a net due to the coupling of its neighbors and their switching activity.
- ❑ Glitches (voltage spikes) that can be caused in one signal route by the switching of a neighbor resulting in a logic malfunction.

This command has its maximum effect once most other timing-related issues have been corrected. It uses CelTIC NDC to run the SI analysis and corrects problems found by using a combination of buffer resizing and routing modifications.

**Note:** Use the `optDesign -postRoute -si` command only after using the `optDesign -postRoute` command.

- To run signal integrity-aware setup and hold repair, use the following commands:

```
optDesign -postRoute -si  
optDesign -postRoute -hold -si
```

**Note:** Use the `optDesign -postRoute -si` command only after using the `optDesign -postRoute` command.

`optDesign` performs the following operations sequentially:

- ❑ Analyzes cross coupling capacitance effects on glitch and noise
- ❑ Back-annotates delays due to cross coupling capacitance
- ❑ Reruns timing analysis
- ❑ Repairs setup violations
- ❑ Repairs hold violations

- To input a SPEF file for use during postroute signal integrity optimization, use the following commands.

```
spefIn spefFileName  
optDesign -postRoute -si
```

## Changing Default Settings in Postroute Mode

You can change or add parameters for the following commands and parameters that optDesign runs automatically:

<u><a href="#">setAnalysisMode</a></u>	optDesign sets -clockPropagation autoDetectClockTree and -clkSrcPath true. You can override other parameters.
<u><a href="#">setClockDomains</a></u>	optDesign uses the parameters you specify. The default is all path groups.
<u><a href="#">setExtractRCMode</a></u>	optDesign sets the extraction mode to detail. You cannot change this mode. Ensure that you set the appropriate extraction scale factor.
<u><a href="#">setTrialRouteMode</a></u>	You can add parameters, but never override the default settings. optDesign sets the -handlePreroute true parameter.

## Optimizing Power During optDesign

During timing optimization, the tool is also able to reduce leakage power and dynamic power.

### Leakage Power Optimization

To activate leakage power optimization during timing optimization, run the following:

```
setOptMode -leakagePowerEffort none|low|high
```

- When effort is set to `none`, optDesign will not optimize the leakage power.  
This is the default option.
- When effort is set to `low`, optDesign will optimize leakage power only in the postroute stage. The leakage reduction will happen during postroute setup optimization and no high leakage cells will be inserted during hold fixing.
- When effort is set to `high`, optDesign will optimize leakage power in all stages. Leakage reduction will happen during each setup optimization and no high leakage cells will be inserted during hold fixing.

**Note:** To achieve the best leakage power results, load all the different Vth libraries.

## Dynamic Power Optimization

To activate dynamic power optimization during timing optimization, run the following:

```
setOptMode -DynamicPowerEffort none|low|high
```

- When effort is set to none, optDesign will not optimize dynamic power.  
This is the default option.
- When the effort is set to low, optDesign will only optimize dynamic power in the postroute setup optimization phase.
- When the effort is set to high, optDesign will optimize dynamic power in the entire setup optimization phases.

**Note:** Cadence recommends that you provide an activity file. If you do not provide an activity file, Encounter automatically generates a default activity file based on a propagated toggle rate of 0.2 on each input port.

## Using Useful Skew

The useful skew feature in the Encounter software modifies the clock arrival time on sequential elements in order to improve the datapath timing between sequential elements.

The software provides two approaches to using useful skew, depending on whether you have run CTS:

- Pre-CTS mode  
Advances the clock signal for critical path start points. The start point must be a sequential element: No input paths are allowed.
- Post-CTS mode  
Delays the clock signal for critical path end points. The end point must be a sequential element: No output paths are allowed.

## Using Useful Skew in Pre-CTS Mode

To take advantage of useful skew during pre-CTS optimization, use the following commands:

```
setOptMode -usefulSkew true  
optDesign -preCTS
```

The software determines the sequential instances whose clock signals can be advanced, then generates the following two files:

- **latency\_file.sdc**

This latency file models the proposed clock advancement for timing analysis.

- **scheduling\_file.cts**

This file contains scheduling information for clock tree synthesis. You must specify this file when you specify the CTS constraints, for example:

```
specifyClockTree -clkfile scheduling_file.cts  
specifyClockTree -clkfile original_constraints.cts
```

You can change the names of the latency and scheduling files by using the following commands:

```
setLatencyFile fileName  
setSchedulingFile fileName
```

Use the following commands to report the names of the latency and schedule files:

- [getLatencyFile](#)
- [getSchedulingFile](#)

For more information, see “Timing Optimization Commands” in the *Encounter Text Command Reference*:

## Using Useful Skew in Post-CTS Mode

To take advantage of useful skew during post-CTS optimization, use the following commands:

```
setOptMode -usefulSkew true  
optDesign -postCTS
```

In this case, the clock tree is already in place. The software determines the sequential instances whose clock signals can be delayed, and adds buffers or inverters to their clock nets accordingly. If the clock is already detail routed, these commands perform ECO routing on the clock tree after useful skew optimization.

## Controlling Useful Skew Optimization

You can control how the Encounter software employs useful skew, use the following command:

- [setUsefulSkewMode](#)

If you choose specific cells for clock tree synthesis, use `setUsefulSkewMode -useCells` to specify the cells to use for padding the clock nets. If you have no constraint on the type of

## Encounter User Guide

### Optimizing Timing

---

cells allowed in the clock tree, you can omit this parameter, and the software selects the best combination of cells to achieve the required delay.

For example, if you want clock buffers or inverters only, specify the following command:

```
-setUsefulSkewMode -useCells {...}
```

To advance or delay sequential elements more aggressively than it does by default, without degrading the worst negative slack, use the following `setUsefulSkewMode` parameter:

```
-maxSkew true
```

When you specify this parameter, the tool skews other registers as much as possible regardless of the worst slack on a particular register. This approach can help with difficult timing closure situations. In post-CTS mode, critical paths probably have been fully optimized, so further traditional optimization cannot dramatically improve timing.

To close timing, you might need to delay the endpoint clock pins more than the useful skew feature would do by default, by only padding the clock nets until the data path meets the target slack. To take advantage of this feature, use the following command:

```
setUsefulSkewMode -maxSkew true
```

To exclude boundary sequential cells in useful skew calculations, use the following command:

```
setUsefulSkewMode -noBoundary true
```

If you do not specify this parameter, the software takes boundary cells and ordinary sequential elements into account when calculating useful skew.

To use NanoRoute detailed routing to route nets that are added or changed during useful skew optimization, use the following command:

```
setUsefulSkewMode -ecoRoute true
```

To limit the amount of slack the Encounter software can borrow from neighboring flip-flops when performing useful skew operations, use the following command:

```
setUsefulSkewMode -maxAllowedDelay true
```

The Encounter delay calculation and RC extraction methods might differ from those of sign-off tools, so other setup violations might occur if the Encounter tool borrows too much slack. By having control over slack borrowing, you can prevent these setup violations. Limiting borrowed skew also limits the clock tree skew to avoid large hold violations. If you do not specify this parameter, the Encounter software automatically borrows the amount of slack needed (there is no maximum) to reduce setup violations.

To report the current `setUsefulSkewMode` settings, use the following command:

```
getUsefulSkewMode
```

For more information, see “Timing Optimization Commands” in the *Encounter Text Command Reference*.

## Using Active Logic View for Chip-Level Interface Circuit Timing Closure

The Encounter software provides a top-level interface timing operation flow to perform partitioning and budgeting on a trimmed-down version of the timing graph: an active logic view. This flow helps you close the timing issues of the interface top-level paths as your design has gone through the hierarchical flow until the post-route stage. This flow also saves the memory usage and provides faster runtime on large designs.

To perform optimization using an active logic view at the post-route stage, complete the following steps:

1. Load the hierarchical design in the database that is created by `-assembleDesign` using the entire postrouted block partition and the top-level partition. Specify the partition information in the database.

```
restoreDesign assembled.enc.dat toplevel_design_name
```

2. Perform timing analysis on the design to identify the timing of the full-chip design.

```
timeDesign -postRoute -prefix preOpt
```

3. Set the optimization mode to use active logic view. If you specify this parameter, `optDesign` observes the floorplan fence constraint when moving or adding cells.

```
setOptMode -virtualPartition true
```

4. Run `optDesign`. The `optDesign` command honors active logic view.

```
optDesign -postRoute
```

5. Perform timing analysis again to ensure that there are no timing issues.

```
timeDesign -postRoute -prefix postOpt
```

## Optimizing Timing in On-Chip Variation Analysis Mode

Optimize timing in on-chip variation (OCV) analysis mode to account for variations in process, voltage, and temperature (PVT) across the die. When it takes OCV into account, the software calculates early and late delays, and uses them to evaluate setup and hold timing checks. You introduce the delays into the analysis by specifying different min/max corner timing libraries and operating conditions. Early/late variation might also be present due to slew merging effects of multiple input gates in the clock path.

To enable the software to consider multiple libraries and operating conditions, you must specify a multi-mode/multi-corner (MMMC) environment. If the MMMC environment is not specified, and you try to run timing optimization in OCV mode, the `optDesign` command exits with an error message.

## Related Topics

To see this step in the design flow, see [Route the Design and Run Postroute Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

For more information on OCV and MMMC, see

- [On-Chip Variation \(OCV\) Timing Analysis Mode](#) on page 922
- [Performing Multi-Mode Multi-Corner Timing Analysis and Optimization](#) on page 1183

## Specifying the MMMC Environment

There are three MMMC scenarios for timing optimization in OCV mode:

- [One library and one operating condition per corner](#)
- [One library and two operating conditions per corner](#)
- [Two worst-case libraries and two best-case libraries per corner](#)

The operating condition specifications you provide to the `create_delay_corner` command determine the MMMC scenario for OCV mode. These specifications give the software the values to use for early and late timing.

The following sections show the specifications necessary for each scenario. The differences are highlighted in bold-face type.

- **One library and one operating condition per corner**

```
create_library_set -name libs_min -timing [list $bestcase_lib]
create_library_set -name libs_max -timing [list $worstcase_lib]
create_rc_corner -name rc_worst -cap_table CMAX.capTbl
create_rc_corner -name rc_best -cap_table CMIN.capTbl
create_constraint_mode -name postCTS [list xxx.sdc]
create_delay_corner -name delay_corner_max \
    -library_set libs_max \
    -opcond_library stdcmos90T125 \
    -opcond cmos90T125 \
    -rc_corner rc_worst
create_delay_corner -name delay_corner_min \
    -library_set libs_min \
    -opcond_library stdcmos90Tm40 \
    -opcond cmos90Tm40 \
```

# Encounter User Guide

## Optimizing Timing

---

```
-rc_corner rc_best
create_analysis_view -name postCts_max \
    -delay_corner delay_corner_max \
    -constraint_mode postCTS
create_analysis_view -name postCts_min \
    -delay_corner delay_corner_min \
    -constraint_mode postCTS
set_analysis_view -setup postCts_max -hold postCts_min
```

### ■ One library and two operating conditions per corner

```
create_library_set -name libs_min -timing [list $bestcase_lib]
create_library_set -name libs_max -timing [list $worstcase_lib]
create_rc_corner -name rc_worst -cap_table CMAX.capTbl
create_rc_corner -name rc_best -cap_table CMIN.capTbl
create_constraint_mode -name postCTS [list xxx.sdc]
create_delay_corner -name delay_corner_max \
    -library_set libs_max \
    -late_opcond_library stdcmos90T125 \
    -late_opcond cmos90T125_slow \
    -early_opcond_library stdcmos90T125 \
    -early_opcond cmos90T125 \
    -rc_corner rc_worst
create_delay_corner -name delay_corner_min \
    -library_set libs_min \
    -late_opcond_library stdcmos90Tm40 \
    -late_opcond cmos90Tm40 \
    -early_opcond_library stdcmos90Tm40 \
    -early_opcond cmos90Tm40_fast \
    -rc_corner rc_best
create_analysis_view -name postCts_max \
    -delay_corner delay_corner_max \
    -constraint_mode postCTS
create_analysis_view -name postCts_min \
    -delay_corner delay_corner_min \
    -constraint_mode postCTS
set_analysis_view -setup postCts_max -hold postCts_min
```

### ■ Two worst-case libraries and two best-case libraries per corner

```
create_library_set -name libs_min_std -timing [list $bestcase_lib_std]
create_library_set -name libs_max_std -timing [list $worstcase_lib_std]
create_library_set -name libs_min_fast -timing [list $bestcase_lib_fast]
create_library_set -name libs_max_fast -timing [list $worstcase_lib_fast]

create_rc_corner -name rc_worst -cap_table CMAX.capTbl
create_rc_corner -name rc_best -cap_table CMIN.capTbl

create_constraint_mode -name postCTS [list xxx.sdc]

create_delay_corner -name delay_corner_max
-late library_set libs_max_std \
-late_opcond_library stdcmos90T125 \
-late_opcond cmos90T125 \
-early library_set libs_max_fast \
-early_opcond_library fastcmos90T125 \
-early_opcond cmos90T125 \
-rc_corner rc_worst

create_delay_corner -name delay_corner_min
-late library_set libs_min_std \
```

```
-late_opcond_library stdccmos90Tm40 \
-late_opcond cmos90Tm40 \
-early_library_set libs_min_fast \
-early_opcond_library fastcmos90Tm40 \
-early_opcond cmos90Tm40 \
-rc_corner rc_best

create_analysis_view -name postCts_max \
-delay_corner delay_corner_max \
-constraint_mode postCTS
create_analysis_view -name postCts_min \
-delay_corner delay_corner_min \
-constraint_mode postCTS

set_analysis_view -setup postCts_max -hold postCts_min
```

## Optimizing Timing in OCV Mode Using the Default Delay Calculator

After you specify the MMMC environment, use the following commands:

```
setAnalysisMode -analysisType onChipVariation
optDesign -postRoute [-hold]
```

For more information, see documentation for the following commands:

- [setAnalysisMode](#)
- [optDesign](#)

## Optimizing Timing in OCV Mode Using the Sign-Off Delay Calculator

After you specify the MMMC environment, use the following commands:

```
setDelayCalMode -engine signalStorm [-signoff]
setAnalysisMode -analysisType onChipVariation
optDesign -postRoute [-hold]
```



The `setDelayCalMode -engine signalStorm` command is necessary for full support of the `rise/fall_pin_cap_range` construct.

For more information, see documentation for the following commands:

- [setDelayCalMode](#)
- [setAnalysisMode](#)
- [optDesign](#)

## Using Conformal Constraint Designer During Timing Optimization

The Encounter software is tightly linked to the Conformal Constraint Designer (CCD) software. One of the features CCD provides is the ability to analyze critical false paths based on Encounter-CTE timing information and constraints. CCD outputs a file that lists a set of false paths, and the file can be loaded back into the Encounter software. Identifying the false paths in this way eliminates unnecessary netlist optimizations and improves design area, power, and timing.

You can use CCD to improve timing optimization in one of the following ways:

- [Post-Processing Approach](#) on page 1037
- [Integrated Approach](#) on page 1038

### Post-Processing Approach

Even after optimizing the design, you might not have achieved the frequency target. Usually, this problem is due to tight timing constraints that are difficult to meet. By using CCD to verify whether the worst slack paths are valid, you may find that your critical paths are actually invalid and therefore need not be taken in account.

In this approach, the CCD tool is used post-processing only, and the Encounter timing closure flow is unchanged.

The following example shows the commands in the post-processing approach. Use these commands after running `timeDesign` or `optDesign` in setup mode.

```
deriveFalsePathCCD -outputDir . outputFile trv.sdc  
loadTimingCon -incr trv.sdc
```

The `deriveFalsePathCCD` command runs CCD in batch mode and generates a list of critical false paths. The `loadTimingCon` command reads the list into the Encounter software.

After running these commands, re-analyze timing by running `timeDesign`. The slack should be better because false paths have been removed from consideration.

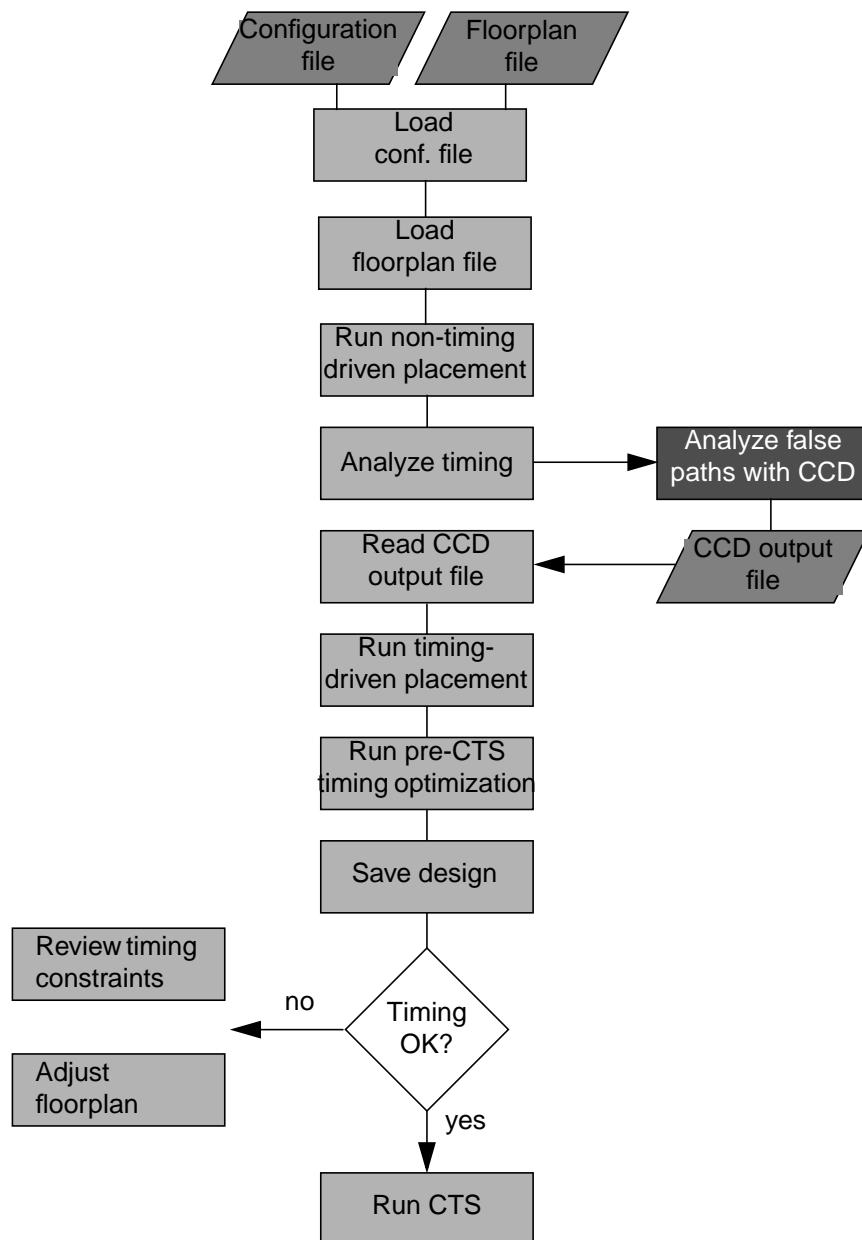
For more information, see documentation for the following commands:

- [deriveFalsePathCCD](#)
- [loadTimingCon](#)

## Integrated Approach

A second approach is to integrate CCD with the Encounter timing optimization flow. Cadence recommends this method because it gives you the advantage of identifying false paths that are timing critical earlier in the implementation process. In addition, this method enables faster timing closure and leads to a better optimized netlist in area/leakage/power.

The following figure shows the recommended flow for this approach:



## Encounter User Guide

### Optimizing Timing

---

1. Load the floorplanned design and place standard cells in non-timing driven mode. Placing the cells in non-timing driven mode speeds up placement. You run placement again later, in timing-driven mode, in this flow.

For example,

```
loadConfig myConfigFile.conf
loadFPlan myFloorPlan.fp
setPlaceMode -timingDriven false
placeDesign
```

2. Run `timeDesign` to build a clean timing graph.

For example,

```
timeDesign -preCTS -outDir timing_place_noTD
```

3. Analyze and report setup timing violated paths using CCD.

For example,

```
deriveFalsePathCCD -outputDir . -outputFile trv.postnotdp.sdc
```

This command runs CCD in batch mode. It outputs an SDC file that lists the false paths. CCD exits when the command completes.

For more information, see [`deriveFalsePathCCD`](#) in the *Encounter Text Command Reference*.

4. Read the CCD output file into the Encounter software.

For example,

```
loadTimingCon -incr trv.postnotdp.sdc
```

5. Place the design in timing-driven mode and run pre-CTS timing optimization. Save the design.

For example,

```
setPlaceMode -timingDriven true
placeDesign
timeDesign -preCTS -outDir timing_place_TD
setOptMode -effort high
optDesign -preCTS -outDir timing_optimized
saveDesign post_opt_prects.enc
```

If the design meets timing, you can go on to clock tree synthesis; if not, look at the timing constraints and the floorplan to make sure they are reasonable. If CCD reports too many false paths, the CPU run time for timing analysis and timing optimization might be affected.

 *Important*

If WNS and TNS do not change when reverting to initial timing constraints, it may be that the false paths identified by CCD could have met timing easily. This can happen because the false paths generated by CCD are based on timing post-placement timing. At this stage no timing optimization has been performed and removing the false paths does not necessarily uncover violated paths.

## Optimizing Timing Using a Rule File

In a partitioned design, top-level and leaf partitions are generated. Before implementation, the leaf partitions' timing models are not completely accurate. Because accurate timing cannot be derived without accurate timing models for leaf partitions, rule-based optimization is a more suitable option than timing analysis-based optimization at this design stage. You can use a rule file for the top-level design by using the following command:

- insertRepeater

## Optimizing Timing When the Constraint File Includes the `set_case_analysis` Constraint

If you include the `set_case_analysis` constraint in the timing constraint file, the Encounter software sets a constant value on specified signals before performing timing analysis. This constant value is then propagated through the path.

If you use the same timing constraint file for timing optimization, the software does not perform timing optimization on the constant nets because the delays are 0.

To run timing optimization on these nets, you must first specify the following command:

- setAnalysisMode -caseAnalysis false

## Using the Footprintless Flow

By default, the Encounter software creates an internal footprint structure based on cell functionality. This methodology is referred to as the footprintless flow, and has the following advantages over a flow that relies on footprint information from the libraries:

- The libraries do not need to contain footprints, and you do not need to specify a footprint file.

## Encounter User Guide

### Optimizing Timing

---

- The following commands are not necessary because the software detects the functionality for inverters and buffers and decides whether a buffer is a delay cell, based on the cell's timing characteristic. The commands have no effect if specified in this flow.
  - `setBufFootPrint`
  - `setInvFootprint`
  - `setDelayFootPrint`
- The software considers cells with the same functionality but different function syntax as equivalent and allows sizing between such cells.
- The software prints the list of usable and unusable (“don’t use”) buffers, inverters, and delay cells to the log file after reading in the libraries, for example:

```
Total number of combinational cells: 620
Total number of sequential cells: 247
Total number of tristate cells: 42
Total number of level shifter cells: 0
Total number of power gating cells: 0 Total number of isolation cells: 0
List of usable buffers: BFX1 BFX2 BFX3 BFX4
Total number of usable buffers: 4
List of unusable buffers: BFX20 BFX32
Total number of unusable buffers: 2
List of usable inverters: IVX1 IVX2 IVX3 IVX4
Total number of usable inverters: 4
List of unusable inverters:
Total number of unusable inverters: 0
List of identified usable delay cells: DLY2 DLY4 DLY8
Total number of identified usable delay cells: 3
List of identified unusable delay cells:
Total number of identified unusable delay cells: 0
```

To revert to the behavior in previous releases (that is, to rely on footprint information in the libraries), use the `loadFootPrint` command. As in those releases, you must specify buffers, inverters, and delay cell footprints according to what was loaded in the footprint file. For more information, see “[Using Cell Footprints](#)” on page 1042.

To exclude cells from timing optimization, for example, if the libraries have clock buffers or clock inverters that should be used during CTS but not during timing optimization, set the “don’t use” attribute in the timing constraints file, library, or command shell. Timing optimization can resize a “don’t use” cell, but does not insert it.

For more information see [`setDontUse`](#) in the “Timing Optimization Commands” chapter of the *Encounter Text Command Reference*.

## Using Cell Footprints

Timing optimization can use information in a footprint file. For example, the buffering mechanisms in `optDesign` add cells only if they are defined in the buffer footprint file.

To disable the footprintless flow (the default timing optimization flow) and load a footprint file, specify the following command:

```
loadFootPrint -infile footprint_file_name
```

For more information, see [loadFootPrint](#) in the “Timing Optimization Commands” chapter of the *Encounter Text Command Reference*.

Define footprints in your library or a footprint file by using the following commands, which are enabled when you specify `loadFootPrint`:

- `setBuffFootPrint`
- `setInvFootPrint`
- `setDelayFootPrint`

## Viewing Added Buffers, Instances, and Nets

After running timing optimization, use the Design Browser to view the added buffers, instances, and nets. The names of the buffers, instances, and nets added as a result of timing optimization are annotated with the prefix `FE_`.

For information on using the Design Browser, see [Design Browser](#) in the “Tools Menu” chapter of the *Encounter Menu Reference*.

## Default Naming Conventions

Prefix	Description	Command
<code>FE_MDBC</code>	Instance added by multi-driver net buffering	<a href="#">optDesign</a>
<code>FE MDBN</code>	Net added by multi-driver net buffering	<a href="#">optDesign</a>
<code>FE_OCP_DRV_C</code>	Instance added by DRV fixing	<a href="#">optDesign</a>
<code>FE_OCP_DRV_N</code>	Net added by DRV fixing	<a href="#">optDesign</a>
<code>FE_OCP_RBC</code>	Instance added by rebuffering	<a href="#">optDesign</a>

## Encounter User Guide

### Optimizing Timing

---

<b>Prefix</b>	<b>Description</b>	<b>Command</b>
FE_OCP_RBN	Net added by rebuffering	<a href="#"><u>optDesign</u></a>
FE_OCPC	Instance added by critical path optimization	<a href="#"><u>optDesign</u></a>
FE_OCPN	Net added by critical path optimization	<a href="#"><u>optDesign</u></a>
FE_OFC	Buffer instance added by rule-based buffer insertion	<a href="#"><u>insertRepeater</u></a> / <a href="#"><u>optDesign</u></a>
FE_OFN	Buffer net added by rule-based buffer insertion	<a href="#"><u>insertRepeater</u></a> / <a href="#"><u>optDesign</u></a>
FE_PHC	Instance added by hold time repair	<a href="#"><u>optDesign</u></a>
FE_PHN	Net added by hold time repair	<a href="#"><u>optDesign</u></a>
FE_PSBC	Instance added by buffer insertion in optDesign -postRoute	<a href="#"><u>optDesign</u></a>
FE_PSBN	Net added by buffer insertion in optDesign -postRoute	<a href="#"><u>optDesign</u></a>
FE_PSC	Instance added by postroute setup repair	<a href="#"><u>optDesign</u></a>
FE_PSN	Net added by postroute setup repair	<a href="#"><u>optDesign</u></a>
FE_RC	Instance created by netlist restructuring	<a href="#"><u>optDesign</u></a>
FE_RN	Net created by netlist restructuring	<a href="#"><u>optDesign</u></a>
FE_USC	Instance added during useful skew optimization	<a href="#"><u>optDesign</u></a>

## **Encounter User Guide**

### Optimizing Timing

---

---

## Interactive ECO

---

- [Overview](#) on page 1046
- [Before You Begin](#) on page 1046
- [Results](#) on page 1046
- [Adding Buffers](#) on page 1046
- [Changing the Cell](#) on page 1049
- [Deleting Buffers](#) on page 1051
- [Displaying Buffer Trees](#) on page 1052
- [Running ECO Placement](#) on page 1054
- [Naming Conventions for Interactive ECO](#) on page 1055
- [Comparing Physical Design Data](#) on page 1055

## Overview

The Interactive ECO feature enables you to run manual incremental updates to the design to repair timing or transition time violations. You can run Interactive ECO after running placement, timing optimization, or signal integrity analysis (CeltIC NDC).

If you performed trial route and RC extraction on the design, and the timing graph was built before running an ECO, then the trial route data, RC extraction data, and timing graph are incrementally updated.

## Before You Begin

Before you can perform interactive ECO, the following conditions must be met:

- You must place and route the design,
- You must load the design into the current session.

## Results

The following output files are generated:

- Updated netlist
- Updated placement

## Adding Buffers

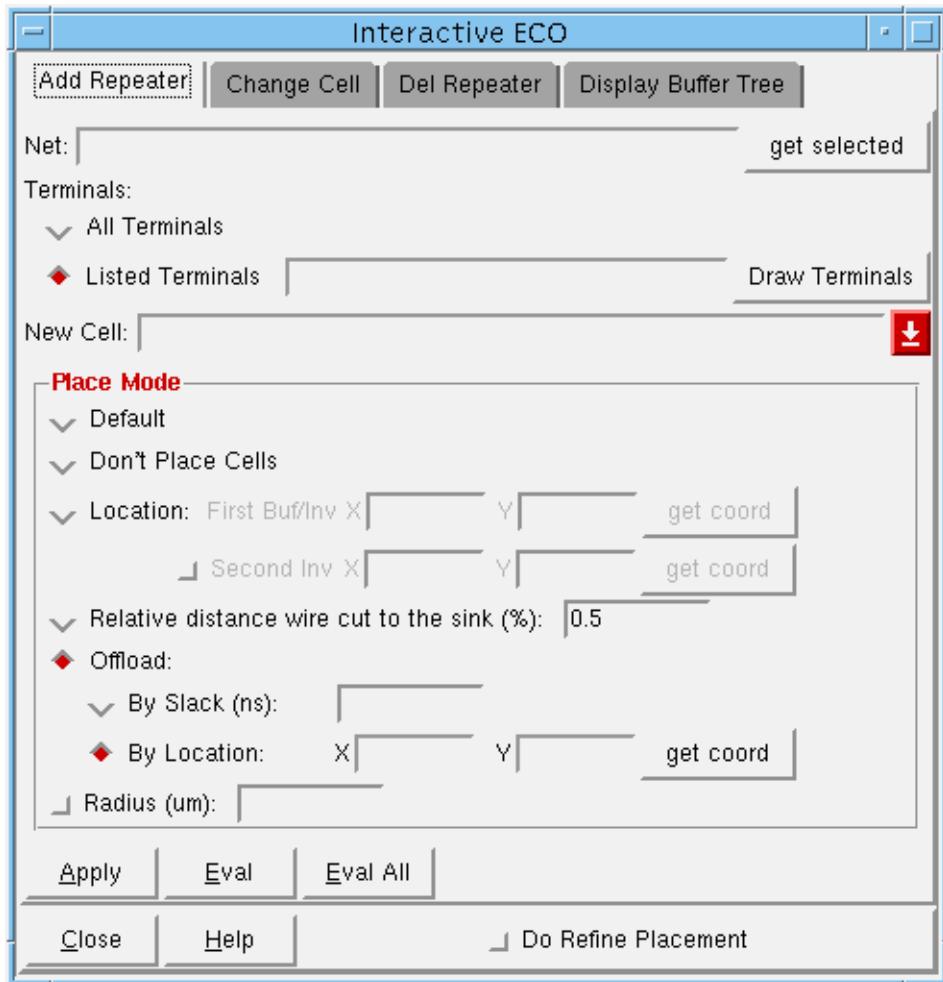
You can add one buffer at a time on a net.

1. To open the Interactive ECO form, select Timing – Interactive ECO from the Encounter menu.

## Encounter User Guide

### Interactive ECO

This opens the Interactive ECO form. The *Add Repeater* page is selected.



**2. Enter the net name in the *Net* field.**

Type the net name, or click on a displayed net in the design display window and click *get selected*.

**3. To select the terminals, choose one of the following:**

- To connect the added buffer to drive all the receivers, specify *All Terminals*. Use this to reduce the delay and output transition time of a weak driver driving a large capacitive load.
- To connect the added buffer to drive the listed receivers, specify *Listed Terminals*. This provides full flexibility for building an arbitrary buffer connection.

- Draw Terminal* button - Allows you to draw an area covering the terminals to which you want to add the buffer.
4. In the *New Cell* field, enter the cell type name of the repeater to add, or click on the arrow to right of the field and select a buffer from the list.
5. In the *Place Mode* pane, you can choose one among several options:
- Default*  
The software automatically determines a location and places the new cell.
  - Don't Place Cells*  
Specifies that the inserted cells should not be placed. Only the logical change in connectivity will be made.
  - Location*  
Enter the location for the buffer using one of the following methods:
    - You can use the automatically assigned locations, enter the locations, or click on an area in the design display window and click *get coord*.
  - Relative Distance to the Sink*  
Specifies the location of the buffer based on its distance from the sink or the driver pin. The value is a number between 0 and 1. A low value (0.1) places the buffer near the sink; a high value (0.9) places the buffer near the driver. The fraction is based on the length of the wire.  
This option works when one term is provided; it does not work if no term or multiple terms are specified.
  - Offload*
    - a. To connect the added buffer to drive only noncritical receivers, select *By Slack*. This checks the timing graph for noncritical receivers and offloads those from the critical path, and could improve critical path timing by penalizing noncritical path delays.
    - b. To add a buffer at a specific location, select *By Location* and enter the x, y coordinates.
6. Specify a radius.
- Specify the radius in which the added instances are free to move. If no legal location can be found in the specified radius, the cells would be placed at the specified location resulting in an overlap with other cells. In that case, you should perform legalization.

7. (Optional) To legalize placement of the ECO changes, click *Do Refine Placement*.
8. Click *Apply*.
9. (Optional) Click the *Eva* button to evaluate the effect on timing if you add a new cell. The values are not applied in the database.
10. (Optional) Click the *Eval All* button to evaluate the effect on timing for all the cell types available for the new cell. The timing report shows the effects of all the cell types, enabling you to select the best cell for your design. The values are not applied in the database.

**Note:** You can also add a buffer around the I/O pin of a block using the [attachIOBuffer](#) command.

The following text command and parameters provide equivalent or additional functionality:

- [ecoAddRepeater](#)

For more information, see “[Interactive ECO Commands](#)” in the *Encounter Text Command Reference*.

## Changing the Cell

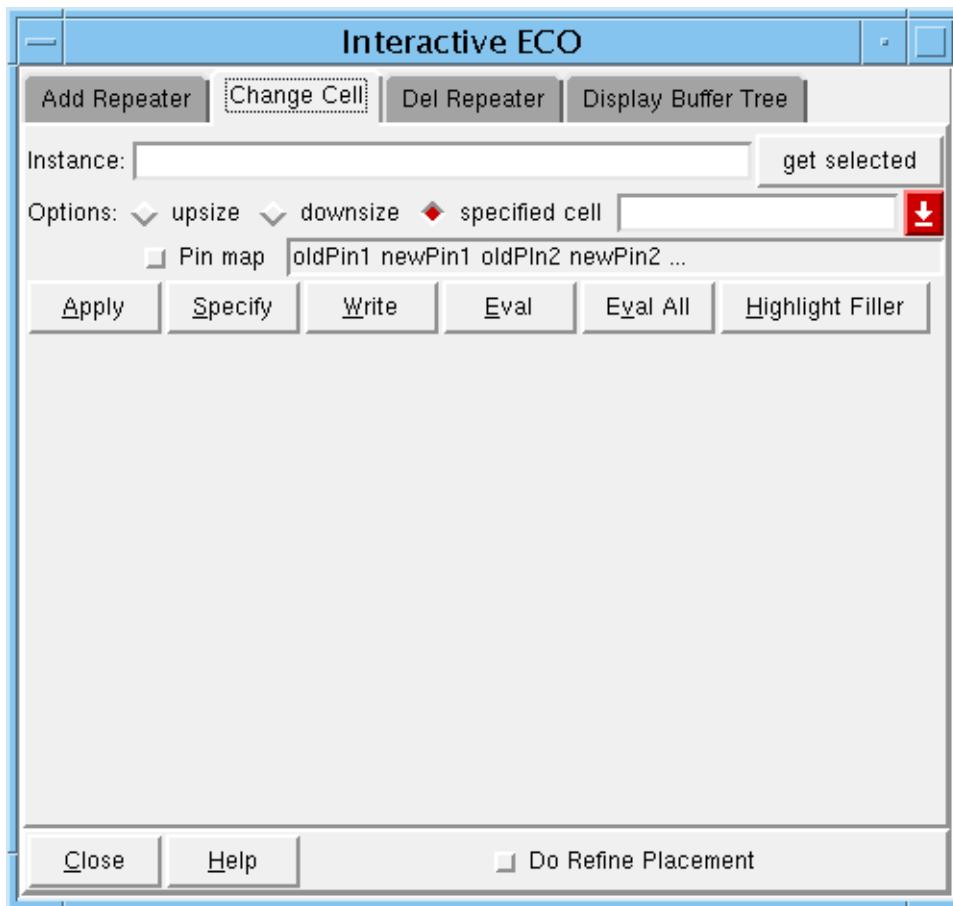
You can upsize or downsize instances. Upsizing an instance that drives a large load can improve the driver delay and the transition time at the receivers. You can also downsize an instance on the noncritical path to reduce the loading of its driver on the critical path.

1. To open the Interactive ECO form, select Timing – Interactive ECO from the Encounter menu, and click the *Change Cell* tab.

## Encounter User Guide

### Interactive ECO

The Change Cell page is displayed.



2. In the Instance field, enter the hierarchical instance name to be changed.

Type the instance name, or click an instance in the design display window and click *get selected*.

3. Select either upsize, downsize, or specified cell. If you select specified cell, enter the replacement cell name in the adjacent field.

Type the cell name, or use the pull-down menu to select a cell.

4. (Optional) Specify the pin mapping for the new cell based on the old cell.

5. (Optional) Click the *Eval* button to evaluate the effect on timing if you add a new cell. The values are not applied in the database.

6. (Optional) Click the *Eval All* button to evaluate the effect on timing for all the cell types available for the new cell. The timing report shows the effects of all the cell types,

enabling you to select the best cell for your design. The values are not applied in the database.

7. (Optional) To legalize placement of ECO changes, click *Do Refine Placement*.
8. Click *Apply*.

The following text command and parameters provide equivalent or additional functionality:

- ecoChangeCell

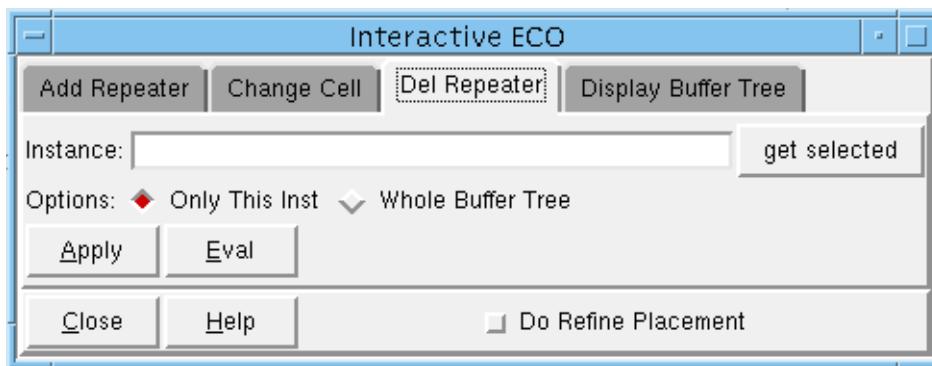
For more information, see “[Interactive ECO Commands](#)” in the *Encounter Text Command Reference*.

## Deleting Buffers

You can delete redundant buffers that cause extra delay. Buffers are typically over-added by synthesis tools based on wireload models.

1. To open the Interactive ECO form, select Timing – Interactive ECO from the Encounter menu, and click the *Delete Repeater* tab.

The Delete Repeater page is displayed.



2. Enter the buffer instance name to be removed in the *Instance* field.

Type the instance name, or click an instance in the design display window and click *get selected*.

3. Select a deletion option: *Only This Instance* or *Whole Buffer Tree*.
4. (Optional) Click the *Eval* button to evaluate the effect on timing if you delete the cell. The values are not applied in the database.
5. (Optional) To legalize placement of ECO changes, click *Do Refine Placement*.

**6. Click *Apply*.**

The following text command and parameters provide equivalent or additional functionality:

- [ecoDeleteRepeater](#)

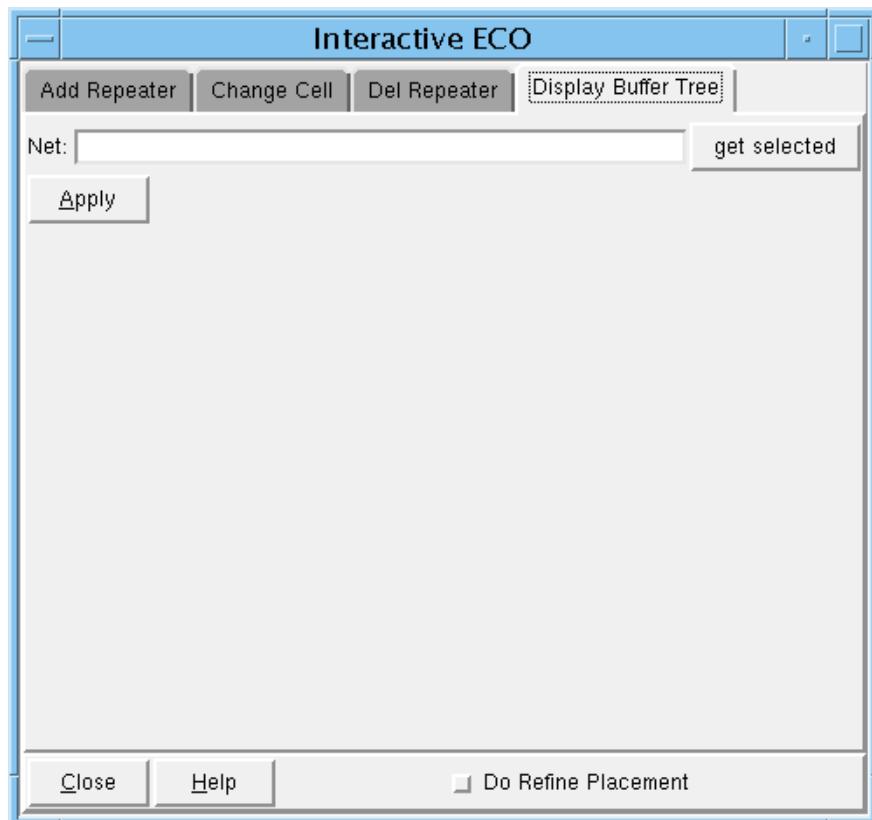
For more information, see “[Interactive ECO Commands](#)” in the *Encounter Text Command Reference*.

## Displaying Buffer Trees

You can inspect the routing topology of the buffer tree after it is created. If the buffer tree requires correction, you can rebuild or modify it through the other three pages in the Interactive ECO form.

1. To open the Interactive ECO form, select Timing – Interactive ECO from the Encounter menu, and click the *Display Buffer Tree* tab.

The Display Buffer Tree page is displayed.



## **Encounter User Guide**

### Interactive ECO

---

- 2.** To select a buffer tree, enter the net name in the *Net* field.

You can type the net name, or click a net in the design display window and click *get selected*.

- 3.** (Optional) To legalize placement of ECO changes, click *Do Refine Placement*.
- 4.** Click *Apply*.

The following text command provides equivalent or additional functionality:

- displayBufTree

For more information, see “[Interactive ECO Commands](#)” in the *Encounter Text Command Reference*.

## Running ECO Placement

ECO placement updates the placement from a prior Encounter session to reflect the netlist changes, merging the new netlist changes into the prior netlist’s placement. The modified netlist can then be imported into an Encounter session so that the result is a new placement that reflects the changes made in the modified netlist.

You can run either an incremental timing or logic change to the design. You can run ECO after running placement, although ECO is usually run after analyzing speed or RC data.

To update the placement with the ECO netlist, complete the following steps:

1. Save the pre-ECO netlist placement data.
2. Start a new Encounter session.
3. Import the (ECO) design.
4. Load the floorplan.
5. Run ECO Placement.

This references the pre-ECO netlist placement data. The changes reflected in the new netlist are ECO’d into the pre-ECO placement. All designs are placed in the resulting placement.

After running ECO successfully, you can run Trial Route to view the routing congestion and analyze the design for timing.

# Naming Conventions for Interactive ECO

After running interactive ECO, you can use the Design Browser to view the newly added instance names, prefixed with `FE_`. The interactive ECO operation naming conventions are described in the following table:

Name Prefix	Description
FE_ECON	A net added by interactive ECO
FE_ECOC	An instance added by interactive ECO

# Comparing Physical Design Data

After making changes to a DEF file, you can compare the new file to the information stored in the Encounter database. You can perform this comparison after you perform ECO.

Use the following command to compare physical design data:

```
defComp defFile -reportFile fileName
```

The default filename is defPhyDiff.rpt

The report file includes the following information:

- ## ■ VERSION statement

VERSION *num*

*num* Specifies the file version number.

- ## ■ UNITS statement

UNITS *num*

*num* Specifies the unit for the values such as coordinates, width, and so on.

- #### ■ ADDINST statement

**ADDINST** *instName* *cellName* *x* *y* *orientation*

*instName*      Specifies the instance name.

`cellName` Specifies the master cell name of the instance.

## Encounter User Guide

### Interactive ECO

---

<i>x y</i>	Specifies the coordinates of the instance.
<i>orientation</i>	Specifies the orientation of the instance. The orientation can be one of the following: N, FN, S, FS, W, FW, E, FE, as used by DEF file.

**Note:** The report provides the connectivity of added instances in the NETS section described below.

■ **DELINST statement**

```
DELINST instName cellName x y orientation
```

The arguments have the same meaning as described in ADDINST statement.

■ **CHANGECELL statement**

```
CHANGECELL instName oldCellName newCellName
```

The master cell of the instance *instName* is changed from *oldCellName* to *newCellName*. The coordinate, orientation, and connectivity of the instance are not changed. If a master cell is changed along with any of the coordinates, orientation, or connectivity of the instance, Encounter considers this change as a deletion and addition of the instance. Rather than including a CHANGECELL statement, the file contains one pair of DELINST and ADDINST statements.

■ **MOVEINST statement**

```
MOVEINST instName [COORD oldX oldY newX newY] [ORIENT oldOrientation newOrientation]
```

The statement contains the COORD phrase if the instance *instName* has moved, and the ORIENT phrase if the instance has changed orientation.

■ **ADDNET statement**

```
ADDNET netName
```

*netName* Specified the name of a added net.

■ **DELNET statement**

```
DELNET netName
```

*netName* Specifies the name of the deleted net.

## Encounter User Guide

### Interactive ECO

---

#### ■ ADDPIN statement

```
ADDPIN netName instName pinName
```

<i>netName</i>	Specifies the net from which the pin is added.
<i>instName</i>	Specifies the instance name of the added pin.
<i>pinName</i>	Specifies the name of the added pin.

#### ■ DELPIN statement

```
DELPIN netName instName pinName
```

<i>netName</i>	Specifies the net from which the pin is deleted.
<i>instName</i>	Specifies the instance name of the deleted pin.
<i>pinName</i>	Specifies the name of the deleted pin.

#### ■ CHANGEROUTE and ENDCHANGEROUTE statements

```
CHANGEROUTE netName
```

```
ENDCHANGEROUTE
```

These statements mark the beginning and end of the CHANGEROUTE section that contains changes on the routing segment on net *netName*. The wire change statements are included between the CHANGEROUTE and ENDCHANGEROUTE statements.

#### ■ POWERROUTE and ENDPOWERROUTE statements

```
POWERROUTE netName
```

```
ENDPOWERROUTE
```

These two statements mark the beginning and the end of the POWERROUTE section that contains the power routing differences between two DEF files. The wire change statements are included between the POWERROUTE and ENDPOWERROUTE statements.

#### ■ Wire change statements

The ADDWIRE, DELWIRE, ADDVIA, and DELVIA statements appear between the CHANGEROUTE and ENDCHANGEROUTE statements, or POWERROUTE and ENDPOWERROUTE statements.

##### □ ADDWIRE statement

```
ADDWIRE layerName width x1 y1 x2 y2
```

<i>layerName</i>	Specifies the layer name of wire segment added to the net.
<i>width</i>	Specifies the width of the wire segment added to the net.
<i>x1 y1</i>	Specifies the left or bottom coordinate of wire segment added to the net.
<i>x2 y2</i>	Specifies the right or top coordinate of wire segment added to the net.

❑ DELWIRE statement

```
DELWIRE layerName width x1 y1 x2 y2
```

<i>layerName</i>	Specifies the layer name of wire segment deleted to the net.
<i>width</i>	Specifies the width of the wire segment deleted to the net.
<i>x1 y1</i>	Specifies the right or top coordinate of wire segment deleted from the net.
<i>x2 y2</i>	Specifies the left or bottom coordinate of wire segment deleted from the net.

❑ ADDVIA statement

```
ADDVIA [botLayerName bx1 by1 bx2 by2] topLayerName tx1 ty1 tx2 ty2
```

<i>botLayerName</i>	Specifies the bottom layer name of the via added to the net.
<i>bx1 by1</i>	Specifies the lower-left coordinate of bottom layer of the via added to the net.
<i>bx2 by2</i>	Specifies the top-right coordinate of bottom layer of the via added to the net.
<i>topLayerName</i>	Specifies the top layer name of via added to the net.
<i>tx1 ty1</i>	Specifies the lower-left coordinate of top layer of the via added to the net.
<i>tx2 ty2</i>	Specifies the top-right coordinate of top layer of the via added to the net.

**Note:** For turnvias, Encounter reports *topLayerName* only.

## Encounter User Guide

### Interactive ECO

---

#### □ DELVIA statement

```
DELVIA [botLayerName bx1 by1 bx2 by2] [topLayerName tx1 ty1 tx2 ty2]
```

*botLayerName*      Specifies the bottom layer name of via deleted from the net.

*bx1* *by1*      Specifies the lower-left coordinate of bottom layer of the via deleted from the net.

*bx2* *by2*      Specifies the top-right coordinate of bottom layer of the via deleted from the net.

*topLayerName*      Specifies the top layer name of via deleted from the net.

*tx1* *ty1*      Specifies the lower-left coordinate of top layer of the via deleted from the net.

*tx2* *ty2*      Specifies the top-right coordinate of top layer of the via deleted from the net.

**Note:** For turnvias, Encounter reports *topLayerName* only.

#### ■ ADDOBS statement

```
ADDOBS layerName x1 y1 x2 y2
```

*layerName*      Specifies the layer name for the obstruction added.

*x1* *y1*      Specifies the lower-left coordinate of the obstruction.

*x2* *y2*      Specifies the top-right coordinate of the obstruction.

#### ■ DELOBS statement

```
DELOBS layerName x1 y1 x2 y2
```

*layerName*      Specifies the layer name of the deleted obstruction.

*x1* *y1*      Specifies the lower-left coordinates of the obstruction.

*x2* *y2*      Specifies the upper-right coordinates of the obstruction.

**Encounter User Guide**  
Interactive ECO

---

---

## Analyzing and Repairing Crosstalk

---

- [Overview](#) on page 1062
- [Inputs and Outputs for SI Analysis](#) on page 1063
- [Setting Up Encounter for SI Analysis](#) on page 1064
  - [RC Extraction Settings](#) on page 1064
  - [Noise Analysis Settings](#) on page 1066
  - [Static Timing Analysis \(STA\) Settings](#) on page 1068
  - [Advanced Settings for SI Analysis](#) on page 1069
  - [Example of Setting Up Encounter for SI Analysis](#) on page 1073
- [Preventing Crosstalk Violations](#) on page 1074
- [Fixing Crosstalk Violations](#) on page 1075
  - [Data Preparation](#) on page 1075
  - [Using optDesign to Fix Setup Violations with Crosstalk Effects](#) on page 1076
  - [Using optDesign to Fix Hold Violations with Crosstalk Effects](#) on page 1078
  - [Using optDesign to Fix Transition Time Violations with Crosstalk Effects](#) on page 1080
- [Performing XILM-Based SI Analysis and Fixing](#) on page 1083

## Overview

Crosstalk is the undesired electromagnetic coupling between signal lines that causes functional failures and delay variation. The effects of crosstalk might slow down or speed up the delay depending on the transition direction of the two coupling nets.

The Encounter™ software supports signal integrity (SI) operations that include crosstalk prevention, analysis, and repair. The software uses an advanced crosstalk repair algorithm which features:

- Gate sizing
- Buffer insertion
- Victim spacing and protection

**Note:** NanoRoute™ is seamlessly integrated with these tools to perform all crosstalk analysis and repair operations.

Analyzing and repairing crosstalk is part of the signal integrity closure repair loop, which reduces both timing and crosstalk violations starting from the prevention stage to the post-route optimization stage in the design flow.

## Inputs and Outputs for SI Analysis

The following design input files are required in order to repair crosstalk violations:

- Netlist
- SDC (timing information)
- Routed Encounter database or DEF file (placement and routing information)
- LEF file (physical library)
- .cdb noise library
- XILM data (for backward compatibility, ECHO models are also supported.)
- Liberty library (.lib)
- Encounter extended capacitance table file
- QRC standalone extraction technology file and library (optional)

**Note:** Before you begin, ensure that your routed design meets all the timing requirements.

When finished, the software produces an Encounter database optimized for crosstalk violations. The following files are generated:

- Incremental SDF file
- Incremental transition time file
- Timing reports with and without incremental delays



Use the `-detailedReports` parameter of the `setSIMode` command to control the report files generated during SI analysis. Set this parameter to `true` if you want to enable detailed reporting for debugging purposes.

## Setting Up Encounter for SI Analysis

- [RC Extraction Settings](#)
- [Noise Analysis Settings](#)
- [Static Timing Analysis \(STA\) Settings](#)
- [Advanced Settings for SI Analysis](#)
- [Example of Setting Up Encounter for SI Analysis](#)

### RC Extraction Settings

The RC extraction settings for SI analysis include the extraction engine and the extraction filters.

#### Extraction Engine

The extraction engine can be one of detail or CCE. By default, Encounter uses detailed extraction for post-route timing and/or optimization. However, if you want better quality extraction compared to the detail engine and best correlation with standalone QRC signoff extraction, use the CCE engine.

**Note:** CCE extraction requires a separate QRC license.

To use the CCE extraction engine, use the following command:

```
setExtractRCMode -engine CCE
```

**Note:** In 7.1 USR3 and prior releases, `timeDesign` and `optDesign` commands did not honor the `setExtractRCMode -engine` settings and forced the use of detailed extraction.

#### Extraction Filters

Extraction filters enable you to reduce the total number of parasitic capacitors in the design, by decoupling coupled capacitors from nets whose total capacitance does not exceed a specific amount, or whose coupling capacitance does not exceed a specific amount, or merely by decoupling very small coupling capacitors.

#### Effect of RC Extraction Settings on SI Analysis

Extraction coupled capacitance filtering has the most significant impact on SI analysis and run time. The Encounter software automatically sets the default values for the RC extraction

filters based on the process node specified using the `-process` parameter of the `setDesignMode` command. To set the process node, use the following command:

```
setDesignMode -process process_node
```

**Note:** For more information on the default values assigned to the filtering parameters with respect to the specified process node, see the `setDesignMode` command.

If you do not want to use the default filtering values for RC extraction, specify the following parameters of the `setExtractRCMode` command to tweak the coupling capacitance filters:

- `-total_c_th`: Specifies the threshold value (femtoFarads) that determines when the extractor lumps a net's coupling capacitance to ground. The software grounds the coupling capacitances for nets which have a total capacitance value less than the value specified with this parameter.
- `-coupling_c_th`: Specifies the threshold value that determines when the extractor lumps a net's coupling capacitance to ground. The software decouples the coupling capacitance of nets when the total coupling capacitance between the pair of nets is lower than the threshold specified with this parameter.
- `-relative_c_th`: Sets a ratio threshold value that determines when the extractor lumps a net's coupling capacitance to ground. If the total coupling capacitance between a pair of nets is less than the percentage (specified with this parameter) of the total capacitance of the net with the smaller total capacitance in the pair, the coupling capacitance between these two nets will be considered for grounding.

## Guidelines for RC Extraction Settings

Use the following guidelines while setting up extraction:

- Note that the detailed extraction engine (default extraction engine for SI analysis) is significantly faster compared to CCE. However, it trades off accuracy of the extraction results for performance.
- Ensure that the filters that you are using in Encounter for SI fixing are the same as the ones used for SI signoff analysis.
- The default filtering values set by Encounter based on the process node (`setDesignMode -process`) attempt to capture the most significant effects of coupling capacitances on SI analysis. It is strongly recommended that you correlate your RCs using these default filters (set by Encounter) with the RCs from your signoff extractor.
- While setting the filtering thresholds, ensure that you retain small coupling capacitors because SI analysis lumps these together into a single virtual attacker model. Multiple small coupling capacitors can result in a significant virtual attacker.

- Exercise caution while setting low-value filters because this can increase the run time significantly.

## Noise Analysis Settings

Noise analysis settings include loading the input noise model, configuring the timing windows, choosing the noise analysis engine, setting the delta delay threshold, and specifying the virtual attacker mode.

### Noise Models

The primary input for noise analysis is a **cdb** library, which contains characterized noise data. In the absence of a **cdb** file, you can use a **Liberty** library (**.lib**) file. However, it is strongly recommended that you use a **cdb** noise library for SI analysis.

**Note:** For hierarchical designs, you need XILM data. For more information on XILM-based SI analysis, see the [Using Interface Logic Models in Hierarchical Designs](#) chapter in the *Encounter User Guide*.

### Timing Windows

Timing windows are used to filter out signals that are not switching simultaneously. The internal timing engine computes the timing windows and slew rates automatically. To use a conservative approach for analysis, set the timing windows to infinite switching mode as shown:

- `setSIMode -noiseTwfMode "-infSW"`

### Noise Analysis Engine

The Encounter software uses native signal integrity analysis to perform crosstalk analysis. This engine is same as the one used by Encounter Timing System. If you are using Encounter Timing System as the SI signoff tool, the same settings can be applied in Encounter for performing noise analysis.

To use Encounter Timing System commands in Encounter, use the **-insCeltICPreTcl** and **-insCeltICPostTcl** parameters of the [setSIMode](#) command, as shown:

```
setSIMode -insCeltICPreTcl { command_name; }
setSIMode -insCeltICPostTcl { command_name; }
```



### *Tip*

The Cadence® Encounter® Timing System provides a sign-off timing and signal integrity solution for a design flow and is shipped with the ETS release.

## **Delta Delay Threshold**

You can set the delta delay threshold for noise-on-delay analysis using the following command:

- `setSIMode -deltaDelayThreshold value`

## **Virtual Attacker Mode**

To efficiently model the many small attackers on a victim net, the SI analysis engine replaces them with an imaginary net, called a virtual attacker. The two main characteristics of a virtual attacker are : coupling capacitance, and the voltage source waveform used as a transition on the virtual attacker.

There are two methods to create and control the characteristics of the virtual attacker.

- Coupling Capacitance-Based Method

Matches the total coupling capacitance of a virtual attacker with the small attackers that are being virtualized.

- Current Matching-Based Method

Models the transitions on virtual attackers using piece-wise linear (PWL) waveforms. The PWL waveforms are generated in a way that the current induced by the transition matches the total current induced by all virtual attacker components.

Use the following command to setup the virtual attacker mode:

```
setSIMode -insCeltICPreTcl { set_virtual_attacker option1 option2 }
```

## **Guidelines for Noise Analysis Settings**

- When using a standalone tool for SI analysis such as Encounter Timing System, use the same noise analysis settings in Encounter for best correlation results.
- If you are using Encounter Timing System or any other third-party signoff solution, the following settings will have the most significant impact on correlation of analysis results:
  - Delta delay threshold

## Encounter User Guide

### Analyzing and Repairing Crosstalk

---

Set the delta delay threshold value to 0 to ensure that the software retains the small incremental delay pushouts:

```
setSIMode -deltaDelayThreshold 0
```

- Virtual attacker mode

Use the following command for specifying the virtual attacker settings:

```
setSIMode -insCeltICPreTcl { set_virtual_attacker -gtol 0.025 -mode current; }
```

**Note:** Although these settings will capture most of the noise violations in a pessimistic way, it is still recommended that you match the settings with the settings of your signoff tool.

## Static Timing Analysis (STA) Settings

Static timing analysis settings include the timing analysis engine and the analysis conditions.

### Input Timing Library

The primary input for timing analysis is a Liberty (.lib) library.

### STA Engine

Encounter uses the native timing analysis engine to perform static timing analysis. This engine is same as the one used by Encounter Timing System. You can choose to use the built-in timing analysis engine or a third-party tool for performing timing analysis.

### Analysis Conditions

The important analysis conditions for running SI analysis include:

- Setting Up the Analysis Mode

Encounter software provides different timing analysis modes and performs different calculations for setup and hold checks for each mode. For SI analysis, set the analysis mode to On-Chip Variation using the [setAnalysisMode](#) command:

```
setAnalysisMode -analysisType onChipVariation
```

OCV mode assumes that capture clock, launch clock, and data path can have maximum or minimum delays in setup and hold analysis.

 *Important*

To specify the OCV analysis mode, you must use the multi-mode multi-corner (MMMC) framework regardless of whether your design is an MMMC design. For more information on setting up Multi-Mode Multi-Corner, see [Performing Multi-Mode Multi-Corner Timing Analysis and Optimization](#).

■ **Removing Common Path Pessimism**

To remove the common path pessimism, use the following command:

```
setAnalysisMode -cppr true
```

The `-cppr` parameter removes pessimism from clock paths that have a portion of the clock network in common between the clock source and clock destination paths. The pessimism is introduced when the timing analysis tools assume that the common path has different delay values for two different paths in case of on-chip variation.

■ **Enabling Accurate CPPR Analysis When Incremental Delays are Present**

To enable accurate CPPR analysis in the presence of incremental delays, set the following variable:

```
set_global timing_enable_si_cppr true
```

When this variable is set to `true`, the incremental delay is not included in the CPPR calculation during setup analysis, but is included in the CPPR calculation during hold analysis.

## Advanced Settings for SI Analysis

- [Multi-CPU Processing Settings](#)
- [Path Group Settings for SI Optimization](#)

### Multi-CPU Processing Settings

Encounter supports multi-threaded, distributed, and super-threaded noise analysis for MMMC designs, and multi-threaded and distributed noise analysis for non-MMMC designs. Multi-CPU processing support, in general, reduces the noise analysis run time significantly.

The following command considers the multi-CPU processing settings during noise analysis:

- `optDesign -postRoute -si`
- `timeDesign -postRoute -si`



For information on multi-CPU processing, see [Accelerating the Design Process by Multiple-CPU Processing](#).

### **Setting Up Multi-Threading for Noise Analysis**

Multi-threading enables you to run noise analysis on multiple CPUs of a single host. The host machine could be the one on which you are running Encounter or a different host.

- To setup multi-threaded noise analysis for non-MMMC designs on the same host, use the following command:

```
setMultiCpuUsage -numThreads number  
optDesign -postRoute -si
```

Alternatively, you can use the `-superThreadsNumThreads` parameter to specify the number of threads for multi-threading. This parameter will have the same effect as the `-numThreads` parameter when used without `-numThreadsNumHosts` parameter:

```
setMultiCpuUsage -superThreadsNumThreads number  
optDesign -postRoute -si
```

- To setup multi-threaded noise analysis for non-MMMC designs on a different host, use the following set of commands:

```
setDistributeHost -rsh -add { hostname }  
setMultiCpuUsage -superThreadsNumHosts 1 -superThreadsNumThreads number  
optDesign -postRoute -si
```

### **Setting Up Distributed Processing for Noise Analysis**

Distributed processing enables you to divide a noise analysis job between two or more networked computers running concurrently.

- To setup distributed processing using RSH, specify the following commands:

```
setDistributeHost -rsh -add { host1 host2 host3 host4 ... hostN }  
setMultiCpuUsage -numHosts number  
optDesign -postRoute -si
```

- To setup distributed processing using LSF, specify the following commands:

```
setDistributeHost -lsf -queue myLSFqueue  
setMultiCpuUsage -numHosts number  
optDesign -postRoute -si
```



When setting up distributed processing for MMMC designs, ensure that there is a corresponding host computer for each view definition. For example, if your MMMC design has four view definitions, divide these between four host computers.

### **Setting Up Super-Threaded Noise Analysis for MMMC Designs**

Super-threading enables you to run a noise analysis job on both distributed processing and multi-threaded modes; that is, two or more networked computers, each with multiple processors, can be called to concurrently run noise analysis. The super-threaded mode is supported for MMMC designs only.

Enable super-threaded noise analysis for MMMC designs:

- To setup super-threading using RSH, use the following commands:

```
setDistributeHost -rsh -add {host1 host2 host3 .... hostN}  
setMultiCpuUsage -superThreadsNumHosts number \  
                  -superThreadsNumThreads number  
optDesign -postRoute -si
```

- To setup super-threading using LSF, use the following commands:

```
setDistributeHost -lsf -queue myLSFqueue  
setMultiCpuUsage -superThreadsNumHosts number \  
                  -superThreadsNumThreads number  
optDesign -postRoute -si
```

**Note:** The `-superThreadsNumThreads` and `-superThreadsNumHosts` parameters can be used exclusive of each other.

### **Examples of Setting Up Multi-CPU Processing**

- The following distributed host settings distribute one view each on `host1` and `host2` machines for a two-view MMMC design using RSH:

```
setDistributeHost -rsh -add { host1 host2 }  
setMultiCpuUsage -numHosts 2  
optDesign -postRoute -si
```

- The following multi-threading settings run each view (in eight threads) for a two-view MMMC design sequentially:

```
setMultiCpuUsage -numThreads 8  
optDesign -postRoute -si
```

- The following settings distribute one view each on host1 and host2 machines for a two-view MMMC design using RSH:

```
setDistributeHost -rsh -add { host1 host2 }
setMultiCpuUsage -numHosts 2 -numThreads 8
optDesign -postRoute -si
```

**Note:** When used together, the `-numHosts` parameter is given preference over the `-numThreads` parameter. In this case, the `-numThreads` parameter is ignored. If you intend to use multiple hosts and multiple threads at the same time during SI optimization, use `-superThreadsNumHosts` and `-superThreadsNumThreads` parameters instead.

- The following super-threading settings distribute one view each on host1 and host2 machines for a two-view MMMC design using RSH, and run each view divided between eight threads each:

```
setDistributeHost -rsh -add { host1 host2 }
setMultiCpuUsage -superThreadsNumHosts 2 \
                 -superThreadsNumThreads 8
optDesign -postRoute -si
```

**Note:** In this case, a total of 16 CPUs will be used.

### ***Guidelines for Setting Up Multi-CPU Processing for SI Analysis***

- Multi-threading scales better for large designs during single corner analysis.
- Distributed processing scales better for multi-mode multi-corner designs.

### **Path Group Settings for SI Optimization**

The SI optimization flow accounts for the SDC path groups. To enable path groups, use the following command:

- `setAnalysisMode -honorClockDomains false`

When you specify this command, the software disables the use of standard and user-defined clock domains during optimization and uses path groups for timing and SI optimization instead. This enables you to take advantage of the path group effort level during SI optimization. In other words, optimization is run on path groups for which the effort level is set to high (`setPathGroupOptions -effortLevel high`).

In addition, you can use:

- The `-slackAdjustment` parameter of the [setPathGroupOptions](#) command for the slack adjustment value.
- The `-acceptableWNS` parameter of the [setSIMode](#) command for the target slack value.

The path group settings provide you the flexibility to run optimization on selective paths that belong to critical parts of the design.

The following command accounts for path groups:

- [optDesign -postroute -si](#)



To create, modify, and report path groups, use the following commands:

- [group\\_path](#)
- [reset\\_path\\_group](#)
- [report\\_path\\_groups](#)
- [createBasicPathGroups](#)
- [setPathGroupOptions](#)
- [resetPathGroupOptions](#)
- [reportPathGroupOptions](#)

## Example of Setting Up Encounter for SI Analysis

The following example provides a summary of the settings that are required for performing SI analysis:

```
## Extraction Settings ##
if { $PROCESS == "65nm" } {
    setExtractRCMode -engine detail -rcdb /tmp/extract -optMemory true \
        -relative_c_th 1.0 -total_c_th 0.00001 -coupling_c_th 0.01 \
        -capFilterMode relAndCoup
}

## SI Settings ##
setSIMode -noiseTwfMode "-infSW" \
```

```
-noiseProcess $PROCESS \
-deltaDelayThreshold 0 \
-insCeltICPreTcl { set_virtual_attacker -gtol 0.025 -mode current; }
```

```
## STA settings ##
## Make sure MMMC/SMSC is used
setAnalysisMode -analysisType onChipVariation -cppr true
## CPPR Removal
set_global timing_enable_si_cppr true
```

## Preventing Crosstalk Violations

The following steps enable you to prevent crosstalk violations:

1. Place and optimize your design. Set reasonable max transition thresholds.
2. Fix transition time violations on the Clock tree aggressively.
3. Run timing and signal integrity driven routing using the [routeDesign](#) command.

**Note:** To enable the reduction of SI effects while fixing the base timing during post-route optimization, set the `-postRouteSiAware` parameter of the [setOptMode](#) command to `true`. When you set this parameter to `true`, the software reduces the SI violations which results in faster timing closure at the signoff stage.

## Fixing Crosstalk Violations

- [Data Preparation](#)
- [Using optDesign to Fix Setup Violations with Crosstalk Effects](#)
- [Using optDesign to Fix Hold Violations with Crosstalk Effects](#)
- [Using optDesign to Fix Transition Time Violations with Crosstalk Effects](#)

### Data Preparation

In order to analyze and repair crosstalk violations properly, you must prepare your data correctly.

**1.** Read in signal integrity related data and libraries.

**a.** Generate the noise library.

Use the `make_cdb` utility to create the characterized noise library. For blocks, you can save a block-level cdb in Encounter if the block will be represented as a black-box on the timing side.

The `make_cdb` utility can be run in interactive mode or in batch mode. For information on how to use the `make_cdb` utility, see the *make\_cdB Noise Characterizer User Guide*.

**2.** Check that the data and libraries are correct and that there are no timing violations.

**a.** Check your congestion map and resolve any highly congested areas using the following command:

`congOpt`

**b.** Correct transition time violations.

**c. Perform RC Extraction and Correlation**

You can use native detailed extraction, CCE, or QRC standalone sign-off extraction to extract the routed design. To correlate native extraction results with sign-off extraction, you compare SPEF files from basic and sign-off extraction to generate the new scaling factors for total capacitance, cross-coupling capacitance, and resistance. Using these scaling factors, the native extraction results are closer to the sign-off extraction results, while only taking a fraction of the run time required for sign-off extraction. For more information, see [“Correlating Native Extraction With Sign-Off Extraction”](#).

**3. Read in a placed and routed database.**

```
restoreDesign your_design.dat your_topCell
```

## Using optDesign to Fix Setup Violations with Crosstalk Effects

Use the `optDesign` command with the `-postRoute` and `-si` parameters to correct glitch and setup violations caused by incremental delays, which occur due to coupling capacitance. The `optDesign` command fixes additional setup time violations caused by incremental delay up to a target slack equaling the worst negative slack without incremental delays. It is recommended that a design be optimized to the worst negative slack without incremental delays before performing SI fixing. The `optDesign` command avoids the degradation of the worst negative slack without incremental delays while fixing the setup time violations with incremental delays.

The setup time violations are fixed before glitch violations. By default, two iterations are done for SI fixing. However you can use the `-maxNriter` parameter of the `setSIMode` command to specify a different number for the iterations to be performed.



*Important*  
For best results, setup Encounter SI analysis to match SI sign-off analysis before using the `optDesign` command.

## Using RC Data Generated by an External Tool for SI Fixing

You can load the RC data generated by an external tool by using the `spefIn` command in Encounter to do SI fixing. When RC data from an external tool is used, SI fixing is limited to one iteration. RC data needs to be regenerated using the third-party tool to perform SI analysis and generate reports after fixing.



For MMMC designs, ensure that you use the `specIn` command to load the parasitic data for each corner.

## Using SDF Data Generated by an External Tool for SI Fixing

You can use SDF data generated by an external tool in Encounter to do limited SI fixing. When SDF data from an external tool is used, SI fixing is limited to one iteration of fixing. SDF data needs to be regenerated with the external tool to perform SI analysis and generate reports after fixing.

### ***Fixing Setup Violations Using External SDF for Non-MMMC Designs***

For non-MMMC designs, the SDF file specified with the `-setupSdfFile` parameter of the `setOptMode` command is used for fixing setup violations.

- Use the following command to specify the SDF file for SI fixing in non-MMMC designs:

```
setOptMode -setupSdfFile filename
```

The other requirements for using the external SDF file for SI setup fixing flow are:

- Specify the target slack using the `-acceptableWNS` parameter of the `setSIMode` command:

```
setSIMode -acceptableWNS value
```

**Note:** This is a mandatory setting.

- Enable the external SDF based SI fixing flow:

```
optDesign -postRoute -si -useSDF
```

### ***Fixing Setup Violations Using External SDF for MMMC Designs***

For MMMC designs, use the `read_sdf` command to load an SDF file for each view.

- Use the following commands to load separate SDF files for two different views:

```
read_sdf -view viewname1 filename1  
read_sdf -view viewname2 filename2
```

- Enable the external SDF based SI setup fixing flow:

```
optDesign -postRoute -si -useSDF
```

## Using optDesign to Fix Hold Violations with Crosstalk Effects

Use the `optDesign` command with the `-postRoute`, `-hold`, and `-si` parameters to correct hold violations caused by incremental delays, which occur due to coupling capacitance. The `optDesign` command corrects additional hold violations caused by incremental delays up to a target slack equaling the hold worst negative slack without incremental delays. It is assumed that a design has been optimized to the best hold worst negative slack without incremental delays before performing SI fixing. The `optDesign` command does not degrade the hold worst negative slack (without incremental delays) and setup worst negative slack (without incremental delays) while fixing the hold violations with incremental delays.



By default, setup SI analysis is not performed. However, you can choose to enable setup SI analysis while performing SI hold fixing. In this case, the `optDesign` command will try not to degrade setup violations with incremental delays while fixing hold violations with incremental delays. To account for crosstalk setup timing information while performing SI hold fixing, use the following setting:

```
setSIMode -fixHoldIncludeXtalkSetup true
```

## Using RC Data Generated by an External Tool for SI Hold Fixing

You can load the RC data generated by an external tool by using the `spefIn` command in Encounter to do limited SI hold fixing. When RC data from an external tool is used, SI hold fixing is limited to one iteration. RC data needs to be regenerated with the external tool to perform SI analysis and generate reports after fixing.



For MMMC designs, ensure that you use the `spefIn` command to load the parasitic data for each corner.

## Using SDF Data Generated by an External Tool for SI Hold Fixing

You can also use SDF data generated by an external tool in Encounter to do limited SI hold fixing. When SDF data from an external tool is used, SI hold fixing is limited to one iteration. SDF data needs to be regenerated with the external tool to perform SI analysis and generate reports after fixing.

### ***Fixing Hold Violations Using External SDF for Non-MMMC Designs***

For non-MMMC designs, the SDF file specified with the `-holdSdfFile` parameter of the [setOptMode](#) command is used for SI hold fixing.

- Use the following command to specify the SDF file for both setup and hold fixing:

```
setOptMode -setupSdfFile filename -holdSdfFile filename
```

The other requirements for using the external SDF file for SI fixing flow are:

- Specify the target slack using the `-acceptableWNS` parameter of the [setSIMode](#) command:

```
setSIMode -acceptableWNS value
```

**Note:** This is a mandatory setting.

- Enable the external SDF based SI hold fixing flow:

```
#When you are performing setup and hold fixing in the same flow.
```

```
setOptMode -setupSdfFile filename  
optDesign -postRoute -si -hold -useSDF
```

### ***Fixing Hold Violations Using External SDF for MMC Designs***

For MMC designs, use the [read\\_sdf](#) command to load an SDF file for each view.

- Use the following commands to load separate SDF files for two different views:

```
read_sdf -view viewname1 filename1  
read_sdf -view viewname2 filename2
```

- Enable the external SDF based SI fixing flow:

```
optDesign -postRoute -si -hold -useSDF
```

## Using optDesign to Fix Transition Time Violations with Crosstalk Effects

You can now fix SI induced maximum transition violations in Encounter by using:

- A transition file generated during noise analysis
- External transition file(s) generated by third-party tools.

### SI Transition Violation Fixing

In the default flow, use the `-fixDRC` parameter of the `setSIMode` command to perform maximum transition violation fixing. After setting this parameter to `true`, run the `optDesign -postRoute -si` command to perform maximum transition violation fixing using transition information obtained from the transition file (`celtic.slew`) generated during noise analysis.

**Note:** The transition values will be used for maximum transition violation fixing only and will not affect the timing calculation in Encounter's timing engine. This is because the transition effect is considered by the internal SI engine during delay pushout calculation.

The following commands support the `-fixDRC` parameter of the `setSIMode` command, and report the transition violations in the Timing Summary report.

- `timeDesign -postRoute -si`
- `timeDesign -signoff -si`
- `timeDesign -reportOnly -si`

### Transition Violation Fixing Using Transition File Generated During Noise Analysis Without Glitch or Noise-On-Delay Fixing

Use the `-drv` parameter of the `optDesign` command to perform DRV fixing only:

```
optDesign -postRoute -si -drv
```

In this flow, the software does not perform glitch and noise-on-delay fixing.



The `-hold` parameter cannot be specified when using the `-drv` parameter with the `-si` parameter.

## Transition Violation Fixing Using External Transition File(s)

Use the `readTransitionFile` command to read the external transition file before performing maximum transition violation fixing. The transition file should contain the transition values in the following format:

```
setTranTime [-add] [-maxR slewValue | -maxF slewValue] { pinName }
```



The transition values specified in the transition file must be in nanoseconds (ns).

Where:

Option	Description
<code>-add</code>	Specifies that the transition value is incremental.
<code>-maxR</code>   <code>-maxF</code>	Specifies the maximum rise or fall transition.
<code>pinName</code>	Specifies the pin name for which the transition value is specified. To specify multiple pin names, use separate <code>setTranTime</code> statements as follows:  <code>setTranTime -add -maxR 3.5 inst/A</code> <code>setTranTime -add -maxR 3.5 pinB</code>
	<b>Note:</b> Minimum transition values will be ignored while performing transition violation fixing.

- Specify the following command for reading an external transition file for non-MMMC designs:  
  
`readTransitionFile -file fileName`
- Specify the following command for reading an external transition file for MMC designs:  
  
`readTransitionFile -view viewName -file fileName`  
**Note:** Ensure that you specify the transition files for all setup views.

After you read in the transition file, specify the following command to fix the maximum transition violations:

```
optDesign -postRoute -si -useTransitionFiles -drv
```

To fix the maximum transition violations for selected nets, use the `-selectedNets` parameter of the `optDesign` command.

**Note:** In this flow, the software will exit after the `ecoRoute` command.



*Tip*

Use the following command to report the maximum transition violations when specifying external transition files:

```
timeDesign -postRoute -si -useTransitionFiles
```

### External Transition and SDF Files Used for Transition Violation and Delay Fixing

To perform maximum transition violation fixing along with delay pushout fixing, you will need to specify the external transition file as well as the external SDF file.

- Use the following commands for non-MMMC designs:

```
setOptMode -setupSdfFile fileName  
setSIMode -acceptableWNS value #This is a mandatory setting.  
readTransitionFile -file fileName  
optDesign -postRoute -si -useSDF -useTransitionFiles
```

**Note:** To perform transition violation fixing using an external transition file, use the `-selectedNets` parameter of the `optDesign` command. This parameter works with the `-useTransitionFiles` parameter but not with the `-useSDF` parameter.

- Use the following commands for MMC designs:

```
read_sdf -view viewName1 fileName1  
read_sdf -view viewName2 fileName2  
readTransitionFile -view viewName1 -file fileName1  
readTransitionFile -view viewName2 -file fileName2  
setSIMode -acceptableWNS value #This is a mandatory setting.  
optDesign -postRoute -si -useSDF -useTransitionFiles
```

**Note:** To perform maximum transition violation fixing on selected nets, use the `-selectedNets` parameter of the `optDesign` command. This parameter is valid with the external transition file flow only.

## Performing XILM-Based SI Analysis and Fixing

The model-based flow in Encounter involves generating timing accurate interface logic models (ILMs) for hierarchical blocks. To perform SI analysis, the model data generation process should account for the characterized noise library of the blocks, the timing window information of non-ILM timing path nets inside the blocks, and cross-coupling information. This data, which is an extension to ILMs, is called eXtended Interface Logic Model (XILM). An XILM is built with:

- Cells and RC network (including cross-coupling data) connected from each I/O pin to and from the first latch or flip-flop.
- eXtended Timing Window Format (XTWF) file that contains timing window and slew information of non-ILM timing paths inside the block, which might be aggressors to the nets on the ILM timing path or the top-level nets.

**Note:** For more information on XILM-based SI analysis, see the [Using Interface Logic Models in Hierarchical Designs](#) chapter of the *Encounter User Guide*.

**Encounter User Guide**  
Analyzing and Repairing Crosstalk

---

---

## Power and Rail Analysis

---

Encounter Power System (EPS) sign-off power and rail analysis engines are fully integrated in SoC Encounter. The TCL and GUI use-models are identical in stand-alone EPS and its integration in SoC Encounter. The power and rail analysis in SoC Encounter is available under the *Power* menu (EPS is under *Power & Rail* menu). These menus are arranged differently between the two product (See [Encounter and EPS menu differences](#)). The Early Rail Analysis (ERA) feature is available through SoC Encounter and is not available in Encounter Power System (EPS).

- [Early Rail Analysis](#)
- [Signoff-Rail Analysis](#)
- [Encounter and EPS menu differences](#)

## Early Rail Analysis

ERA uses EPS power and rail analysis engines and can be used during power-grid prototypes to analyze power, IRdrop and power-grid integrity.

- [Early Rail Analysis key features](#)
- [Prior to running early rail analysis](#)
- [Setting up and running early rail analysis](#)
- [Viewing Early Rail Analysis results](#)

### Early Rail Analysis key features

- Rail analysis during floorplanning stage using grid based interactive current specification use-model
- Power and rail analysis during placement stage using virtual power-grid connectivity to the instance power pin
- Power and rail analysis on placed and routed database
- No requirement for extraction tech file or power-grid cell libraries. Optionally, power-grid libraries can be used for the placed macros to achieve better accuracy.
- Flexible power-constraints specification including:
  - Total power and current regions
  - Calculating power for placed instances and specify current regions for unplaced regions
  - Instance current and current region files for macros
- Package and pad location optimization based on IRdrop analysis
- Early power-switch analysis to refine power-switch placement

### Prior to running early rail analysis

Prior to running Early Rail Analysis it is recommended to check the power rail and vias for problems. This can be done with the [verifyGeometry](#), [verifyConnectivity](#), and [verifyPowerVia](#) commands described in the *Encounter Text Command Reference*. You can find additional information in the Design Sanity Checks chapter of the Encounter Power System User Guide.

## **Encounter User Guide**

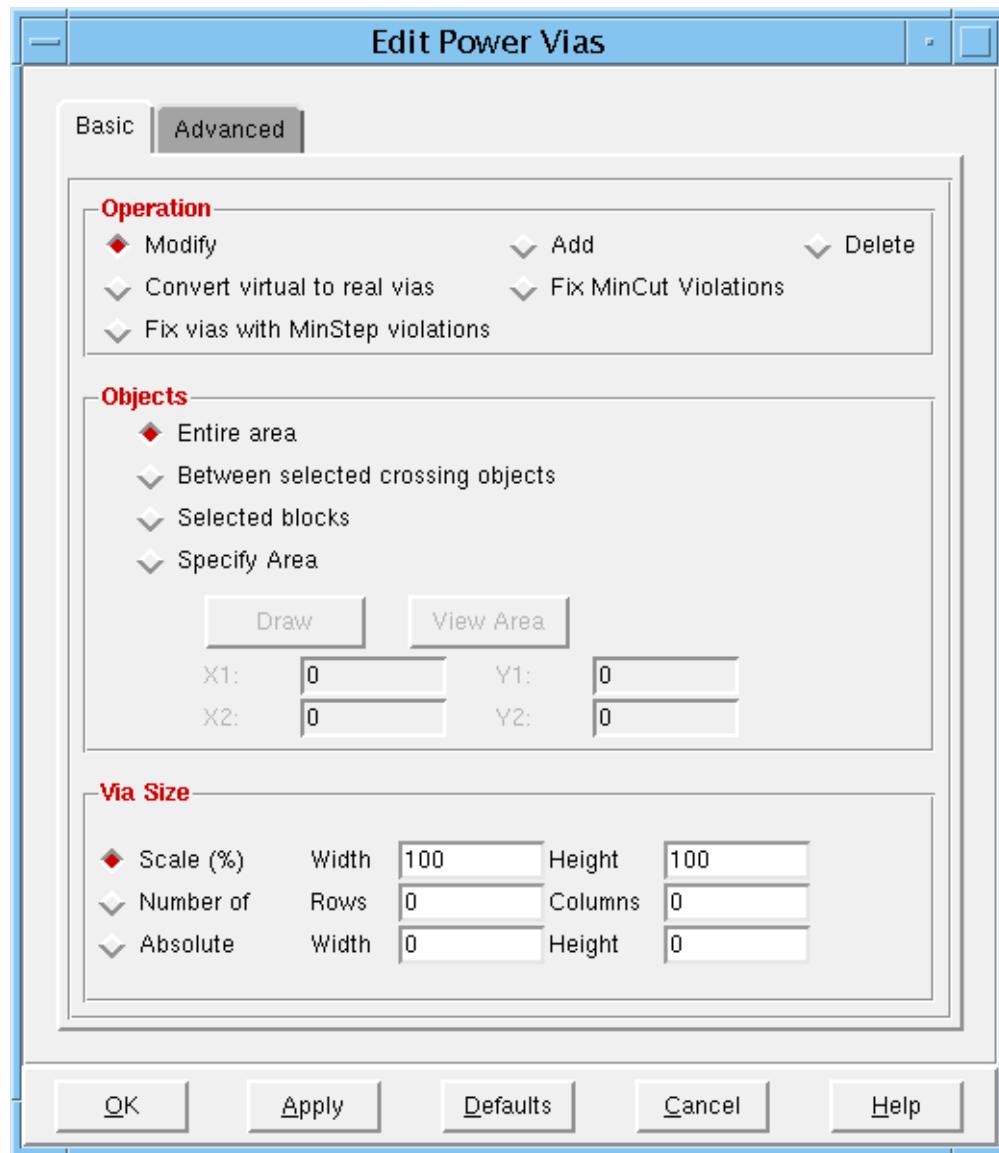
### Power and Rail Analysis

---

If there are missing power vias, you can use the *Power - Power Planning - Edit Power Via* form shown in [Figure 33-1](#) on page 1088, to correct the problem. This form can be used for adding, removing, and sizing the power vias.

The design must be loaded into Encounter. Early Rail Analysis (ERA) has the ability to analyze power grid integrity early in the floorplanning state, after placement, as well as post routing. ERA helps fix power-grid problems early in the flow, rather than waiting for when the layout is mostly done and the problems are much more difficult to correct. ERA will take whatever blocks, macros, standard cells, and routing that is available to help improve the accuracy of early rail analysis.

**Figure 33-1 Edit Power Vias**

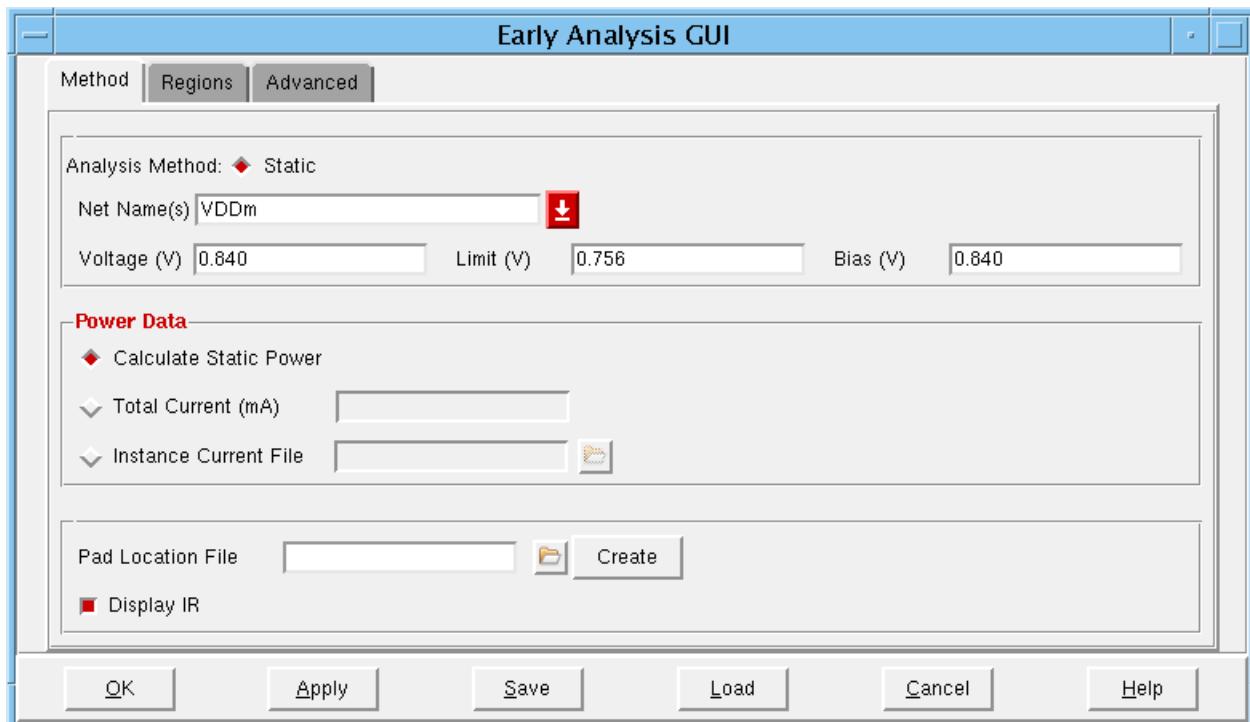


## Setting up and running early rail analysis

After the design is loaded use the following steps to setup and run Early Rail Analysis:

- Select *Power -> Rail Analysis - Early Rail Analysis* menu. The form shown in [Figure 33-2](#) on page 1089 will appear with the tab set to *Method* by default.

**Figure 33-2 Early Analysis - Method Form**

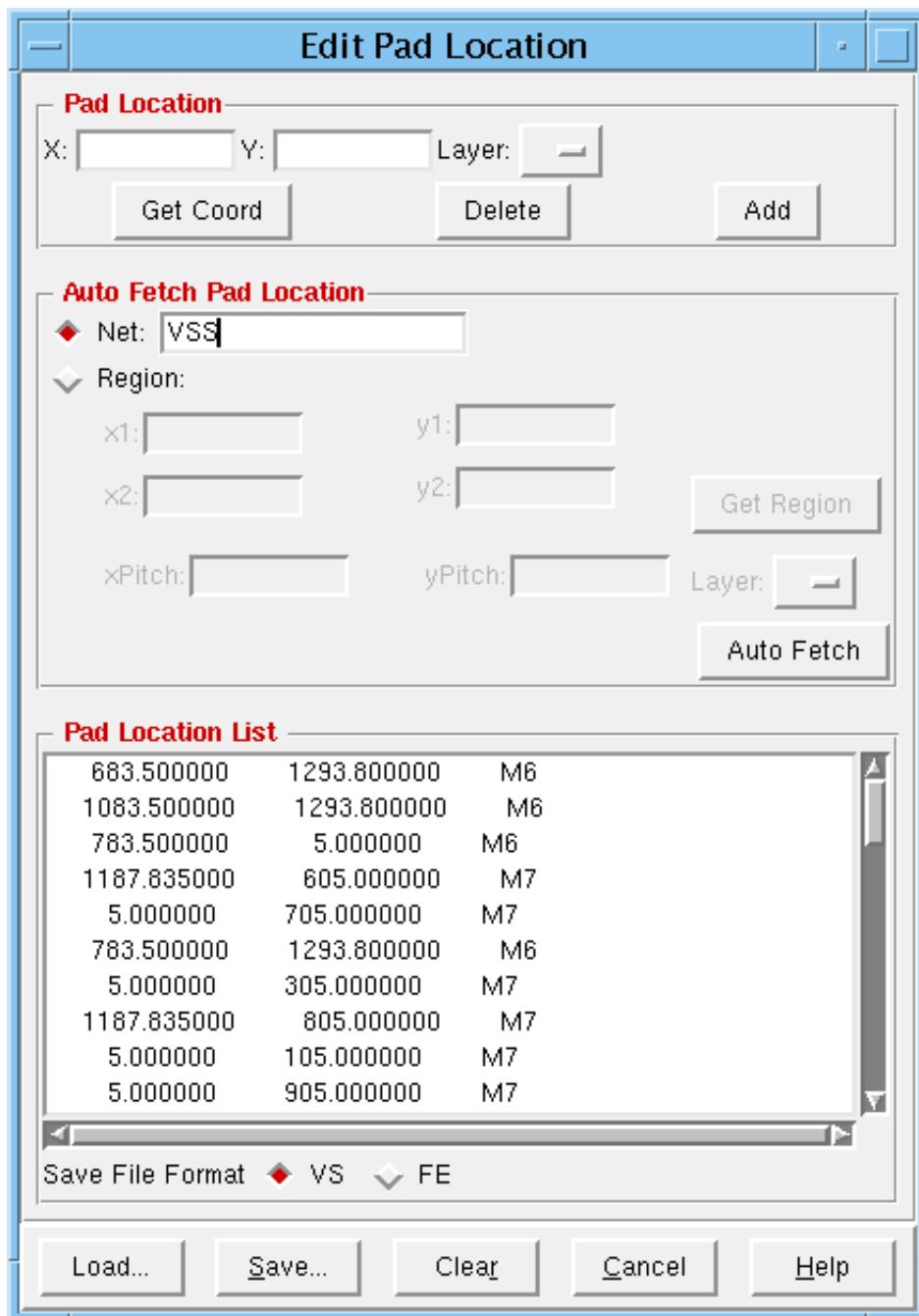


- Next the locations of the power rail voltage sources (VDD, VSS ...) need to be specified and placed in a text file. This can be done automatically by selecting the *Create* button. The form in [Figure 33-3](#) on page 1090 will appear. By entering one of the power or ground net names, such as VSS, in the *Net* field and selecting the *Auto Fetch* button, the VSS *Pad Location List* will automatically be filled in with location and layer values based on the VSS pins in the DEF. Repeat the process for all of the other power and ground sources that you are interested in analyzing.
- Next the *Power Location List* must be saved to a file. This file can be saved in either the First Encounter (*FE*) or VoltageStorm (*VS*) format. After specifying the *Save File Format* select the *Save* button.

## Encounter User Guide

### Power and Rail Analysis

**Figure 33-3 Edit Pad Location**



- Looking back at [Figure 33-2](#) on page 1089, select the *Net Name* that you want to analyze from the menu selection. The Voltage, Limit, and Bias fields will be automatically filled in.
  - Voltage

Specifies the voltage of the power or ground net that you will be analyzing

**Limit**

Specifies the IR drop constraint for analysis. All power grid elements operating below this limit (or above if it is a ground net) will be flagged as a violation and can be seen in the violation browser.

**Bias**

Specifies a bias voltage (supply range) for power and ground nets. This voltage is used to compute current based on power specification. For the power net, the net voltage is used by default. For the ground net, the rail analysis does not know how to convert the specified power into current or at what voltage power was computed. The bias voltage will be needed to be specified in this case.

**Note:** Bias is only used when you specify *Total Current*. It is not used if *Calculate Static Power* is selected.

- Next one must specify what method you will be using for defining Power (Current) information:

***Calculate Static Power***

Select this if you want the Common Power Engine (CPE) to calculate the power for each instance in the design. This requires that your design is placed.

***Total Current***

If your design is not placed, you can specify a total current in mA.

***Instance Current File***

You can also specify an instance current file from a previous power run.

- Next specify the *Pad Location File* that was created earlier.

- If you select *Display IR*, then at the end of early rail analysis, the ir (voltage) results display will automatically be loaded in the layout window.

- You can select the *Regions* tab, and will see the fields shown in [Figure 33-4](#) on page 1094.

Selecting *Current Region*, provides the ability to:

***Point to Static Current Region File***

***Draw Current Regions* to create a *Current Region List*.**

Use this when you have an area that has not been placed, but you would like to have its power consumption influence the overall grid.

Specifies the coordinates of the region, the layer, and the current. The *Add* button can be selected to add the region to the *Current Region List*. The *Delete* button will delete a selected item on the list.

*x1, y1, x2, and y2* specifies a rectangular region that the current will be distributed within.

*Rectilinear* specifies the rectilinear current region that the current will be distributed within.

**Note:** In the static mode, ERA splits the rectilinear region into several rectangular regions and distributes current to the rectangular regions based on the area.

Current in rectangular region = Area of the rectangle / Area of the rectilinear

*Layer* specifies the metal layer that the current sink will be placed on.

If you select *Draw*, then select a box in main window, the coordinate of this box will be automatically populated in *x1 y1 x2 y2*. Use the left mouse button to draw the box in Encounter.

If you select *View* after selecting an item in the list, the selected region will be displayed in the main window.

*label* specifies a name for the region. If not specified, Encounter will provide a name (region1, region2...)

You can also specify a rectilinear box to add a current region. The rectilinear box enables you to specify multiple x,y points to add current regions in the areas that are not rectangular in shape. To draw the rectilinear box in Encounter, use the left mouse button and select multiple points. Use the 'Esc' key to the last point of the rectilinear box to finish and capture the box co-ordinates.

**Note:** The tool will connect the region through a "virtual" power connection to the power pin, since there is no power routing.

- You can select the *Advanced* tab, and will see the fields shown in [Figure 33-5](#) on page 1095.

Selecting *Current Region*, provides the ability to:

- Specify a Macro Power File

This is a text file with a list of macros and the power (current) that they consume.

An example macro file is shown below:

```
RAM1 0.1  
RAM2 0.2  
INVX1 0.001
```

- Specify Power (Currents) for levels of hierarchy.
- Specify a power-grid library.

This is for characterized cells that have been placed. See [Power and Rail Analysis](#) on page 1085

- Specify a power-gate file

The power-gate file syntax is as follows:

```
CELL cellname SUPPLY supplypin SWITCHED switchedpin  
RON val IDSAT val ILEAK val
```

If this is specified it is expected that the power-switches are fully connected to the appropriate alwaysOn and Switched power nets. ERA will simply extract the power-grid and perform rail analysis. See [Power and Rail Analysis](#) on page 1085 in this manual for details.

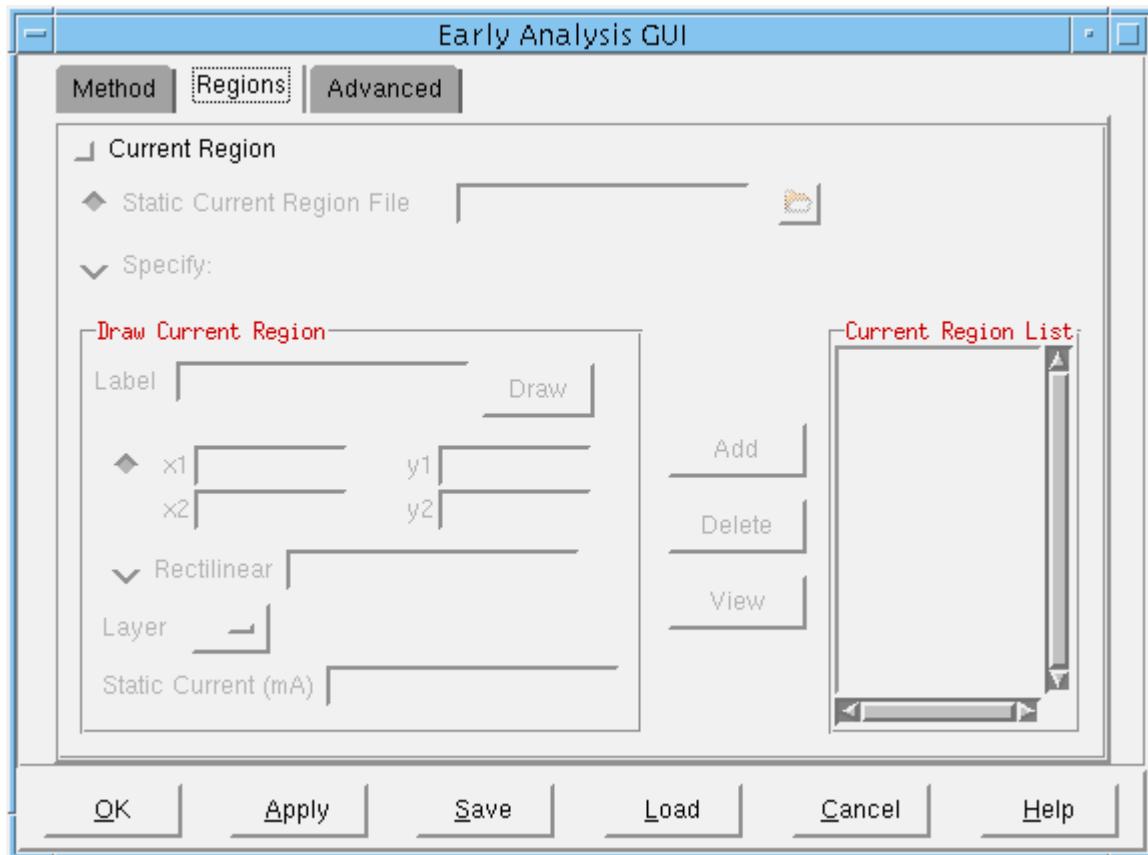
- Specify a temperature

See [early rail analysis](#) command for details on these and other options.

- Select *OK* or *Apply* and the early rail analysis will be run.

**Encounter User Guide**  
Power and Rail Analysis

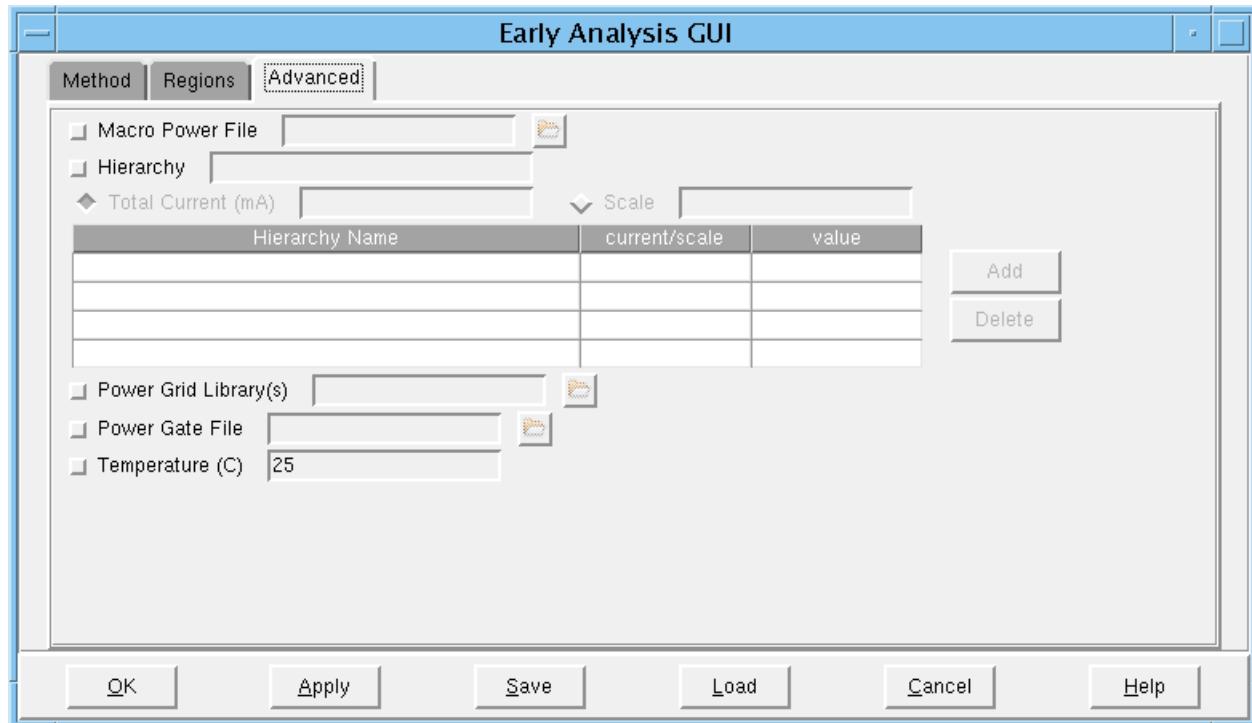
**Figure 33-4 Early Analysis - Regions Form**



## Encounter User Guide

### Power and Rail Analysis

**Figure 33-5 Early Rail Analysis - Advanced Form**



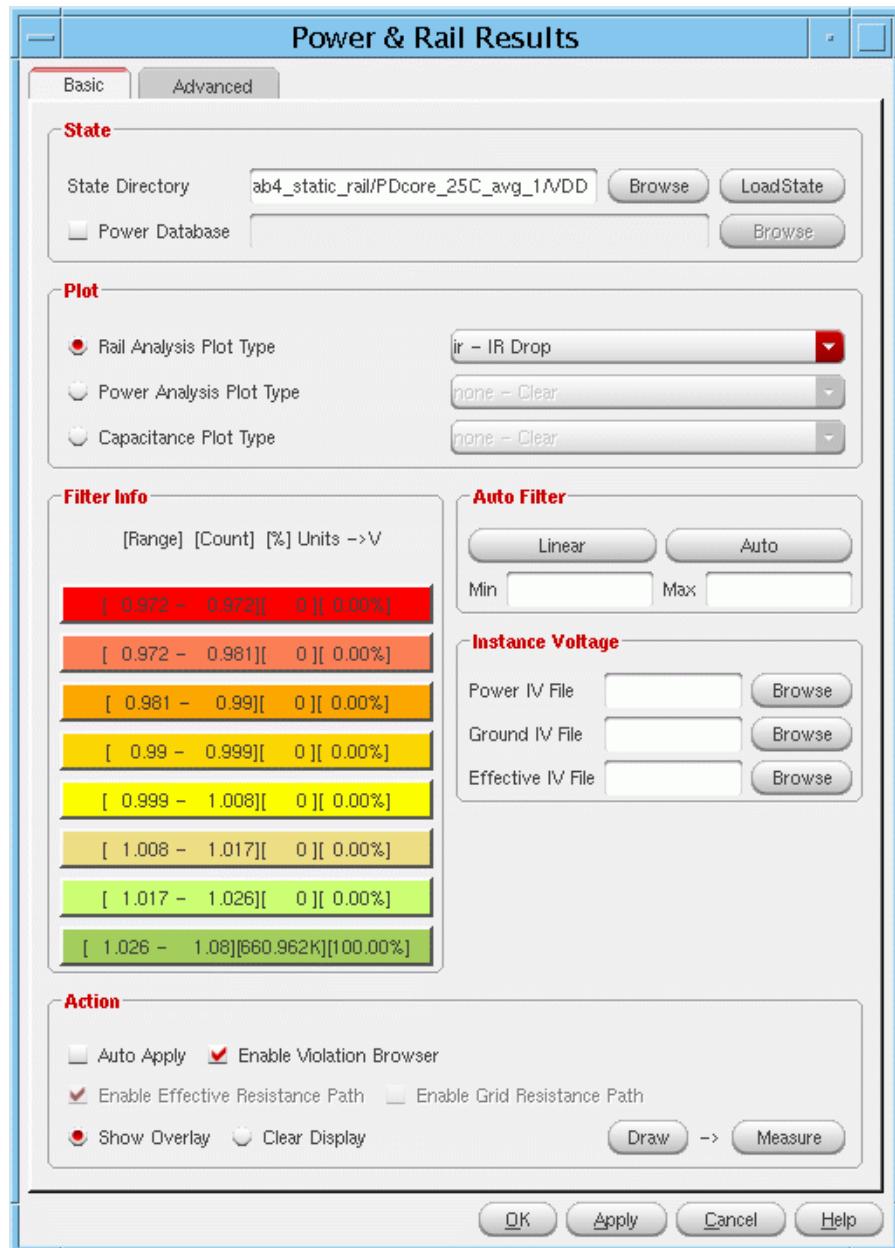
#### The TCL command syntax is as follows:

```
analyze_early_rail \
    -method static \
    -net net_name+ \
    -net_voltage voltage \
    -volt_limit voltage \
    -bias_voltage voltage \
    -[calculate_power | -total_current value| -instance_current_file file] \
        | -macro_power_file file \
    -current_region_file file \
    -power_gate_file file \
    -power_grid_library libraries \
    -scale_hierarchy_current { {hier_name[scale | current] value}+} ] \
    -pad_location_file file \
    -temperature value \
    -displayIR
```

## Viewing Early Rail Analysis results

Selecting the Power - Report - Power & Rail results will bring up the form shown in [Power & Rail Results](#) on page 1096.

**Figure 33-6 Power & Rail Results**



The *State Directory* will automatically be filled in with the location of the most recent analysis that was run. You can also specify values for previous runs. All of the ERA runs will be located in a `FE2VSEarlyRA` directory.

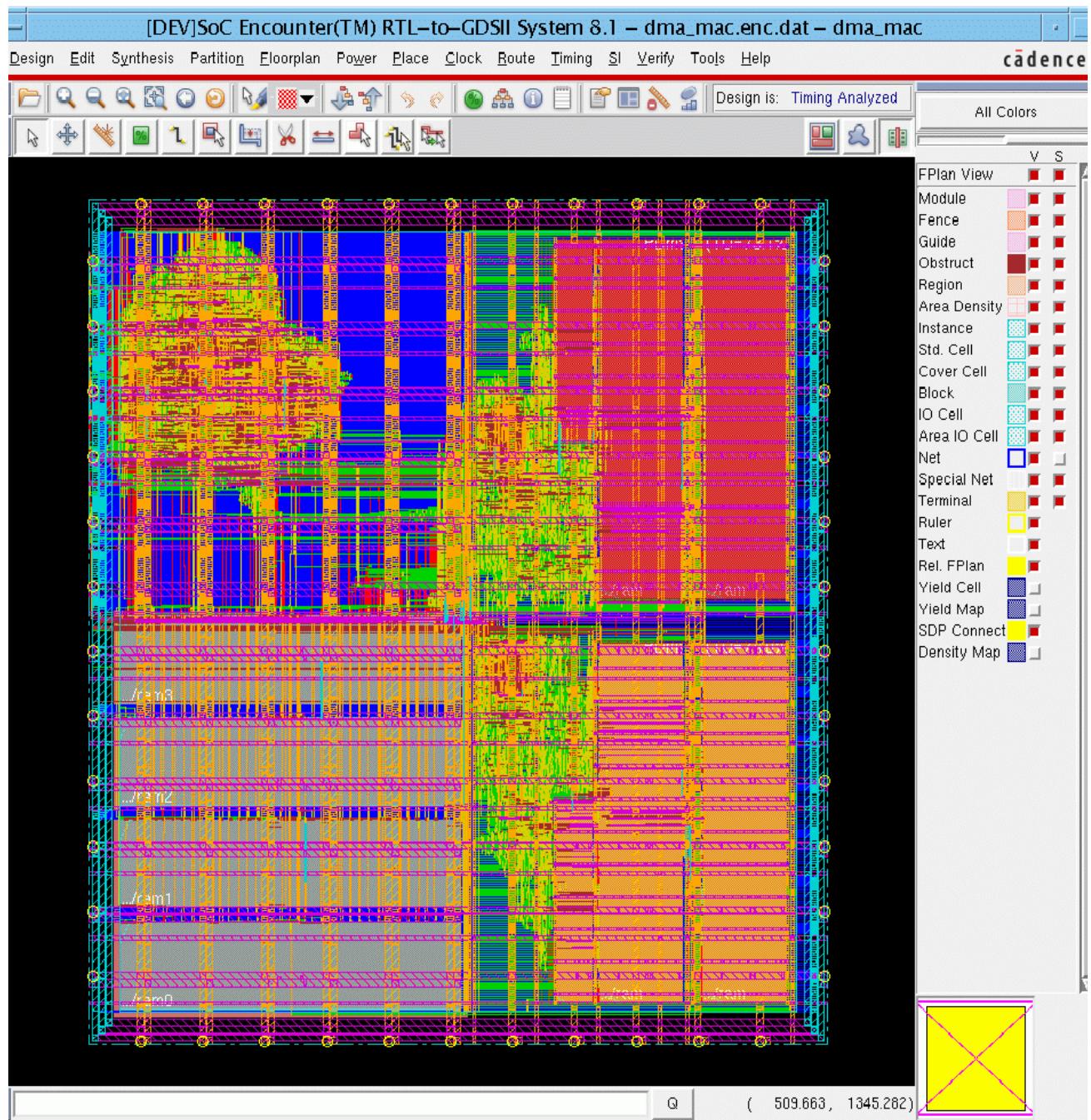
You can specify the type of plot (Rail Analysis, Power Analysis, or Capacitance) and then select the specific plot type. An instance power (ip) and load capacitance (load) plot are shown in [Instance Power Plot on page 1098](#) and [Load Capacitance Plot on page 1099](#) respectively.

For Early Rail Analysis the viewing of Power & Rail Results is the same as that used for Sign-off Analysis. For additional information on viewing the plots, see [Power and Rail Analysis on page 1085](#) and [Power and Rail Analysis on page 1085](#).

## Encounter User Guide

### Power and Rail Analysis

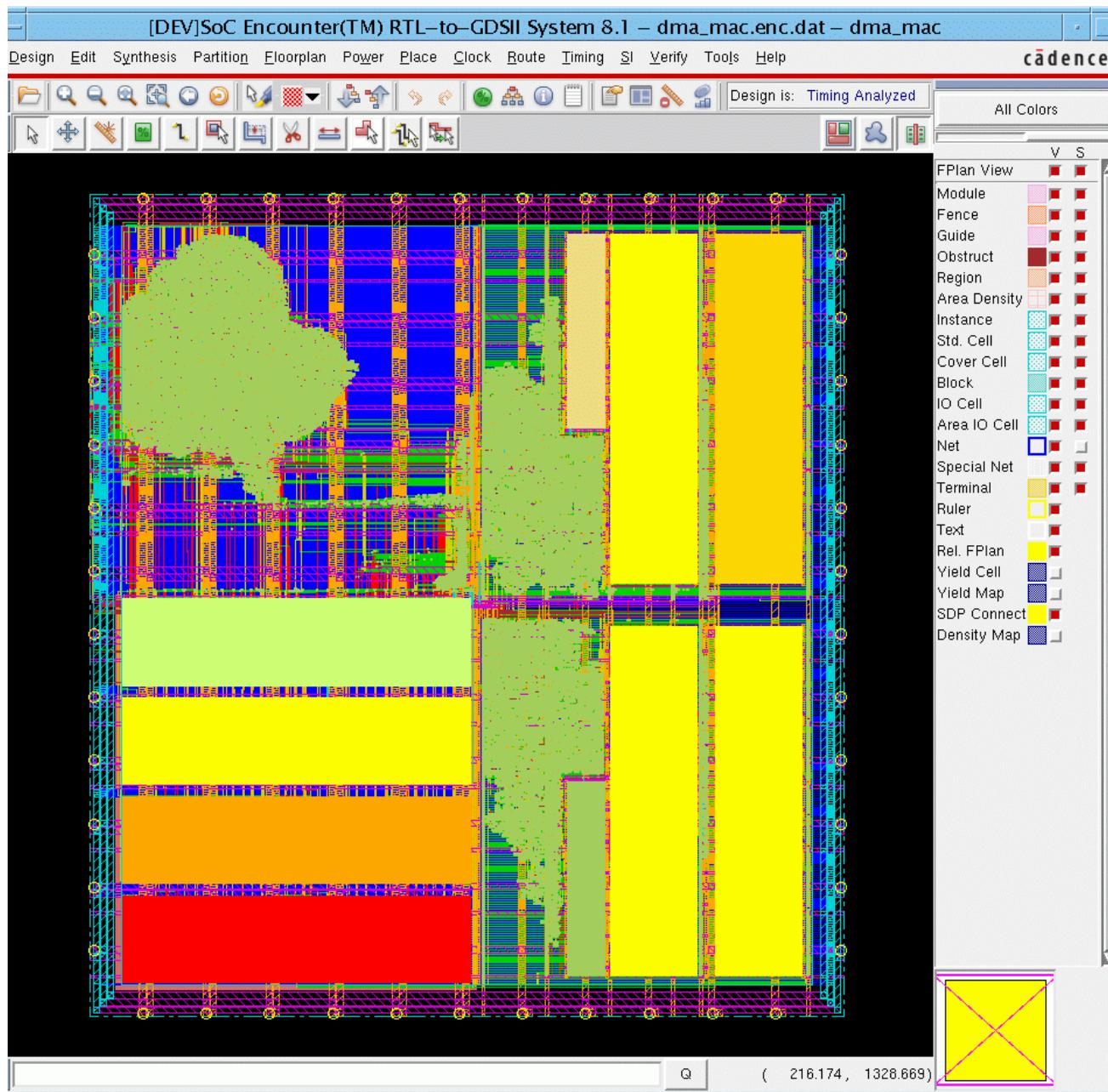
**Figure 33-7 Instance Power Plot**



## Encounter User Guide

### Power and Rail Analysis

**Figure 33-8 Load Capacitance Plot**



## Signoff-Rail Analysis

For details on running Signoff Power and Rail Analysis within Encounter, see [Encounter Power System User Guide](#) chapters 5-12.

The Encounter Power and Rail Analysis commands and forms are identical to those in Encounter Power System, but the menus are organized differently. The forms are accessed in different places in the pull-down menus for each of the two products. This is shown on the next page.:

## Encounter and EPS menu differences

<b>Form</b>	<b>Encounter Menu</b>	<b>EPS Menu</b>
<i>Set Power Analysis Mode</i>	<i>Power - Power Analysis</i>	<i>Power &amp; Rail Analysis</i>
<i>Run Power Analysis</i>	<i>Power - Power Analysis</i>	<i>Power &amp; Rail Analysis</i>
<i>Early Rail Analysis</i>	<i>Power - Rail Analysis</i>	not part of EPS
<i>Set Power Library Mode</i>	<i>Power - Rail Analysis - PowerGrid Library</i>	<i>Power &amp; Rail Analysis - PowerGrid Library</i>
<i>Create PowerGrid Library</i>	<i>Power - Rail Analysis - PowerGrid Library</i>	<i>Power &amp; Rail Analysis - PowerGrid Library</i>
<i>Set Rail Analysis Mode</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail Analysis</i>
<i>Run Rail Analysis</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail Analysis</i>
<i>Create Hierarchical View</i>	<i>Power - Rail Analysis</i>	<i>Power &amp; Rail Analysis</i>
<i>PowerGrid Library Report</i>	<i>Power - Report</i>	<i>Power &amp; Rail Analysis-Report</i>
<i>Power Report</i>	<i>Power - Report</i>	<i>Power &amp; Rail Analysis-Report</i>
<i>Power Histograms</i>	<i>Power - Report</i>	<i>Power &amp; Rail Analysis-Report</i>
<i>Power &amp; Rail Results</i>	<i>Power - Report</i>	<i>Power &amp; Rail Analysis-Report</i>
<i>Dynamic Movies</i>	<i>Power - Report</i>	<i>Power &amp; Rail Analysis-Report</i>
<i>Dynamic Waveforms</i>	<i>Power - Report</i>	<i>Power &amp; Rail Analysis-Report</i>

**Encounter User Guide**  
Power and Rail Analysis

---

---

## Verifying Violations

---

- [Overview](#) on page 1104
- [Verifying Connectivity](#) on page 1107
- [Verifying Metal Density](#) on page 1109
- [Verifying Geometry](#) on page 1110
- [Verifying Process Antennas](#) on page 1114
- [Verifying Maximum Floating Area Violations](#) on page 1117
- [Verifying AC Limit](#) on page 1118
- [Viewing Violations With the Violation Browser](#) on page 1119
- [Clearing Violations](#) on page 1122

## Overview

The Encounter software's verification commands check and report on the following types of violations:

- Connectivity

Checks for opens, unconnected wires (geometrical antennas), loops, and partial routing.

Verify the connectivity of the design after the following design step:

- Detailed routing

For more information, see

- [Verifying Connectivity](#) on page 1107
  - `verifyConnectivity` in the “Verify Commands” chapter of the *Encounter Text Command Reference*

- Metal density

Checks that the metal density of the metal layers is within the minimum and maximum metal density values specified by the LEF file or the `setMetalFill` command. Also checks the density of the cut layers.

Verify the metal density after the following design step:

- Inserting metal fill

For more information, see `verifyMetalDensity` and `verifyCutDensity` in the “Verify Commands” chapter of the *Encounter Text Command Reference*.

- Geometry

Checks the physical layout of the design, including width, length, spacing, area, overlap, enclosure, wire extension, and via stacking violations. If you modify or edit any part of the design, run `verifyGeometry` to make sure the design is still DRC clean.

Verify the geometry of the design after the following design steps:

- Placement
  - Power routing
  - Detailed routing
  - Wire editing

For more information, see

## Encounter User Guide

### Verifying Violations

---

- [Verifying Geometry on page 1110](#)
  - [verifyGeometry](#) in the “Verify Commands” chapter of the *Encounter Text Command Reference*.
- Process antennas and unconnected metal segments (floating areas)

Checks the charge that builds up on pins caused by routing that does not have a discharge path to a gate. The `verifyProcessAntenna` command checks for pin routing that violates the maximum antenna charge for the pins, and reports violations on pins that have an antenna ratio larger than the maximum allowed antenna ratio specified for the routing layer.

The `verifyProcessAntenna` command also checks for unconnected metal segments that violate the maximum area specified in the LEF file. An unconnected (floating) metal segment is a segment that is not connected to diffusion (or a polysilicon gate) through the same layer or a lower layer.

Verify process antenna and maximum floating area violations after the following design step:

  - Detailed routing

For more information, see [verifyProcessAntenna](#) in the “Verify Commands” chapter of the *Encounter Text Command Reference*.
- AC limit

Checks for AC current violations on signal nets.

Verify the AC limit after the following design step:

  - Detailed routing

For more information, see [verifyACLimit](#) in the “Verify Commands” chapter of the *Encounter Text Command Reference*.
- Lithography hotspots

The software can interpret hotspot interchange format (HIF) files. For more information, see [loadViolationReport](#) in the “Verify Commands” chapter of the *Encounter Text Command Reference*.
- Placement

For more information on the types of violations, see [checkPlace](#) in the “Placement Commands” chapter of the *Encounter Text Command Reference*.

You can use text commands or GUI forms to check the violations and create the reports.

## **Encounter User Guide**

### Verifying Violations

---

You create violation markers with the Encounter commands, or import markers from another verification tool, such as Assura<sup>TM</sup> or Calibre, and view the markers with the Violation Browser. The Encounter software saves the markers with the database.

For more information, see [“Viewing Violations With the Violation Browser”](#) on page 1119.

## Verifying Connectivity

Verify the connectivity of your design to detect and report conditions such as opens, unconnected pins, dangling wires, loops, and partial routing. You can use the command to create violation markers in the design window and generate a violation report. There is no database impact from using this command unless you save the design, which saves the violation markers.

For regular wires, the Encounter software checks connectivity by using the center line of the wire segments and center of the vias. For special wires, the command checks the whole geometry. If a via or wire is slightly offset from where it should be, the software reports an error.

The software also detects connectivity loops based on the end points of a regular wire segment center line or the center of a via. It reports geometry loop violations.

### Before You Begin

Before you verify connectivity, the following conditions must be met:

- The design must be routed.
- The design must be loaded into the current Encounter session.

### Types of Connectivity Violations Reported

- Antennas

Unconnected wires (dangling wires). For more information, see “[Types of Antenna Violations Reported](#)” on page 1112.

- Opens

Parts of nets, such as wires or pins, that are connected to each other but are missing a connection to the net as a whole. Marks each part of a net that is missing a connection as an open and displays a violation marker between the parts.

Violation markers for opens are displayed as polygons that include all wires, pins, and vias that connect to an island.

- Loops

- Unconnected pins

Pins that are not connected to any other objects

**Note:** In releases prior to 7.1, `verifyConnectivity` marked nets with connected pins, but without any wiring, as unrouted nets. `verifyConnectivity` no longer marks these nets as unrouted, so they do not cause violations.

## Results

After verifying connectivity, you can use information in the violation report to repair connectivity violations. You can use the Violation Browser for interactive viewing and highlighting of violation markers.

You can see incremental results in the Violation Browser. For more information, see [Verify Connectivity](#) in the “Verify Menu” chapter of *Encounter Menu Reference*.

## Verifying Metal Density

Verify the metal density of the design for each routing layer, to ensure within the minimum and maximum density values specified in the LEF file or by the `setMetalFill` command.

### Before You Begin

Before you verify metal density, the following conditions must be met:

- Metal density values must be specified in the LEF file or by the `setMetalFill` command.
- The design must be detail routed.
- The design must be loaded into the current Encounter session.

### Metal Density Statements in the LEF File

The following statements in the Layer (Routing) section of the LEF file define the minimum and maximum metal density and how to analyze the density.

- `MINIMUMDENSITY`
- `MAXIMUMDENSITY`
- `DENSITYCHECKWINDOW`
- `DENSITYCHECKSTEP`
- `FILLACTIVESPACING`

For more information, see the “[LEF Syntax](#)” chapter in the *LEF/DEF Language Reference*.

### Results

The verification process generates a report file containing information about the metal density that is not within the minimum and maximum density range.

## Verifying Geometry

Verify the physical layout of the design by checking the width, spacing, internal geometry, and other characteristics of objects. Use the `verifyGeometry` command to specify the checks to perform, disable checking, and set limits for errors and warnings to report.

The disable feature is useful when false violations arise because of discrepancies in the way mask-level data is presented. For example, cell internal obstructions and pins might be represented in a way that causes the verifier to report design rule violations that do not exist in the mask-level layout.

Verify geometry at the following stages in the design flow:

- After placing the design.
- After adding power stripes and rings and running power routing.
- After running detailed routing.

### Before You Begin

- Ensure the following LEF statements are specified:
  - `CLEARANCEMEASURE`
  - `USEMINSPACING` statements for obstructions and pins

For more information, see the “[LEF Syntax](#)” chapter in the *LEF/DEF Language Reference*.
- If you plan to run `verifyGeometry` in multiple-CPU processing mode, use the Encounter multiple-CPU commands or select the appropriate options on the Multiple CPU Processing form. For more information, see “[Verifying Geometry in Multi-Thread Mode](#).”
- Route the design.

### Verifying Geometry in Multi-Thread Mode

You can accelerate geometry checking by running the software in multi-thread mode. Use one of the following methods:

- On the text command line:

Run the following command before running `verifyGeometry`:

[setMultiCpuUsage](#)

For example,

```
setMultiCpuUsage -numThreads 4  
verifyGeometry
```

■ In the GUI:

- a. On the Verify Geometry – Advanced page, click the *Set Multiple CPU* button to open the Multiple CPU Processing form.
- b. On the Multiple CPU Processing form, specify the number of threads.
- c. Optionally, select *Release License at The End of Each Command*.
- d. Click *OK* to close the Multiple CPU Processing form and return to the Verify Geometry form.

When you return to the Verify Geometry form, the *Number of Threads* option is updated with the value you specified on the Multiple CPU Processing form.

- e. Run Verify Geometry.

## Related Topics

- [Accelerating the Design Process by Using Multiple-CPU Processing](#)
- [Multiple-CPU Processing Commands](#) chapter in the *Encounter Text Command Reference*
- [Verify Geometry – Advanced](#) in the *Encounter Menu Reference*

## Spacing Violation Checks

- `verifyGeometry` uses the minimum dimension of an object to check for spacing violations. The minimum dimension is the width of the object.
- The command does not detect objects with width greater than `WIDTH` and length greater than `LENGTH` that exist within a distance (`WITHIN`) greater than 10 µm for the `MINIMUMCUT` check in the LEF file.
- The command categorizes spacing violations as `SameNet`, `NonDefault`, and `ParallelRun` violations. If it finds a violation caused by a blockage between two instances of different cells, it treats the violation as a `SameNet` violation because it does not belong to a net.

## Encounter User Guide

### Verifying Violations

---

- The command considers OBS CUT layer shapes as within the same metal if they are within the same OBS ROUTING layer shape (the layer above or below). This avoids -sameCellViols flags on SPACING violations inside the cells.
- To check implant layers for violations, specify an implant rule in the LEF file. To skip implant layer checking, specify the verifyGeometry -noImplantCheck parameter.
- To check spacing between cut layers and metal layers, specify a cut-metal spacing rule in the LEF file. For example, the following rule triggers a check of the spacing between CUTG1 and MET5 layers:

```
LAYER CUTG1 TYPE CUT ;
  SPACING 0.42 ;
  SPACING 0.28 LAYER MET5 ;
END CUT G1
```

For more information, see the [“LEF Syntax”](#) chapter of the *LEF/DEF Language Reference*.

## Types of Antenna Violations Reported

verifyGeometry flags an antenna violation when it finds an unconnected wire.



In the context of verifyGeometry and verifyConnectivity, antenna violations are different from process antenna violations. The verifyProcessAntenna command checks for process antenna violations, which are caused by pins whose process antenna ratio is larger than the maximum allowed ratio specified in the LEF file for the routing layer. For more information, see [verifyProcessAntenna](#) in the “Verify Commands” chapter of the *Encounter Text Command Reference*.

To avoid antenna violations, wires and nets must meet the following conditions:

- Regular wires
  - Must terminate on a pin or the center of a via.
  - If a vertical wire intersects a horizontal wire along its axis, the end of the vertical wire must be covered by the horizontal wire.
  - If the ends of a vertical wire and horizontal wire meet, they must meet at their end points.
- Regular net vias
  - Must be covered by a pin.

- ❑ The center of the via must be the start or end point of a wire.
- ❑ The center point of stacked vias must be coincident.
- Special wires
  - ❑ A point that is one-quarter of a wire width from the center of the ending edge of the special wire must be covered by a via, pin, or another wire on the same layer.
- Special net vias
  - ❑ The metal rectangle of the special net via must overlap with a special wire or via of the same net.

## Support for Via Rules

`verifyGeometry` uses the rules defined in the `VIARULE` section of the LEF file to check for violations caused by vias.

- The command checks the master via only, and flags violations on only one instance of the via.
- The command considers the content of the via during verification when checking for spacing violations.

## Results

`verifyGeometry` creates markers corresponding to geometry violations in the database. Use the [Violation Browser](#) to see the markers.

## Verifying Process Antennas

Verify process antenna violations by checking for routing that violates the maximum charge caused by the process antenna effect (PAE) on pins. The software finds violations when a pin's process antenna ratio is larger than the maximum ratio specified in the LEF file for the routing layer.

The report file lists all the violated nets and includes process antenna information. Optionally, it can also report all other nets.

### Before You Begin

Before performing process antenna verification, complete the following tasks:

- Perform signal routing.
- Ensure the antenna keywords are specified in the LEF file; for example,
  - `ANTENNAAREARATIO` for LEF layers
  - `ANTENNAGATEAREA` and `ANTENNADIFFAREA` for macro pins

For more information, see the “[LEF Syntax](#)” chapter in the *LEF/DEF Language Reference*.

### Verifying PAE

Checks for pin routing that violates the maximum antenna charge for the pins and reports violations on pins that have an antenna ratio larger than the maximum allowed antenna ratio specified for the routing layer. Handles PAE violations on any metal layer on flat or hierarchical designs. Uses a geometry-based approach and does not double count metal areas for vias or wires. Provides a detailed process antenna report including the metal area, diffusion area, and target ratio for each pin. The report file lists all violated nets with process antenna information. Optionally, it can also report all other nets. For more information on PAE, see the “[Calculating and Fixing Process Antenna Violations](#)” appendix in the *LEF/DEF Language Reference*.

### Results

After verifying process antenna violations, you can use information in the violation report to repair process antenna violations. You can use the *Tools – Violation Browser* command for interactive viewing and highlighting of violation markers.

## Encounter User Guide

### Verifying Violations

---

### Sample Process Antenna Report

The following example shows section of a detailed process antenna report file:

```
D1 (2)

U0 (BUF) A

[1]      MET:   Area: 1.10    S.Area:  2.10    G.Area: 100.00 D.Area:  0.00
          Fact: 0.90     PAR:    0.01    Ratio:   0.10  (Area)
          Fact: 1.00     PAR:    0.02    Ratio:   0.10  (S.Area)
                      CAR:    0.01    Ratio:   0.00  (C.Area)
                      CAR:    0.02    Ratio:   0.00  (C.S.Area)

[1]      THO:   Area: 0.20    S.Area:  0.00    G.Area: 100.00 D.Area:  0.00
          Fact: 1.00     PAR:    0.00    Ratio:   0.00  (Area)
                      CAR:    0.00    Ratio:   0.00  (C.Area)

[1]      WMET:  Area: 13.27   S.Area: 51.20    G.Area: 100.00 D.Area: 300.00
          Fact: 1.10     PAR:    0.15    Ratio:   2.50  (Area)
          Fact: 0.90     PAR:    0.46    Ratio:   0.00  (S.Area)
                      CAR:    0.16    Ratio:   0.00  (C.Area)
                      CAR:    0.48    Ratio:   0.00  (C.S.Area)
```

The report uses the following terms:

Area:	Metal area
S.Area:	Metal side area
G.Area:	Gate area
D.Area:	Diffusion area
Fact:	Metal (side) area adjusted factor
PAR:	Partial antenna ratio
CAR:	Cumulated antenna ratio
Ratio:	Target antenna ratio
(Area)	Metal area
(S.Area)	Metal side area

## **Encounter User Guide**

### Verifying Violations

---

(C.Area)      Cumulated metal area

(C.S.Area)    Cumulated metal side area

## Verifying Maximum Floating Area Violations

Verify maximum floating area violations (unconnected metal segments whose area is greater than the maximum area specified in the LEF file) by using the `verifyProcessAntenna` command. The Encounter software checks for maximum floating area violations by default when you run this command. For more information, see [verifyProcessAntenna](#) in the *Encounter Text Command Reference*.

The LEF 5.6 property `MAXFLOATINGAREA` specifies the maximum area. The following global properties are also associated with this property:

- `GATEISGROUND`  
Does not check metal layer connected to a polysilicon gate.
- `CONNECTED`  
Checks the sum of areas on the same metal that are connected through a lower metal layer.

For more information, see “[Defining Routing Layer Properties to Create 45 nm and 65 nm Rules](#)” in the “LEF Syntax” chapter of the *LEF/DEF Language Reference*.

**Note:** To skip maximum floating area violation verification, but run process antenna verification, type the following command:

```
verifyProcessAntenna -noMaxFloatingArea
```

## Verifying AC Limit

Verify AC current violations on signal nets by using the `verifyACLimit` command. This command calculates the root mean square current ( $I_{rms}$ ) at the driver output and compares it to the `ACCURRENTDENSITY` tables in the LEF file that contain the  $I_{rms}$  limits for routing layers. It generates an error and attaches a violation marker to a net if the calculated  $I_{rms}$  for a net exceeds the `ACCURRENTDENSITY`  $I_{rms}$  limit for a routing layer or width used by the net.

Computes the  $I_{rms}$  from the slew rates of the signal, the capacitance of the net, and the toggle-rate frequency as computed by timing analysis commands like `buildTimingGraph` and `check_timing` (and the values can be written out with the `-report` parameter). If there is more than one timing view in use, uses the default setup view (controlled by `set_default_view -setup viewName` command).

The software checks the `ACCURRENTDENSITY` tables for the following conditions and takes the following actions:

- If there is no table for a routing layer, the software gives a warning and assumes an infinite limit for the layer.
- If `PEAK` and `AVERAGE` tables are present, the software ignores them.

**Note:** This command reports the AC current density in  $\text{mA}/\text{micron}$ . The LEF file specifies it in  $\text{mA}/\text{micron}$ . To convert the LEF specification, the `verifyACLimit` command reads the value in  $\text{mA}/\text{micron}$  and multiplies it by the wire width.

For more information, see the "[LEF Syntax](#)" chapter of the *LEF/DEF Language Reference*.

## Before You Begin

Before verifying AC limit, complete the following tasks:

- Perform RC extraction.
- Perform timing analysis.

## Results

After verifying AC limit violations, you can use information in the violation report to repair AC current limit violations. Use the `fixACLimitViolation` command to repair the violations. You can use the *Tools – Violation Browser* command for interactive viewing and highlighting of violation markers generated after you use this command.

## Viewing Violations With the Violation Browser

Use the Violation Browser form or the `violationBrowser` text command to view and highlight violation and lithography hotspot markers interactively.

### Viewing Geometry or Metal Density Violations

The Violation Browser updates violation markers generated by the `verifyGeometry` and `verifyMetalDensity` commands incrementally in an Encounter session—that is, it displays the markers generated the first time you run either of these commands and adds new markers, or deletes markers, from subsequent runs during the same session. If the software finds violations during a subsequent run that were already found previously, the browser display does not change, as there is no incremental update.

The browser can make the incremental changes because `verifyGeometry` and `verifyMetalDensity` can check a small area of the design and update the database. As a result of this behavior, the Encounter software saves the information from the first verification run.

### Viewing Connectivity, Process Antenna, or AC Limit Violations

The Violation Browser overwrites violation markers from the `verifyConnectivity`, `verifyProcessAntenna`, and `verifyACLimit` commands if they are run more than once during an Encounter session. These commands are net-based, not area-based, so the browser does not make incremental updates for connectivity, process antennas, or AC limit. As a result of this behavior, the Encounter software does not keep the information from the first verification run.

### Viewing Violation Markers From Assura or Calibre

To view violation markers from Assura or Calibre with the Violation Browser, use the following commands or forms:

- `createMarker`

This command creates markers from data imported from Assura or Calibre. For more information, see [createMarker](#) in the *Encounter Text Command Reference*.

- `loadViolationReport` (*Tools – Load Violation Report*)

## Encounter User Guide

### Verifying Violations

---

This command loads a report file from Assura or Calibre and converts it to a format that the Encounter software can interpret. For more information, see [loadViolationReport](#) in the *Encounter Text Command Reference*.

- **violationBrowser (Tools – Violation Browser)**

This command displays the markers in the Violation Browser. For more information, see [violationBrowser](#) in the *Encounter Text Command Reference*.

## Violation Browser Features

- Click a violation on the violation list on the form to see a description of the violation. The description includes actual and target values for AC limit violations, process antenna violations, and geometry spacing violations.
  - An actual value is the current value
  - A target value is the value defined in the LEF file.
- Click the + or – sign to collapse or expand the listings of each violation type.
- Use the *First*, *Previous*, *Next*, *Last*, *Up*, and *Down* buttons to navigate through the list of violations.
- The browser displays the violations in the following hierarchical order:

```
+ tool
  + type
    + subtype
      Description
```

where the *tool*, *type*, and *subtype* value correspond to the value you specify using the [createMarker](#) command.

- Use cross probing between the design display area and the Violation Browser.  
To display the details of a violation in the Violation Browser form, double-click the violation marker in the design display area.
- If there are violation markers for overlapped objects, select the top-most marker in the design display area and press the space bar on your keyboard to navigate through the other markers. The type and name of the selected violation is displayed in the lower-left corner of the Encounter window. Use the *q* keyboard shortcut key to select a violation and highlight it in the Violation Browser form.
- Use the Zoom buttons to change the magnification level of a violation.

## Encounter User Guide

### Verifying Violations

---

- Use any of the following buttons to change the display for a selected violation:

- Highlight Color*
  - Highlight Violations*
  - De-Highlight Violations*
  - Delete Violations*
  - Mark Violations as False*
  - Mark Violations as True*

- Generate a report file by clicking the Save button.

The report file includes information on the violations shown in the violation browser.

- Limit the number of violations to display by using the *Show Types* panel in the *Settings* page of the form.
- Limit the area to display by using the *Show Area* panel in the *Settings* page of the form.
- Filter the violations to display by using the *Other Filters* panel in the *Settings* page of the form.

## Clearing Violations

Use the *Tools – Clear Violation* menu command to clear the violation markers in your design.

There is no GUI form for this command.

---

## Analyzing Yield

---

- [Overview](#) on page 1124
- [What Effects Does reportYield Consider?](#) on page 1124
- [Calculating Failure Probabilities](#) on page 1125
  - [Critical Area Analysis](#) on page 1126
  - [Defect Data and Cumulative Defect Data Functions](#) on page 1127
- [Before You Begin](#) on page 1127
- [Results](#) on page 1127
- [Interpreting the Yield Map](#) on page 1129
  - [Displaying the Yield Map](#) on page 1129
- [Interpreting the Yield Report](#) on page 1132
  - [Yield Report](#) on page 1132
  - [Detailed Report](#) on page 1136
- [Understanding the Yield Technology File](#) on page 1138
  - [File Format](#) on page 1138
  - [File Sections and Keyword Statement Descriptions](#) on page 1140
  - [Yield Technology File Example](#) on page 1150
- [Formulas and Calculations](#) on page 1153
  - [Calculating the Probability of Failure for a Metal Layer](#) on page 1153
  - [Calculating Defect and Cumulative Defect Data](#) on page 1153
  - [Cost Formulas](#) on page 1156

## Overview

This chapter describes how the Encounter software analyzes the costs and benefits of different design-for-yield techniques. The Encounter `reportYield` command uses a yield technology file with data models for cell, via, and wire defects to calculate certain effects that impact yield. The command then generates a yield map and yield report based on its calculations. You use the data in the report and map as a basis to optimize your design to improve the yield.

**Note:** The software supports yield analysis of 45-degree shapes, using a stair-step approximation during critical area analysis. This feature is suitable for wide-wire routes, as would occur for redistribution layer routing connected to bumps (RDL routing), but not for X-routing, where there are many minimum-width routes.

## What Effects Does `reportYield` Consider?

The `reportYield` command calculates the probability of yield loss due to the following effects:

- Cell failures
- Via failures
- Wire opens
- Wire shorts

These effects are caused by random particles that land on the die during fabrication, causing defects. Yield loss caused by the defects is called *defect-limited yield loss*. The `reportYield` command reports statistics for the *defect-limited yield (DLY)*.

Defect-limited yield loss accounts for only a portion of actual yield loss. For example, a chip with no defect-limited yield loss might have parametric yield loss due to RC variation or systematic yield loss due to lithography problems, and an actual yield much less than `reportYield` calculates.

## Encounter User Guide

### Analyzing Yield

As you optimize your design for defect-limited yield, you increase the actual yield, as shown by the following table:

DLY	Yield based on other effects	Actual yield	Actual yield if DLY increases 1 percentage point	Increase in good die per wafer
90%	90%	81%	81.9%	1.11%
90%	50%	45%	45.5%	1.11%
50%	90%	45%	45.9%	2.0%
50%	50%	25%	25.5%	2.0%

An increase of 1 percent in DLY gives an increase of 1/DLY in good die per wafer.

The report generated by `reportYield` gives you the defect-limited yield for the design and includes an estimation of the probability of cell failures, via failures, wire shorts, and wire opens. You use this information to understand the impact of cell yield optimization and routing yield optimization choices.

## Calculating Failure Probabilities

The `reportYield` command uses cell and via failure rates from data supplied by the fab or library vendor. It also uses fab or library vendor-supplied data to determine the probabilities of wire shorts and opens. The data is included in the yield technology file.

- Via failure rates

The `reportYield` command calculates failure rates for single-cut vias, double-cut vias, and all other vias. For more information, see ["Via Probability"](#) on page 1145.

- Cell failure rates

The `reportYield` command calculates failure rates for a particular instance of a cell and for any instances of a cell. For more information, see ["Cell Probability"](#) on page 1148.

- Wire failures caused by shorts or opens

The `reportYield` command calculates failure rates for wires by performing critical area analysis. Critical area analysis uses the size and shape of the particles that land on the die and the width and spacing of the wires to calculate the probability that the particles

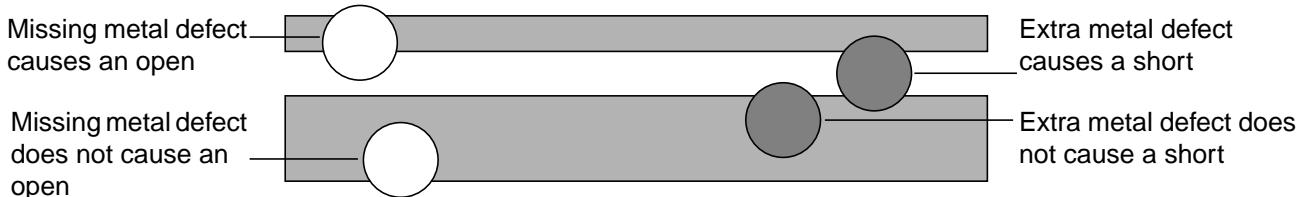
will create opens or shorts. For more information, see “[Critical Area Analysis](#)” on page 1126, “[Open Point Defect](#)” on page 1140, and “[Short Point Defect](#)” on page 1144.

## Critical Area Analysis

The critical area is the area where the center of a particle must land to cause a short or an open. The size and shape of the critical area are dependent on the size and geometry of the particle and the wire width and spacing on the die.

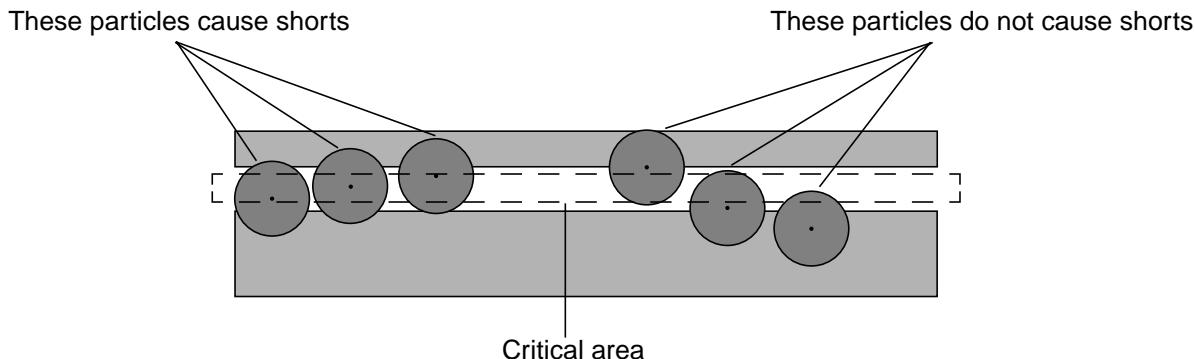
The following figure shows four particles of the same size that have fallen on a die.

- The two particles on the left cause missing metal defects, but only one causes an open.
- The two particles on the right cause extra metal defects, but only one causes a short.



The following figure shows six particles of the same size that have fallen on a die. The critical area is outlined by dashed lines.

- The particles on the left side of the figure cause shorts. The centers of these particles are in the critical area.
- The particles on the right side of the figure do not cause shorts. The centers of these particles are not in the critical area.



**Note:** In reality, defects are not circles. The `reportYield` command uses a square defect model, which is a commonly used approximation in critical area analysis literature that makes calculations faster with little impact on accuracy.

For information on calculating failure probabilities based on critical area analysis, see [“Formulas and Calculations”](#) on page 1153.

## Defect Data and Cumulative Defect Data Functions

The Encounter software can perform critical area analysis calculations based on defects of a specific size or based on a range of defect sizes. In many cases the fabs supply data based on size ranges.

- Direct measurements are called *defect data*, and the associated function is the *defect data function*.
- Size range measurements are called *cumulative defect data*, and the associated function is the *cumulative defect data function*.

The functions are defined by one or more values in a table in the yield technology file and can be derived from each other. The `reportYield` command supports both functions.

For more information, see [“Calculating Defect and Cumulative Defect Data”](#) on page 1153.

## Before You Begin

- Load the yield technology file by running `loadConfig` or `loadYieldTechFile`.
- To report yield statistics on cells, the design must be placed.
- To report yield statistics on vias and wires, the design must be routed.

## Results

- Yield report

Reports the impact on the yield from using different cells, vias, and wire spacing. Also reports a final yield result. The detailed report includes details for each metal and via layer, for different wire widths and spacing, and for each cell.

For more information, see [“Interpreting the Yield Report”](#) on page 1132

- Yield map

## **Encounter User Guide**

### Analyzing Yield

---

Displays a map of the design in the main Encounter window, using up to ten colors to represent different levels of yield loss.

For more information, see ["Interpreting the Yield Map"](#) on page 1129.

## Interpreting the Yield Map

After you run the `reportYield` command, the Encounter software can display a yield map, similar to a congestion map, that represents yield graphically. The software uses different colors to indicate ten different levels of yield. The levels are relative; that is, level 0 represents the lowest yield, level 1 represents the next lowest yield, and so on, to level 9. In general, cooler colors (for example, blue) indicate higher yield levels.

The sections that follow describe how to perform the following actions:

- Display the yield map
- Change the size of the grid
- Change the objects on which the display is based (cells, vias, or wires)
- Change the colors

### Displaying the Yield Map

1. Run the `reportYield` command or select *Tools – Report Yield* from the main Encounter menu and click *OK* on the Report Yield form.
2. Select the following options in the main Encounter window:
  - Physical view
  - Yield Map* visibility toggleTo see the *Yield Map* visibility toggle, move the slider bar under the *All Colors* button to the right.
3. Turn off visibility for the following options in the main Encounter window so you can see the yield map clearly:
  - Instances*
  - Nets*
  - Snets*

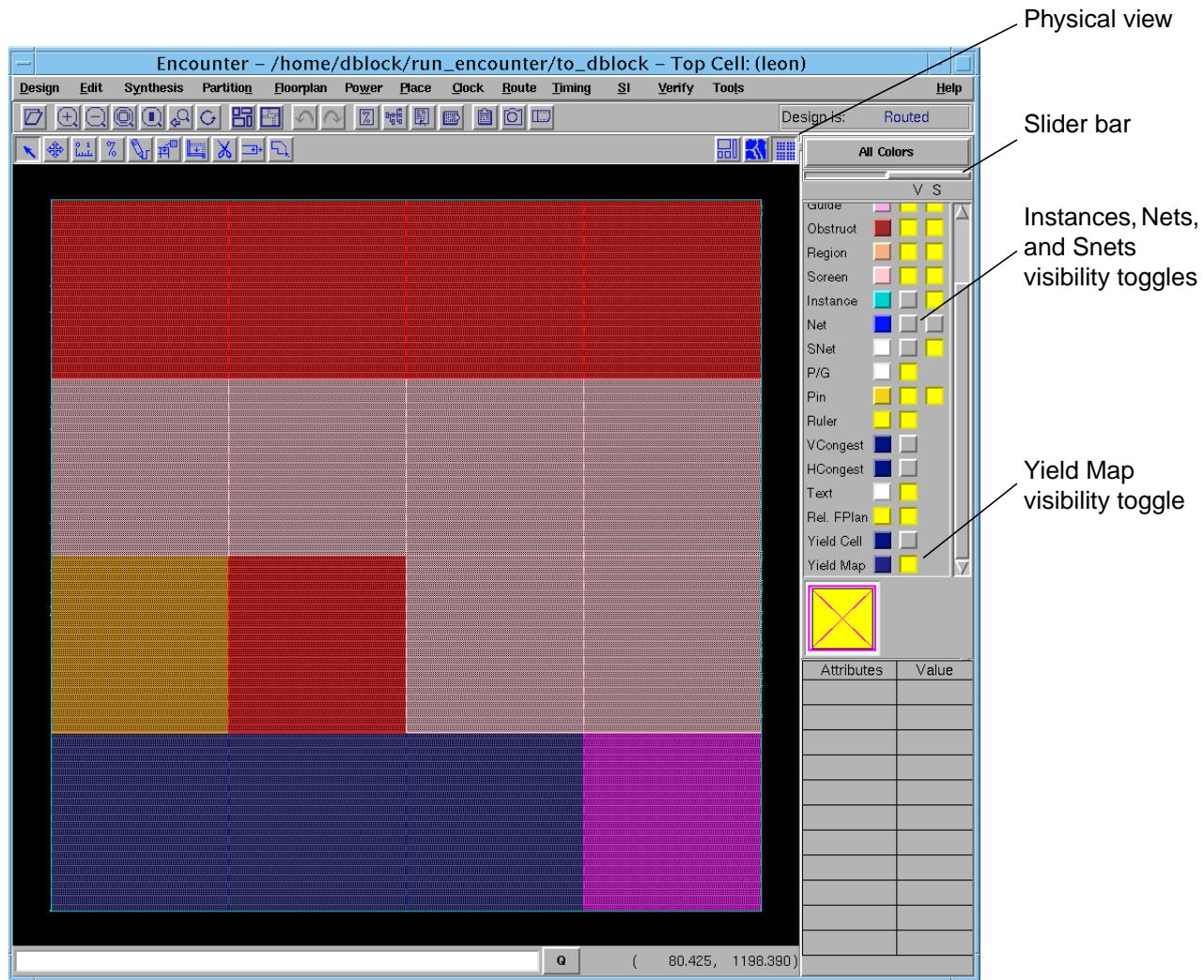


You can also select the layers to display in the map and control the display of yield cells by selecting options on the *Display* page of the *Design – Preferences* form. For more information, see [Yield Map Mode](#) in the “Design Menu” chapter of the Encounter Menu Reference.

## Encounter User Guide

### Analyzing Yield

The following figure shows the main Encounter window with a yield map in the design display area.



### Changing the Bin Size

The default grid size is 50 microns in each direction. To change it, complete the following steps:

1. Select *Display Options* in the Report Yield form.
2. Change the value for one or both of the following options:
  - Display Grid Size X*
  - Display Grid Size Y*.

3. Click *OK* or *Apply*.

### Changing the Object Types

By default, the map shows yield for instances, wires, and vias. To display yield for one or two types of objects, instead of all three types, complete the following steps:

1. Select *Design – Preferences* on the main Encounter menu.
2. Select the *Display* tab on the Preferences form.
3. Select one or more of the following *Yield Map Mode* options:
  - Instance*
  - Wire*
  - Via*
4. Click *OK* or *Apply*.

### Changing the Yield Map Colors

The yield map displays ten relative yield levels, with each level a different color. To change a color, complete the following steps:

1. Click the *All Colors* button in the main Encounter window.
2. On the *View-Only* tab, click the *Yield Map* button.
3. Double-click the color button next to *Yield Map* to open the *Yield Map Color Selection* form.
4. On the *Yield Map Color Selection* form, click the color button next to a level and change the value for *Red*, *Green*, or *Blue*.
5. Click *OK* and close the form.
6. To save the new colors, click the *Save* button at the bottom of the *Color Preferences* form and give the preferences file a name in the *Save Color Preference* form.

## Interpreting the Yield Report

The `reportYield` command generates reports that provide yield data as probabilities and as costs. Data for the chip overall, and for specific features (cells, vias, and routes), are reported.

### ■ Probabilities

Yield probabilities are expressed as percentages. The higher the probability, the greater the yield.

You multiply yield probabilities for the features to see yield probability data for the chip:

$$\text{Cell Yield \%} \times \text{Via Yield \%} \times \text{Route Yield \%} = \text{Chip Yield \%}$$

**Note:** The Summary section, which appears in the text command window and at the end of the report, also includes values for Yield Loss. The Yield Loss is the difference between 100 percent and Yield. The total Yield Loss for the chip is the difference between 100 percent and the total chip yield percent.

### ■ Costs

Yield costs are derived from the yield probabilities. Costs make yield calculations more convenient than probabilities because they can be added or subtracted rather than multiplied. The formula for the cost of a feature is

$$C_i = -\ln(1-P_i)$$

where

$C_i$  = cost of feature  $i$

$P_i$  = probability of failure of feature  $i$

The costs are expressed as floating-point numbers. The greater the cost, the lower the yield.

You add the costs for the features to see the total yield cost for the chip:

$$\text{Cell Cost} + \text{Via Cost} + \text{Routing Cost} = \text{Total Cost}$$

For more information, see [“Cost Formulas”](#) on page 1156.

## Yield Report

Following are sections from yield report for a design named `test`. The report is saved in the working directory as `test.yield.rpt`. The report has the following sections:

### ■ [Header](#) on page 1133

- [Cell](#) on page 1133
- [Via](#) on page 1133
- [Routing](#) on page 1134
- [Summary](#) on page 1136

## Header

Displays the software name and build, the operating system and host name, the date the report was generated, and the command used to generate the report. In addition, the header includes a title for the report, the design name and dimension.

```
#####
# Generated by:      Cadence First Encounter 08.10-b039_1
# OS:                Linux x86_64(Host ID abcde)
# Generated on:     Thu Aug  7 16:02:08
# Command:          reportYield
#####
Encounter Defect Limited Yield and Cost Report

Title      : Created by SOC Encounter with generic values
Design     : test
Dimension  : 465.000um x 460.000um
Shrink Factor: 0.999
```

## Cell

Gives yield and cost data for each type of cell (core, pad, blocks, and other), and total cell yield data.

	Cost	Yield %
Core Cells:	5.0682e-03	99.494
Pad Cells :	0.0000e+00	100.000
Blocks    :	0.0000e+00	100.000
Other     :	0.0000e+00	100.000
-----		
Total     :	5.0682e-03	99.494

## Via

Gives yield and cost data for single-cut vias, double-cut vias, and array vias (vias with more than two cuts) for each via layer and for all via layers, the total via cost per via layer, and the total via yield.

## Encounter User Guide

### Analyzing Yield

---

Layer	Number of Vias			Cost			Total Cost
	1-cut	2-cut	Array	1-cut	2-cut	Array	
via12	2.0791e+05	0.0000e+00	0.0000e+00	2.0791e-04	0.0000e+00	0.0000e+00	2.0791e-04
via23	7.0367e+04	0.0000e+00	0.0000e+00	3.0763e-04	0.0000e+00	0.0000e+00	3.0763e-04
via34	1.6723e+04	0.0000e+00	0.0000e+00	2.2463e-04	0.0000e+00	0.0000e+00	2.2463e-04
via45	1.4420e+03	0.0000e+00	0.0000e+00	7.1809e-05	0.0000e+00	0.0000e+00	7.1809e-05
via56	0.0000e+00	0.0000e+00	0.0000e+00	1.6723e-05	0.0000e+00	0.0000e+00	1.6723e-05
-----	-----	-----	-----	-----	-----	-----	-----
total	2.9644e+05	0.0000e+00	0.0000e+00	8.2870e-04	0.0000e+00	0.0000e+00	8.2870e-04

Total via yield: 99.917%

## Routing

Gives the following data:

- Routing costs and yield loss for the following:
  - Opens, shorts, and vias on each metal layer
  - Totals per metal layer
  - Totals for opens, shorts and vias
- Total routing yield percent
- Combined horizontal and vertical wire width and spacing data
- Critical area for shorts and opens, per layer

Costs of various design features, per layer

Layer	opens	shorts	via	total
METAL1	0.0002	0.0000	0.0000	0.0003
METAL2	0.0009	0.0004	0.0000	0.0013
METAL3	0.0012	0.0007	0.0000	0.0018
METAL4	0.0009	0.0005	0.0000	0.0014
METAL5	0.0004	0.0002	0.0000	0.0006
METAL6	0.0001	0.0000	0.0000	0.0001
	-----			
Total	0.0038	0.0018	0.0000	0.0056

Yield losses in percent. [Same data presented differently.]

Layer	opens	shorts	via	total
METAL1	0.0245	0.0026	0.0000	0.0271
METAL2	0.0915	0.0412	0.0000	0.1327
METAL3	0.1165	0.0664	0.0000	0.1828

## Encounter User Guide

### Analyzing Yield

---

METAL4   0.0904 0.0495 0.0000   0.1398
METAL5   0.0437 0.0180 0.0000   0.0616
METAL6   0.0084 0.0025 0.0000   0.0109
-----
Total   0.3744 0.1801 0.0000   0.5538

Total routing yield 99.4462%

Combined horizontal and vertical wires for layer METALL

wire widths (nm)	length(m)	approximate cost
0.0 < x <= 200.0,	0.077785720	0.00006870
200.0 < x <= 300.0,	0.106160940	0.00008057
300.0 < x <= 400.0,	0.000000000	0.00000000
400.0 < x <= 500.0,	0.000820480	0.00000030
500.0 < x <= 1200.0,	0.008865920	0.00000252
1200.0 < x <= 2000.0,	0.001695840	0.00000010
2000.0 < x <= 4000.0,	0.001165920	0.00000002

Combined horizontal and vertical spaces for layer METALL

spacing (nm)	length(m)	approximate cost
0.0 < x <= 200.0,	0.004468480	0.00000377
200.0 < x <= 300.0,	0.005448040	0.00000413
300.0 < x <= 400.0,	0.000499400	0.00000033
400.0 < x <= 500.0,	0.000133120	0.00000006
500.0 < x <= 1200.0,	0.020538250	0.00000400
1200.0 < x <= 2000.0,	0.019310920	0.00000094
2000.0 < x <= 4000.0,	0.036233835	0.00000055

Critical area for shorts, layer METALL

Diameter(nm)	Radius(nm)	Crit area(cm <sup>2</sup> )
0.0	0.0	0.000000e+00
200.0	100.0	4.575570e-07
300.0	150.0	7.698167e-06
400.0	200.0	1.808128e-05
500.0	250.0	2.861692e-05
1200.0	600.0	1.797329e-04
2000.0	1000.0	4.704228e-04
4000.0	2000.0	1.575977e-03

Critical area for opens, layer METALL

## Encounter User Guide

### Analyzing Yield

---

Diameter (nm)	Radius (nm)	Crit area (cm <sup>2</sup> )
0.0	0.0	0.000000e+00
200.0	100.0	3.111429e-05
300.0	150.0	1.648091e-04
400.0	200.0	3.487557e-04
500.0	250.0	5.327854e-04
1200.0	600.0	1.801876e-03
2000.0	1000.0	3.131847e-03
4000.0	2000.0	5.871899e-03
...		

Data for additional metal layers follows the data for *METAL 1*.

## Summary

Gives summary cost, yield, and yield loss for cells, vias, and routing, total yield percent, and yield loss percent.

	Cost	Yield %	Yield Loss %
Cell :	5.0682E-03	99.494	0.506
Via :	8.2870E-04	99.917	0.083
Routing:	5.5536E-03	99.446	0.554
Total :	1.1450E-02	98.861	1.139

## Detailed Report

The detailed report contains all the statistics in the non-detailed report, plus the following additional detailed sections:

- [Vias on page 1136](#)
- [Cells on page 1137](#)
- [Subareas on page 1137](#)

## Vias

The via section includes statistics for each via. Following is a via detail section for layer *via 12*:

Layer	Number Cuts	Number Vias	Approx Cost	Via Name
Vial2	1	5.4287e+04	1.0857e-04	Via2tsl
	1	1.5245e+05	3.0490e-04	Via2

## Encounter User Guide

### Analyzing Yield

1	5.7500e+02	1.1500e-06	Via2tsls
1	5.8900e+02	1.1780e-06	Via2tsh
1	7.0000e+00	1.4000e-08	Via2tshs
	-----   -----	-----   -----	-----   -----
	2.0791e+05	4.1582e-04	

## Cells

The cell section includes statistics for each cell. The cells are listed in the order they appear in the Encounter database. Following are entries for several cells:

name	number	cost
SPN_OR3_4	22	1.5716e-06
SPN_OR3_2	28	1.5483e-06
SPN_OR3_1	14	6.7594e-07
SPN_OR2_12	0	0.0000e+00
SPN_OR2_8	0	0.0000e+00
SPN_OR2_6	0	0.0000e+00
SPN_OR2_4	183	1.1356e-05
SPN_OR2_3	0	0.0000e+00
SPN_OR2_2	89	4.0853e-06
SPN_OR2_1	196	7.5502e-06

## Subareas

The last section includes yield statistics by subarea. Subareas correspond to the bin size in reportYield.

SubArea	Location (um)				Yield	Cost
x1	y1	x2	y2	Cell	Via	Routing
=====	=====	=====	=====	=====	=====	=====
0, 17	[ 816,	0]	[ 864,	48]	0.000000,	0.000000,
0, 18	[ 864,	0]	[ 913,	48]	0.000002,	0.000000,
0, 19	[ 913,	0]	[ 961,	48]	0.000010,	0.000002,
0, 20	[ 961,	0]	[ 1009,	48]	0.000010,	0.000002,
0, 21	[ 1009,	0]	[ 1057,	48]	0.000011,	0.000002,
0, 22	[ 1057,	0]	[ 1105,	48]	0.000010,	0.000002,
0, 23	[ 1105,	0]	[ 1153,	48]	0.000010,	0.000002,
...						
					total cost:	cell 0.005068, via 0.000829, routing 0.005554

## Understanding the Yield Technology File

The yield technology file contains models for cell and via defects, and for open and short defects in routing. The Encounter software integrates the models with critical area analysis to calculate how much the yield is affected by using different cells, vias, wire spacing and wire widening.

The yield technology file is typically supplied by the fab or the library vendor. However, if the fab or library vendor does not supply the file, the software generates a template file from generic values computed from the minimum width, minimum spacing, vias, and cells specified in the LEF file. The generic values come from public sources such as the International Technology Roadmap for Semiconductors (ITRS), so users without access to fab data can have an approximate estimate of yield impacts from different yield optimization choices. You can use the template file to experiment with `reportYield`.



Cadence recommends that you use real yield coefficients from the fab or library vendor in order to get accurate yield estimates.

The following commands load the yield file automatically during design import:

- `loadConfig`
- `loadYieldTechFile`

The following commands require that the yield technology file is loaded:

- `reportYield`
- `optCellYield`

## File Format

The file uses XML syntax with the following basic XML format:

- Statements
  - XML statements have the following format:  
`<keyword> ... </keyword>`  
For example,  
`<layers> m1 m2 m3 </layers>`
  - Unrecognized keyword statements are ignored.

## Encounter User Guide

### Analyzing Yield

---

- Statements can be nested inside other statements.
- Comments
  - XML comments are included between `<!--` and `-->` delimiters. For example,  
`<!-- This is an XML comment.-->`

For more information on XML, see <http://www.w3c.org> or any other XML reference.

## Conventions

### ■ Layer names

The yield technology file uses the layer names from the LEF technology file. The names are case-sensitive.

### ■ Distance units values

In the yield technology file, many values are specified in distance units. Supported distance units include the following:

- nanometer (nm)

1 nm = 1e-9 meters

- micron (um)

1 um = 1e-6 meters

- centimeter (cm)

1 cm = 1e-2 meters

In some cases, a squared or cubed unit value, such as  $\text{cm}^2$ , is required. This value is written in the file as  $\text{cm}^2$ . A value like 5.0 defects per  $\text{cm}^2$  is written as 5.0 /  $\text{cm}^2$ .

### ■ Probabilities of failure

Probabilities of failure are expressed as floating point numbers between 0.0 and 1.0.

### ■ File sections

Each section of the file has a separate keyword.

## File Sections and Keyword Statement Descriptions

The yield technology file contains the following sections:

- [Header](#) on page 1140
- [Open Point Defect](#) on page 1140
- [Short Point Defect](#) on page 1144
- [Via Probability](#) on page 1145
- [Cell Probability](#) on page 1148

### Header

```
<yield_file>
  [<title> title]
...
</yield_file>
```

Declares that the file is a yield file and contains the entire file. For example:

```
<yield_file>
  <title> 90nm Standard Process Version 1.21</title>
...
</yield_file>
```

### Keyword Statements

<pre>&lt;yield_file&gt;</pre>	The first statement in the yield file. Statements outside of (before or after) the <code>yield_file</code> statement are ignored.
<pre>&lt;title&gt;</pre>	The title is optional. The title is written to the report file.

### Open Point Defect

```
<open_point_defect>
  <layers> layer1 [layer2 ...] </layers>
  <defect_width_range> minDefectWidth maxDefectWidth </defect_width_range>
  <reporting_widths> width1 [width2 ...] </reporting_widths>
  {<cumulative_defect_width_and_density> defectWidth defectsPerArea
   </cumulative_defect_width_and_density>
   |<defect_width_and_density> defectWidth defectDensity
   </defect_width_and_density>}
</open_point_defect>
```

Describes the density of defects that cause opens. A defect that lands on a metal wire and creates a hole in the metal causes an open. The file can have more than one open point defect table.

## Encounter User Guide

### Analyzing Yield

---

An open point defect table can include defect data statements, which describe the defect size distribution directly, or cumulative defect data statements, which aggregate the data from defects up to a specified size. The following example includes cumulative defect data statements.

```
<open_point_defect>
  <layers> m1 m2 m3 </layers>
  <defect_width_range> 100nm 2000nm </defect_width_range>
  <reporting_widths> 100nm 150nm 200nm 300nm 400nm 1000nm 2000nm
    </reporting_widths>
  <cumulative_defect_width_and_density> 100nm 1.00/cm^2
    </cumulative_defect_width_and_density>
  <cumulative_defect_width_and_density> 200nm 0.25/cm^2
    </cumulative_defect_width_and_density>
</open_point_defect>
```

#### **Keyword Statements**

`<layers> layer1 [layer2 ...] </layers>`

Specifies the layers to which this table applies. You must specify at least one layer.

`<defect_width_range> minDefectWidth maxDefectWidth
 </defect_width_range>`

Specifies the minimum and maximum width of defects to consider.

Typically, `minDefectWidth` is the same as the minimum width for the layer and `maxDefectWidth` is the largest defect that matters. Since the defect size normally falls off as  $1/\text{size}^3$ , a value of 10 or 20 times minimum width is sufficient.

A distance units value is required for both `minDefectWidth` and `maxDefectWidth`.

`<reporting_widths> width1 [width2 ...] </reporting_widths>`

Specifies the widths for the detailed report file that is output by the `reportYield` command. The detailed report includes total length of each width, critical area analysis for each width, and additional statistics.

Typically, `width1` is the minimum width for the layer and the last width is the maximum width for the layer. You must specify a value for at least one width. A distance units value is required for each width.



Each open point defect table must contain either defect width or cumulative defect width and density keyword statements. It cannot contain both types of statements.

```
<defect_width_and_density> defectWidth defectDensity  
</defect_width_and_density>
```

The defect width is the diameter of the defect. The defect density of a defect of width *defectWidth* is *defectDensity*.

- *defectWidth* is specified in distance units, typically nm.
- *defectDensity* is specified in units of 1/distance<sup>3</sup>, either 1/cm<sup>2</sup>/nm or 1/nm<sup>3</sup> (1/cm<sup>2</sup>/nm means there are *defectDensity* defects per cm<sup>2</sup> between *defectWidth* and *defectWidth* + 1 nm in size).

<defect\_width\_and\_density> values are used to define the DSD(x) function. For more information, see [“Defect Data Function”](#) on page 1153.

The table can have one or more defect data statements.

- If only one defect data statement is specified, the defect density is assumed to fall off as 1/width<sup>3</sup>.
- If more than one defect data statement is specified, intermediate data points are interpolated by using a log scale which assumes cumulative defects fall off as 1/width<sup>p</sup>.

## Encounter User Guide

### Analyzing Yield

---

```
<cumulative_defect_width_and_density> defectWidth defectsPerArea  
</cumulative_defect_width_and_density>
```

There are *defectsPerArea* with a width (diameter) that is greater than or equal to *defectWidth*.

- *defectWidth* is specified in distance units, typically nm.
- *defectsPerArea* is specified in units of 1/distance<sup>2</sup>.

<cumulative\_defect\_width\_and\_density> values are used to define the CDSD(x) function. For more information, see [“Cumulative Defect Data Function” on page 1154](#).

An open point defect table can have one or more cumulative defect data statements.

- If only one cumulative defect data statement is specified, the cumulative defect density is assumed to fall off as 1/width<sup>2</sup>.
- If more than one cumulative defect data statement is specified, intermediate data points are interpolated. For more information, see [“Converting defect bin size data to defect data statements” on page 1143](#).

## Examples

- Using data measured as total defects greater than or equal to x size (cumulative data statements)

In the following example, the fab measured defects on *m1* and *m2* with a minimum width of 0.10 um. The density was:

```
0.1 defects/cm2 >= 100nm defect size  
0.025 defects/cm2 >= 200nm defect size
```

The <cumulative\_defect\_width\_and\_density> statements for these values are:

```
<open_point_defect>  
  <layers> m1 m2 </layers>  
  <cumulative_defect_width_and_density> 100nm  
  0.1/cm2 </cumulative_defect_width_and_density>  
  <cumulative_defect_width_and_density> 200nm  
  0.025/cm2 </cumulative_defect_width_and_density>  
</open_point_defect>
```

- Converting defect bin size data to defect data statements

## Encounter User Guide

### Analyzing Yield

---

In the following example, the fab measured the total number of defects for various defect size bins and made the following measurements for  $m1$  and  $m2$ :

Bin 1    Defect width between 100 nm and 150 nm = 0.05 defects/cm<sup>2</sup>

Bin 2    Defect width between 150 nm and 200 nm = 0.02 defects/cm<sup>2</sup>

Assume an exponent of 3 for the distribution within a given bin range. Then, for a given bin, from  $x_1$  to  $x_2$  with a measured number of defects/cm<sup>2</sup> =  $D$  for the bin, the formula is:

$$C_D = 2 * D / [(1/x_1)^2 - (1/x_2)^2]$$

At any point  $x$  inside the bin, the number of defects of size  $x$  is

$$C_D * (1/x)^3$$

Solving for the midpoint of the bins, the two bins (scaling the 1/cm<sup>2</sup> to 1/nm<sup>2</sup> with 1e-14), results in the following values:

Bin 1     $C_D = 2 * 0.05 \times 1e-14 / [ (1/100)^2 - (1/150)^2 ] = 0.1e-14 * 1.8e4 = 1.8e-11 \text{ DSD}(125) = C_D * (1/125)^3 = 1.8e-11 * (1/125)^3 = 9.22e-18/\text{nm}^3 \text{ at } 125\text{nm}$

Bin 2     $C_D = 2 * 0.02 \times 1e-14 / [ (1/150)^2 - (1/200)^2 ] = 0.04e-14 * 5.15e4 = 2.06e-11 \text{ DSD}(175) = C_D * (1/175\text{nm})^3 = 2.06e-11 * (1/175)^3 = 3.84e-18/\text{nm}^3 \text{ at } 150\text{nm}$

For Bin 1, the data means that the number of defects between 125 nm and 126 nm wide is

$$\sim 9.22e-18/\text{nm}^3 * 1\text{nm} = 9.22e-18/\text{nm}^2 = 9.22e-4/\text{cm}^2$$

The <defect\_width\_and\_density> statements for these values are:

```
<open_point_defect>
  <layers> m1 m2 </layers>
  <defect_width_and_density> 125nm 9.22e-18/nm^3 </defect_width_and_density>
  <defect_width_and_density> 175nm 3.84e-18/nm^3 </defect_width_and_density>
</open_point_defect>
```

## Short Point Defect

```
<short_point_defect>
  <layers> layer1 [layer2 ...] </layers>
  <defect_width_range> minDefectWidth maxDefectWidth </defect_width_range>
  <reporting_spacings> spacing1 [spacing2 ...] </reporting_spacing>
  <cumulative_defect_width_and_density> defectWidth defectsPerArea
  </cumulative_defect_width_and_density>
</short_point_defect>
```

## Encounter User Guide

### Analyzing Yield

---

Describes the density of defects that cause shorts. A defect that lands between two metal wires, adding extra metal, causes a short. The file can have more than one short point defect table.

A short point defect table can include cumulative defect data, as in the following example, or defect data.

```
<short_point_defect>
  <layers> m1 m2 m3 </layers>
  <defect_width_range> 100nm 2000nm </defect_width_range>
  <reporting_spacings> 100nm 150nm 200nm 300nm 400nm 1000nm 2000nm
    </reporting_spacings>
  <cumulative_defect_width_and_density> 100nm 1.00/cm^2
    </cumulative_defect_width_and_density>
  <cumulative_defect_width_and_density> 200nm 0.25/cm^2
    </cumulative_defect_width_and_density>
</short_point_defect>
```

### **Keyword Statements**

The syntax of the short point defect table is identical to that of the open point defect table, except that reporting spacings are used, instead of reporting widths:

```
<reporting_spacings> spacing1 [spacing2 ...]</reporting_spacings>
```

Specifies the spacing ranges to use when reporting wire statistics. The first value is typically the minimum spacing for the layer and the last value is the largest spacing of interest.

### **Via Probability**

```
<via_probability>
  <layers> bottomLayer cutLayer topLayer </layers>
  <via> probability_of_failure
    <cuts> number_of_cuts
      <lower_along> lowerAlong1 lowerAlong2 </lower_along>
      <lower_across> lowerAcross1 lowerAcross2 </lower_across>
      <upper_along> upperAlong1 upperAlong2 </upper_along>
      <upper_across> upperAcross1 upperAcross2 </upper_across>
    </via>
  </via_probability>
```

Describes the probability of failure for different types of vias, where a via is defined as a combination of metal below, via cuts, and metal above. Each via probability table has one or more layers statements and one or more via statements.

```
<via_probability>
  <layers> M1 via12 M2 </layers>
  <layers> M2 via23 M3 </layers>
  <via> 1.0e-9
    <cuts> 1 </cuts>
```

```
<lower_along> 50nm 50nm </lower_along>
<lower_across> 5nm 5nm </lower_across>
<upper_along> 50nm 50nm </upper_along>
<upper_across> 5nm 5nm </upper_across>
</via>
</via_probability>
```

### **Keyword Statements**

```
<layers> metalLayerBelow viaLayer metalLayerAbove </layers>
```

Each layers statement specifies the following three layers:

- Metal layer below the via layer
- Via layer itself
- Metal layer above the via layer

```
<via> probFailure
  <cuts> numCuts </cuts>
  <lower_along> lowerAlong1 lowerAlong2 </lower_along>
  <lower_across> lowerAcross1 lowerAcross2 </lower_across>
  <upper_along> upperAlong1 upperAlong2 </upper_along>
  <upper_across> upperAcross1 upperAcross2 </upper_across>
</via>
```

Each `<via>` statement specifies the following:

- `probFailure`  
The probability of failure for this via
- `<cuts>`  
The number of cuts in this via
- `<lower_along> <lower_across> <upper_along> <upper_across>`  
Overhang values for upper and lower layers, specified in distance units.

### **Examples**

- Single-cut via

## Encounter User Guide

### Analyzing Yield

---

In the following example, the first <via> statement below says a single-cut via with overhangs of 50 nm along and 5 nm across for both metal layers has a probability of failure of 1.0e-9.

```
<via_probability>
<layers> M1 vial2 M2 </layers>
<layers> M2 via23 M3 </layers>
<via> 1.0e-9
  <cuts> 1 </cuts>
  <lower_along> 50nm 50nm </lower_along>
  <lower_across> 5nm 5nm </lower_across>
  <upper_along> 50nm 50nm </upper_along>
  <upper_across> 5nm 5nm </upper_across>
</via>
<via> 0.8e-9
  <cuts> 1 </cuts>
  <lower_along> 80nm 80nm </lower_along>
  <lower_across> 5nm 5nm </lower_across>
  <upper_along> 80nm 80nm </upper_along>
  <upper_across> 5nm 5nm </upper_across>
</via>
<via> 0.7e-9
  <cuts> 1 </cuts>
  <lower_along> 30nm 30nm </lower_along>
  <lower_across> 30nm 30nm </lower_across>
  <upper_along> 30nm 30nm </upper_along>
  <upper_across> 30nm 30nm </upper_across>
</via>
</via_probability>
```

If a via with different metal overhangs than given in the example is encountered, it has the probability of failure of the via in the table with the lowest failure rate, as long as it meets all the overhang values for that via.

In the example above,

- ❑ A via with 60 nm along and 5 nm across on both metal layers only meets the 50 nm/5 nm overhang of the first <via> statement above, so it uses that data (1.0e-9).
- ❑ A via with 90 nm along and 5 nm across on both metal layers meets the 80 nm/5 nm overhang of the second <via> statement, so it uses that data (0.8e-9).
- ❑ If a via cannot meet all the overhangs of any of the <via\_probability> statements, it uses the data for the via with the highest probability of failure for a via with the same number of cuts (the first <via> statement's value of 1.0e-9).

#### ■ Double-cut via

Double-cut vias typically do not vary much by overhang, so normally only one entry is necessary in the table.

```
<via_probability>
<layers> M1 vial2 M2 </layers>
<layers> M2 via23 M3 </layers>
<via> 0.05e-9
  <cuts> 2 </cuts>
```

## Encounter User Guide

### Analyzing Yield

---

```
<lower_along> 50nm 50nm </lower_along>
<lower_across> 5nm 5nm </lower_across>
<upper_along> 50nm 50nm </upper_along>
<upper_across> 5nm 5nm </upper_across>
</via>
</via_probability>
```

#### ■ Via with more than two cuts

By default, the yield technology file reports failure rates for three types of vias: single-cut vias, double-cut vias, and array vias (vias with more than two cuts).

To report failure probabilities for array vias, define `<via_probability>` statements with a probability of failure for vias with `<cuts> 3 </cuts>` and add a statement like the following for triple-cut vias:

```
<via_probability>
<layers> M1 via12 M2 </layers>
<layers> M2 via23 M3 </layers>
<!-- vias with > 2 cuts might have 0 probability of failure -->
<via> 0.0
<cuts> 3 </cuts>
<lower_along> 0nm 0nm </lower_along>
<lower_across> 0nm 0nm </lower_across>
<upper_along> 0nm 0nm </upper_along>
<upper_across> 0nm 0nm </upper_across>
</via>
</via_probability>
```

If no array-cut via probability is given, array vias use the probability of failure for the via with two cuts with the lowest probability of failure.

## Cell Probability

```
<cell_probability>
<cell> cell_name </cell>
  <instance> probInstFail </instance>
  <systematic> probSystematicFail </systematic>
</cell>
</cell_probability>
```

Describes the probability of failure for different types of cells, including the probability that a particular instance of a cell will fail and the probability that one or more instances of a cell will fail.

```
<cell_probability>
<cell> and2_1x
  <instance>    1.2e-8    </instance>
  <systematic>  1.0e-7    </systematic>
</cell>
<cell> and2_2x
  <instance>    1.6e-8    </instance>
  <systematic>  1.2e-7    </systematic>
</cell>
</cell_probability>
```

## **Keyword Statements**

```
<cell> cellName
  [<instance> probInstFail </instance>]
  [<systematic> probSystematicFail </systematic>]
</cell>

<cell> cellName
  Specifies a cell in the .lib or LEF file. The
  cellName is case sensitive. Each cell
  probability section has zero or more
  <cell> statements.

<instance> probInstanceFail
  Specifies the probability that one instance of
  the cell will fail. The probability is specified
  as a floating point number between 0.0 and
  1.0.

<systematic> probSystematicFail
  Specifies the probability of a systematic
  failure of the cell, independent of the number
  of instances. The probability is specified as a
  floating point number between 0.0 and 1.0.
```

## **Example**

In the following example, the and2\_1x cell has a 1.2e-8 probability of failure for each instance of and2\_1x. In addition, there is a systematic probability of failure of 1.0e-7 if any and2\_1x cells are used.

Therefore, a design that has five and2\_1x instances, would have a defect-limited yield equal to  $(1 - 1.2\text{e-}8)^5 * (1 - 1.0\text{e-}7)$  from the usage of and2\_1x cells.

The cell probability statement for these values is

```
<cell_probability>
<cell> and2_1x
  <instance> 1.2e-8    </instance >
  <systematic> 1.0e-7   </systematic>
</cell>
...
</cell_probability>
```

## **Yield Technology File Example**

The following yield technology file is for a technology with three thin metal layers and two thick layers (twice the minimum width).

```
<?xml version="1.0"?>
<yield_file>
<title> 65nm Standard Process Version 1.21 </title>
<!-- This is an XML comment -->
<!-- This section describes the density of defects-->
<!-- of the kind that removes metal (cause opens).-->
<open_point_defect>
<layers> m1 m2 m3 </layers>
<!-- Widths of defects to do critical area analysis -->
<defect_width_range> 100nm 2000nm </defect_width_range>
<!-- Width-ranges to use when reporting wire statistics. -->
<!-- The first value is typically min-width. -->
<reporting_widths> 100nm 150nm 200nm 300nm 400nm 1000nm 2000nm
</reporting_widths>
<!-- Defect width and the number of defects/area >= width -->
<cumulative_defect_width_and_density> 100nm 1.00/cm^2
</cumulative_defect_width_and_density>
<cumulative_defect_width_and_density> 200nm 0.25/cm^2
</cumulative_defect_width_and_density>
</open_point_defect>
<open_point_defect>
<!-- 2x metal layers -->
<layers> m4 m5 </layers>
<!-- Widths of defects to do critical area analysis -->
<defect_width_range> 200nm 2000nm </defect_width_range>
<!-- Width-ranges to use when reporting wire statistics. -->
<!-- The first value is typically min-width. -->
<reporting_widths> 200nm 300nm 400nm 500nm 1000nm 2000nm </reporting_widths>
<!-- Defect width and the number of defects/area >= width -->
<cumulative_defect_width_and_density> 200nm 1.00/cm^2
</cumulative_defect_width_and_density> <cumulative_defect_width_and_density>
400nm 0.25/cm^2 </cumulative_defect_width_and_density>
</open_point_defect>
<!-- This section describes the density of defects-->
<!-- of the kind that add extra metal (causes shorts).-->
<short_point_defect>
<layers> m1 m2 m3 </layers>
<!-- Widths of defects to do critical area analysis -->
<defect_width_range> 100nm 2000nm </defect_width_range>
<!-- Spacing-ranges to use when reporting wire statistics. -->
<!-- The first value is typically min-spacing. -->
<reporting_spacings> 100nm 150nm 200nm 300nm 400nm 1000nm 2000nm
</reporting_spacings>
<!-- Defect width and the number of defects/area >= width -->
<cumulative_defect_width_and_density> 100nm 1.00/cm^2
</cumulative_defect_width_and_density> <cumulative_defect_width_and_density>
200nm 0.25/cm^2 </cumulative_defect_width_and_density>
</short_point_defect>
<short_point_defect>
<!-- 2x metal layers -->
<layers> m4 m5 </layers>
<!-- Widths of defects to do critical area analysis -->
<defect_width_range> 200nm 2000nm </defect_width_range>
```

# Encounter User Guide

## Analyzing Yield

---

```
<!-- Spacing-ranges to use when reporting wire statistics. -->
<!-- The first value is typically min-spacing. -->
<reporting_spacings> 200nm 300nm 400nm 500nm 1000nm 2000nm
  </reporting_spacings>
<!-- Defect width and the number of defects/area >= width -->
<cumulative_defect_width_and_density> 200nm 1.00/cm^2
</cumulative_defect_width_and_density>
<cumulative_defect_width_and_density> 400nm 0.25/cm^2
</cumulative_defect_width_and_density>
</short_point_defect>

<!-- This section describes different via's probability of failure -->
<via_probability>
  <!--the layers for these vias-->
  <layers> M1 via12 M2 </layers>
  <layers> M2 via23 M3 </layers>
  <!-- Probability of failure for a given via. Each via has -->
  <!-- the number of cuts, and various overhang values for the -->
  <!-- metal layers below and above the via layer. -->
  <via> 1.0e-9    <!-- min-overhang -->
    <cuts> 1 </cuts>
    <lower_along> 50nm 50nm </lower_along>
    <lower_across> 5nm 5nm </lower_across>
    <upper_along> 50nm 50nm </upper_along>
    <upper_across> 5nm 5nm </upper_across>
  </via>
  <via> 0.8e-9    <!-- extra-overhang -->
    <cuts> 1 </cuts>
    <lower_along> 80nm 80nm </lower_along>
    <lower_across> 5nm 5nm </lower_across>
    <upper_along> 80nm 80nm </upper_along>
    <upper_across> 5nm 5nm </upper_across>
  </via>
  <via> 0.7e-9    <!-- equal overhang -->
    <cuts> 1 </cuts>
    <lower_along> 30nm 30nm </lower_along>
    <lower_across> 30nm 30nm </lower_across>
    <upper_along> 30nm 30nm </upper_along>
    <upper_across> 30nm 30nm </upper_across>
  </via>
  <!-- double-cut vias typically don't vary much by overhang -->
  <!-- so only one entry is necessary. -->
  <via> 0.05e-9
    <cuts> 2 </cuts>
    <lower_along> 50nm 50nm </lower_along>
    <lower_across> 5nm 5nm </lower_across>
    <upper_along> 50nm 50nm </upper_along>
    <upper_across> 5nm 5nm </upper_across>
  </via>
  <!-- vias with > 2 cuts have 0 probability of failure -->
  <via> 0.0
    <cuts> 3 </cuts>
    <lower_along> 50nm 50nm </lower_along>
    <lower_across> 5nm 5nm </lower_across>
    <upper_along> 50nm 50nm </upper_along>
    <upper_across> 5nm 5nm </upper_across>
  </via>
</via_probability>
<via_probability>
  <layers> M3 via34 M4 </layers>
  <layers> M4 via45 M5 </layers>
```

# Encounter User Guide

## Analyzing Yield

---

```
<via> 1.0e-9      <!-- min-overhang -->
<cuts> 1 </cuts>
<lower_along> 60nm 60nm </lower_along>
<lower_across> 10nm 10nm </lower_across>
<upper_along> 60nm 60nm </upper_along>
<upper_across> 10nm 10nm </upper_across>
</via> <via> 0.8e-9      <!-- extra-overhang -->
<cuts> 1 </cuts>
<lower_along> 90nm 90nm </lower_along>
<lower_across> 10nm 10nm </lower_across>
<upper_along> 90nm 90nm </upper_along>
<upper_across> 10nm 10nm </upper_across>
</via>      <!-- double-cut via -->
<via> 0.05e-9
<cuts> 2 </cuts>
<lower_along> 60nm 60nm </lower_along>
<lower_across> 10nm 10nm </lower_across>
<upper_along> 60nm 60nm </upper_along>
<upper_across> 10nm 10nm </upper_across>
</via>
<!-- vias with > 2 cuts have 0 probability of failure -->
<via> 0.0
<cuts> 3 </cuts>
<lower_along> 60nm 60nm </lower_along>
<lower_across> 10nm 10nm </lower_across>
<upper_along> 60nm 60nm </upper_along>
<upper_across> 10nm 10nm </upper_across>
</via>
</via_probability>

<!-- This section has the probability of failure for each cell -->
<!-- <instance> is the probability of failure for each -->
<!-- instance of the cell. <systematic> is the probability -->
<!-- of failure is 1 or more instances of the cell are used (e.g. -->
<!-- the failure rate is independent of the number of instances) -->
<cell_probability>
<cell> and2_1x
    <instance>      1.2e-8      </instance>
    <systematic>    1.0e-7      </systematic>
</cell>
<cell> and2_2x
    <instance>      1.6e-8      </instance>
    <systematic>    1.2e-7      </systematic>
</cell>
...
</cell_probability>
</yield_file>
```

## Formulas and Calculations

### Calculating the Probability of Failure for a Metal Layer

The formula for calculating the probability of failure of a metal layer for each defect width is:

$$\lambda = \int_{x_0}^{\infty} CA(x)DSD(x) dx$$

where

$\lambda$  is the expected number of failures for one die (defects that cause a failure).

$CA(x)$  is the critical area for a particle of width  $x$ .

$x_0$  is the reference defect width, which is the smallest defect width of interest. It is typically less than or equal to the minimum width for a given layer.

$DSD(x)$  is the number of defects equal to size  $x$ .

$Y$  = the yield per die, which is the probability of 0 failures for a die that has  $\lambda$  expected failures. This can be computed using Poisson's formula like:

$$Y = e^{-\lambda}$$

### Calculating Defect and Cumulative Defect Data

#### Defect Data Function

The defect data function is described as:

$$DSD(x) = C_D * 1/x^p \text{ for } x \geq x_0$$

where

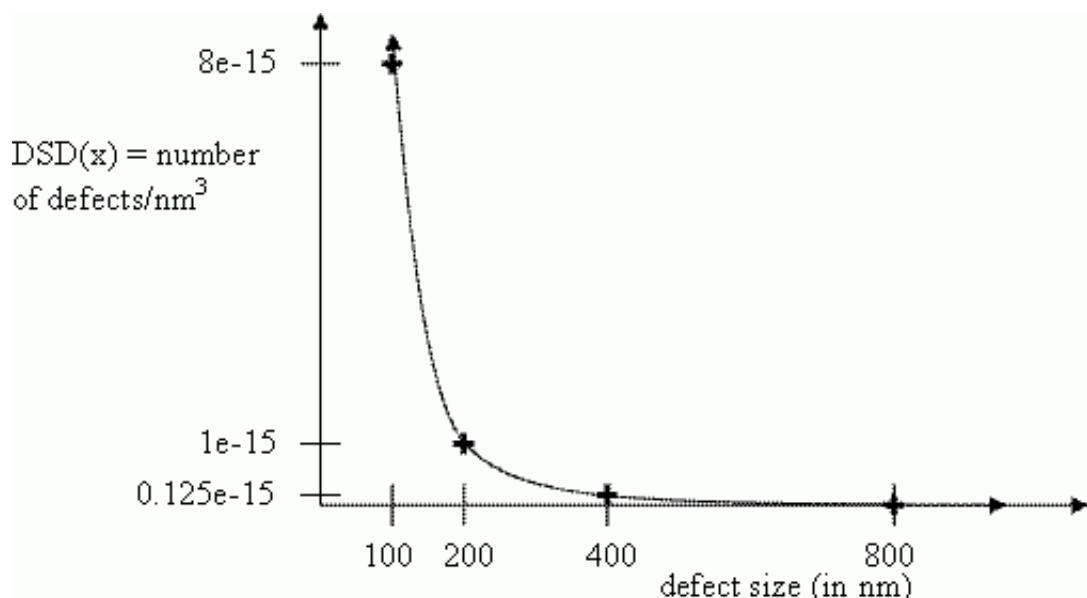
$C_D$  is a constant related to the total number of defects per  $\text{cm}^2$ .

$p$  is typically 3, which means the number of defects for a given size  $x$  falls off as  $1/x^3$ .

The units of  $DSD(x)$  are most naturally expressed as defects per  $\text{cm}^2$  per nm of defect size.

**Note:** Units are converted to defects per  $\text{nm}^2$  per nm or defects/ $\text{nm}^3$  in the data below and in the defect data table entries.

The following graph shows a typical  $DSD(x)$  function:



You can derive the complete function by giving various  $DSD(x)$  values for different  $x$  values in the yield technology file.

### Cumulative Defect Data Function

The cumulative defect data function is described as:

$$CDSD(x) = \int_x^{\infty} DSD(x) dx$$

where

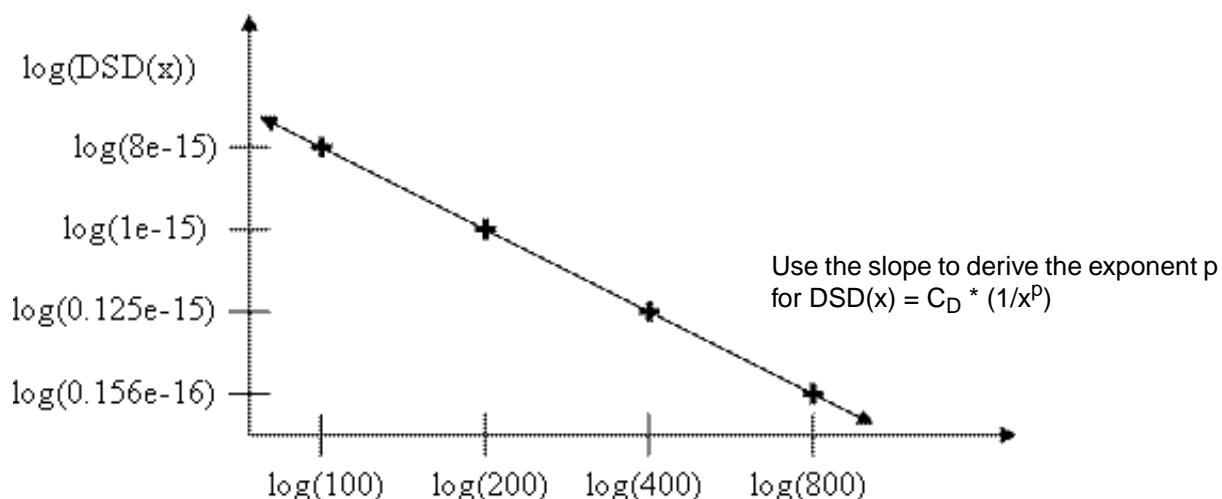
$CDSD(x)$  is the total number of defects greater than or equal to  $x$ .

### Deriving the Functions from Each Other

The  $DSD(x)$  and  $CDSD(x)$  functions can be derived from each other if you know the exponent  $p$ . You can estimate  $p$  from the data in the yield technology file. Solving the integral give the following formula:

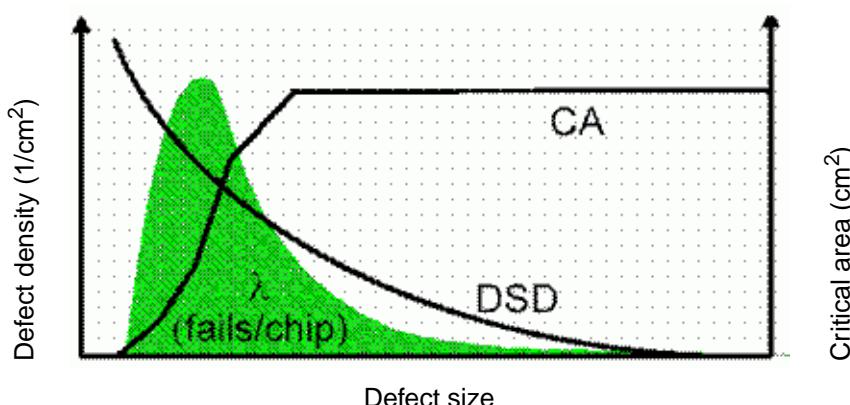
$$CDSD(x) = (1/(p-1)) * C_D * 1/x^{(p-1)} = 0.5 * C_D * 1/x^2 \text{ for } p = 3$$

The exponent  $p$  is derived from the slope of the log of the values shown in the figure below:



If only one value is given in the table, an exponent of 3 is assumed for the defect data function and an exponent of 2 is assumed for the cumulative defect data function.

The following graph shows the relationship between the defect size, defect density, the critical area, and the chip failure rate.



## Cost Formulas

Yield costs are determined using the following formulas, where

$Y$  = the probability the chip works (the yield)

$P_i$  = the probability of failure of one feature  $i$  (for example, a cell, a via, a  $10 \mu$  minimum-width wire)

$C_i = -\ln(1-P_i)$  = the cost of feature  $i$  (higher costs have worse yield)

$C_T = \text{total cost} = \sum C_i$

For  $N$  features on the chip:

$$Y = (1-P_1) * (1-P_2) * \dots * (1-P_N)$$

Taking the  $\ln$  of both sides gives:

$$\ln(Y) = \ln((1-P_1) * (1-P_2) * \dots * (1-P_N)) = \sum \ln(1-P_i) = -\sum C_i = -C_T$$

so

$$Y = e^{-C_T}$$

and

$$\text{Yield loss} = 1-Y$$

For values of  $P_i \ll 1$ , which is true for these types of probability values, it is useful to remember the following approximation:

$$-\ln(1-P_i) \approx P_i$$

---

## Creating An Initial Floorplan Using Masterplan

---

- [Overview](#) on page 1158
- [Masterplan Automatic Floorplanner Flow](#) on page 1159
- [Data Preparation](#) on page 1161
  - [Selecting Seeds](#) on page 1161
- [Importing the Design](#) on page 1167
- [Setting Masterplan Global Parameters](#) on page 1168
- [Creating an Initial Floorplan](#) on page 1168
- [Creating Floorplan for Hierarchical Design](#) on page 1169
  - [Macro placement](#) on page 1170
  - [Full-chip Floorplan](#) on page 1171
  - [Power-Domain Aware Floorplan](#) on page 1172
- [Creating Multiple Alternative Floorplans](#) on page 1174
- [Analyzing the Floorplan](#) on page 1174
- [Adjusting Macro Placement](#) on page 1176
  - [Manual Macro Adjustment](#) on page 1176
  - [Masterplan Macro Adjustment](#) on page 1176
  - [Marking Refinement Steps](#) on page 1180
  - [Restoring Refinement Steps](#) on page 1181
- [Saving the Floorplan](#) on page 1181

## Overview

Masterplan automatic floorplanner is a set of natively-integrated automatic floorplan capabilities that can create a quick, prototype floorplan. Given a gate-level netlist and design physical boundary, Masterplan can analyze the signal flow and generate a floorplan that includes automatic module and macro placement for large chips.

By default, Masterplan takes timing constraints into account during floorplan generation. The advantage of using Masterplan is that you can quickly create multiple alternative floorplans. You can then test the floorplans to find the one that gives you the best placement and routing results, and use it as a starting point for making the final floorplan.

Ideal designs for use with Masterplan are:

- Designs with hierarchical logical netlists
- Large designs that contain more than 50 hard macros
- Designs in which no more than 50 percent of the chip area is made up of hard macros

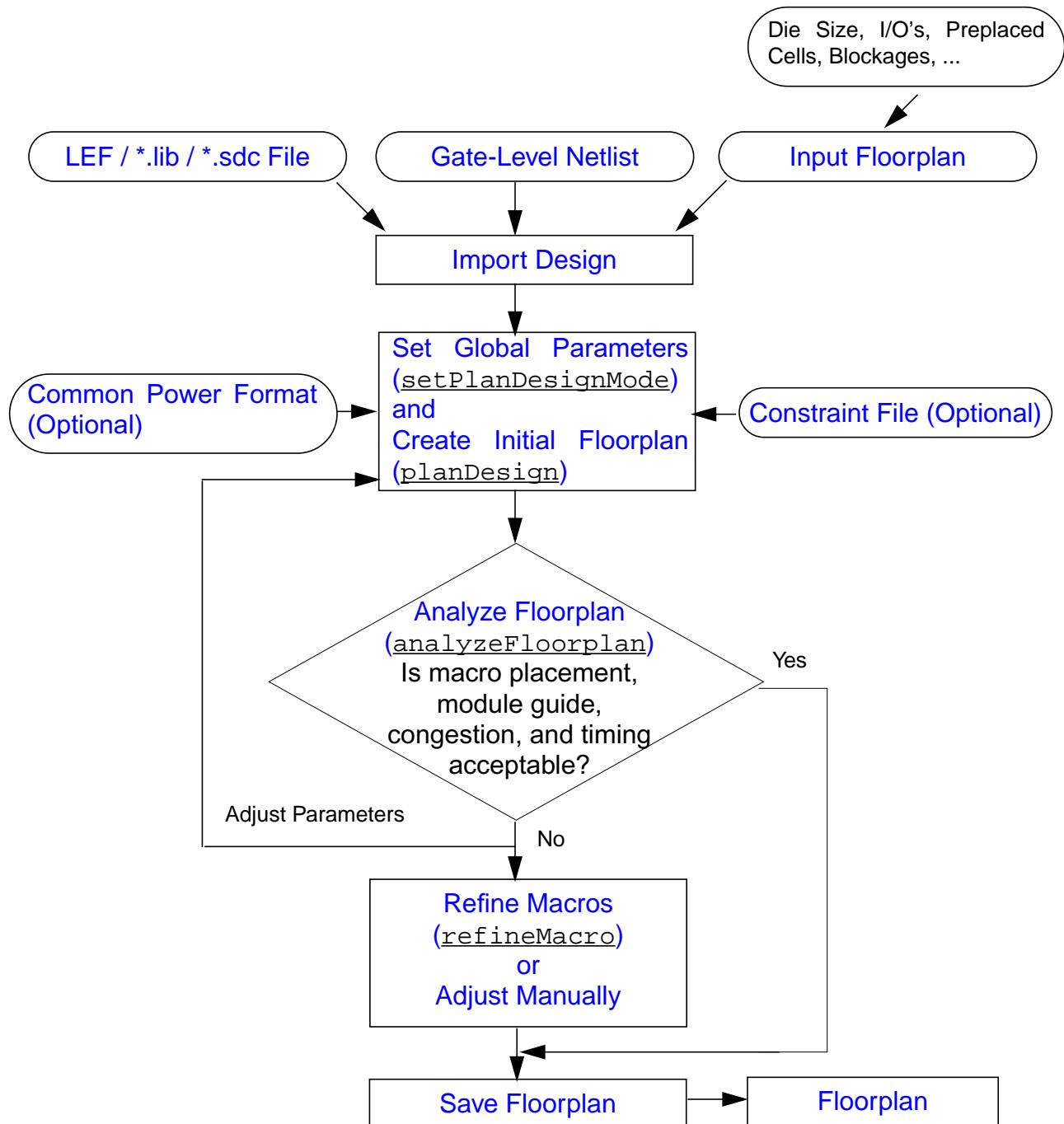
## **Masterplan Automatic Floorplanner Flow**

Figure 36-1 on page 1160 shows the typical task flow for using Masterplan automatic floorplanner to create an initial floorplan:

# Encounter User Guide

## Creating An Initial Floorplan Using Masterplan

Figure 36-1 Masterplan Flow



## Data Preparation

The Masterplan automatic floorplanner uses the following input files when creating floorplans:

- Gate-level netlist
- LEF file that contains models for standard cells, hard macros, I/O pads
- Floorplan file (or DEF file) that defines:
  - Die size and core area
  - Fixed I/O pad and pin locations
  - Any preplaced hard macros or placement blockages
  - Power stripes and rings (optional)

**Note:** Cells with direct I/O connections should be preplaced around the chip boundary before running Masterplan. These include Jtag cells, boundary scan cells, and I/O interface logic cells. Masterplan automatically ignores high fanout global nets, such as clock, reset, scan, and enable. Use the following commands to preplace Jtag and boundary scan cells:

[specifyJtag](#)  
[placeJtag](#)

- Constraint file (optional)

## Selecting Seeds

Seeds are typically design blocks that represent functional units. For example, USB controllers, PCI controllers, Cache sub blocks, and FPUs make good seed candidates. For most designs, hierarchical modules make good seed candidates if they represent the functional units in the design. Hierarchical modules (soft seeds) are not the only candidates for seeds; a seed can also be a hard macro (hard seed) or an instance group made up of strongly connected instances.

Masterplan analyzes the data flow between seeds (design blocks) based on their connectivity and their location. Masterplan then places the seeds in the core area in a way that minimizes wire length and congestion. You should select seeds that identify the data flow in the design so that when `planDesign` is run, Masterplan can optimize your data flow to reduce overall interconnect and placement.

## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

---

Seed selection and seed location also influence how Masterplan places hard macros. Seeds eventually become module guides in the Masterplan generated floorplan.

There are two methods for selecting seeds:

- Automatic seed selection
- User-Specified seed selection

#### **Automatic Seed Selection**

By default, Masterplan selects seeds using the following methods in the specified order:

1. Chooses modules that fit an internally calculated size range
2. Chooses large hard macros as seeds
3. Groups small modules or single standard cells to fit an internally calculated size range

However, you can force automatic seed selection to favor a certain constraint during the seed selection process by specifying one of the following setPlanDesignMode parameters:

-setSeedHierLevel, -numSeed, or -seedSize.

For example, some designs might not require Masterplan to look through their entire hierarchies to select seeds. Going down two to three levels from the top in the hierarchy might be enough to select good seeds. In this case, you can force Masterplan to favor logic level when selecting seeds by typing the following commands:

```
setPlanDesignMode -setSeedHierLevel 3  
planDesign
```

**Note:** After automatic seed selection, Masterplan writes out a seed file called MP\_seed under the current directory.

#### **User-Specified Seed Selection**

You can provide your own choice of seeds to influence Masterplan macro placement and module guide generation. You can select seeds based on a module's function, size, connectivity, or any combination of the three.

For most designs, the optimal number of seeds to select is between 20 to 50. You should not select fewer than 10 seeds, or more than 100. You do not have to provide a complete list of seeds for the design. Masterplan will select the remainder of the seeds required for the design size, and will attempt to select seeds that match the size of the ones you selected.

## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

---

To provide Masterplan with your choice of seeds, you must create a seed section in the constraint text file. This seed section lists the names of the hierarchical modules, hard macros, and instance groups that you picked as seeds. You also can specify utilization values for individual modules or instance groups.

You can specify a seed section using the following format:

```
BEGIN SEED
  seed_name [utilization_value]
END SEED
```

Where:

*seed\_name*                      Specifies the name of the hierarchical module, hard macro, or instance group that you want to choose as a seed.

*utilization\_value*      Specifies the utilization value for the specified module or instance group.

For example:

```
BEGIN SEED
  Module-A/A1/abc      0.75
  Module-B/B2/xyz
  ...
  HM2
  Module-C/C1/HM2
  ...
  instGrp-1          0.85
  instGrp-2
  ...
END SEED
```

### Creating a Seed Section In the Constraint File

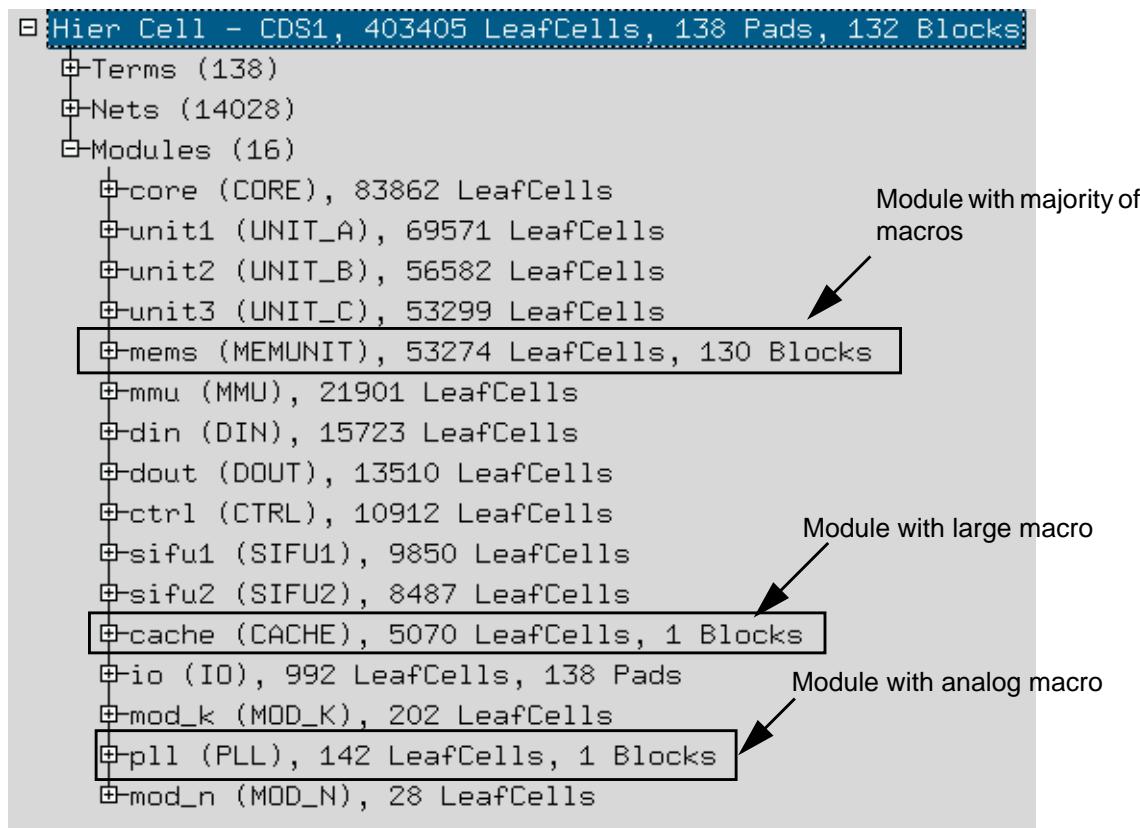
The purpose of creating a seed section in the constraint file is to select seeds that will identify the data flow of your design so as to generate better floorplan results. The following steps show you one way to determine which modules in your design to select as seeds.

1. Import your design.

## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

#### 2. Open the Design Browser and examine the design hierarchy.



Consider the following points when examining the modules:

The relationship between modules

Look at the relationships between modules to find which modules best identify the data flow in your design. The block diagram for your design can give you a good idea as to which modules these are.

The size and area of modules

- Do not select seeds that differ widely in size. The size of a seed is based on an area estimate. The size difference between the largest seed and the smallest seed in the design should be no more than 10x.
- Select large hard macros with sizes greater than 1/4 of the core area as seeds, except if they are marked as FIXED.
- Only select small modules or single standard cells as seeds if you know they are important to the global signal flow. At the end of the seed selection process,

## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

Masterplan groups small modules and individual standard cells into instance group seeds based on existing seed size and connectivity.

- Do not select I/O modules that include pad cells as seeds because I/O pads should already be pre-placed.
- The number of macros in a module

If your design hierarchy includes a single module that contains almost all of the macros in the design, do not select it as a seed. Instead, examine its sub modules for seed candidates (step 3).

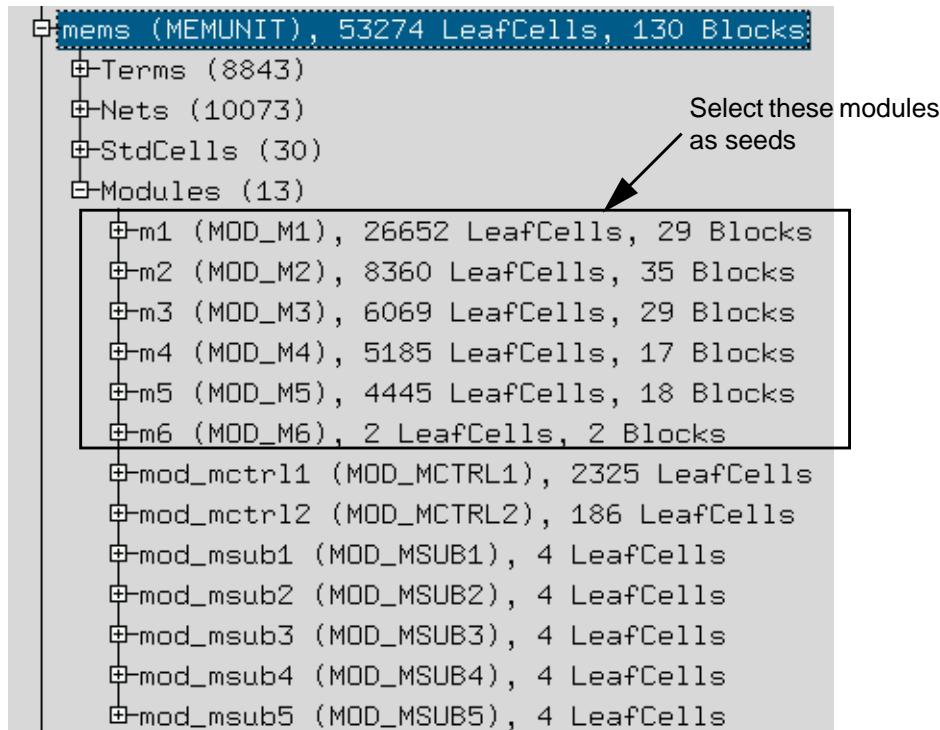
- Any special modules in the design

For example, look for modules that include a very large macro, or an analog macro.

### 3. Examine the sub modules of a module for seed candidates.

In [Figure 36-2 on page 1165](#), you should select the modules at this level as seeds because it will produce better grouping for macro placement.

**Figure 36-2**

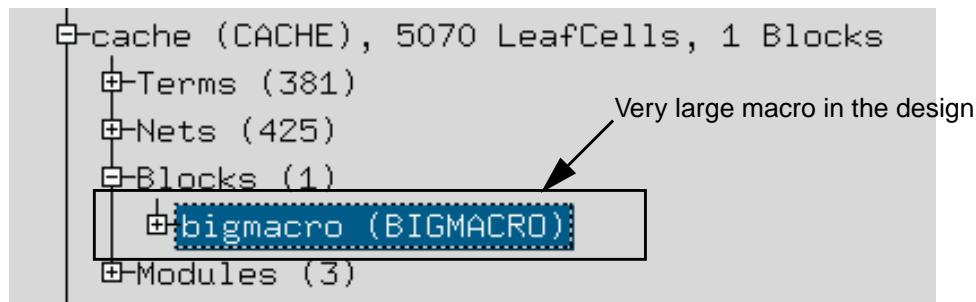


In [Figure 36-3 on page 1166](#), you should select this macro as a seed because it is larger than 1/4 of the core area.

## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

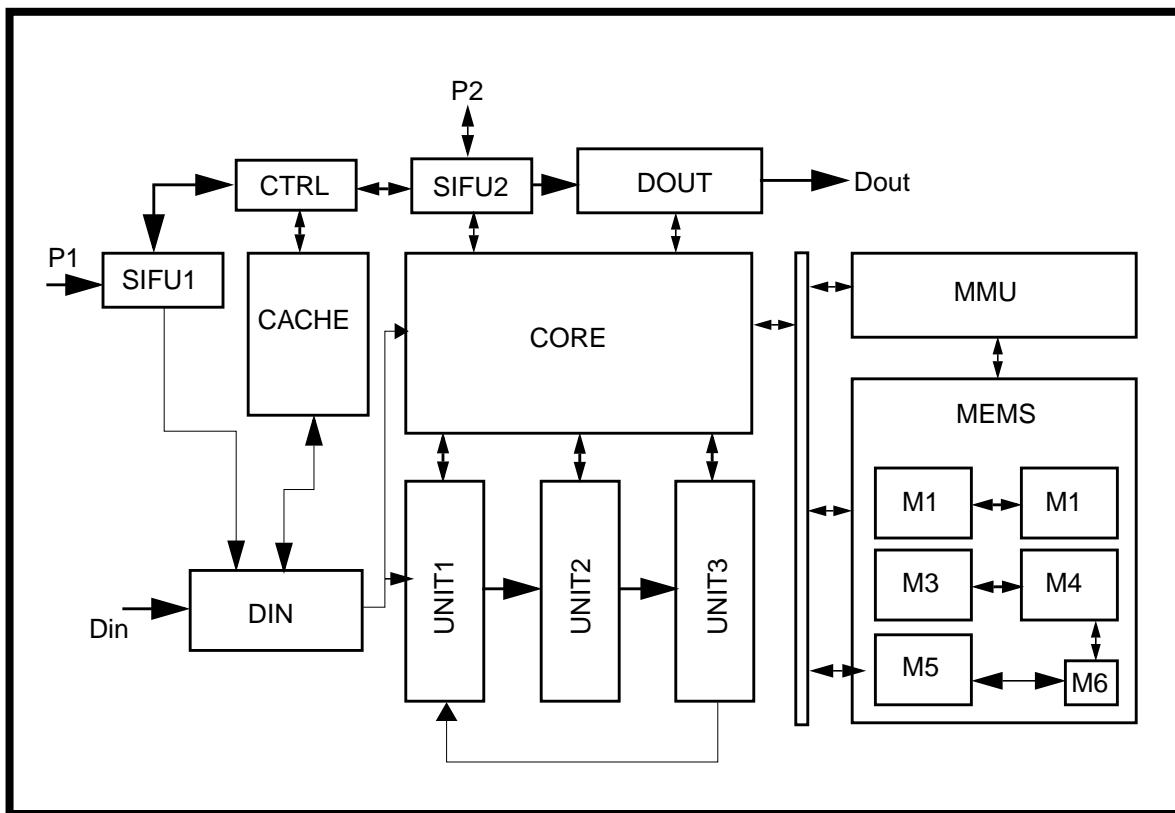
**Figure 36-3 Large Macro**



4. Create a seed section.

For example, [Figure 36-4 on page 1166](#) shows the block diagram for the design.

**Figure 36-4 Design Block Diagram**



Based on this diagram:

## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

---

1. Examine the module `mems` because it contains the majority of the macros in the design. Select seeds from its sub modules in order to get better macro placement ([Figure 36-2 on page 1165](#)).
2. Examine the module `cache`, and select the macro `bigmacro` because it is very large ([Figure 36-3 on page 1166](#)).
3. Do not select the modules named `mod_k` and `mod_n` as seeds because they are so small that they do not show up in the block diagram.
4. Do not select the module named `io` because it includes I/O pad cells that have already been pre-placed.

The resulting seed section is as follows:

```
core          #Hinst: good for data flow
unit1         #Hinst: good for data flow
unit2         #Hinst: good for data flow
unit3         #Hinst: good for data flow
mmu          #Hinst: good for data flow
din           #Hinst: good for data flow
dout          #Hinst: good for data flow
ctrl          #Hinst: good for data flow
siful         #Hinst: good for data flow
sifu2         #Hinst: good for data flow
mems/m1       #Hinst: good for data flow and macro grouping
mems/m2       #Hinst: good for data flow and macro grouping
mems/m3       #Hinst: good for data flow and macro grouping
mems/m4       #Hinst: good for data flow and macro grouping
mems/m5       #Hinst: good for data flow and macro grouping
mems/m6       #Hinst: good for data flow and macro grouping
pll           #Hinst: maintain for analog macro
cache/bigmacro #Inst: maintain for big macro
```

## Importing the Design

1. To import a design, type the following command:

```
loadConfig design_name.conf 1
```

2. To load a floorplan file, type the following command:

```
loadFPlan design_name.fp
```

## Setting Masterplan Global Parameters

Use the `setPlanDesignMode` command to specify Masterplan global parameters. These parameters are used by the `planDesign` command every time you call it to create a floorplan.

You can set the following parameters:

- The method to use for selecting seeds (See [Automatic Seed Selection](#) on page 1162)
- Hard macro spacing
- Target utilization for floorplan guides
- Place macros close to the chip boundary
- Control macro placement status and color
- Flow control options

## Creating an Initial Floorplan

- Type the following command to generate an initial floorplan:

```
planDesign [-constraints constraint_file]
```

In general, you use Masterplan to create multiple alternative floorplans, which you can then compare. Run the `planDesign` command the first time using automatic seed selection, and analyze the results. For each subsequent run, modify the seed section of the constraint file that was created automatically (`MP_seed`) with several different seed options. Then specify the modified constraint file when you run `planDesign` to produce different results.

By default, Masterplan takes timing constraints into account during floorplan generation, if timing libraries (.lib) and SDC constraint files are loaded in the design. Masterplan will not perform timing aware floorplanning if either the timing library or constraint file is not loaded.

When specified, Masterplan performs the following internal functions in order:

- Selects the floorplan objects (seeds) to be placed

Masterplan performs seed selection either automatically, or using the seed section of the specified constraint file, then clusters the netlist.

- Places the seeds

## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

---

Masterplan calls the placer to place the seeds of zero size in order to find the best relative locations for the seeds. Masterplan gives I/O nets a higher weight (100), and ignores high fanout nets (0 weight).

- Refines the seeds

Masterplan adjusts seed size, location, and aspect ratio to reflect real module and macro size, and to reduce seed-to-seed overlap area. The tool preserves the relative location of seeds throughout refinement.

- Places the macros

Masterplan firsts groups and packs hard macros from the same seed that are of the same type, or are similar in size and aspect ratio. Macro packs furthest from the center of the core area are moderately pushed out toward the chip or block boundary.

Masterplan determines a macro's location and orientation based on a combination of many factors, including parent seed location, size, aspect ratio, chip or core aspect ratio, wire length and congestion optimization, macro pin layer, and the preferred routing layer direction of the layer. Masterplan calculates the space between adjacent macros based on pin density, track estimation, and metal layer pitch. You can also specify a spacing value for the tool to use instead.

After placement, macros are marked `PLACED` in the database.

## Creating Floorplan for Hierarchical Design

Given a hierarchical top-level floorplan that contains fences or power domains that represent predefined partitions ([Figure 36-5 on page 1170](#)), Masterplan can be used for:

- Macro placement

Places the macros within the predefined partition fences.

If no predefined fences or power domains are present in heirarchical top-level floorplan, Masterplan can be used for:

- Full-chip Floorplan

Creates fences and places macros at chip level.

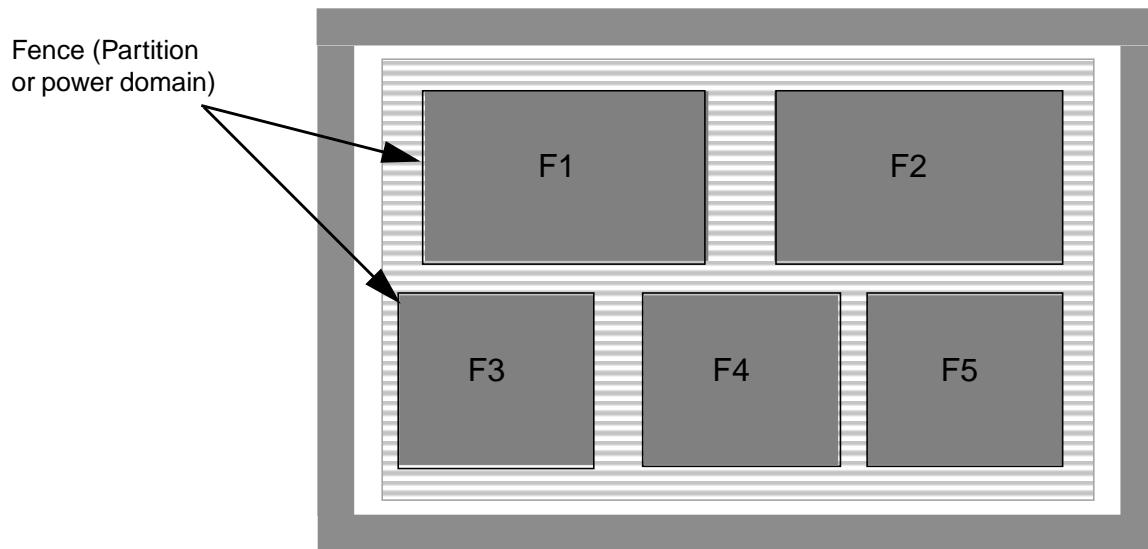
- Power-Domain Aware Floorplan

Reads CPF, creates power domains, and places macros.

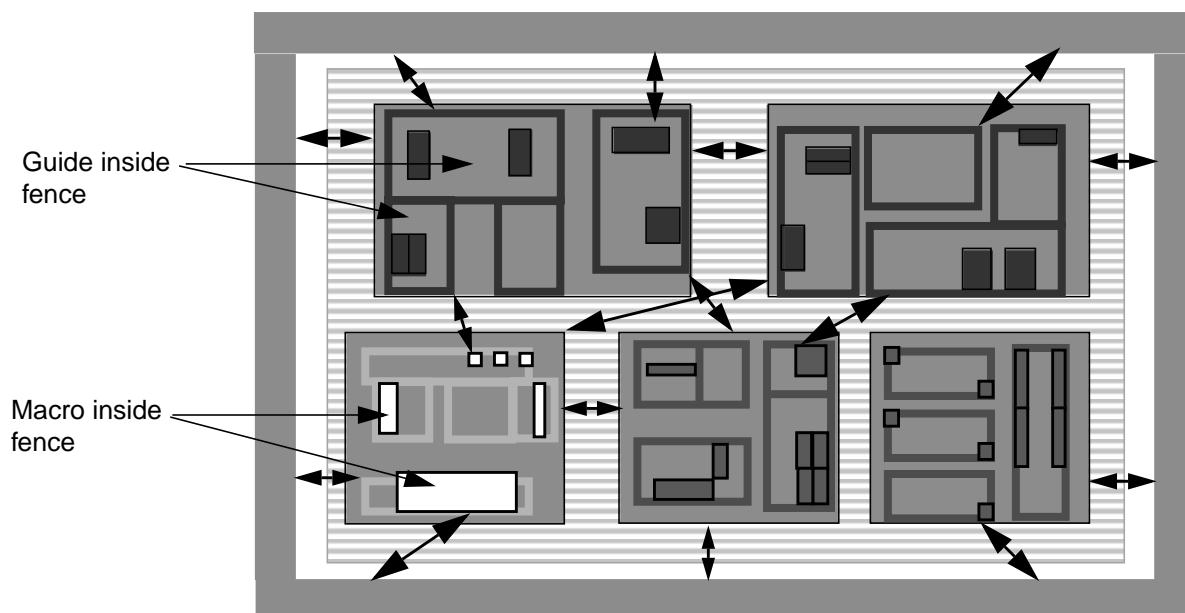
## Macro placement

Masterplan places hard macros, and generates guides for the submodules within the fence boundary and also on the top level ([Figure 36-6 on page 1170](#)), considering the global connectivity.

**Figure 36-5 Hierarchical Top-Level Floorplan**



**Figure 36-6 Masterplan Result**



## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

Note the following guidelines when performing hierarchical floorplanning:

- If you specify a seed section in a constraint file (-constraints), the fence module should not be specified in it as a seed, because Masterplan does not touch any fences in the floorplan.
- The macro placer looks at nets globally, but does not see hierarchical ports on partition boundaries because the ports have not been assigned yet.
- In order for the macro placer to produce good results, you must set appropriate fence utilization and aspect ratio values in the input floorplan by manually stretching or reshaping them.

## Full-chip Floorplan

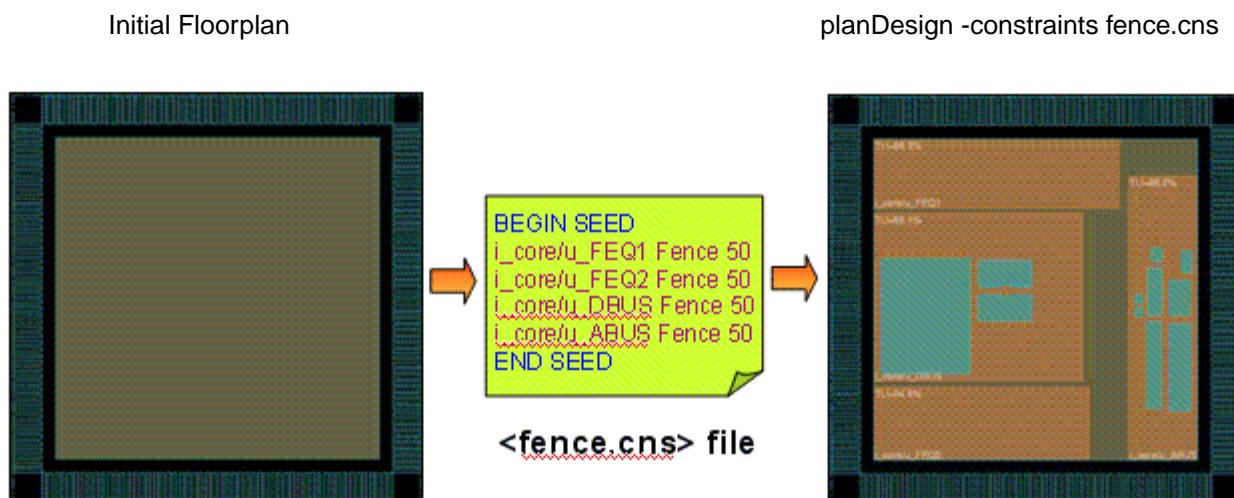
Floorplanning a top-level hierarchical design or creating power domain physical boundary for a MSV design requires rectilinear module fence generation. Masterplan generates a quick top-level initial floorplan for such designs.

For Masterplan to create fences and place macros at chip level, you must specify the fence module(s) in the constraint file using the planDesign command.

- `planDesign -constraints constraint_file`

Specify a single-module fence in the Seed Section and multiple-modules fence in the Relative Constraint Section.

The results of the `planDesign` command is as follows:



## Power-Domain Aware Floorplan

The `planDesign` command is power-domain aware and supports placement of power domains. If your design uses the Common Power Format (CPF) flow, you can run the `planDesign` command to automatically create fences around the power domains and place the power domains along with other hard blocks in the design, without having to create any seed definitions in the constraint file. The `planDesign` command honors power domain attributes, such as the minimum gap (`modifyPowerDomainAttr -minGaps`) if they are defined before running the command. You can also perform congestion aware power-domain placement by specifying `setPlanDesignMode -congAware true`, which is also applicable for standard cell placement.

### Use Flow

1. Load the configuration file.

```
loadConfig design_name.conf 1
```

2. Load the floorplan.

```
loadFPlan design_name.fp
```

3. Specify the floorplan dimensions.

```
floorPlan -site core
```

4. Load the Common Power Format file. To read-in the CPF that contains CPF commands, use the following command:

```
loadCPF design.cpf
```

This command reads the file and performs lint, parsing, and semantics checking. This command does not execute any of the command within the CPF file.

5. Commit the CPF file.

```
commitCPF
```

This command executes the CPF commands loaded by `loadCPF`. Running this command does the following:

- Creates power domains
- Creates logical power/ground net connections
- Specifies timing libraries for power domains
- Inserts shifter cells and isolation cells
- Replaces regular registers with state-retention (SRPG) registers

6. Modify the power domain attributes.

## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

```
modifyPowerDomainAttr -minGaps T B L R
```

The minimum gaps attribute, `-minGaps`, defines the distance, in microns, that must be reserved outside the power domain boundary edges for power routing.

7. Specify congestion aware placement of power domains.

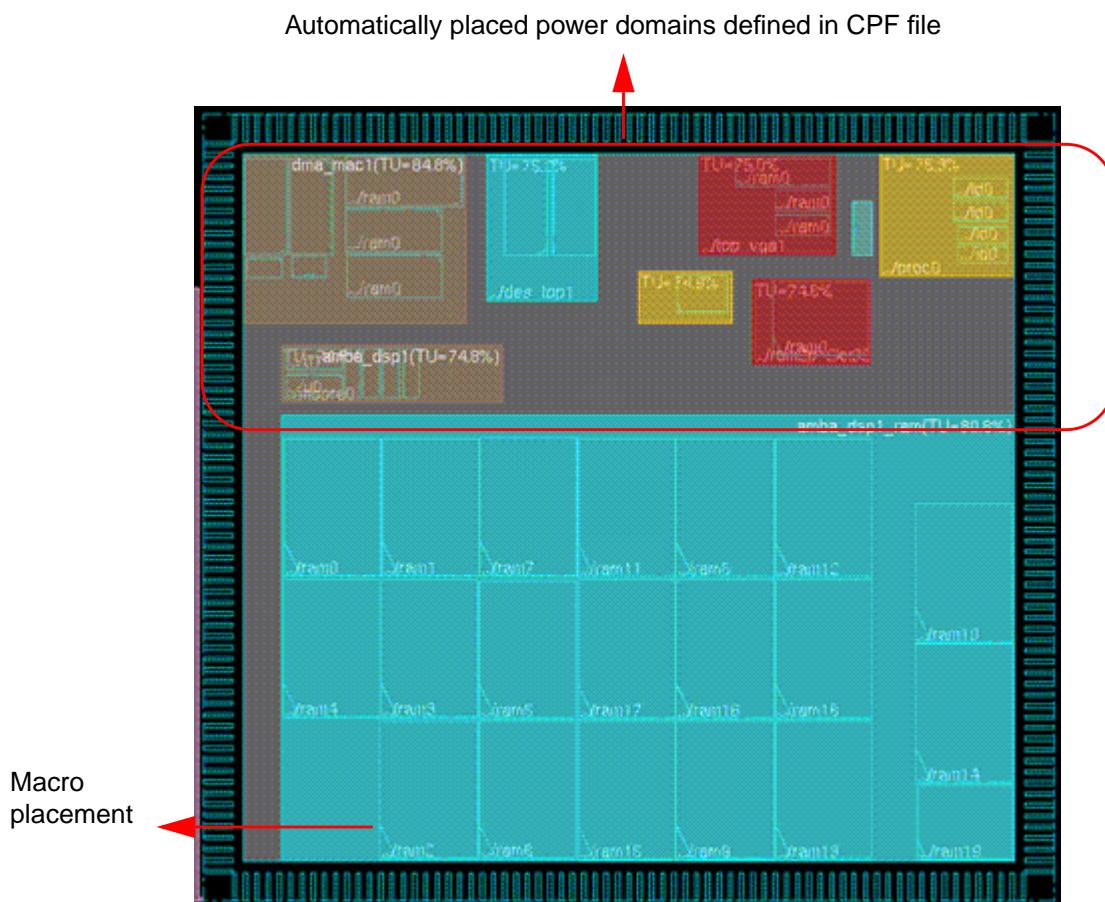
```
setPlanDesignMode -congAware true
```

8. Place power domains and macros.

```
planDesign
```

The following example describes the output of the `planDesign` command which has automatically placed 8 power domains and hard macros in the design:

Automatic placement of power domains and hard macros for a sample design with 8 power domains



## Creating Multiple Alternative Floorplans

Use the `multiPlanDesign` command to create multiple alternative floorplans by running different variations of the `setPlanDesignMode` and `planDesign` commands. Masterplan then analyzes the resulting floorplans, and ranks their usability using estimated wire length, congestion, and timing criteria.

The `multiPlanDesign` command generates floorplans based on the parameter settings you specify. For some parameters, you can instruct Masterplan to always perform the functionality (`on`), or to perform or not perform the functionality on floorplans in a predefined order (`on_off`).

You can run `planDesign` jobs sequentially on one machine, or in parallel on multiple host machines. To run `multiPlanDesign` in parallel, you must first use the `setDistributeHost` and `setMultiCpuUsage` commands to set up the configuration for the distributed processing. For example:

```
setDistributeHost {-local | -rsh -add {machine1 machine2}}
setMultiCpuUsage -numHosts 2
multiPlanDesign -autoTrials 2
```

Floorplan rankings are automatically displayed in a separate results form after all of the `planDesign` jobs are completed. Additionally, Masterplan saves a text report of the ranking results to a user-specified file.

If you run this command on a master instance blackbox with non-R0 orientation, it automatically converts the new orientation to R0. For more information, see [Handling of Blackboxes with Non-R0 Orientation](#) in the “Partitioning the Design” chapter of the *Encounter User Guide*.

## Analyzing the Floorplan

After generating a floorplan, analyze the results in order to assess whether improvements are needed:

- Run the `analyzeFloorplan` command to analyze the floorplan, and report basic design information.
- Visually check the floorplan for macro placement issues, such as:
  - Macros placed far away from their related modules or connected I/Os.
  - Too many wire cross-overs among modules, macros, and I/Os.
  - Too many macro-to-macro, or module-to-module, overlaps.

## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

---

- Too many dead areas in the design (that is, holes in the middle of placed macros).
- Inadequate macro orientation or spacing.

**Note:** Use the Preferences form and the display control panel on the Encounter main window to control the information displayed when visually checking the floorplan.

- Read the log file to see Masterplan intermediate results, and error and warning messages that might indicate to more serious issues.

Check the following items in the log file:

- Seed Report
  - Do the selected seeds match those specified in the seed section of the constraint file?
  - How many seeds exist in the design after clustering? Are there too many seeds, or too few?

The following example shows a typical seed report in the log file:

```
----- Clustering summary -----
*** Clustered Preplaced Objects ***
# of preplaced io is: 0
# of preplaced hard macro is: 1
# of preplaced standard cell is: 0

*** Clustered FPlan Seeds ***
# of hard macro seeds is: 1
# of standard cell seeds is: 0
# of soft module seeds is: 9
# of instance group seeds is: 1

*** Normal Netlist Clustering ***
# of clustered instances is: 0
----- End of Clustering summary -----
```

- Macro Placement
  - Read the options reported back by the macro placer.
  - Are there any reported overlaps between PLACED macros?
  - What total wire length did the macro placer report?

## Adjusting Macro Placement

Typically after generating a floorplan, macro adjustment is required to improve the layout. There are two ways you can refine the macro placement:

- Manually adjust the macros using interactive commands
- Use `refineMacro` to adjust the macro locations

### Manual Macro Adjustment

The Encounter main window includes widgets that allow you to interactively adjust macro placement. Actions you can perform include move, align, flip, rotate, redo, and undo. For more information on the toolbar and tool widgets, see “[Toolbar Widgets](#)” and “[Tool Widgets](#)” in the *Encounter Menu Reference*.

You can also use the Edit Floorplan submenu on the Floorplan menu to perform many of the same actions. For more information, see “[Edit Floorplan](#)” in the *Encounter Menu Reference*.

### Masterplan Macro Adjustment

You can also use the `refineMacro` command to adjust the placement of macros in the floorplan. By default, the command performs global incremental macro adjustment. You can also adjust the placement of specific macro packs, or all of the macros in a specific area of the design.

#### Adjusting the Placement of a Specific Macro Pack

1. Select the macro pack you want to adjust in the main window. A macro pack is a set of macros from the same seed that have similar sizes and aspect ratios and have been grouped together. You can select one macro to select the entire pack.

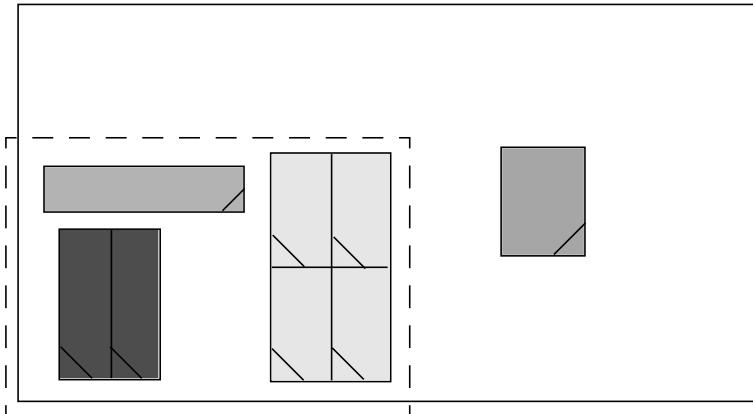
## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

---



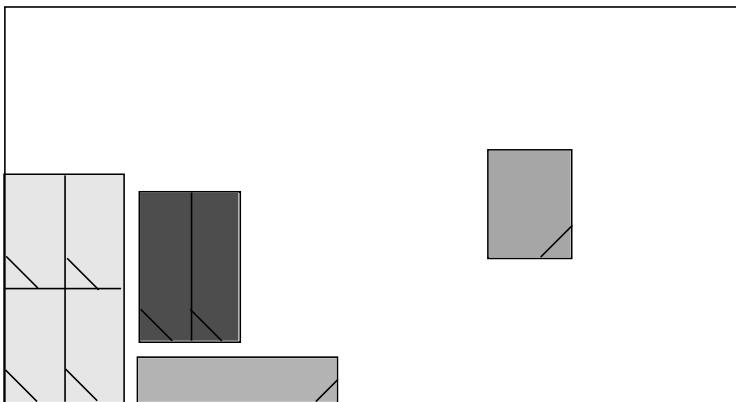
Macro pack information is stored temporarily in the data base. If you quit the Encounter session, the information is lost.



2. Type the following command:

```
refineMacro -permutePack
```

Masterplan adjusts the macros' locations to reduce empty area between them.

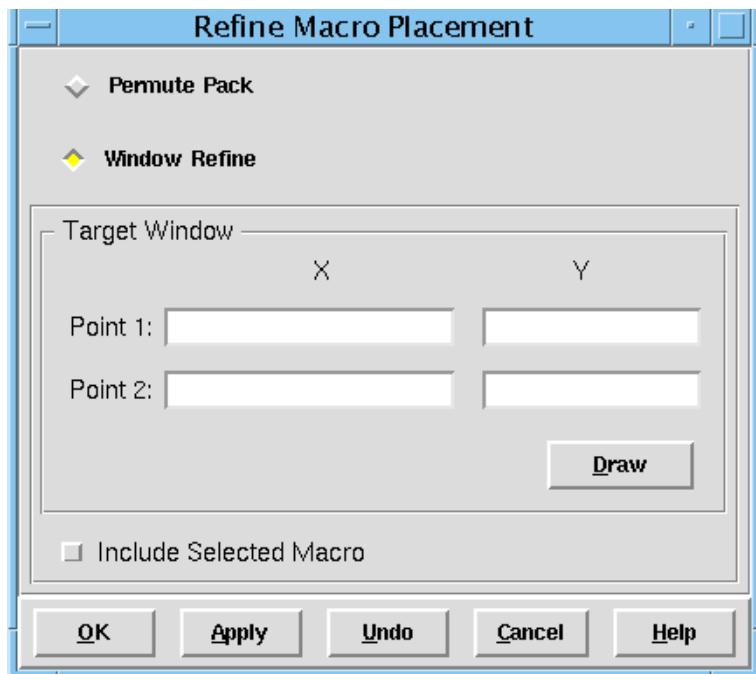


## Encounter User Guide

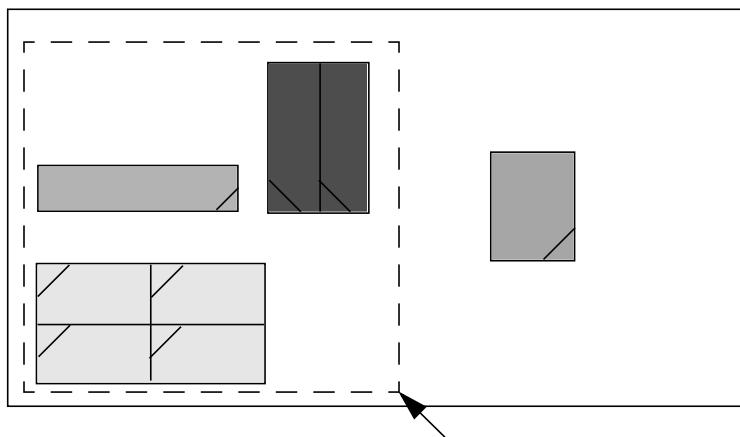
### Creating An Initial Floorplan Using Masterplan

#### Adjusting Macro Placement Within a Specified Area

1. Choose *Floorplan – Automatic Floorplan – Refine Macro Placement.*



2. Select the *Window Refine* option.
3. Click the *Draw* button to enable drawing mode.
4. Hold down the left mouse button and drag the cursor to draw a box around the area of the design in which you want to adjust macro placement.

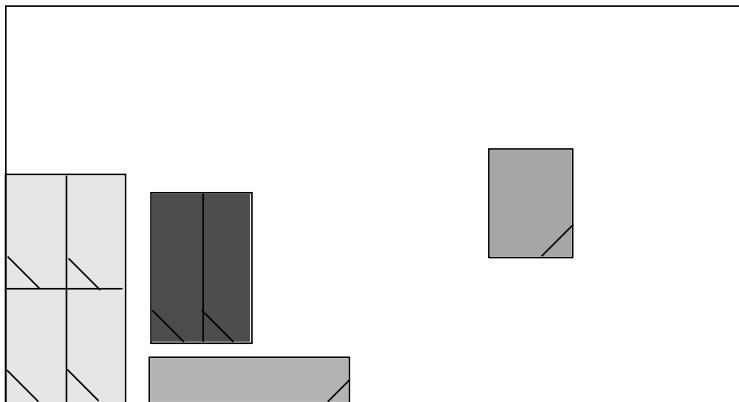


## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

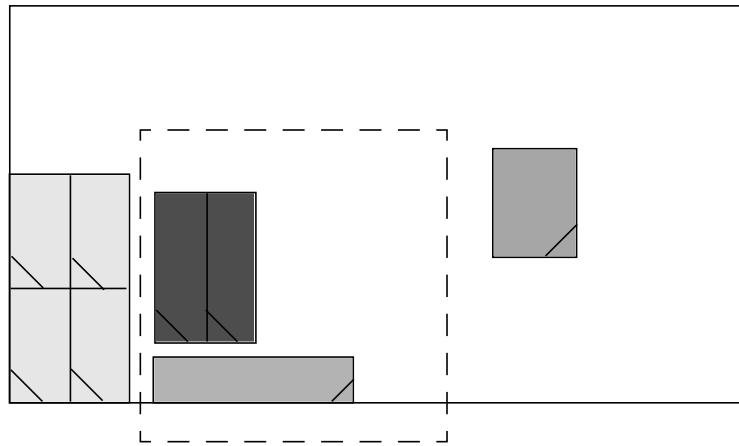
---

5. Click *Apply* to perform the adjustment.



You can also move selected macros to a specified area and adjust them along with any macros already within the area.

1. Select the macros you want to move in the main window.

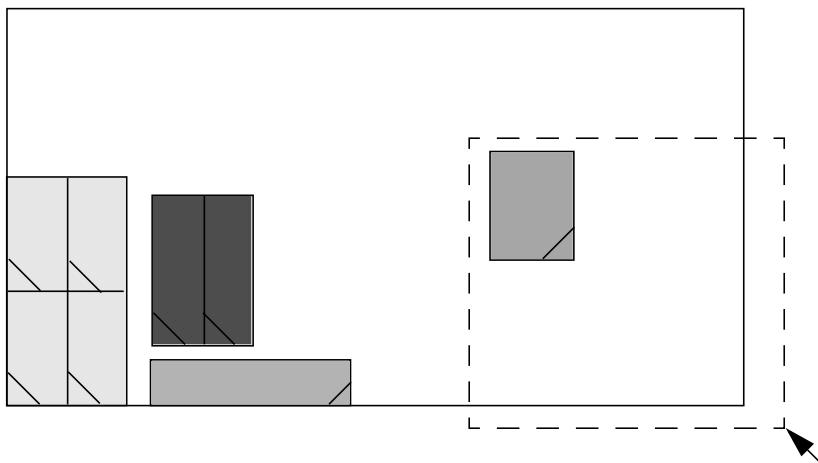


2. Open the Refine Macro Placement form, then select *Include Selected Macro*.

## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

3. Click the *Draw* button and draw a box around the area of the design to which you want to move the selected macros.



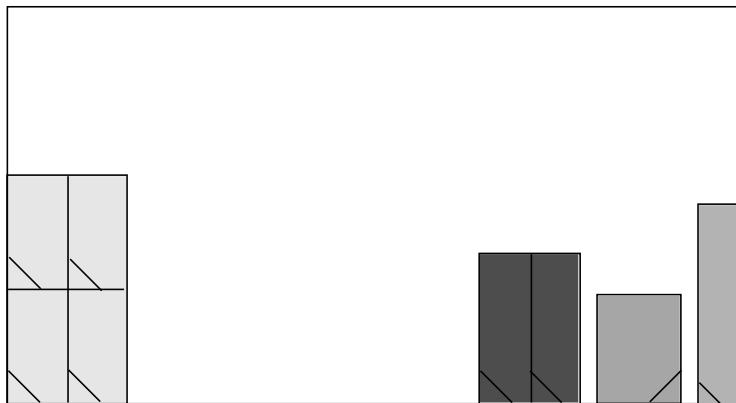
Alternatively, if you know the coordinates of the area to which you want to move the macros, you can perform one of the following steps:

- Specify the coordinates in the *Path1* and *Path2* X/Y text fields on the Refine Macro Placement form and click *Apply*.

- Type the following command:

```
refineMacro -selected -area l1x l1y urx ury
```

4. Click *Apply* to perform the adjustment.



#### Marking Refinement Steps

Masterplan considers each adjustment that you make to the floorplan a step. By default, Masterplan saves all incremental steps done with the `refineMacro` command and lists

## Encounter User Guide

### Creating An Initial Floorplan Using Masterplan

---

them in the log file with a number. The tool can store up to 1,000 steps in the database before overwriting them. Masterplan does not save steps done through manual refinement (such as move, rotate, and flip).

- To mark manual refinement steps (or any other important step that you want to save), type the following command:

```
refineMacro -markStep
```

Masterplan assigns a number to the mark (as well as a step number) and stores the step in the database. Masterplan saves up to 20 marks before overwriting them.

### **Restoring Refinement Steps**

You can also restore the design to a previous state during the refinement process.

- To return the design to the state it was in after the specified intermediate non-marked step, type the following command:

```
refineMacro -restoreStep step_number
```

- To return the design to the state it was in after the specified marked step, type the following command:

```
refineMacro -restoreMark mark_number
```

- To return the design to the state it was in after the previous adjustment, type the following command:

```
refineMacro -restoreStep -1
```

You can find the mark and step numbers listed in the log file.

### **Saving the Floorplan**

- To save the floorplan, type the following command:

```
saveFPlan fileName.fp
```

**Encounter User Guide**  
Creating An Initial Floorplan Using Masterplan

---

---

## Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

---

- [Overview](#) on page 1185
  - [Multi-Mode Multi-Corner Licensing](#) on page 1185
  - [Multi-Mode Multi-Corner Flow Support](#) on page 1185
- [Configuring the Setup for Multi-Mode Multi-Corner Analysis](#) on page 1186
  - [Creating Library Sets](#) on page 1187
    - [Editing A Library Set](#) on page 1188
  - [Creating Virtual Operating Conditions](#) on page 1188
    - [Editing A Virtual Operating Condition](#) on page 1189
  - [Creating RC Corner Objects](#) on page 1189
    - [Editing An RC Corner Object](#) on page 1190
  - [Creating Delay Calculation Corner Objects](#) on page 1190
    - [Editing A Delay Corner Object](#) on page 1191
  - [Adding A Power Domain Definition To A Delay Calculation Corner](#) on page 1192
    - [Editing A Power Domain Definition](#) on page 1192
  - [Creating Constraint Mode Objects](#) on page 1193
    - [Editing A Constraint Mode Object](#) on page 1194
    - [Entering Constraints Interactively](#) on page 1194
    - [Constraint Support in Multi-Mode and Multi-Mode Multi-Corner Analysis](#) on page 1195
  - [Creating Analysis Views](#) on page 1197

## **Encounter User Guide**

### Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

---

- [Editing An Analysis View Object](#) on page 1197
- [Setting Active Analysis Views](#) on page 1198
  - [Guidelines For Setting Active Analysis Views](#) on page 1198
  - [Changing the Default Active Analysis View](#) on page 1199
- [Checking the Multi-Mode Multi-Corner Configuration](#) on page 1199
  - [Changing How the MMMC Browser Displays Configuration Information](#) on page 1199
  - [Saving Multi-Mode Multi-Corner Configurations](#) on page 1200
- [Controlling Multi-Mode Multi-Corner Analysis Through the Flow](#) on page 1200
- [Performing Timing Analysis](#) on page 1202
- [Generating Timing Reports](#) on page 1203

## Overview

As feature sizes decrease, it becomes more difficult to determine the worst-case combinations of device corners (timing libraries and PVT operating conditions), RC corners, and constraint modes at which to validate timing requirements for a design. Multi-mode multi-corner analysis and optimization provides the ability to configure the software to support multiple combinations of modes and corners, and to evaluate them concurrently.

To analyze multiple combinations of modes and corners concurrently, a significant amount of data typically imported for a single analysis run must be replicated and expressed in a way that is easy to define and manage throughout the flow. Multi-mode multi-corner analysis uses a tiered approach for defining the required data. It allows top-level definitions (analysis views) that share common information to be specified by referencing the same lower-level objects. The active analysis views defined in the software represent the different design variations that will be analyzed.

Multi-mode multi-corner analysis also provides a methodology that allows multi-mode multi-corner optimization on a select set of analysis views, and timing signoff on a potentially large set of text criteria.

**Note:** To see this step in the design flow, see [Route the Design and Run Postroute Optimization](#) in the *Encounter Flat Implementation Flow Guide*.

### Multi-Mode Multi-Corner Licensing

Multi-corner analysis and optimization requires an XL or GXL license. Multi-mode analysis does not require an XL or GXL license, and is limited to only one delay calculation corner for setup analysis, and one delay calculation corner for hold analysis.

### Multi-Mode Multi-Corner Flow Support

Multi-mode multi-corner analysis can be used with flat implementation flows.

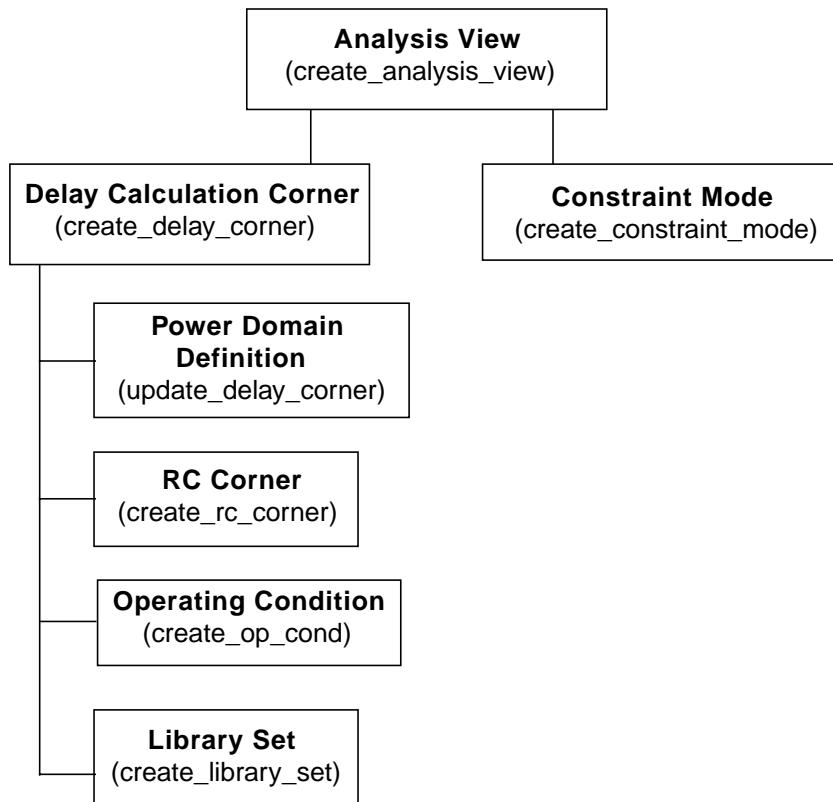
In a flat, block implementation flow, Encounter applications support multi-mode multi-corner analysis to different degrees. For information, see [“Controlling Multi-Mode Multi-Corner Analysis Through the Flow”](#) on page 1200.

## Configuring the Setup for Multi-Mode Multi-Corner Analysis

Multi-mode multi-corner analysis uses a tiered approach to assemble the information necessary for timing analysis and optimization. Each top-level definition (called an analysis view) is composed of a delay calculation corner and a constraint mode. The active analysis views defined in the software represent the different design variations that will be analyzed.

**Note:** Some Encounter min/max mode commands do not operate under multi-mode multi-corner analysis mode because they conflict with the multi-mode multi-corner use model. These commands include `setTimingLibrary`, `setOpCond`, `defineRCCorner`, `setRCFactor`, `loadTimingCon`, and `unloadTimingCon`.

**Figure 37-1 Hierarchical Analysis View Configuration**



## Creating Library Sets

Complex designs can require the specification of multiple library files to define all of the standard cells, memory devices, pads, and so forth, included in the design. Different library sets can be defined to provide uniquely characterized libraries for each delay corner or power domain.

Library sets allow a group of library files to be treated as a single entity so that higher-level descriptions (delay calculation corners) can simply refer to the library configuration by name. A library set can consist of just timing libraries, or it also can include cdb libraries to keep timing and signal integrity libraries in sync throughout a multi-corner flow.

The same library set can be referenced multiple times by different delay calculation corners.



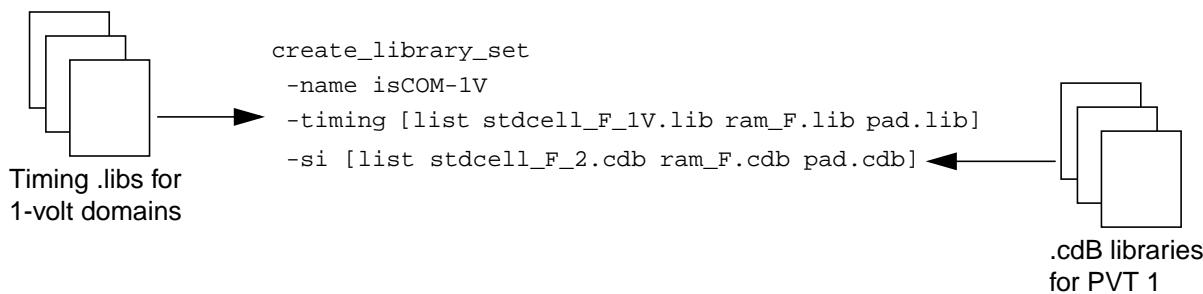
You must specify a set of timing libraries in the configuration file as part of the design initialization. Create your multi-mode multi-corner library sets as needed, to support all the delay calculation corners that will be included in the multi-mode multi-corner analysis.

The order in which you define timing libraries is important. The software considers the first library you specify in the list as the master library, with each successive library having a lower priority.

- To create a library set, use the following command:

[create\\_library\\_set](#)

The following figure shows the creation of a library set that associates timing libraries and cdb libraries with a nominal voltage of 1 volt with the library name `IsCOM-1V`:



## Editing A Library Set

To change the timing and cdb library files for an existing library set, use the following command:

update\_library\_set

You also can make changes to a library set using the Edit Library Set form:

- Choose *Design – Import Design*, click the *Advanced* tab, select *MMMC*, click the *MMMC Browse* button, and double click on the name of the library set you want to edit.
- or:
- Choose *Timing – Configure MMC*, and double click on the name of the library set you want to edit.



You can use the `update_library_set` command or the Edit Library Set form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

## Creating Virtual Operating Conditions

Generally in most user environments, the process, voltage, and temperature (PVT) point is specified by referring to a predefined operating condition definition in a specific timing library. The library operating condition provides the system with values for P,V, and T, and these then are used to calculate derating parameters and other aspects of the analysis. However, there are situations when there are no predefined operating conditions in the user timing libraries, or the pre-existing operating conditions are not consistent with the user's operating environment.

Instead of actually modifying the timing libraries to add or adjust operating condition definitions, you can create a set of virtual operating conditions for a library, to define a PVT operating point. These virtual operating conditions can then be referenced by a delay corner as if they actually existed in the library.

- To create a virtual operating condition for a library, use the following command:

create\_op\_cond

For example, the following command creates a virtual operating condition called `PVT1` for the library `IsCOM-1V`:

# Encounter User Guide

## Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

---

```
create_op_cond -name PVT1  
    -library_file IsCOM-1V.lib  
    -P 1.0  
    -V 1.2  
    -T 120
```

### Editing A Virtual Operating Condition

You can add, delete, or change attributes for a defined virtual operating condition using the Edit Operating Condition form.



You can edit a virtual operating condition before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

- Choose *Design – Import Design*, click the *Advanced* tab, select *MMMC*, click the *MMMC Browse* button, and double click on the name of the operating condition you want to edit.

or:

- Choose *Timing – Configure MMC*, and double click on the name of the operating condition you want to edit.

### Creating RC Corner Objects

An RC corner object provides the software with all of the information necessary to properly extract, annotate, and use the RCs for delay calculation. RC corner objects also control the attributes for running sign-off extraction sequentially on each RC corner.

For each active RC corner in the design, the software extracts and stores a unique set of parasitics. You must use the RC corner attributes to control RC scaling when running the software in multi-mode multi-corner analysis mode. Scaling factors set using the setRCFactor command are ignored in this mode.

RC corner objects are referenced when creating delay calculation corner objects.

- To create an RC corner, use the following command:

```
create_rc_corner
```

## Encounter User Guide

### Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

---

For example, the following command creates an RC corner called `rc-typ` that uses the capacitance table `myTech_nc.CapTbl`, and derates the resistance values based on the temperature of 50 Celsius:

```
create_rc_corner -name rc-typ -cap_table myTech_nc.CapTbl -T 50
```

### Editing An RC Corner Object

To add or change attribute values for an existing RC corner object, use the following command:

```
update_rc_corner
```

You also can make changes to an RC corner object using the Edit RC Corner form:

- Choose *Design – Import Design*, click the *Advanced* tab, select *MMMC*, click the *MMMC Browse* button, and double click on the name of the RC corner object you want to edit.

or:

- Choose *Timing – Configure MMMC*, and double click on the name of the RC corner object you want to edit.



You can use the `update_rc_corner` command or the Edit RC Corner form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

### Creating Delay Calculation Corner Objects

A delay calculation corner provides all of the information necessary to control delay calculation for a specific analysis view. Each corner contains information on the libraries to use, the operating conditions with which the libraries should be accessed, and the RC extraction parameters to use for calculating parasitic data. Delay corner objects can be shared by multiple top-level analysis views.

- To create a delay calculation corner, use the following command:

```
create_delay_corner
```

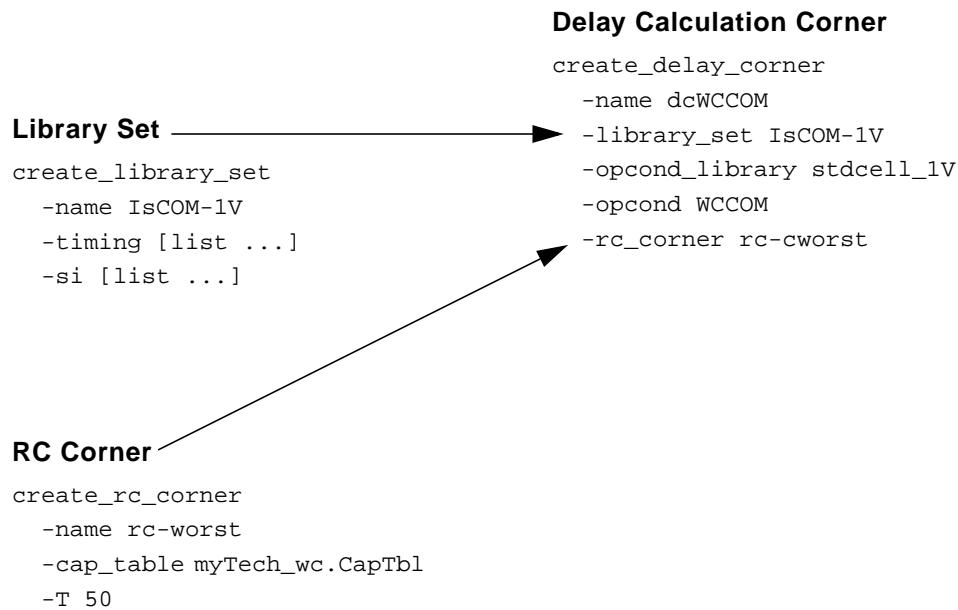
- Use separate delay calculation corners to define major PVT operating points (for example, Best-Case and Worst-Case).

# Encounter User Guide

## Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

- ❑ Use the `-early_*` and `-late_*` parameters within a single delay calculation corner to control on-chip variation.

The following figure shows the creation of a delay calculation corner called `dcWCCOM`. This corner uses the libraries from `IsCOM-1V`, sets the operating condition to `WCCOM`, as defined in the `stdcell_1V` timing library, and uses the `rc-cworst` RC corner:



## Editing A Delay Corner Object

To add or change attribute values of an existing delay calculation corner object, use the following command:

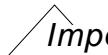
[update delay corner](#)

You also can make changes to a delay calculation corner object using the Edit Delay Corner form:

- Choose *Design – Import Design*, click the *Advanced* tab, select *MMMC*, click the *MMMC Browse* button, and double click on the name of the delay calculation corner you want to edit.

or:

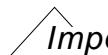
- Choose *Timing – Configure MMMC*, and double click on the name of the delay calculation corner you want to edit.

 *Important*

You can use the `update_delay_corner` command or the Edit Delay Corner form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

## **Adding A Power Domain Definition To A Delay Calculation Corner**

A single delay calculation corner object specifies the delay calculation rules for the entire design. If a design includes power domains, the delay calculation corner can contain domain-specific subsections that specify the required operating condition information, and any necessary timing library rebinding for the power domain.

 *Important*

You must use the same power domain names that you intend to specify when using the `createPowerDomain` command to define the physical aspects of the domain.

- To create a power domain definition for a delay calculation corner, use the following command:

`update_delay_corner`

For example, the following command adds a definitions for the power domain `domain-3V` to the `dcWCCOM` delay calculation corner:

```
update_delay_corner -name dcWCCOM  
-power_domain domain-3V  
-library_set libs-3volt  
-opcond_library delayvolt_3V  
-opcond slow_3V
```

**Note:** When the Encounter software is not in multi-mode multi-corner mode, the binding of timing libraries and operating conditions to power domains are specified using `createPowerDomain -minTimingLib` and `-maxTimingLib`, and `setOpCond -powerDomain` parameters. When the software is in multi-mode multi-corner mode, this use model is disabled.

## **Editing A Power Domain Definition**

To add or change attribute values for an existing power domain definition, use the following command:

## Encounter User Guide

### Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

---

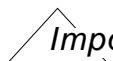
#### update\_delay\_corner

You also can make changes to a power domain definition using the Edit Power Domain form:

- Choose *Design – Import Design*, click the *Advanced* tab, select *MMMC*, click the *MMMC Browse* button, click the + next to *Delay Corners* to list the available delay corners, click the + next to the delay corner to which the power domain definition belongs, and double click on the name of the power domain definition.

or

- Choose *Timing – Configure MMC*, click the + next to *Delay Corners* to list the available delay corners, click the + next to the delay corner to which the power domain definition belongs, and double click on the name of the power domain definition.



*Important*  
You can use the `update_delay_corner` command or the Edit Power Domain form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

## Creating Constraint Mode Objects

A constraint mode object defines one of possibly many different functional, test, or Dynamic Voltage and Frequency Scaling (DVFS) modes of a design. It consists of a list of SDC constraint files that contain timing analysis information, such as the clock specifications, case analysis constraints, I/O timings, and path exceptions that make each mode unique. SDC files can be shared by many different constraint modes, and the same constraint mode can be associated with multiple analysis views.

- To create a constraint mode object, use the following command:

#### create\_constraint\_mode

## Encounter User Guide

### Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

The following figure shows the grouping of the SDC files `io.sdc`, `mission1-clks.sdc`, and `mission1-except.sdc` to create a mode object named `missionSetup`:



### Editing A Constraint Mode Object

To change the SDC constraint file information for an existing constraint mode object, use the following command:

[update\\_constraint\\_mode](#)

You also can make changes to a constraint mode object using the Edit Constraint Mode form:

- Choose *Design – Import Design*, click the *Advanced* tab, select *MMMC*, click the *MMMC Browse* button, and double click on the name of the constraint mode object you want to edit.

or:

- Choose *Timing – Configure MMC*, and double click on the name of the constraint mode object you want to edit.



You can use the `update_constraint_mode` command or the Edit Constraint Mode form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

### Entering Constraints Interactively

You can use the [set\\_interactive\\_constraint\\_modes](#) command to update assertions for a multi-mode multi-corner constraint mode object, and have those changes take immediate effect.

## Encounter User Guide

### Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

---

Specifying `set_interactive_constraint_modes` puts the software into interactive constraint entry mode for the specified constraint mode objects. Any constraints that you specify after this command will take effect immediately on all active analysis views that are associated with these constraint modes. By default, no constraint modes are considered active.

For example, the following commands put the software into interactive constraint entry mode, and apply the `set_propagated_clock` assertion on all views in the current session that are associated with the constraint mode `func1`:

```
set_interactive_constraint_modes func1
set_propagated_clock [all_clocks]
```

The software stays in interactive mode until you exit it by specifying the `set_interactive_constraint_modes` command with an empty list:

```
set_interactive_constraint_modes { }
```

All new assertions are saved in the SDC file for the specified constraint mode when you save the design (`saveDesign`).

The `all_constraint_modes` command can be used to generate a list of constraint modes as the argument for this command.

For example, the following commands put the software into interactive constraint entry mode, and apply the `set_propagated_clock` assertion on all active analysis views in the current session.

```
set_interactive_constraint_modes [all_constraint_modes -active]
set_propagated_clock [all_clocks]
```

Use the `get_interactive_constraint_modes` command to return a list of the constraint mode objects in interactive constraint entry mode.

**Note:** Interactive constraint mode only works when the software is in multi-mode multi-corner timing analysis mode. In min/max analysis mode, constraints are always accepted interactively.

### Constraint Support in Multi-Mode and Multi-Mode Multi-Corner Analysis

The Encounter software isolates the following SDC constraints from conflicting with each other, in both multi-mode and multi-mode multi-corner analysis modes:

- `create_clock`
- `create_generated_clock`
- `set_annotated_check`

## Encounter User Guide

### Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

---

- set\_annotated\_delay
- set\_annotated\_transition
- set\_case\_analysis
- set\_clock\_gating\_check
- set\_clock\_groups
- set\_clock\_latency
- set\_clock\_sense
- set\_clock\_transition
- set\_clock\_uncertainty
- set\_disable\_timing
- set\_drive
- set\_driving\_cell
- set\_false\_path
- set\_fanout\_load
- set\_input\_delay
- set\_input\_transition
- set\_load
- set\_max\_delay
- set\_max\_time\_borrow
- set\_max\_transition
- set\_min\_delay
- set\_min\_pulse\_width
- set\_multicycle\_path
- set\_output\_delay
- set\_propagated\_clock
- set\_resistance

## Encounter User Guide

### Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

**Note:** Path groups defined with `group_path` are considered to be global definitions across all analysis views.

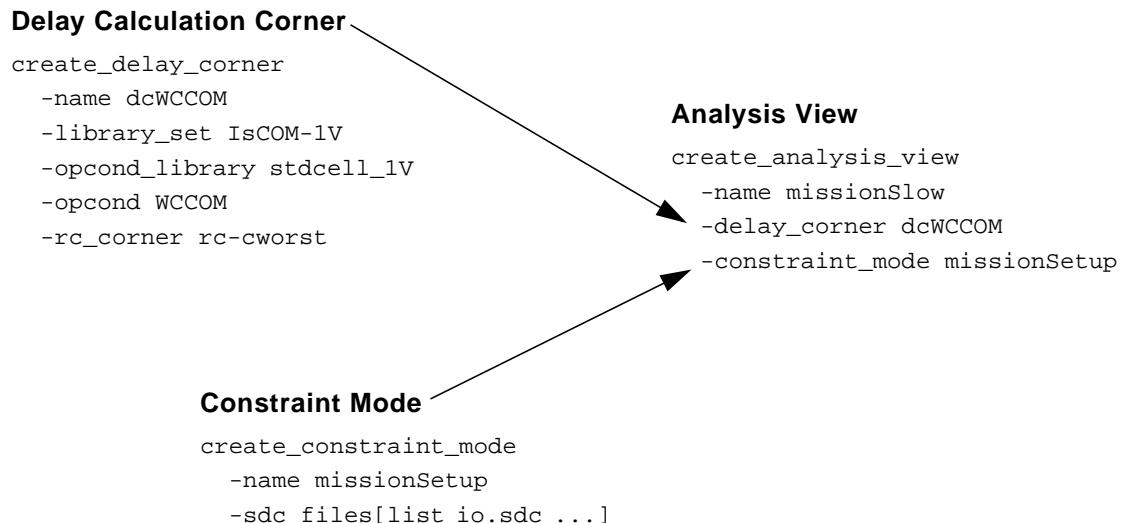
## Creating Analysis Views

An analysis view object provides all of the information necessary to control a given multi-mode multi-corner analysis. It consists of a delay calculation corner and a constraint mode.

- To create an analysis view, use the following command:

```
create_analysis_view
```

The following figure shows the creation of the analysis view `missionSetup`:



## Editing An Analysis View Object

To change the attribute values for an existing analysis view, use the following command:

```
update_analysis_view
```

You also can make changes to an analysis view using the Edit Analysis View form:

- Choose *Design – Import Design*, click the *Advanced* tab, select *MMMC*, click the *MMMC Browse* button, and double click on the name of the analysis view you want to edit.

or:

## Encounter User Guide

### Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

---

- Choose *Timing – Configure MMMC*, and double click on the name of the analysis view you want to edit.
- 



You can use the `update_analysis_view` command or the Edit Analysis View form before multi-mode multi-corner view definitions are loaded into the design, or after. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

## Setting Active Analysis Views

After creating analysis views, you must set which views the software should use for setup and hold analysis and optimization. These “active” views represent the different design variations that will be analyzed. Active views can be changed throughout the flow to utilize different subsets of views. Encounter applications can handle the views concurrently or sequentially, depending on their individual capabilities. Libraries and data are loaded into the system, as required to support the selected set of active views.

- To set active analysis views, use the following command:

```
set_analysis_view
```

**Note:** You must define at least one setup and one hold analysis view.

For example, the following command sets `missionSlow` and `mission2Slow` as the active views for setup analysis, and `missionFast` and `testFast` as the active views for hold analysis:

```
set_analysis_view
  -setup {missionSlow mission2Slow}
  -hold {missionFast testFast}
```

## Guidelines For Setting Active Analysis Views

- The order in which you specify views using the `set_analysis_view` command is important. By default, the first views defined in the `-setup` and `-hold` lists are the default views. Certain Encounter applications that do not support multi-mode multi-corner can only process the data defined for a single view. These applications use the information defined for the default view.
- Concurrent analysis of views for timing optimization costs memory and CPU.

## Changing the Default Active Analysis View

Some Encounter applications can function only on a single analysis view at a time. By default, these single-view applications use the default analysis view. If an application or flow step does not provide a native option for specifying which view to use, you can use the `set_default_view` command to temporarily change the default view to a different active view that is better suited to that flow step.

For example, if the analysis view `missionSlow` is currently the default active setup view in the design, the following command temporarily changes the default view to `mission2Slow`:

```
set_default_view -setup mission2Slow
```

**Note:** Using the `set_default_view` command does not affect software performance because it only uses views that are already active in the design. If you use the `set_analysis_view` command to change the default views, the existing timing, delay calculation, and RC data is reset.

## Checking the Multi-Mode Multi-Corner Configuration

- Use the following command to generate a hierarchical report of your current multi-mode multi-corner configuration:

```
report analysis views
```

You can customize the report to show only the active setup or hold analysis views, all of the active views, or all of the defined views in the design, including those that are currently inactive.

## Changing How the MMMC Browser Displays Configuration Information

By default, the MMMC Browser displays configuration information in two columns: one displays the different existing analysis views, and the other displays the different existing multi-mode multi-corner objects.

You can change how the MMMC Browser displays configuration information using the MMMC Preferences form. You can use this form to change the number of columns displayed, and rename the titles of the columns.

You also can use this form (and the Add Object form) to add and delete objects from columns, and rearrange the order in which the objects are listed, by clicking and holding the left mouse button on an object name, and dragging it to a different position in the list.

- Choose *Design – Import Design*, click the *Advanced* tab, select *MMMC*, click the *MMMC Browse* button, and click the *Preferences* button.

or:

- Choose *Timing – Configure MMMC*, and click the *Preferences* button.

## Saving Multi-Mode Multi-Corner Configurations

The software stores the multi-mode multi-corner configuration as Tcl commands in a view definition file. The view definition file contains all of the library set, RC corner, delay calculation corner, constraint mode, and analysis view definitions that you created. When you specify the `saveDesign` command, the software saves the file to the save directory, and updates the configuration file with a pointer to the file. This multi-mode multi-corner configuration will be reloaded automatically by the subsequent use of the `restoreDesign` command.

Updated SDC files for each mode are saved to the save directory, if ECO changes were made that affect pins that have constraint assertions.

## Controlling Multi-Mode Multi-Corner Analysis Through the Flow

The following levels of support for multi-mode multi-corner analysis exist for Encounter applications:

- Does not support multi-mode multi-corner

Applications that do not support multi-mode multi-corner analysis use the timing, delay, and RC information in the default active analysis views. Use the `set_analysis_view` and `set_default_view` commands to select the appropriate analysis view for the specific flow step.

- Supports single view analysis

Applications that support single view multi-mode multi-corner analysis generally use the default active analysis view, but provide controls that allow you to select specific views to use. You can also use scripting to iterate through the active views.

- Supports sequential

Applications that support sequential view multi-mode multi-corner analysis work on only one analysis view at a time, but provide their own automation to iterate through the active views. These applications can cycle through the views one at a time.

- Supports concurrent view analysis

## Encounter User Guide

### Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

Applications that support concurrent view multi-mode multi-corner analysis work on all active analysis views at the same time.

The following table describes how Encounter applications behave in multi-mode multi-corner analysis mode throughout the flow:

**Table 37-1 Application Behavior in Multi-Mode Multi-Corner Analysis Mode**

Flow Step	Commands	Application Behavior
Placement	<code>setPlaceMode -timingDriven true</code> <code>placeDesign</code>	Uses data from all active analysis views concurrently.
	<code>setPlaceDesign -inPlaceOpt</code> <code>placeDesign</code>	Runs <code>optDesign -preCTS</code> on all active analysis views.
Pre-CTS Timing Optimization	<code>optDesign -preCTS</code>	Optimizes all active analysis views concurrently.
	<code>timeDesign -preCTS</code>	Analyzes all active analysis views concurrently.
Clock Tree Synthesis	CTS is in MMMC mode by default.	
	<code>specifyClockTree</code>	Reports the RC information for each active analysis view, including the estimated capacitance, resistance, and via resistance for each view.
	<code>ckSynthesis</code> <code>ckECO</code>	Attempts to balance the skew and transition for all active setup and hold analysis views. For other constraints, such as capacitance, and delay, CTS attempts to meet the requirements for the default analysis view, or the specified analysis view.
	<code>reportClockTree -view view</code>	Reports the trigger slew for all active analysis views in the design, or for a specified view (-view). Also reports other analysis information, such as the phase delay and transition values, for the default analysis view.

**Encounter User Guide**  
Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

---

Flow Step	Commands	Application Behavior
Post-CTS Timing Optimization	optDesign -postCTS	Optimizes all active analysis views concurrently.
	timeDesign -postCTS	Analyzes all active analysis views concurrently.
NanoRoute	setNanoRouteMode -routeWithTimingDriven true	CTE passes a timing graph file that contains the worst-case slack values across all active setup analysis views.
Metal Fill	addMetalFill -timingAware sta	Uses the timing slack values from the default active analysis view.
Post-Route Timing Optimization	optDesign -postRoute	Optimizes all active analysis views concurrently.
	timeDesign -postRoute	Analyzes all active analysis views concurrently.
Leakage Power Optimization	optLeakagePower	Optimizes all active setup analysis views concurrently.
Signal Integrity Repair	optDesign -postRoute -si	Merges glitch and noise violations across all views; corrects the violations in all active views concurrently.
Timing Signoff	timeDesign -signoff	Calls runQRC iteratively to generate and annotate SPEF files for RC corners.
	timeDesign -signoff -si	Analyzes all active analysis views concurrently. Currently, does not correct crosstalk violations.
Power Analysis	setPowerAnalysisLibrary -view viewName   -internalPowerView view -leakagePowerView view updatePower	Uses the same timing library groups from the specified view for both internal and leakage power (-view), or uses different timing library groups from the specified views for internal and leakage power.

## Performing Timing Analysis

The software performs multi-mode multi-corner timing analysis concurrently on all active analysis views in the design. To run multi-mode multi-corner timing analysis, use one of the following commands:

■ report\_timing

By default, the `report_timing` command analyzes all active analysis views in the design concurrently. Information in the timing report indicates which view was responsible for the path. By default, the tool returns only one path per end point.

If you have a large number of views, you can use `report_timing -sequential` to have the software process the active analysis views one at a time. The software stores results for each view, as well as an aggregated report, in machine-readable timing report files (`.mtarpt`). You can view the results graphically in the Encounter Timing Debug environment. Use this procedure to help determine which critical view subset should be used to drive the implementation flow.

■ timeDesign

The `timeDesign` command reports an aggregated summary. You can use the `-expandedViews` parameter to generate a per view summary report.

## Generating Timing Reports

In multi-mode multi-corner analysis mode, the software can generate various reports that contain timing information for each active analysis view. For more information, see the `report_*` commands in the Timing Analysis Commands chapter of the *Encounter Text Command Reference*.

## Performing Timing Optimization

The software performs multi-mode multi-corner timing optimization concurrently on all active analysis views in the design. To run multi-mode multi-corner timing optimization, use the following command:

■ `optDesign`

By default, the `optDesign` command optimizes timing for all active analysis views in the design concurrently. It automatically prunes the views that are not relevant for timing optimization, in order to reduce turn around time.

The timing closure flow in multi-mode multi-corner mode using `optDesign` is the same one as in Min-Max mode.

For hold optimization, it is important to specify all hold views for which timing violations must be corrected, and all setup views in which the setup timing must be maintained. Do not reduce the active setup view list when performing hold optimization.

## **Encounter User Guide**

### Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

---

The `optDesign` command reports results in an aggregated summary. Use the `timeDesign` command to see an expanded timing per view report.

## Creating the ICT File

---

The first step involved in modeling the parasitic interconnect capacitance and resistance of your design is to specify the fabrication process information in an Interconnect technology (ICT) file by using the syntax defined in this section. You can use any text editor to enter this information.

**Note:** Although there are no file-naming restrictions for ICT files, you should name your ICT file by using the process name with the `.ict` file extension, as follows:

`process_name.ict` (ICT file)

Fabrication process information consists of the following requirements:

- Minimum spacing and minimum width of the conductors as specified in the design rules for the conductor layers
- Thicknesses of the conductor layers
- Heights of the conductor layers above the substrate (measuring height from the field) or as a delta from a previously defined lower-level conductor layer
- Resistivities of the conductor layers
- Interlayer planar dielectric constant, its height above the substrate (measuring height above the field), and its thickness
- Names of the top conductor layer of a via, the bottom conductor or diffusion layer of the via, and the contact resistance of the via
- Names of the wells

The rest of this section describes the syntax and format of the ICT file containing the process information for your design.

For more information on generating the ICT files, see the QRC TechGen manual.

## Format

Lines in the ICT file are in the following general format:

*command name {argument\_list}*

where *argument\_list* is a list of field-value pairs. The fields in this syntax are separated by white space. ViewICT, IceCaps, and RCgen ignore blank lines.

**Note:** A backslash (\) is generally required for line continuation, but it is not required if you are using braces ({} ) to define a list.

## Data

All data entered into the ICT file should be the actual physical fabrication process information, not the drawn data.

## Comments

A pound-sign character (#) at the beginning of a line indicates text that ViewICT, IceCaps, and RCgen are treated as comments.

## Case Sensitivity

All keywords in the ICT file are case-insensitive. However, the arguments are case-sensitive. Keywords consist of all command and field names.

## Warnings and Errors

The ViewICT utility displays all errors, warnings, and informational messages on screen and writes them in a log file. Warnings and errors include the corresponding line number.

## Invalid Layer Names

The “NX” string is an invalid layer name.

## Commands

This section describes the commands available in the ICT file.

All command fields are enclosed in braces ({}).

## Process

The `process` command specifies the background dielectric constant. Use it only once in the ICT file.

### Syntax

```
process name {background_dielectric_constant value}
```

or

```
process name {  
    background_dielectric_constant value  
}
```

This syntax contains the following parameters:

- *name*  
Specifies the name of the process.
- *background\_dielectric\_constant value*  
Specifies the dielectric constant for the region above the top passivation layer or last dielectric layer. This field is required.

### Example

```
process "Process_Example" {  
    background_dielectric_constant 1.0  
}
```

## Well

The `well` command which defines the well layers is an optional command that you can use to differentiate capacitance to a well from capacitance to the substrate.

### Syntax

```
well name { }
```

*Name* specifies the name of the well layer.

Anything placed in the brackets is ignored.

### **Example**

```
well nwell { }
well pwell { }
```

### **Conductor**

The `conductor` command defines conductor layers.

You can specify the height of a conductor layer in three ways:

- Height (absolute)
- Delta height (relative)
- Upto (maximum top down)

You can use more than one of these methods per conductor definition, as long as the numbers are valid.

All measurements are in microns, unless otherwise specified.

### **Syntax**

```
conductor name {field1 value1 ... fieldN valueN}
```

or

```
conductor name {
    field1 value1
    ...
    fieldN valueN
}
```

You can specify the *field-value* pairs in any order.

This syntax contains the following parameters:

- *name*

Specifies the name of the conductor layer.

■ `min_spacing value`

Specifies the minimum spacing permitted by the technology between two conductors (wires) on a layer.

■ `min_width value`

Specifies the minimum width of a conductor.

■ `height value`

Specifies the layer's height above the substrate.

■ `delta_height value`

Specifies the layer's height relative to the top of another layer. This parameter must be used with `delta_layer`.

■ `delta_layer layer_name`

Specifies the reference layer for `delta_height`. It must be a layer that has already been defined. The reference layer must be a conducting layer, a dielectric layer, or a passivation layer. This parameter must be used with `delta_height`.

■ `thickness value`

Specifies the layer's thickness.

■ `upto value`

Specifies the layer's top surface height above the substrate. This value is equal to the height plus the thickness. You only need to specify two of the three or four parameters (`height`, `{delta_height, delta_layer}`, `thickness`, `upto`) to complete the geometrical definition of a conductor layer.

■ `resistivity value | [value width]+`

Specifies the layer's sheet resistance, in ohms per square. You can enter the resistivity value as a constant, or you can enter value-width pairs as a piecewise linear function. You may want to use the value-width pairs to account for width-dependent resistivity.

If you enter value-width pairs, the syntax is as follows:

```
resistivity value1 width1 value2 width2 ... valuen widthn
```

If the width of the wire is less than the minimum width, `width1`, use the minimum value, `value1`. If the width of the wire is greater than the maximum width, `widthn`, use the maximum value, `valuen`. For widths between value-width pairs, the resistivity value through linear interpolation.

**Note:** The width in the value-width pair refers to the silicon width of the wire.

- **rho**
  - **rho\_widths** W<sub>1</sub> ... W<sub>n</sub>
  - **rho\_spacings** S<sub>1</sub> ... S<sub>m</sub>
  - **rho\_values** R<sub>11</sub> .... R<sub>1n</sub>  
....  
R<sub>m1</sub> ... R<sub>mn</sub>

This parameter is for specifying resistivity as a function of both width and spacing. For values that fall between specified points, linear interpolation is applied. When values are outside of the boundary values, it uses the boundary values.

- **gate\_forming\_layer** [true|false]

Specifies that this layer forms the gate. The polycide or polysilicon layer is a typical gate-forming conducting layer.

- **field\_poly\_diffusion\_spacing value**

Specifies the lateral spacing between field polycide and diffusion for transistor-level parasitic extraction. There is no lateral separation between gate polycide and diffusion.

- **PnR\_widths [value] +, PnR\_spacings [value] +**

Allows you to provide design widths and spacings used in the layout to the technology file generation program. These are not necessary for accurate extraction of parasitics if the design widths and spacings are within small perturbations of the minimum process widths and spacings. However, if the design widths and spacings are routinely different from the specified process parameters, it is recommended that you provide these values to the technology file generator.

- **capacitor\_only\_layer\_to layer\_name**

Specifies that the current layer be used solely to create high capacitance values in the design and that it is located a few angstroms above or below the *layer\_name* layer. Layers having the **capacitor\_only\_layer\_to** keyword set are not extracted.

- **wire\_top\_enlargement\_c E<sub>top</sub>**

**wire\_top\_enlargement\_r E<sub>top</sub>**

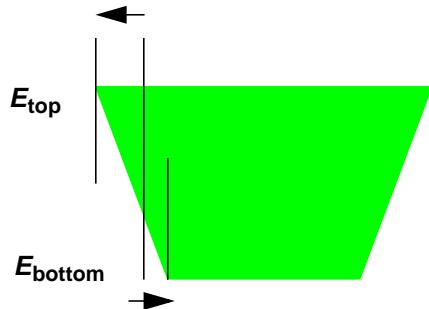
Specifies the enlargement, either positive or negative, of the top edge of the wire. Specify values for both R and C to account for different bias values for R and C. See [Figure 38-1](#) on page 1211.

- **wire\_bottom\_enlargement\_c E<sub>bottom</sub>**

`wire_bottom_enlargement_r  $E_{bottom}$`

Specifies the enlargement, either positive or negative, of the bottom edge of the wire. Specify values for both R and C to account for different bias values for R and C. See [Figure 38-1](#) on page 1211.

**Figure 38-1 Trapezoidal Wire Shape Resulting from Manufacturing Processes**



■ `wire_edge_enlargement | wire_edge_enlargement_[r|c]`

`wee_widths  $W_1 \dots W_n$`

`wee_spacings  $S_1 \dots S_m$`

`wee_adjustments  $E_{11} \dots E_{1n}$`

.

.

.

`$E_{m1} \dots E_{mn}$`

Models the effect of wire-edge enlargement, if the `wire_edge_enlargement`, `wee_width`, `wee_spacings`, and `wee_adjustments` keywords are specified. The `wee_adjustments` table describes the amount of enlargement applied when certain spacings and widths are observed. For example, the wire is enlarged by  $E_{ij}$  for spacing  $S_i$  and width  $W_j$ . Positive enlargements oversize and negative enlargement undersize the wire. A piecewise constant interpolation is used to obtain enlargements for intermediate spacings and widths. For width/spacings outside of the boundary width/spacing points, the boundary values are used.

`Wire_edge_enlargement_r` and `Wire_edge_enlargement_c` can be used if one wants to specify different values for resistance and capacitance.

□ `wee_widths  $W_1 \dots W_n$`

Specifies the widths of the wires in the design. Typically, variation is only seen for widths less than 1.5 microns.

- wee\_spacings  $S_1 \dots S_m$

Specifies the spacings of the wires in the design. Typically, variation is only seen for spacings less than 1.5 microns.

- wee\_adjustments  $E_{11} \dots E_{1n} \dots E_{m1} \dots E_{mn}$

Specifies the enlargement, either positive or negative, of the wire edge.

See “[Wire-Width Values](#)” on page 1213 for information on the wire-width values to use.

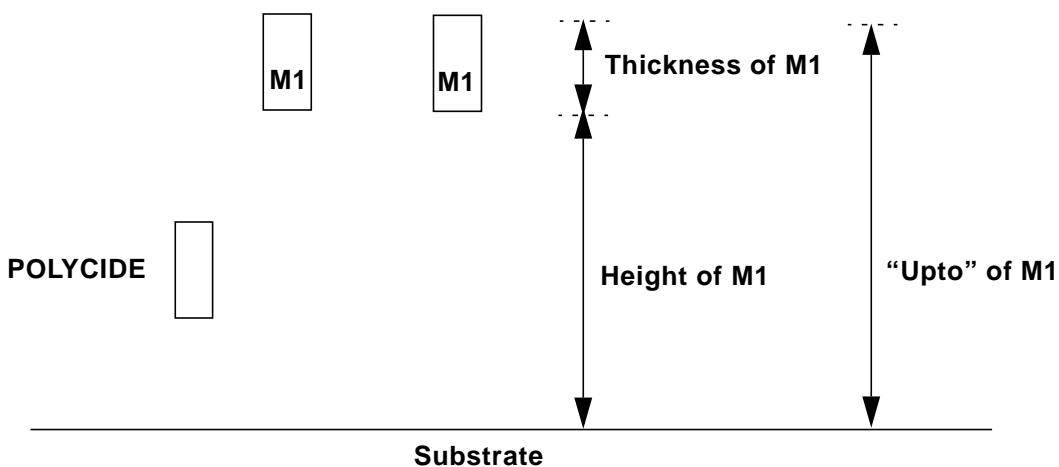
### **Required Conductor Command Fields**

The required fields in this syntax are min\_spacing, min\_width, resistivity, gate\_forming\_layer, min\_net\_fill\_spacing, X\_fill\_fill\_spacing, Y\_fill\_fill\_spacing, unit\_fill\_region, and two of the following three parameters:

- height (or delta\_height and delta\_layer)
- thickness
- upto

[Figure 38-2 on page 1212](#) illustrates these parameters.

### **Figure 38-2 Geometric Fields in a Conducting Layer**



### **Wire-Width Values**

You can use the `wire_edge_enlargement` statement with the `wire_top_enlargement` statement or the `wire_bottom_enlargement` statement, or both in the ICT file. If you use the `wire_edge_enlargement` statement with either or both of these statements, the width of the wires defined by `wee_widths` must be biased as follows:

$$\text{drawn\_width} + ((\text{top} + \text{bottom}) / 2)$$

When calculating resistivity as a function of width, you must use the `wire_top_enlargement` and `wire_bottom_enlargement` values to correct the resistance-width pairs. If a table of wire-edge enlargement values is available, the RC extractor uses the wire widths in the table, which always include biasing and wire-edge enlargement. If this table is not available, the resistance is calculated as follows:

$$\rho * L / (\text{drawn\_width} + (\text{top} + \text{bottom}) / 2 + (\text{top} + \text{bottom}) / 2)$$

where `rho` is the sheet resistivity.

Wire-width values are used in the following order:

1. Drawn width
2. Biased width
3. Edge-enlarged width
4. Resistivity as a function of width

Figure 38-3 on page 1214 illustrates the defining of the conductor layer.

## Encounter User Guide

### Creating the ICT File

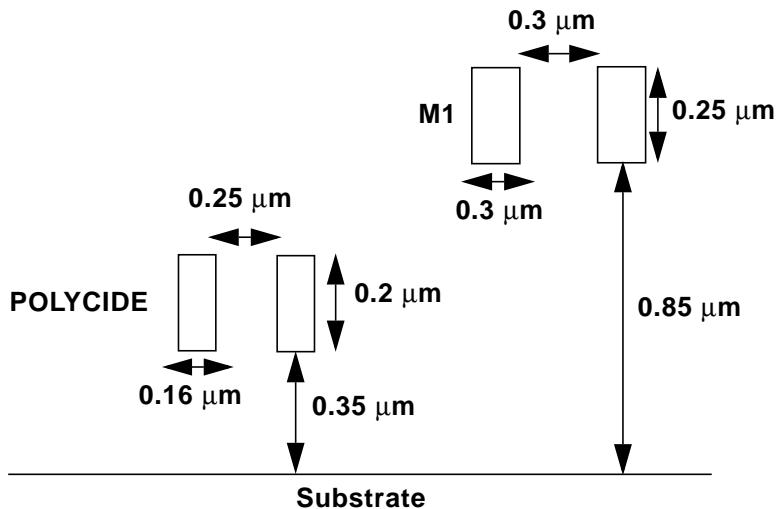
---

**Figure 38-3 Example Conductor Definition**

```

    :
conductor "POLYCIDE" {
    min_spacing      0.25
    min_width        0.16
    height           0.35
    thickness         0.20
    resistivity       8.6
    gate_forming_layer true
}
conductor "M1" {
    min_spacing      0.30
    min_width        0.30
    height           0.85
    thickness         0.25
    resistivity       8.0
    gate_forming_layer false
}
:

```



#### **Example File for Conductor Definition**

```

conductor "POLYCIDE" {
    min_spacing      0.25
    min_width        0.16
    height           0.35
    upto             0.55
    resistivity       8.6
    gate_forming_layer true
}

conductor "M1" {
    min_spacing      0.30
    min_width        0.30
    delta_layer      POLYCIDE
    delta_height     0.30
    thickness         0.25
    resistivity       8.0
    gate_forming_layer false
    wire_top_enlargement 0.01
    wire_bottom_enlargement -0.01
    wire_edge_enlargement {
        wee_widths      0.18  0.00   0.26   0.30   0.34
        wee_spacings    0.18  0.00   0.26   0.30   0.34   0.38
        wee_adjustments 0.00  0.00  -0.10  -0.10  -0.20
    }
}

```

## Encounter User Guide

### Creating the ICT File

---

```
    0.00  0.00  0.00  -0.10  -0.20
    0.10  0.00  0.00   0.00  -0.10
    0.10  0.10  0.00   0.00   0.00
    0.20  0.20  0.10   0.00   0.00
    0.30  0.20  0.20   0.10   0.00
}
}
```

## Dielectric

The `dielectric` command defines dielectric layers.

All measurements are in microns unless otherwise specified.

### Syntax

```
dielectric name {conformal value field1 value1 ... fieldN valueN}
```

or

```
dielectric name {
    conformal value
    field1 value1
    ...
    fieldN valueN
}
```

You can specify the *field-value* pairs in any order.

The syntax for planar dielectrics contains the following parameters:

- `name`  
Specifies the name of the dielectric layer.
- `conformal false`  
Specifies that the dielectric is planar. This field is required.
- `height value`  
Specifies the layer's height above the substrate.
- `thickness value`

Specifies the layer's thickness.

- `dielectric_constant value`

Specifies the dielectric constant for this material.

- `delta_height value`

Specifies the layer's height relative to the top of another layer.

- `delta_layer layer_name`

Specifies the reference layer for `delta_height`. It must be a layer that has already been defined. A reference layer can be a conducting layer or a dielectric layer.

- `upto value`

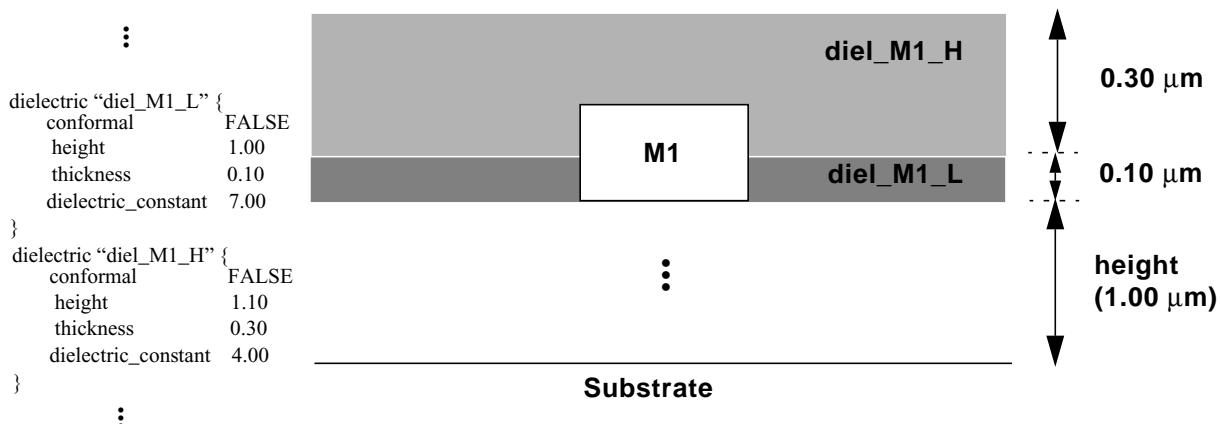
Specifies the layer's top surface height above the substrate. This value is equal to the height plus the thickness. You only need to specify two of the three parameters (`height` (or `{delta_height, delta_layer}`), `thickness`, `upto`) to complete the geometrical definition of a dielectric layer.

The required fields in the specification for planar dielectrics are `conformal`, `dielectric_constant`, and two of the following three parameters:

- `height (or {delta_height and delta_layer})`
- `thickness`
- `upto`

[Figure 38-4 on page 1216](#) illustrates the planar dielectric syntax.

#### Figure 38-4 Planar Dielectric Syntax



## Passivation

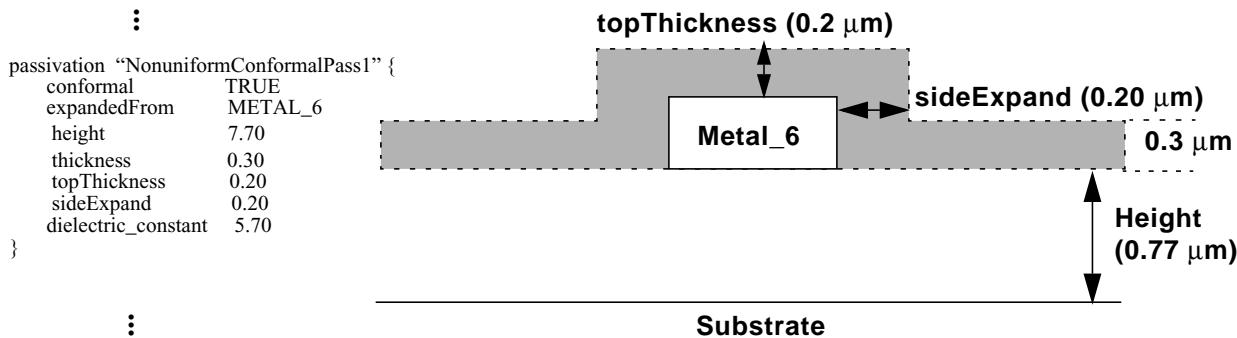
The passivation command defines passivation layers. The passivation layers are usually placed on top of the last metal layer and should be placed higher than the dielectric layers.

### Syntax

The syntax of this command is the same as that of the dielectric command, except that *name* specifies the name of the passivation layer. See “[Dielectric](#)” on page 1215 for information on this syntax. The passivation layers are usually placed on top of the last metal layer and should be placed higher than the dielectric layers.

[Figure 38-5 on page 1217](#) illustrates the defining of the passivation layer.

### Figure 38-5 Passivation Syntax



### Example

```
passivation "NonuniformConformalPass1" {
    conformal      TRUE
    expandedFrom   METAL_6
    height         7.70
    thickness      0.30
    topThickness   0.20
    sideExpand     0.20
    dielectric_constant 5.70
}
```

## Via

The `via` command defines vias or contacts.

### Syntax

```
via name {top_layer value bottom_layer value contact_resistance value}
```

This syntax contains the following parameters:

- `top_layer value`  
Specifies the name of the top conductor.
- `bottom_layer value`  
Specifies the name of the bottom conductor or diffusion layer.
- `contact_resistance value`  
Specifies the contact resistance of the via, in ohms.

### Example

```
via "V A1" {  
    top_layer METAL_2  
    bottom_layer METAL_1  
    contact_resistance 7.9  
}
```

Following is a sample specification of a local interconnect via layer. The name of the conductor and the name of the via are the same.

```
conductor "LI" {  
    min_spacing      0.3  
    min_width        0.35  
    height           0.55  
    thickness         0.60  
    resistivity       0.40  
    gate_forming_layer FALSE  
}
```

```
via "LI" {  
    top_layer LI  
    bottom_layer POLYCIDE  
    contact_resistance 2.000
```

## Encounter User Guide

### Creating the ICT File

---

}

**Note:** Local interconnect is a layer, usually thicker than the polysilicon layer, that can be deposited after polysilicon and can connect to source-drain regions on the polysilicon layer.

## Sample ICT File

```
#  
#      Copyright (c) 2003 Cadence Design Systems, Inc.  
  
#####  
# Process declaration.  
#####  
  
process "DIFFERENT_KINDS_OF_DIELECTRIC" {  
    background_dielectric_constant 1.0  
}  
  
#####  
# Well declarations.  
#####  
  
well nwell {}  
well pwell {}  
  
#####  
# Diffusion declarations.  
#####  
  
diffusion "N_SOURCE_DRAIN" {  
    # Tox is (height of POLYCIDE - thickness of diffusion) = (0.35 - 0.3455) = 0.0045um  
    thickness          0.3455  
    resistivity        7.7  
}  
diffusion "P_SOURCE_DRAIN" {  
    thickness          0.3455  
    resistivity        8.3  
}  
  
#####
```

## Encounter User Guide

### Creating the ICT File

---

```
# Conducting layer declarations.  
#####  
  
conductor "POLYCIDE" {  
    min_spacing          0.25  
    min_width            0.16  
    height               0.35  
    thickness             0.20  
    resistivity           8.6  
    gate_forming_layer   true  
}  
conductor "METAL_1" {  
    min_spacing          0.23  
    min_width            0.23  
    height               1.05  
    thickness             0.53  
    resistivity           0.086  
    gate_forming_layer   false  
}  
conductor "METAL_2" {  
    min_spacing          0.28  
    min_width            0.28  
    height               2.38  
    thickness             0.53  
    resistivity           0.086  
}  
# The key words TRUE and FALSE are not case sensitive.  
    gate_forming_layer   FALSE  
}  
conductor "METAL_3" {  
    min_spacing          0.28  
    min_width            0.28  
    height               3.71  
    thickness             0.53  
    resistivity           0.086  
    gate_forming_layer   false  
}  
conductor "METAL_4" {  
    min_spacing          0.28  
    min_width            0.28  
}  
# delta_height + delta_layer is an alternative to height.  
    delta_height          0.80
```

## Encounter User Guide

### Creating the ICT File

---

```
delta_layer           METAL_3
# "height" is then redundant but it's okay to specify.
#   height             5.04
#   thickness          0.53
#   resistivity        0.086
#   gate_forming_layer false
}
conductor "METAL_5" {
    min_spacing      0.28
    min_width        0.28
    height           6.37
    thickness         0.53
    resistivity       0.086
    gate_forming_layer false
}
conductor "METAL_6" {
    min_spacing      0.46
    min_width        0.44
    height           7.70
    thickness         0.99
    resistivity       0.035
    gate_forming_layer false
}

#####
# Dielectric and passivation layer declarations.
#####

#####
# Base dielectric from substrate...
#####

dielectric "First_dielectric" {
# Starts at height zero.
    conformal        FALSE
    height            0.00
    thickness         0.35
    dielectric_constant 3.90
}
# Simple planar dielectric starts at the bottom of POLYCIDE
```

## Encounter User Guide

### Creating the ICT File

---

```
# and ends at 1.08um which is 0.03um above the bottom of M1.

dielectric "SimplePlanar1" {
# Starts at height of Poly
    conformal           FALSE
    height              0.35
    thickness           0.73
    dielectric_constant 4.00
}

#####
# M1 level...
#####

# Now a planar intra-metal (M1) dielectric starts 0.03um above from the
# bottom of M1.

dielectric "PlanarIntraMetal1" {
    conformal           FALSE
#
# Starts at height of M1
    height              1.08
# Laterally intersect with M1
    thickness           0.03
    dielectric_constant 7.00
}

#
# The second intra-metal dielectric across M1
# and on top of "PlanarIntraMetal1".

dielectric "PlanarIntraMetal2" {
# Yet another intra-metal planar dielectric layer.
    conformal           FALSE
    height              1.11
    upto                1.15
# OR
#    thickness           0.04
#    dielectric_constant 3.00
}

#
# A conformal dielectric.
```

## Encounter User Guide

### Creating the ICT File

---

```
# When specifying a conformal dielectric (whether it is uniform or
# non-uniform, we must use "conformal TRUE", "expandedFrom", "sideExpand",
# and "topThickness" together.

#
# 1. "conformal" must be set to TRUE.
# 2. "expandedFrom" can be a metal layer or a dielectric/passivation layer.
#    The conformal dielectric layer must be expanded from its immediate
#    lower (metal/dielectric/passivation) layer. It cannot be expanded
#    from a planar dielectric layer.
# 3. "thickness" is the bottom dielectric thickness.
# 4. "sideExpand" specifies the side thickness.
# 5. "topThickness" is the thickness of the dielectric above the
#    top of the "expandedFrom" layer.

dielectric "conformalAtTopOFM1" {
# Conformal above M1
    conformal          TRUE
    expandedFrom       METAL_1

# and starts from the top of "PlanarIntraMetal2"
    height            1.15

# Base/Bottom thickness of the conformal dielectric.
    thickness          0.43

# The thickness of the dielectric above the "expandedFrom" object, i.e. M1.
    topThickness       0.43

# This is the side thickness of the dielectric.
    sideExpand         0.43
    dielectric_constant 4.10
}

dielectric "SimplePlanar2" {
# From top of M1 to bottom of M2
    conformal          FALSE
    height             1.58
    thickness          0.80
    dielectric_constant 4.00
}
```

## Encounter User Guide

### Creating the ICT File

---

```
#####
# M2 level...
#####

# An uniform conformal dielectric starting from the bottom of M2.

dielectric "UniformConformal1" {
    conformal          TRUE
    expandedFrom       METAL_2
    # Height of M2
    delta_height       0.00
    delta_layer        SimplePlanar2
    # height           2.38
    thickness          0.50
    topThickness       0.50
    sideExpand         0.50
    dielectric_constant 3.00
}

# A nonuniform conformal dielectric is one when any one of "thickness",
# "sideExpand", and "topThickness" are different.

dielectric "NonuniformConformal1" {
    conformal          TRUE
    height             2.88
    thickness          0.10
    expandedFrom       UniformConformal1
    sideExpand         0.03
    topThickness       0.05
    dielectric_constant 7.00
}

dielectric "SimplePlanar3" {
    conformal          FALSE
    height             2.98
    thickness          0.73
    dielectric_constant 4.10
}

#####
```

## Encounter User Guide

### Creating the ICT File

---

```
# M3 level...
#####
# A special case of conformal dielectric.
dielectric "NonuniformConformal2" {
    # Humps over M3 with side and top thicknesses equal to 0.17 um and 0.50 um,
    # respectively.
    conformal          TRUE
    expandedFrom       METAL_3
    height             3.71
    # Note that the bottom thickness is thicker than M3!
    thickness          0.90
    topThickness       0.50
    sideExpand         0.17
    dielectric_constant 4.10
}

dielectric "SimplePlanar5" {
    conformal          FALSE
    height             4.61
    # Upto the bottom of M4.
    upto               5.04
    dielectric_constant 3.00
}

#####
# M4 level...
#####

dielectric "NonuniformConformal3" {
    conformal          TRUE
    expandedFrom       METAL_4
    # Height of M4
    height             5.04
    thickness          0.30
    topThickness       0.30
    sideExpand         0.10
    # Special case. See SimplePlanar6.
    dielectric_constant 4.10
}
dielectric "PlanarIntraMetal3" {
```

## Encounter User Guide

### Creating the ICT File

---

```
# Planar intrametal dielectric.
    conformal          FALSE
    height            5.34
    upto              5.44
    dielectric_constant 3.10
}
dielectric "PlanarIntraMetal4" {
# Top off the top of NonuniformConformal3.
    height          5.44
    upto            5.87
    dielectric_constant 3.00
}
dielectric "SimplePlanar6" {
    conformal          FALSE
    height            5.87
# Upto the bottom of M5.
    upto              6.37
# NOTE that it has the same dielectric constant as NonuniformConformal3.
# This makes "NonuniformConformal3" a special case.
    dielectric_constant 4.10
}

#####
# M5 level...
#####

dielectric "UniformConformal3" {
    conformal          TRUE
    expandedFrom      METAL_5
    height            6.37
    thickness          0.10
    topThickness       0.10
    sideExpand          0.10
    dielectric_constant 7.20
}
dielectric "PlanarIntraMetals5" {
# Special planar dielectric which intersects "UniformConformal3"
    conformal          FALSE
    height            6.47
    thickness          0.40
    dielectric_constant 3.00
}
```

## Encounter User Guide

### Creating the ICT File

---

```
}

dielectric "PlanarIntraMetal6" {
# Another special planar dielectric which intersects "UniformConformal3"
    conformal          FALSE
    height             6.87
    thickness          0.10
    dielectric_constant 4.00
}

dielectric "PlanarIntraMetal7" {
# Yet another special planar dielectric which intersects "UniformConformal3"
    conformal          FALSE
    height             6.97
    thickness          0.03
    dielectric_constant 7.00
}

#####
passivation "PlanarPass1" {
# From top of M5 to bottom of M6.
    conformal          FALSE
    height             7.00
    thickness          0.70
    dielectric_constant 4.00
}

#####
# M6 level...
#####

passivation "NonuniformConformalPass1" {
    conformal          TRUE
    expandedFrom        METAL_6
    height             7.70
    thickness          0.30
    topThickness       0.20
    sideExpand          0.20
    dielectric_constant 5.70
}

passivation "PlanarPass2" {
    conformal          FALSE
}
```

## Encounter User Guide

### Creating the ICT File

---

```
height          8.00
upto           8.89
dielectric_constant 4.30
}

#####
passivation "PlanarPass3" {
    conformal      FALSE
    height         8.89
    thickness      1.00
    dielectric_constant 3.00
}

#####
# Contacts and Via declarations.
#####

via "CONT" {
    top_layer METAL_1
    bottom_layer POLYCIDE
    contact_resistance 7.8
}
via "CONT" {
    top_layer METAL_1
    bottom_layer N_SOURCE_DRAIN
    contact_resistance 11
}
via "CONT" {
    top_layer METAL_1
    bottom_layer P_SOURCE_DRAIN
    contact_resistance 10
}
via "VAl" {
    top_layer METAL_2
    bottom_layer METAL_1
    contact_resistance 7.9
}
via "VA2" {
    top_layer METAL_3
    bottom_layer METAL_2
    contact_resistance 8.2
}
```

## Encounter User Guide

### Creating the ICT File

---

```
}
```

```
via "VA3" {
```

```
    top_layer METAL_4
```

```
    bottom_layer METAL_3
```

```
    contact_resistance 8.1
```

```
}
```

```
via "VA4" {
```

```
    top_layer METAL_5
```

```
    bottom_layer METAL_4
```

```
    contact_resistance 8.0
```

```
}
```

```
via "VA5" {
```

```
    top_layer METAL_6
```

```
    bottom_layer METAL_5
```

```
    contact_resistance 4.0
```

```
}
```

## **Encounter User Guide**

### Creating the ICT File

---

---

## **ECO Flows**

---

This appendix summarizes the variety of Engineering Change Order (ECO) flows possible with Encounter, and outlines the current approach for each flow.

- [Overview](#) on page 1232
- [Pre-Mask ECO Changes from a New Verilog File](#) on page 1234
- [Pre-Mask ECO Changes from a New DEF File](#) on page 1238
- [Pre-Mask ECO Changes from an ECO File](#) on page 1242
- [Post-Mask ECO Changes from a New Verilog Netlist](#) on page 1246
- [Post-Mask Gate Array Style ECO from a New Verilog Netlist](#) on page 1252

## Overview

Many types of ECO flows are possible. The ones outlined in this appendix cover the most common cases. You can use these flows directly, or you can modify them for your design.

For an [example ECO file](#), see the “ECO Directives” chapter in the *Encounter User Guide*.

## Assumptions

The ECO flows in this appendix assume the following:

- The old Verilog netlist and the new Verilog netlist have already been uniquified so that no Verilog module is instantiated more than once.
- Your design uses an existing floorplan.
- Your old placement, special routing, and routing information is saved in one of the Encounter formats, DEF, and so forth.

## Flows

This appendix describes various types of ECO flows:

- [Pre-Mask ECO Changes from a New Verilog File](#) on page 1234

If you make changes to the netlist, use this flow to use to load the new netlist and restore all the physical data from the previously saved design.

- [Pre-Mask ECO Changes from a New DEF File](#) on page 1238

Allows you to make external changes that include new cell placements from a DEF file, while preserving your previous placement, optimization, and optionally, previous routing information; for example, a clock tree with placements and specialized buffer insertion with placements.

- [Pre-Mask ECO Changes from an ECO File](#) on page 1242

Allows you to use an ECO file to make changes to the netlist.

- [Post-Mask ECO Changes from a New Verilog Netlist](#) on page 1246

Allows you to make late logic changes after the masks are made. This flow uses pre-existing spare cells, so no poly/diffusion changes are allowed, and only the routing is modified. You can direct the software to make routing changes only on specific layers.

## **Encounter User Guide**

### ECO Flows

---

- [Post-Mask Gate Array Style ECO from a New Verilog Netlist](#) on page 1252

Allows you to make late logic changes after the masks are made. This flow uses pre-existing gate array style filler cells to create new cells. No poly/diffusion changes are allowed, and only the routing is modified.

## Pre-Mask ECO Changes from a New Verilog File

In this flow, your design is placed and possibly routed, and you want to make a few changes. The changes are done before the masks are made, so it is a pre-mask ECO flow and there is no need to keep the original poly/diffusion patterns.

### Preparation

Before starting the flow steps, you should have the following files available:

- `oldchip.conf`

Create this file by using the `saveConfig` command, which creates data for libraries, timing constraints, global power connections, and so on.

- `oldchip.fp`  
`oldchip.fp.spr`

Create these files (floorplan, special routing, placement, and routing) by using the `saveFPlan` command.

*or*

- `oldchip.def`

Create this file (DEF formats for floorplan, placement, special routing, optionally routing) by using the `defOut` command.

- `newchip.v`

The new Verilog file is typically created by manually editing the old Verilog netlist.

**Note:** If you want to preserve routing, your existing design must contain the antenna diode cells that were added during the previous routing.

### Flow

1. Read the new netlist
2. Load old floorplan/placement/routing data
3. Remove filler cells and notches (optional)
4. Perform incremental placement
5. Add filler cells (optional)

6. Perform incremental or final route
7. Add notches
8. Trim metal fill

Or,

1. Use the ecoDesign command to perform the pre-mask ECO operations.

## Steps

1. Read the new netlist.

```
loadConfig newchip.conf # same as oldchip.conf except use newchip.v
```

or

```
loadConfig oldchip.conf 0  
set rda_Input(ui_netlist) "newchip.v"  
commitConfig
```

The netlist includes old libraries, global power connections, and so forth. It typically uses old timing constraints, however new timing constraints can be used.

2. Read the old floorplan, special routes, placements and routing from the old netlist files.

```
loadFPlan oldchip.fp  
ecoDefIn -reportFile ecoDefIn.rpt oldchip.def  
applyGlobalNets
```

During this step,

- Matching instances receive old placements.

Soft matching happens when a DEF net name does not have an equivalent name and another net is found in memory, that has the same connections as described in the DEF.

The most common case for soft matching is when nets have multiple aliases in a hierarchical design “net1” = “inst1/net2” = “inst1/inst2/net3” and so on. Any of these net names can be used in the DEF and can have the same connections.

Without soft matching, net ‘a’, for example, is removed and net ‘b’ is created in its place, resulting in ripping of the wire.

- Instances existing only in the oldchip.def file are ignored, so they are not added to the current netlist.

## Encounter User Guide

### ECO Flows

---

- Changed instances (new cell) are assigned a new cell and are left unplaced.
  - Physical-only cells in the old netlist (marked with +SOURCE DIST in the DEF) get added (for example, well taps, end caps, and filler cells).
  - Instances that are only in the new netlist are left unplaced.
  - Routing for existing or modified nets is restored (possibly with opens or shorts).
  - Routing for deleted nets is removed.
- 3.** (Optional) Add level shifter or isolation cell for low power design.

loadShifter -infile oldchip.vsf

or

addIsolationCell

During this step, level shifters or isolation cells will be added to new ECO nets that cross the power domain. The newly added cells will be unplaced. The `ecoPlace` command will place them in their respective power domain boundaries.

- 4.** Remove filler cells or notch fill (if present).

deleteFiller -prefix FILL  
deleteNotchFill

- 5.** Perform incremental placement.

ecoPlace

Unplaced instances are placed, previously placed cells are not moved, and routing is unaffected. You can manually preplace critical cells before using the `ecoPlace` command by placing the cell in the bottom, left corner, selecting it, and then moving it graphically.

placeInstance i1/i2/i3 0 0  
selectInst i1/i2/i3

- 6.** Add filler cells back into the rows.

addFiller -cell FILL4 -prefix FILL -fillBoundary  
addFiller -cell FILL2 -prefix FILL -fillBoundary  
addFiller -cell FILL1 -prefix FILL -fillBoundary

Global power connections are performed automatically based on rules loaded from the configuration or floorplan file earlier.

- 7.** Perform incremental or final route.

ecoRoute

NanoRoute automatically detects modified and new nets, and incrementally routes any nets that are incomplete or have violations.

## Encounter User Guide

### ECO Flows

---

8. (Optional) Add notches.

fillNotch

9. (Optional) If the original design contained metal fill, trim the metal fill:

trimMetalFill

Cadence recommends that you use this command to trim metal fill during the ECO process instead of deleting and adding the metal fill again.

Or,

1. Use the ecoDesign command to perform ECO operations. For example, the following command performs a pre-mask ECO:

```
ecoDesign original.enc.dat top_cell newchip.v
```

## Pre-Mask ECO Changes from a New DEF File

In this flow, your design is placed and possibly routed, and you want to make a few changes with known cell placements from a new DEF file.

Examples of this flow include:

- Bringing in an external clock tree after placement
- Bringing in external optimizations such as new buffers or cell resizing
- Bringing in external post-route fixes such as new buffers or cell resizing

### Preparation

Before starting the flow steps, you should have the following files available:

- `oldchip.enc`  
`oldchip.enc.dat/`

Create these files by using the `saveDesign` command after the previous placement, optimization, and routing steps.

*or*

- `oldchip.conf`  
`oldchip.v`  
`oldchip.def`

Create these files by using the `saveConfig`, `saveNetlist`, and `defOut` commands after the previous placement, optimization and routing steps.

- `newchip.def`

The new DEF file is typically created by an external tool, or possibly done manually to fix a few critical post-route violations with specific placements required. Any necessary physical cells (+SOURCE DIST) are expected to also be in the new DEF.

You must start with the old Verilog and update the Verilog modules with new ports and nets as required to match the new DEF netlist. You need to make sure the DEF instance names match the expected Verilog names (for example, a new buffer added to the output of instance `/i1/i2/i3` should have a name such as `/i1/i2/mynewbuf_i100`), otherwise spurious Verilog ports will be created.

## Flow

1. Read the old floorplan/netlist
2. Compare the old netlist to DEF
3. Load the ECO file
4. Write the modified netlist (optional)
5. Read the new placement data
6. Perform incremental or final routing
7. Trim metal fill

## Steps

1. Read the old Verilog netlist, floorplan, and placement information into Encounter by doing one of the following:

restoreDesign oldchip.enc

or

loadConfig oldchip.conf

defIn oldchip.def

This step reads in the following information:

- Old libraries and global power connections
- Old timing constraints (could be new constraints if necessary)
- Special routing, placements and optionally old routing
- Old filler cells, end caps, well taps, and other cell information

2. Compare the current old netlist to the new DEF netlist to create an ECO file.

ecoCompareNetlist -def newchip.def -outFile oldchip.eco

The ECO file has all the changes required to make the current netlist match the new netlist. Physical-only cells are ignored (for example, +SOURCE DIST cells such as filler, end caps, and well taps). Examine the ECO file to ensure it is correct.

3. Load the ECO file to incrementally update the current netlist to match the new netlist.

loadECO oldchip.eco

During this step,

## Encounter User Guide

### ECO Flows

---

- Instances and nets that are only in the old Verilog are deleted (for example, an old clock tree). Some Verilog ports may now be unconnected due to deleted nets.
- New instances are still unplaced.
- New ports and nets are created in Verilog modules as needed to connect instances in different Verilog modules.
- If any nets are deleted, then any routing attached to the net is also deleted.
- If any nets are modified, then any routing on those nets is left unchanged for later repair.
- Global power connections are done automatically based on the rules from the configuration file or floorplan file loaded earlier.

#### 4. (Optional) Write out the modified Verilog netlist.

```
saveNetlist oldchip_after_eco.v
```

The `oldchip_after_eco.v` file should be the same netlist as `newchip.def`, although it is possible for the net names to be different (any new DEF net names that connect across multiple Verilog modules may be renamed). If you need a DEF file that has exactly the same net names, you can use the `defOut` command.

#### 5. Read in the new placements.

```
defIn newchip.def
```

During this step,

- All instance placements are updated, including unplaced instances. Typically any existing old instances are not moved, but nothing prevents them from moving if the new DEF moved them.
- Remove the `deleteFiller` command before using `defIn` if the new DEF contains different fill, end cap, or well tap cells (+SOURCE DIST). If the filler cells are not changed, the `deleteFiller` command is not necessary. If the new DEF does not have any filler cells, the filler cells (if any) from the old DEF are left in place.
- If any instances are still unplaced, the `ecoPlace` command can be used to place them after removing any notch-fill or metal-fill wiring using the `editDelete` command.
- If only legalization of the placement is needed, the `refinePlace` command can be used.
- If routing is in the new DEF file (typically from the routing done on the old netlist), the routing will also be read in, and it will replace the routing on existing nets.

## Encounter User Guide

### ECO Flows

---

6. Perform incremental or final routing.

[ecoRoute](#)

ecoRoute automatically detects opens and shorts, and incrementally routes any nets that are incomplete or have violations.

7. (Optional) If the original design contained metal fill, trim the metal fill:

[trimMetalFill](#)

Cadence recommends that you use this command to trim metal fill during the ECO process instead of deleting and adding the metal fill again.

8. Continue with the normal post-routing flow (analysis, repair, notch-fill, metal-fill, verify, sign-off, and so forth).

## Pre-Mask ECO Changes from an ECO File

In this flow, your design is placed and possibly routed, and you want to make a small number of changes using an ECO file methodology. The changes are done before the masks are made so it is a pre-mask ECO flow, and there is no need to keep the original poly/diffusion patterns.

For example, you might want to apply a small number of late logical changes, but you want to keep as much of the previous placement, optimization, clock tree, and routing to avoid disturbing previous timing/SI optimization and repair.

### Preparation

Before starting the flow steps, you should have the following files available:

- oldchip.enc  
oldchip.enc.dat /

Create these files by using the `saveDesign` command after the previous placement, optimization, and routing steps.

or

- oldchip.conf  
oldchip.v  
oldchip.def

Create the first three files by using the `saveConfig`, `saveNetlist`, and `defOut` commands after the previous placement, optimization and routing steps. The new Verilog file (`newchip.v`) is typically created by manually editing the old Verilog netlist.

**Note:** If you want to preserve routing, your existing design must contain the antenna diode cells that were added during the previous routing.

or

- oldchip.eco

This file contains the list of changes to be applied to the old netlist. The changes required (see the `loadECO` command syntax) are typically created manually. You might be able to create an ECO file more easily by using `ADDINST` and `DELETEINST` rather than creating a new Verilog file.

## Flow

- Read the old netlist
- Compare netlists
- Load the ECO file
- Write the new netlist (optional)
- Remove filler cells
- Perform incremental placement
- Add filler cells
- Perform incremental or final routing
- Trim metal fill

## Steps

1. Read the old Verilog netlist, floorplan, and placement information into Encounter.

`restoreDesign oldchip.enc`

*or*

`loadConfig oldchip.conf`

`defIn oldchip.def`

This step reads in the following information:

- Old libraries and global power connections
- Old timing constraints or new constraints, if necessary
- Special routing, placement, and optionally, old routing information
- Old filler cells, end caps, well taps, and other cell information

2. Load the ECO file to incrementally update the current netlist to match the new netlist.

`loadECO oldchip.eco`

During this step,

- Instances and nets that are only in the old Verilog are deleted (for example, an old clock tree). Some Verilog ports may now be unconnected due to deleted nets.
- New instances are still unplaced.

## Encounter User Guide

### ECO Flows

---

- ❑ New ports and nets are created in Verilog modules as needed to connect instances in different Verilog modules.
- ❑ If any nets are deleted, the routing attached to the net is also deleted.
- ❑ If any nets are modified, the routing on those nets is left unchanged for later repair.
- ❑ Global power connections are done automatically based on the rules from the configuration file or floorplan file loaded earlier.

#### 3. (Optional) Write out new Verilog netlist.

```
saveNetlist oldchip_after_eco.v
```

The `oldchip_after_eco.v` and `newchip.v` netlists should be identical, with one exception: the newly-created Verilog module ports and nets might have different names because they are automatically generated whenever a new connection is made between separate Verilog modules.

#### 4. Remove filler cells or notch fill (if present).

```
deleteFiller -prefix FILL  
deleteNotchFill
```

#### 5. Perform incremental placement.

```
ecoPlace
```

Unplaced instances are placed; however, previously placed cells are not moved and routing is unaffected.

**Note:** You can manually preplace critical cells before using the `ecoPlace` command by placing the cell in the bottom, left corner, selecting it, and then moving it graphically. For example:

```
placeInstance i1/i2/i3 0 0  
selectInst i1/i2/i3
```

#### 6. Add filler cells back into the rows.

```
addFiller -cell FILL4 -prefix FILL -fillBoundary  
addFiller -cell FILL2 -prefix FILL -fillBoundary  
addFiller -cell FILL1 -prefix FILL -fillBoundary
```

Global power connections are done automatically based on rules loaded from the configuration or floorplan file earlier.

#### 7. Incremental or final route.

```
setNanoRouteMode routeWithEco true      # set for incremental routing  
globalDetailRoute
```

NanoRoute automatically detects opens and shorts, and incrementally routes any nets that are incomplete or have violations.

Or,

You can instead use the [ecoRoute](#) command to perform incremental or final routing

8. (Optional) Add notches.

[fillNotch](#)

9. (Optional) If the original design contained metal fill, trim the metal fill:

[trimMetalFill](#)

Cadence recommends that you use this command to trim metal fill during the ECO process instead of deleting and adding the metal fill again.

10. Continue with the normal post-routing flow (analysis, repair, add metal fill, notch fill, verify, sign-off, and so forth).

## Post-Mask ECO Changes from a New Verilog Netlist

In this flow, the design is taped out and has errors. There is a new Verilog file that only has a few logical changes from the old Verilog file. You want to use pre-existing spare cells so the poly/diffusion and lower layers are not changed, and only the metal and via layer masks need to be modified. To save mask cost, you can direct the software to perform routing changes only on specific layers.

### Preparation

Before starting the flow steps, you must have the following files:

- `oldchip.enc`  
`oldchip.enc.dat/`

Create these files by using the `saveDesign` command after the previous placement, optimization, and routing steps.

*or*

`oldchip.conf`  
`oldchip.v`  
`oldchip.def`

Create these files by using the `saveConfig`, `saveNetlist`, and `defOut` commands after the previous placement, optimization and routing steps.

The old Verilog already has spare cells in it. They are typically added to the original Verilog by creating spare cells at the top-level or inside a Verilog module(s) just to hold the spare cells. For example, a Verilog module named `mySpareCells` is added with as many spare cells as required inside this Verilog module.

You can identify the spare cells before the original placement by using the following command:

`specifySpareGate -inst mySpareCells`

The placer spreads the spare cells evenly throughout the design. If the design is hierarchical, you can add more spare cells inside modules that are likely to change. If the cells are at the top-level with a naming convention such as `SPARE_1`, `SPARE_2`, and so forth, then you can identify them with the following command:

`specifySpareGate -inst SPARE*.`

If you cannot avoid making changes on all layers, you can add spare cells after reading the original Verilog. Create an ECO file containing a list of `ADDINST` commands to create spare cells and `ADDHIERINST` commands to create new Verilog modules. Read the

ECO file with the loadECO command and identify the spare cells by using the specifySpareGate command before placement.

■ **oldchip.fp** (Optional)

You can either save the floorplan in the DEF file or in the floorplan file. If your DEF file does not contain the required floorplan information, you must use `saveFPlan` to generate `oldchip.fp`.

■ **newchip.v**

The new Verilog file is identical to the old Verilog file (including antenna diode cells created during routing) except for a small number of manual fixes for logic errors.

## Flow

1. Read the new netlist
2. Load old floorplan/placement/routing data
3. Specify spare cells
4. Remove notches
5. Perform incremental placement
6. Save the modified design
7. Perform incremental or final route
8. Add notches
9. Trim metal fill

Or,

1. Use the ecoDesign command to perform post-mask ECO operations.

## Steps

1. Read the new netlist.

```
loadConfig newchip.conf # same as oldchip.conf except use newchip.v
```

or

```
loadConfig oldchip.conf 0  
set rda_Input(ui_netlist) "newchip.v"  
commitConfig
```

## Encounter User Guide

### ECO Flows

---

The netlist includes old libraries, global power connections, and so forth. It typically uses old timing constraints, however new timing constraints can be used.

#### 2. Read the old floorplan, special routes, placements and routing from the old netlist files.

```
loadFPlan oldchip.fp #optional  
ecoDefIn -postMask -suffix _spare -reportFile ecoDefIn.rpt oldchip.def  
applyGlobalNets
```

**Note:** You do not need to use the `loadFPlan` command if all floorplan information already exists in the `oldchip.def` file; however, you must use `loadFPlan` if `oldchip.def` contains only placement and routing information.

During this step,

- The `-postMask` option ensures that deleted items are also restored.
- Matching instances get old placements.

Soft matching happens when a DEF net name does not have an equivalent name and another net is found in memory, that has the same connections as described in the DEF.

The most common case for soft matching is when nets have multiple aliases in a hierarchical design “net1” = “inst1/net2” = “inst1/inst2/net3” and so on. Any of these net names can be used in the DEF and can have the same connections.

Without soft matching, net ‘a’, for example, is removed and net ‘b’ is created in its place, resulting in ripping of the wire.

- Any instance existing only in the `oldchip.def` file (deleted cells) is kept in the design, and its name is appended with the string specified by `-suffix`.

For example, if you specify `-suffix _spare`, instance `i1/i2/i3` is changed to `i1/i2/i3_spare`.

- Changed instances (new cell) are assigned a new cell and are left unplaced.
- Physical-only cells in the `oldchip.def` file (marked with `+SOURCE DIST` in the DEF) are added; for example, well taps, end caps, and filler cells.
- Instances that are only in the new netlist are left unplaced.
- Routing for existing or modified nets is restored (possibly with opens or shorts).
- Routing for deleted nets is also restored. The `ecoRoute` command removes the nets according to the `-modifyOnlyLayer` option.
- All unplaced cells are mapped to spare cells during ECO placement in a later step.

**3. (Optional) Make tie connections.**

```
addTieHiLo [-cell "tieHighCellName tieLowCellName"] [-createHierPort {true  
| false}] [-postMask]
```

During this step, the software reuses the existing tie cells to tie off a newly created spare instance in the design, instead of adding or deleting tie cells.

**4. (Optional) Add level shifter or isolation cell for low power design.**

```
loadShifter -infile oldchip.vsf
```

or

```
addIsolationCell
```

During this step, level shifters or isolation cells are added to new ECO nets that cross the power domain. The newly added cells will be unplaced. The `ecoPlace` command will map them to spare cells.

**5. Specify the spare cell list.**

```
specifySpareGate -inst SPARE*
```

**6. Remove notch fill (if present).**

```
deleteNotchFill
```

**Note:** Do not do this step if you plan to freeze metal layers, because this command modifies all layers.

**7. Perform incremental placement.**

```
ecoRemap
```

- The `ecoRemap` command only works for unplaced cells.
- The `ecoRemap` command needs a timing library so that it can compute DRCs and timing slack.
- The `ecoRemap` command might map the unplaced cell to a combination of other cell types to achieve a similar function and yield better DRC and timing results.

```
ecoPlace -useSpareCells true
```

- In the post-mask flow, you must specify `-useSpareCells` to ensure that `ecoPlace` is switched to mapping mode. In this mode, `ecoPlace` maps all unplaced cells to spare cells.
- The `ecoPlace` command automatically chooses a spare cell that is identical to the cell being mapped.

**8. (Optional) Swap spare cells.**

```
ecoSwapSpareCell i_9649 spare1
```

- ❑ At this step of the flow, all the newly-added cells should be mapped. In this step, you can use the `ecoSwapSpareCell` command to manually change the mapping.
- ❑ `i_9649` is the instance name of the placed cell that `ecoSwapSpareCell` swaps with spare cell `spare1`.
- ❑ `spare1` must be a spare cell specified in the previous step. Use the `specifySpareGate` command if necessary.

**9. (Optional) Save the modified design.**

`saveDesign` `design.eco.enc`

Saving the design before you run `ecoRoute` allows you to explore different `modifyOnlyLayers` ranges without repeating all of the previous steps.

**10. Perform incremental or final routing.**

`ecoRoute` `-modifyOnlyLayers 2:3`

- ❑ You can use the `-modifyOnlyLayers` option to restrict the modifications to a specified range of metal layers.
- ❑ If the `-modifyOnlyLayers` range begins with layer 2, and the spare cell pins are only available from metal 1, then the `ecoRoute` command automatically drops a VIA12 via. This behavior is not available if the `-modifyOnlyLayers` range does not begin with 2.
- ❑ The `ecoRoute` command might not be successful if the specified layer range is not sufficient to meet the changes required. You must restore the design from the previous step, then use a different range, such as 2:4, 1:3, and so on.
- ❑ The unused routing segments of deleted and modified nets will appear in the SPECIALNETS section of the DEF file.

**11. (Optional) Add notches.**

`fillNotch`

**Note:** Skip this step if you specify freeze metal layers, because this command modifies all layers.

**12. (Optional) If the original design contained metal fill, trim the metal fill:**

`trimMetalFill`

Cadence recommends that you use this command to trim metal fill during the ECO process instead of deleting and adding the metal fill again.

Or,

## **Encounter User Guide**

### ECO Flows

---

1. Use the `ecoDesign` command to perform post-mask ECO operations. The following command and options are used to implement a post-mask ECO:

```
ecoDesign -postMask -modifyOnlyLayers 2:3 -spareCells *spare* original.enc.dat  
top_cell newchip.v
```

## **Post-Mask Gate Array Style ECO from a New Verilog Netlist**

The design is taped out and has a small number errors. There is a new Verilog netlist that has a few logical changes from the old Verilog netlist. You want to use pre-existing gate array style filler cells that can be programmed with metal layers so the poly/diffusion and lower layers are not changed, and only the metal and via layer masks need to be modified. The new netlist is typically created manually by modifying the old netlist, and any new instances are chosen from cells with a GACORE site.

### **Preparation**

Before starting the flow steps, do the following:

- Create a library of GACORE cells as follows:
  - ❑ All cells have a common transistor pattern.
  - ❑ The cells are a fixed number of CORE sites wide. For example, the width of a GACORE site might be four times the width of a CORE site.
  - ❑ The logical cells are programmed by `meta11` for various AND and OR type gates.
  - ❑ Filler cells use the same transistor pattern (for example, `GAFiller`).
- Pre-place the GACORE filler cells before the final routing in the old design:
  - ❑ Encounter requires an overlayed row pattern for the GACORE cells. Before the final routing (pre-mask), commands such as the following are used:

```
defIn garows.def
addFiller -cell GAFiller -prefix GAFLILL -fillBoundary
addFiller -cell fill2 -prefix FILL -fillBoundary
addFiller -cell fill1 -prefix FILL -fillBoundary
```
- Follow these rules for the Verilog netlist:
  - ❑ Any new instance is only chosen from the GACORE cells.
  - ❑ Any instance can be deleted (it will be left in the netlist as a `sparecell`).

## Encounter User Guide

### ECO Flows

---

Before starting the flow steps, you must have the following files:

- oldchip.enc  
oldchip.enc.dat/

Create these files by using the saveDesign command after the previous placement, optimization, and routing steps.

or

oldchip.conf  
oldchip.v  
oldchip.def

Create these files by using the saveConfig, saveNetlist, and defOut commands after the previous placement, optimization and routing steps.

The old Verilog already has spare cells in it. They are typically added to the original Verilog by creating spare cells at the top-level or inside a Verilog module(s) just to hold the spare cells. For example, a Verilog module named `mySpareCells` is added with as many spare cells as required inside this Verilog module.

You can identify the spare cells before the original placement by using the following command:

`specifySpareGate -inst mySpareCells`

The placer spreads the spare cells evenly throughout the design. If the design is hierarchical, you can add more spare cells inside modules that are likely to change. If the cells are at the top-level with a naming convention such as SPARE\_1, SPARE\_2, and so forth, then you can identify them with the following command:

`specifySpareGate -inst SPARE*.`

If you cannot avoid making changes on all layers, you can add spare cells after reading the original Verilog. Create an ECO file containing a list of ADDINST commands to create spare cells and ADDHIERINST commands to create new Verilog modules. Read the ECO file with the loadECO command and identify the spare cells by using the specifySpareGate command before placement.

- oldchip.fp

You can either save the floorplan in the DEF file or in the floorplan file. If your DEF file does not contain the required floorplan information, you must use saveFPlan to generate `oldchip.fp`.

- newchip.v

The new Verilog file is identical to the old Verilog file (including antenna diode cells created during routing) except for a small number of manual fixes for logic errors.

## Steps

### 1. Read the new netlist.

```
loadConfig newchip.conf # same as oldchip.conf except use newchip.v
```

or

```
loadConfig oldchip.conf 0  
set rda_Input(ui_netlist) "newchip.v"  
commitConfig
```

The netlist includes old libraries, global power connections, and so forth. It typically uses old timing constraints, however new timing constraints can be used.

This procedure reads in the following information:

- The new netlist
- Old libraries and global power connections
- Old timing constraints (could be new constraints if necessary)

### 2. Read the old floorplan, special routes, placements, and routing from the old netlist files.

```
loadFPlan oldchip.fp  
ecoDefIn -useGACells GACORE -suffix _spare -reportFile ecoDefIn.rpt  
oldchip.def  
applyGlobalNets
```

- The -useGACells GACORE option implies -useSpareCells.
- This procedure reads in the following information:
  - Special routing, placements, and old routing
  - Old filler cells, end caps, well taps, and other cell information
- Deleted GACORE cells that are only in the old DEF are deleted (they will be added back by GACORE site filler cells later in the flow).
- Regular standard cells that are only in the old DEF file are implicitly deleted by leaving them in place and changing the name from i1/i2/i3 to i1/i2/i3\_SPARE. The input pins of these new spare cells are tied to the ground net or tie-low cell.
- New GACORE instances are left unplaced.
- Global power connections are made automatically based on the rules from the configuration file or floorplan file loaded earlier.

## Encounter User Guide

### ECO Flows

---

Any GACORE rows in the old design are restored; normal CORE rows are also restored. GACORE rows could optionally come from a separate DEF file if they are not saved with the old design.

#### 3. Specify the spare cell list.

```
specifySpareGate -inst SPARE*
```

#### 4. Remove notch fill (if present).

```
deleteNotchFill
```

**Note:** Do not do this step if you plan to freeze metal layers, because this command modifies all layers.

#### 5. Perform incremental placement.

```
deleteFiller -prefix GAFILL  
ecoPlace -useGACells GACORE  
addFiller -cell -GAFiller -prefix GAFILL -fillBoundary
```

This procedure does the following:

- ❑ Removes GACORE filler cells to leave gaps for the ecoPlace command. The ecoPlace command snaps GACORE cells to the GACORE row sites. Routing is unaffected.
- ❑ Puts back the GACORE filler cells in any leftover gaps.
- ❑ ecoPlace maps cells not matching the GACORE to the available spare cells.
- ❑ ecoPlace places the GACORE cells in a legal placement location.

#### 6. (Optional) Swap spare cells.

```
ecoSwapSpareCell i_9649 spare1
```

- ❑ At this step of the flow, all the newly-added cells should be mapped. In this step, you can use the ecoSwapSpareCell command to manually change the mapping.
- ❑ i\_9649 is the instance name of the placed cell that ecoSwapSpareCell swaps with spare cell spare1.
- ❑ spare1 must be a spare cell specified in the previous step. Use the specifySpareGate command if necessary.

#### 7. (Optional) Save the modified design.

```
saveDesign design.eco.enc
```

Saving the design before you run ecoRoute allows you to explore different modifyOnlyLayers ranges without repeating all of the previous steps.

**8.** Perform incremental or final route.

```
setNanoRouteMode -routeinsertantennadiode false  
ecoRoute -modifyOnlyLayers 2:3
```

During this step,

- ❑ NanoRoute automatically detects opens and shorts, and incrementally routes any nets that are incomplete or have violations.
- ❑ Disable insertion of antenna diode cells. The poly/diffusion layers cannot be modified, so only layer-hopping can be used to avoid process antenna violations.
- ❑ You can use the `-modifyOnlyLayers` option to restrict the modifications to a specified range of metal layers.
- ❑ If the `-modifyOnlyLayers` range begins with layer 2, and the spare cell pins are only available from metal 1, then the `ecoRoute` command automatically drops a VIA12 via. This behavior is not available if the `-modifyOnlyLayers` range does not begin with 2.
- ❑ The `ecoRoute` command might not be successful if the specified layer range is not sufficient to meet the changes required. You must restore the design from the previous step, then use a different range, such as 2:4, 1:3, and so on.
- ❑ The unused routing segments of deleted and modified nets will appear in the SPECIALNETS section of the DEF file.

**9.** (Optional) Add notches.

fillNotch

**Note:** Skip this step if you specify freeze metal layers, because this command modifies all layers.

**10.** (Optional) If the original design contained metal fill, trim the metal fill:

trimMetalFill

Cadence recommends that you use this command to trim metal fill during the ECO process instead of deleting and adding the metal fill again.

Or,

1. Use ecoDesign for post-mask flow.

The following command and options are used to implement a post-mask ECO:

```
ecoDesign -postMask -modifyOnlyLayers 2:3 -spareCells *spare* original.enc.dat  
myDesign myDesign.new.v
```

---

## ECO Directives

---

This appendix describes the directives that you specify in an ECO directives file. After you complete the file, you can then read it into the Encounter™ software by using the `loadECO` command on the Encounter command line. The following command loads `myDirectivesFile`:

```
loadECO myDirectivesFile
```



*Important*

These are ECO directives, not Encounter Tcl commands.

- You can use these directives only in an ECO directives file.
- You cannot use these directives on the Encounter command line.
- You cannot source this file to run the directives.
- You must use the `loadECO` command to read the file.

The names of the directives appear in this appendix in uppercase characters to distinguish them from interactive Encounter commands with the same names; however, the software is case-insensitive.

The ECO File directives do not support Verilog® escape name syntax. For directives that modify or delete existing objects, you cannot specify the name of the object using Verilog escape name syntax.

File format requirements are shown in the section [Example ECO File](#) on page 1279.

The directives are presented in alphabetical order.

- [ADDHIERINST](#) on page 1259
- [ADDINST](#) on page 1260
- [ADDMODULEPORT](#) on page 1262
- [ADDNET](#) on page 1264

## **Encounter User Guide**

### ECO Directives

---

- [ATTACHMODULEPORT](#) on page 1265
- [ATTACHTERM](#) on page 1266
- [CHANGEINSTNAME](#) on page 1268
- [DELETEBUFFER](#) on page 1269
- [DELETEINST](#) on page 1271
- [DELETEMODULEPORT](#) on page 1272
- [DELETENET](#) on page 1273
- [DETACHMODULEPORT](#) on page 1274
- [DETACHTERM](#) on page 1275
- [INSERTBUFFER](#) on page 1276
- [Example ECO File](#) on page 1279

## **ADDHIERINST**

```
ADDHIERINST  
  instName  
  moduleName
```

Creates an instance of a new hierarchical module. You can later use the ADDINST directive to add spare cells inside the new hierarchical instance. The software automatically creates new ports for the hierarchical module when you run the ATTACHTERM directive. Currently, there are no directives available that allow you to manually create new ports for the new hierarchical module.

### **Parameters**

<i>instName</i>	Specifies the name of the new hierarchical instance. If the instance already exists, or if the module containing the instance does not exist, the directive stops and the software displays an error message.
<i>moduleName</i>	Specifies the name of the new hierarchical module. If the module already exists, the software uses the module definition and creates a new hierarchy.

### **Example**

- The following directive creates a new hierarchical cell, sparecell, and an instance of sparecell named i1/i2/i3/spare1:

```
ADDHIERINST i1/i2/i3/spare1 sparecell
```

If the instance i1/i2/i3 does not exist, or instance i1/i2/i3/spare1 already exists, or cell sparecell already exists, the directive stops and the software displays an error message.

## ADDINST

```
ADDINST
  [-moduleBased verilogModule]
  [-physical]
  -cell cellName
  -inst instName
  [-loc lx ly [-ori orient]]
```

Adds an instance and places it in the design.

When *-inst* is specified, the new instance is bound with the correct cell in the power domain.

When *-inst*, and *-loc* are specified, the location also gives a power domain, which is checked against the power domain of the instance. If the power domains are different, a warning is issued.

### Parameters

*-cell cellName*

Specifies the master of the instance.

*-inst instName*

Specifies the name of the instance to add and place.

*-loc lx ly*

Specifies the location of the placed instance.

*Default:* If you do not specify a location, Encounter places the instance at the design origin.

*-moduleBased verilogModule*

Adds the new instance to the specified verilog module.

*-ori orient*

Specifies orientation of the placed instance.

*Default:* If you do not specify an orientation, Encounter uses R0.

*-physical*

Places a physical instance without updating the netlist.

### Example

- The following directive adds buffer instance i1/i2 at location 100, 200:

## **Encounter User Guide**

### ECO Directives

---

```
ADDINST -cell BUFL -inst i1/i2 -loc 100 200
```

- The following directive adds buffer instance `insta` to the verilog module `HIER_2`:

```
ADDINST -moduleBased HIER_2 -cell BUFX2_6 -inst insta
```

## ADDMODULEPORT

```
ADDMODULEPORT
  [-moduleBased verilogModule]
  moduleName | `-
  portName
  {input | output | bidi}
  [-bus n1:n2]
```

Adds a port or a bussed port to a module.

### Parameters

*-bus n1:n2*

Adds a bussed port to the module. Specify the bus range (the beginning and end of the bus). Use integers to specify the range.

*-moduleBased verilogModule*

Adds a new port to the specified verilog module.

*moduleName* | `-

Specifies the module to which you want to attach the port. To specify the top module, enter `-'.

*portName*

Specifies the name of the port to be added.

The specified *portName* can be a new port name or any of the existing net names to which you want to add the port.

The new port name can be the same as the existing net name. This port is attached directly to the existing net name if you specify the port direction (scalar port).

*input | output | bidi*

Specifies whether the port is input, output, or bidirectional.

### Examples

- The following directive creates an input port p1 on instance i1/i2/i3. The port p1 must be a new port name in instance i1/i2/i3. Instance i1/i2/i3 contains no nets name p1:

```
ADDMODULEPORT i1/i2/i3 p1 input
```

## Encounter User Guide

### ECO Directives

---

- The following set of directives adds a hierarchical block and then adds a bussed port to the block.

```
ADDHIERINST -cell i_block1 -hinst i_block1  
ADDMODULEPORT i_block data output -bus 31:0
```

- The following directive adds an output port, testPort to the verilog module hier\_3:

```
ADDMODULEPORT -moduleBased hier_3 testPort output
```

## ADDNET

```
ADDNET
  [-moduleBased verilogModule]
  netName
  [-physical]
  [-bus startID:endID]
```

Adds a net to the design. The net can be logical or physical.

### Parameters

-bus *startID:endID*

Creates a bussed Verilog net. The *startID* and *endID* indicate the first and last bits on the bus. You must separate the start and end IDs with a colon (:).

-moduleBased *verilogModule*

Adds the new net to the specified verilog module.

*netName*

Specifies the name of the net to add. If the module containing the net does not exist, or if the net already exists, the software displays an error message and the directive stops.

-physical

Adds a physical net.

### Example

- The following directive adds net i1/i2/net26 to the netlist:

```
ADDNET i1/i2/net26
```

If the module i1/i2 does not exist, or if the net i1/i2/net26 already exists, the software displays an error message and the directive stops.

## ATTACHMODULEPORT

```
ATTACHMODULEPORT
  {moduleName | '-'}
  portName
  netName
```

Attaches a port in the specified instance (or top level) to a net.

### Parameters

<i>moduleName</i>   '-'	Specifies the module to which you want to attach the port. To specify the top module, enter '-'.
<i>netName</i>	Specifies the net to which you want to create to attach to the port.
<i>portName</i>	Specifies the port you want to create on the module.

### Examples

- The following directives create a port p1 on instance i1/i2/i3 and a net i1/i2/n1. The ATTACHMODULEPORT directive then connects the created port p1 on instance i1/i2/i3 to the net i1/i2/n1:

```
ADDMODULEPORT i1/i2/i3 p1
ADDNET i1/i2/n1
ATTACHMODULEPORT i1/i2/i3 p1 i1/i2/n1
```

- The following directive attaches port in on the top module to net123:

```
ATTACHMODULEPORT - in net123
```

## ATTACHTERM

```
ATTACHTERM
  [-moduleBased verilogModule]
  [-noNewPort]
  instName
  termName
  netName
  [-port portName | -pin refInstName refPinName]
```

Attaches a terminal to a net. If the terminal already connects to the net, the software first detaches the terminal from the current net, then attaches it to the new net.

### Parameters

<i>instName</i>	Specifies the instance containing the terminal. If the instance name does not exist, the software displays an error message and the directive stops.
-moduleBased <i>verilogModule</i>	Attaches the terminal to the specified verilog module.
<i>netName</i>	Specifies the name of the net to attach to the terminal. If the net name does not exist, Encounter displays an error message and the directive stops.
-noNewPort	Prohibits Encounter from creating hierarchical ports when it attaches a terminal. If the terminal cannot connect to the net through existing ports, Encounter displays an error message and the directive stops. <i>Default:</i> If you do not specify this parameter, Encounter creates hierarchical ports as needed.
-pin <i>refInstName</i> <i>refPinName</i>	Specifies the pin <i>refPinName</i> on instance <i>refInstName</i> connected to the net that Encounter connects to the terminal. You cannot specify the -port parameter if you use this parameter.

`-port portName`

Specifies the hierarchical port used to connect the terminal with the net. If you specify this parameter, the hierarchical port must exist in the module that contains the instance. The hierarchical port must connect to the net. This parameter lets you use a specific port to maintain the same netlist topology, which simplifies equivalence checking later in the design flow. You cannot specify the `-pin` parameter if you use this parameter.

*Default:* If you do not specify this parameter, Encounter uses existing ports or creates new hierarchical ports as necessary to connect the terminal to the net.

`termName`

Specifies the name of the terminal that Encounter connects to the specified net. If the terminal name does not exist, Encounter displays an error message and the directive stops.

## Examples

- The following directive attaches terminal `in1` of instance `i1/i2/i3` to net `i1/i2/net26`:

```
ATTACHTERM i1/i2/i3 in1 i1/i2/net26
```

If `i1/i2/i3` does not exist, or if `in1` is not a terminal of `i1/i2/i3`, or if `i1/i2/net26` does not exist, the software displays an error message and the directive stops.

- The following directive attaches terminal `in1` of instance `i1/i2/i3` to net `net27`, using hierarchical port `myPort`:

```
ATTACHTERM i1/i2/i3 in2 net27 myPort
```

The `myPort` port must exist in the module definition for `i1/i2`, and `myPort` must already connect to `net27`. If this is not the case, the software displays an error message and the directive stops.

- The following directive attaches terminal `in3` on instance `i1/i2/i3` through existing ports to `net28`:

```
ATTACHTERM -noNewPorts i1/i2/i3 in3 net28
```

If ports do not exist, the software displays an error message and the directive stops.

- The following directive attaches terminal `Y` of instance `testInst` to the verilog module `hier_t3` using the port `testPort`:

```
ATTACHTERM -moduleBased hier_t3 testInst Y testPort
```

## **CHANGEINSTNAME**

CHANGEINSTNAME  
  -*inst instName*  
  -*newBaseName baseName*

Changes the base name of the specified instance to the given base name.

**Note:** The CHANGEINSTNAME directive does not change the path of hierarchy.

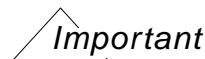
### **Parameters**

-*inst instName*      Specifies the name of the instance.  
-*newBaseName baseName*  
                            Specifies the new base name to use for the instance.

## DELETEBUFFER

```
DELETEBUFFER  
  instName  
  keepNetName  
  [deleteNetName]
```

Deletes a buffer instance after merging the nets on both sides of the buffer into one net.



This directive has been replaced by [ecoDeleteRepeater](#) command.

### Parameters

<i>deleteNetName</i>	Specifies the name of the net connected to the terminal on <i>instName</i> opposite to the terminal that connects to <i>keepNetName</i> . If <i>deleteNetName</i> is not connected to the terminal on <i>instName</i> opposite to the terminal that connects to <i>netName</i> , the software displays an error message and the directive stops. For example, if <i>keepNetName</i> connects to the input of <i>instName</i> , <i>deleteNetName</i> specifies the name of the net connecting to the output. The software uses the <i>deleteNetName</i> for error checking only. When the DELETEBUFFER directive merges the nets, it might detach connections to hierarchical ports, but does not change the direction of hierarchical ports.
<i>instName</i>	Specifies the name of the buffer instance that the DELETEBUFFER directive deletes. The instance must have exactly one input terminal and one output terminal, and one of the terminals must connect to <i>keepNetName</i> .
<i>keepNetName</i>	Specifies the name of the existing net into which the nets from both of the instance's terminals are merged. The <i>keepNetName</i> net must connect to one of the instance's terminals.

### Example

- The following directive deletes buffer *i1/i2/i3* and merges the nets from the buffer's two terminals into net *net26*:

```
DELETEBUFFER i1/i2/i3 net26 i1/net25
```

## **Encounter User Guide**

### ECO Directives

---

Net `net26` already connects to one of the instance's terminals. Net `i1/net25` connects to the terminal opposite the terminal that connects to net `net26`. If buffer `i1/i2/i3` does not exist, or if net `net26` and net `i1/net25` are not already attached to two terminals of buffer `i1/i2/i3`, the software displays an error message and the directive stops.

## **DELETEINST**

```
DELETEINST  
  instanceName  
  [-moduleBased verilogModule]
```

Deletes an instance after deleting all the instance terminal connections to nets.

### **Parameters**

*instName*      Specifies the name of the instance to delete. If the specified instance does not exist, the software displays an error message and the directive stops.

-moduleBased *verilogModule*  
                 Deletes the instance from the specified verilog module.

### **Example**

- The following directive deletes instance i1/i2/i3:

```
DELETEINST i1/i2/i3
```

If instance i1/i2/i3 does not exist, the software displays an error message and the directive stops.

- The following directive deletes the instance *insta* from the verilog module HIER\_2:

```
DELETEINST -moduleBased HIER_2 insta
```

## **DELETEMODULEPORT**

```
DELETEMODULEPORT
  moduleName | '-'
  portName
  netName
```

Disconnects the specified portname from its net and deletes the port.

You can use wildcards (\*?) to specify the nets you want Encounter to delete.

### **Parameters**

<i>moduleName</i>   '-'	Specifies the module from which you want to delete the port. To specify the top module, enter '-'.
<i>netName</i>	Specifies the name of the net from which you want to delete the port.
<i>portName</i>	Specifies the name of the port to be deleted

### **Example**

- The following directive deletes `port1` from the top-level module:

```
DELETEMODULEPORT - port1
```

## **DELETENET**

```
DELETENET  
  netName  
  [-moduleBased verilogModule]
```

Deletes a net after deleting all the instance terminal connections to the net. If routing is connected to the net, the routing is deleted.

You can use wildcards (\*?) to specify the nets you want Encounter to delete.

### **Parameters**

*-moduleBased verilogModule*

Deletes the net from the specified verilog module.

*netName*

Specifies the name of the net to delete.

### **Example**

- The following directive deletes net i1/i2/net26:

```
DELETENET i1/i2/net26
```

If net i1/i2/net26 does not exist, the software displays an error message and the directive stops.

# DETACHMODULEREPORT

**DETACHMODULEPORT**  
    *moduleName*  
    *portName*

Detaches the net connected to the specified port on the specified instance.

## Parameters

<i>moduleName</i>	Specifies the module from which you want to detach the net. To specify the top module, enter '-'.
<i>portName</i>	Specifies the port on the module.

## Example

- The following directive detaches port p1 from moduleA:

```
DETACHMODULEPORT moduleA p1
```

# DETACHTERM

```
DETACHTERM  
    [-moduleBased verilogModule]  
    instName  
    termName  
    [netName] 
```

Disconnects a terminal from a net.

**Note:** Detaching a terminal that drives an output terminal of a module produces a Verilog violation at the output terminal if you use DETACHTERM. Instead, use ATTACHTERM to attach the terminal to a new net. The ATTACHTERM directive automatically detaches the terminal from the net connecting to the output terminal, then attaches the terminal to the net you specify.

## Parameters

*instName*      Specifies the instance that contains the terminal you want to detach.

**-moduleBased verilogModule**  
Detaches the terminal from the verilog module.

*netName* Specifies the net that is already connected to the terminal. If the terminal does not connect to the specified net, or if the net does not exist, the software displays an error message and the directive stops. The software uses this parameter for error checking only.

*termName*      Specifies the terminal to disconnect. If the terminal does not exist on the specified instance, the software displays an error message and the directive stops.

## Example

- The following directive disconnects terminal `in1` of instance `i1/i2/i3`:

DETACHTERM i1/i2/i3 in1 i1/i2/net26

If instance i1/i2/i3 does not exist, or terminal in1 is not a terminal of i1/i2/i3, the software displays an error message and the directive stops. The net i1/i2/net26 is specified, so if net i1/i2/net26 does not exist, or if the terminal is not already connected to net i1/i2/net26, the software displays an error message and the directive stops.

## INSERTBUFFER

```
INSERTBUFFER
  [-noNewPorts]
  netName
  nrNetTerm
  nrBuffer

INST instName cellName nrInstTerm
INSTTERM termName netName [portName]
...
NETTERM instName termName netName
...
```

Inserts a buffer on a net.



This directive has been replaced by [insertRepeater](#) command.

**Note:** You must specify the INST, INSTTERM, and NETTERM directives in the order given in the syntax. You must enter each of these directives on its own line, at the beginning of that line. You can add these directives only after you have specified the [-noNewPorts] *netName* *nrNetTerm* *nrBuffer* parameters.

### Parameters

<i>netName</i>	Specifies the name of the net on which to insert the buffer. You must specify this parameter.
-noNewPorts	Specifies that Encounter must not add new ports when inserting the buffer. If you try to insert a buffer that needs a new port, Encounter issues an error message and does not insert the buffer.
<i>nrBuffer</i>	Specifies the number of buffers attached to the net.
<i>nrNetTerm</i>	Specifies the original number of terminals attached to the net.
INST	Specifies a buffer instance. You must specify the INST directive for each buffer you want to add.
<i>instName</i>	Specifies the name of the buffer instance to insert.

## Encounter User Guide

### ECO Directives

---

	<i>cellName</i>	Specifies the cell master for the buffer instance.
	<i>nrInstTerm</i>	Specifies the number of instance terminals contained in the buffer. Buffers have one input and one output terminal, so specify 2.
INSTTERM		Specifies a terminal on a buffer instance. The <i>termName</i> and <i>netName</i> parameters are required. you must specify the INSTTERM directive for each buffer you want to add.
	<i>termName</i>	Specifies the name of the terminal on the buffer.
	<i>netName</i>	Specifies the net to connect with the terminal.
	<i>portName</i>	Specifies the physical port corresponding to the terminal.
NETTERM		Specifies the net connection between a driver terminal and the added buffer.
	<i>instName</i>	Specifies the instance containing the terminal.
	<i>netName</i>	Specifies the net connected to the terminal.
	<i>termName</i>	Specifies the terminal connected to the net.

### Example

- The following directives insert three buffers, b1, b2, and b3, on net0, which originally connects terminal out on instance i0 to receivers i1, i2, and i3. After the three buffers are added, terminal out drives one terminal: b1/in.

```
INSERTBUFFER net0 4 3
    INST b1 buffer 2
        INSTTERM in net0
        INSTTERM out net1
    INST b2 buffer 2
        INSTTERM in net1
        INSTTERM out net2
    INST b3 buffer 2
        INSTTERM in net1
        INSTTERM out net3
    NETTERM i0 out net0
```

## Encounter User Guide

### ECO Directives

---

```
NETTERM i1 in net2
NETTERM i2 in net1
NETTERM i3 in net3
```

- Buffer b1 has terminal in, connected to net0 and out, connected to net1. Buffer b1 drives buffers b2 and b3, and connects to receiver i2.
- Buffer b2 has terminal in, connected to b1 through net1, and out, connected to receiver i1 through net2.
- Buffer b3 has terminal in, connected to b1 through net1, and out, connected to receiver i3 through net3.

## Example ECO File

The file format consists of directives, each ending with a newline. The keywords are case insensitive.

Comments must begin with a pound symbol (#) as the leading, non-white space character, and end with a newline.



The first directive in the file must be FORMATVERSION 2.

```
#  
# FORMATVERSION  
#  
FORMATVERSION 2  
  
#  
# ADDINST: Add at top level, no connectivity  
#  
ADDINST eco_inst_19 BUFX1  
  
#  
# ADDINST: Add at block level, no connectivity  
#  
ADDINST DTMF_INST/TDSP_CORE_INST/eco_inst_1 BUFX1  
  
#  
# ADDINST: Add at top level, with connectivity  
#  
ADDINST eco_inst_2341 BUFX1 2  
INSTTERM A test_mode  
INSTTERM Y reset  
  
#  
# ADDINST: Add at block level, with connectivity  
#  
ADDINST DTMF_INST/TDSP_CORE_INST/eco_inst_3 BUFX1 2  
INSTTERM A scan_en  
INSTTERM Y reset  
  
#
```

## Encounter User Guide

### ECO Directives

---

```
# DELETEINST: Delete block level instance
#
DELETEINST DTMF_INST/m_clk__L6_I6

#
# ADDNET: Add new top level net
#
ADDNET eco_new_top_net

#
# ADDNET: Add new block level net
#
ADDNET DTMF_INST/eco_new_block_net

#
# DELETENET: Delete top level net
#
DELETENET n_7875

#
# DELETENET: Delete block level net
#
DELETENET DTMF_INST/TDSP_CORE_INST/ALU_32_INST/n_1496

#
# ATTACHTERM: Attach block level inst term to existing net
#
ATTACHTERM DTMF_INST/TDSP_CORE_INST/ACCUM_STAT_INST/i_9529 A DTMF_INST/
Tdsp_CORE_INST/ACCUM_STAT_INST/n_73

#
# DETACHTERM: Detach block level inst term
#
DETACHTERM DTMF_INST/TDSP_CORE_INST/ACCUM_STAT_INST/i_9529 A

#
# ADDHIERINST: create a new module + inst
#
ADDHIERINST DTMF_INST/ECO_NEW_HIER_INST ECO_NEW_HIER

#
```

---

# Clock Mesh Specification File

---

- [Overview](#) on page 1281
- [Routing Type Definitions](#) on page 1282
- [Cutout Definitions](#) on page 1282
- [Clock Mesh Definitions](#) on page 1283
- [Clock Mesh Specification File Example](#) on page 1302

## Overview

Before running clock mesh synthesis, you must create a clock mesh specification file. The clock mesh specification file defines the clock meshes to be created for the design.

A clock mesh specification file contains the following main sections:

- [Routing Type Definitions](#) on page 1282
- [Cutout Definitions](#) on page 1282
- [Clock Mesh Definitions](#) on page 1283

You can specify the following units in a clock mesh specification file:

- Distance unit:  $\mu\text{m}$
- Time unit: ps, ns
- Voltage unit: v, mv
- Power unit: w, mw

If you do not specify units in the file, the clock mesh tool generates warning messages.

## Routing Type Definitions

The routing type definition is a set of routing properties to be used during clock mesh implementation. You must define at least two routing types: one for vertical and one for horizontal mesh trunks or branches. The routing types are then referenced in the clock mesh definition.

The following table describes the entries for the routing type definition:

RouteTypeDef <i>typeName</i>	Specifies the routing type for which you are defining routing attributes. This attribute denotes the beginning of the RouteTypeDef section.
Layer <i>layerName</i>	Specifies the metal layer to be used. The name you specify must be a layer name defined in the LEF file.
Width <i>distance</i>	Specifies the width of the metal layer to be used.
Spacing <i>distance</i>	Specifies the spacing to all other wires in the design. <i>Default:</i> Uses the minimum spacing values specified in the LEF file
End	Denotes the end of the RouteTypeDef section.

## Cutout Definitions

The cutout definition section specifies areas that should not be covered by the clock mesh. When you specify a cutout area, the clock mesh tool must stop mesh routing at the cut out boundary. If you want to completely avoid an area, use a placement or routing blockage.

The Cutout definition section is optional.

You can define a cutout area in one of the following ways:

- X Y coordinates

You can specify a set of upper-right and lower-left X and Y coordinates to define the cutout area. For example:

```
Cutout
+ 0µm 0µm 400µm 280µm
```

- Instance names

You can specify an instance name on which to base the cutout area. The cutout area then covers the area of the instance. For example:

```
Cutout
+ INST1/C1
+ INST2/C1
```

■ **Instance names with instance halos**

You can define a cutout area by specifying the distance in microns for which to increase the size of instance-based cutouts from the instance-cell boundary. For example:

```
Cutout
+ INST1/C1 HALO 50μm
```

**Note:** You also can use the [createClockMeshCutout](#) command to create cutout areas.

## Clock Mesh Definitions

Each clock mesh definition defines a single clock mesh to be synthesized. A clock mesh specification file can contain multiple clock mesh definitions.

`ClockMesh meshName`

Specifies the name of the clock mesh to be synthesized. Each clock mesh definition in the clock mesh specification file must start with a `ClockMesh` statement.

`End`

Marks the end of a clock mesh definition. Each clock mesh definition in the clock mesh specification file must close with an `End` statement.

A clock mesh definition can contain the following information sections:

- [Timing and Power Constraints Section](#) (required)
- [Tracing and Analysis Scope Section](#) (required)
- [Mesh Structure Section](#) (required)
- [Global Mesh Section](#)(required)
- [Top Chain Section](#) (optional)
- [Local Tree Section](#) (optional)

## Timing and Power Constraints Section

This section defines timing and power constraints for the clock mesh.

The following table describes the entries for the timing and power constraints section:

Period <i>timeValue</i>	Specifies the clock period (in picoseconds) for the mesh.  <i>Default:</i> 0 (frequency also will default to 0)
MaxPower <i>timeValue</i>	Specifies the maximum power value (in milliwatts) for the mesh.  <b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.  <i>Default:</i> 0
RootTrans <i>timeValue</i>	Specifies the transition time (in picoseconds) at the root pin.  <i>Default:</i> 0
MinDelay <i>timeValue</i>	Specifies the minimum phase delay (in picoseconds).  <b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.  <i>Default:</i> 0
MaxDelay <i>timeValue</i>	Specifies the maximum phase delay (in picoseconds).  <b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.  <i>Default:</i> Largest possible number, based on the machine
MaxSkew <i>timeValue</i>	Specifies the maximum skew between sink pins (in picoseconds).  <b>Note:</b> Currently, the clock mesh tool does not use this value when implementing the mesh.  <i>Default:</i> 0
MaxBufferTrans <i>timeValue</i>	

Specifies the maximum input transition time for buffers (in picoseconds).

**Note:** Currently, the clock mesh tool does not use this value when implementing the mesh.

*Default:* 0

`MaxLeafTrans timeValue`

Specifies the maximum input transition time for leaf pins (in picoseconds).

**Note:** Currently, the clock mesh tool does not use this value when implementing the mesh.

*Default:* 0

## Tracing and Analysis Scope Section

This section specifies the logical extent of the clock domain, from the root through any mesh drivers to the leaves. Scope is determined by tracing the netlist from the clock root.

The following table describes the entries for the tracing and analysis scope section:

`RootPin instanceName/pinName`

Specifies the complete name of the clock root pin from which the tracing starts.

`LeafPin +instanceName/pinName {rising | falling}`

Defines the specified input pin as a leaf pin for non-clock type instances. The clock mesh tool will stop tracing at this pin, and skew is analyzed only to this pin. This statement also defines the rising or falling trigger edge for the inputs.

For example:

```
LeafPin  
+corem/sync1/reg_3/D rising  
+corem/sync1/reg_4/D rising
```

`LeafCellPin +cellType/pinName {rising | falling}`

Defines the specified input pin as a leaf pin for non-clock type cells. The clock mesh tool will stop tracing at this pin, and skew is analyzed only to this pin. This statement also defines the rising or falling trigger edge for the inputs.

For example:

```
LeafCellPin  
+AND2X/A rising
```

```
DefaultTrigger {rising | falling}
```

Specifies the default trigger edge value for leaves that do not have a specified trigger edge value.

```
AllowGating {true | false}
```

Specifies whether the clock mesh tool traces through gates. If you specify `true`, the tool traces until it finds leaf pins. If you specify `false`, the tool stops at gate inputs.

*Default:* `false`

## Mesh Structure Section

This section defines the overall logical attributes of the clock mesh structure.

The following table describes the entries for the mesh structure section:

```
UseMeshModule [true | false]
```

Specifies whether the clock mesh tool should create a separate module for the mesh buffers. If you specify `true`, the tool creates a module at the level of the root net. If you specify `false`, the tool inserts buffers (flat) at the level of the root net.

*Default:* `true`

```
MeshModule moduleName
```

Specifies the module name to be used when synthesizing the clock mesh.

**Note:** The `UseMeshModule` statement must be set to `true` in order for this statement to apply.

*Default:* The clock mesh tool creates a name based on the mesh name.

## Encounter User Guide

### Clock Mesh Specification File

---

```
LoadCell + cellName1 [+ cellName2] ...
```

Defines the specified cell as a loading cell that can be inserted into the final stage global mesh net, to minimize skew for the global mesh. You can specify buffers, inverters, or single input cells that do not have an output pin.

For example:

```
LoadCell  
+ T2BUFCLXR  
+ CLKBUFX16
```

## Global Mesh Section

This section defines physical attributes of the global clock mesh structure.

The following table describes the entries for the global mesh section:

GlobalMesh	Denotes the beginning of the global mesh definition. The global mesh definition must begin with a GlobalMesh statement, and close with an End statement.
------------	--

MeshDrivePoint {Center   Root   x,y}	Specifies the position of the first-stage pre-drivers.
--------------------------------------	--

If you do not specify this statement, the software positions the first-stage pre-drivers as follows:

- For HTreeMesh and Fishbone meshes, positions the pre-drivers near the center of the clock mesh structure (Center)
- For CTS-generated pre-drive structures, positions the pre-drivers based on whether there is a TopChain section specified in the clock mesh specification file. If there is a TopChain section in the spec file, the drive point will be the center of the mesh structure (Center). If there is no TopChain section defined, the drive point will be the clock root pin (Root).

Center	Positions the pre-drivers near the center of the clock mesh structure.
--------	--

# **Encounter User Guide**

## Clock Mesh Specification File

Root	Positions the pre-drivers near the clock root pin.
$x, y$	Positions the pre-drivers at the specified location.

MeshType {Fishbone | HTreeMesh}

Specifies the type of mesh structure to be synthesized.

**Note:** You must specify the `MeshType` statement if you want to synthesize a clock mesh. If you only want to generate a report, the statement is optional.

TrunkOrientation [Horizontal | Vertical]

Specifies the trunk orientation.

**Note:** You must specify the TrunkOrientation statement if you want to synthesize a clock mesh. If you only want to generate a report, the statement is optional.

TrunkPlacement {UniformPitch | LoadWeighted}

Specifies how trunks are placed if pre-defined target locations are not specified using the `TargetTrunkLocs` statement in the mesh stage section.

**UniformPitch** Places the trunks uniformly according to trunk pitch.

**LoadWeighted** Places each trunk so that it drives a similar load.

`HTreePattern pattern`

Specifies the order for building horizontal (H) and vertical (V) structures for an HTree + Mesh clock mesh.

You can use `H` and `V` in any pattern to specify the order. Specifying an asterisk (\*) causes the tool to repeat alternating structures. For example, specifying `H*` is equivalent to specifying `HVHV`; it is *not* equivalent to specifying `HHHH`.

**Note:** You only can use this statement when you specify MeshType HTreeMesh.

```
TrunkDriveDist {StrictAttach | Uniform | LoadWeighted  
| LoadWeightedMatch}
```

## Encounter User Guide

### Clock Mesh Specification File

---

Specifies how buffers are placed along driving trunks in the final global mesh stage.

`StrictAttach`    Places the buffers according to the specified `BranchAttachFrequency` value.

`Uniform`    Places the buffers in an even, uniform distribution along the trunk.

**Note:** If you specify `Uniform`, the clock mesh tool ignores the `BranchAttachFrequency` value.

`LoadWeighted`    Places the buffers according to load distribution. The `BranchAttachFrequency` value limits how closely the buffers can be placed to each other.

`LoadWeightedMatch`

Places the drivers according to load distribution, and applies the same pattern to all trunks. The `BranchAttachFrequency` value limits how closely the buffers can be placed to each other.

`PreDriveCTS`    Denotes the beginning of the CTS-generated pre-drive structure definition. The pre-drive structure definition must begin with a `PreDriveCTS` statement, and close with an `End` statement.

`Enabled` [true | false]

## Encounter User Guide

### Clock Mesh Specification File

---

Enables the implementation of the pre-drive structure.

When you specify `true`, the clock mesh tool calls CTS to synthesize the clock tree using the last stage of the global mesh drivers as the leaf pin.

The position of the first-level driver in the CTS pre-drive structure is determined by the `MeshDrivePoint` statement in the clock mesh spec file. If the `MeshDrivePoint` statement is not defined, the position depends on whether a `TopChain` section is specified. If there is a `TopChain` section in the spec file, the drive point will be the center of the mesh structure; if there is no `TopChain` section defined, the drive point will be the clock root pin.

By default, the clock mesh tool decides the number of levels for the pre-drive structure.

*Default:* `false`

`DriveCells + cellName1 + cellName2 . . .`

Specifies the types of drive cells to use for the pre-drive structure. You must specify at least one cell choice. Multiple drive cells can be specified.

For example:

```
DriveCells  
+ CLKBUFX20  
+ CLKBUFX16
```

`NonDefaultRule ruleName`

Specifies the LEF `NONDEFAULTRULE` statement for the router to use when routing the nets in the pre-drive structure.

*Default:* The router uses the default routing rule.

`TopPreferlayer layerName`

Specifies the top preferred metal layer to use for routing the pre-drive structure. The name you specify must be a layer defined in the LEF file.

`BottomPrefLayer layerName`

Specifies the bottom preferred metal layer to use for routing the pre-drive structure. The name you specify must be a layer defined in the LEF file.

## **Encounter User Guide**

### Clock Mesh Specification File

---

`PreferredExtraSpace [0 - 3]`

Specifies the extra space to add around clock wires when routing the pre-drive structure.

*Default:* 1

`End`

Denotes the end the pre-drive structure definition. The pre-drive structure definition must begin with a `PreDriveCTS` statement, and close with an `End` statement.

`HTreeSplitDrive [true | false]`

## Encounter User Guide

### Clock Mesh Specification File

---

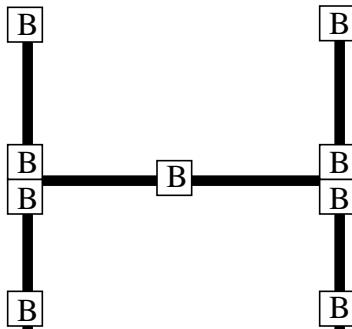
Splits branches at the driving point. Splitting a branch can increase the drive strength without creating a multi-drive net. To split a branch, you must have an even number of drivers at the branching point (for example, 2, 4, 6).

For example, if you define the following in the clock mesh spec file:

```
ClockMesh Clk
...
GlobalMesh
    HTreeSplitDrive true
    Stage
        NumDriver 1
    End
    Stage
        NumDriver 4
        X2
    End
    Stage
        NumDriver 4
    End
...

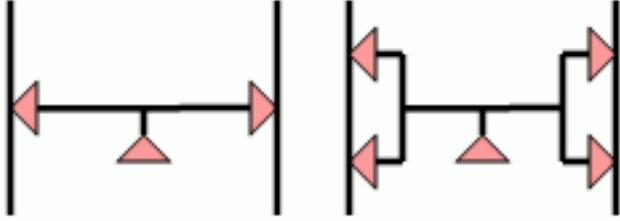
```

The software splits the branches as follows:



## Encounter User Guide

### Clock Mesh Specification File

Stage	<p>Denotes the beginning of a particular stage definition. This section defines stage-specific parameters for the global mesh, including driver cells, route types, and trunk and branch information.</p> <p>Each stage definition must begin with a Stage statement, and close with an End statement.</p> <p><b>Note:</b> You must define an even number of stage definitions if you use an inverter for the drive cell (DriveCell).</p>
NumDriver <i>number</i>	<p>Specifies the number of drivers that should be inserted at the current stage.</p>
X <i>number</i>	<p>Specifies a grouping factor that controls the number of drivers at each node or endpoint at the current stage of an HTree clock mesh structure. If you use the X grouping factor, you can increase the drive at any given level of the tree.</p> <p><b>Note:</b> This statement only can be used when you are defining an HTree + Mesh structure.</p> <p><i>Default:</i> Uses a single driver at the end of each node.</p> <p>In the following illustration, the diagram on the left shows two drivers in stage 2, where X = 1, and the diagram on the right shows four drivers in stage 2, where X=2.</p> 
DriveCell <i>cellName</i>	<p>Specifies the type of driver to be used for the stage being defined.</p>
RouteTypePair <i>type1 type2</i>	<p>Specifies the routing types to be used for the stage. Routing types are defined using the RouteTypeDef statement.</p>

## Encounter User Guide

### Clock Mesh Specification File

---

**NumTrunk *number***      Specifies the number of trunks to create for a particular stage. You can specify this statement for any stage in the clock mesh structure, but it only is useful for the final construction stage.

A Fishbone mesh structure can have one or two trunks (that is, you can specify single or double Fishbone mesh structures). The number of trunks for the second-to-last stage will be 1 or 3, depending on whether the mesh structure is a single or double Fishbone. For all other stages, the number of trunks is 1.

For an HTree + Mesh structure, this statement is relevant only to the last stage. If you do not specify this statement for an HTree + Mesh structure, the clock mesh tool computes a suitable value automatically. If you specify a value that is less than the minimum number of trunks required to implement the structure, the clock mesh tool displays an error message.

**TrunkPitch *distance***

Specifies the pitch for the trunk for a particular stage.

**Note:** The clock mesh tool considers the TrunkPitch statement to be a soft constraint that applies to the final mesh stage. The tool might need to adjust the specified pitch in order to meet physical constraints.

**TrunkAttachFrequency *number***

## Encounter User Guide

### Clock Mesh Specification File

---

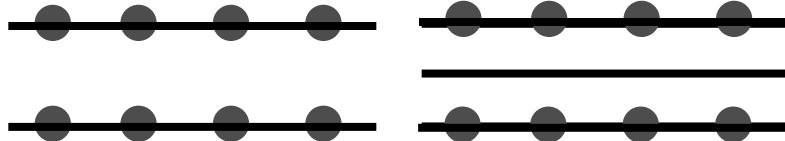
Specifies the frequency with which buffers are attached to the trunks. You must specify a number that is greater than or equal to 1.

*Default:* 1

In the following illustration, assume the design has eight drivers, and the trunk orientation is horizontal.

If TrunkAttachmentFrequency is 1, the minimum number of horizontal trunks is 2, as shown in the diagram on the left (each trunk must have a driver attached to it).

If TrunkAttachmentFrequency is 2, the minimum number of trunks is 3, as shown in the diagram on the right (the middle trunk must not have a driver attached to it). The clock mesh tool generates an error if the specified number of branches does not meet the minimum number of branches.



`NumBranch number`      Specifies the number of branches.

*Default:* The clock mesh tools computes a suitable value automatically.

`BranchPitch distanceValue`

Specifies the pitch for the branches for a particular stage.

**Note:** The clock mesh tool considers the BranchPitch statement to be a soft constraint that applies to the final mesh stage. The tool might need to make adjustments to the specified pitch in order to meet physical constraints.

`TargetBranchOrigin coordinate_in_um`

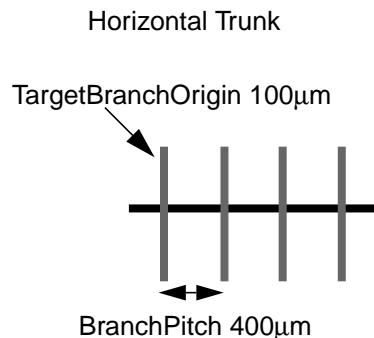
## Encounter User Guide

### Clock Mesh Specification File

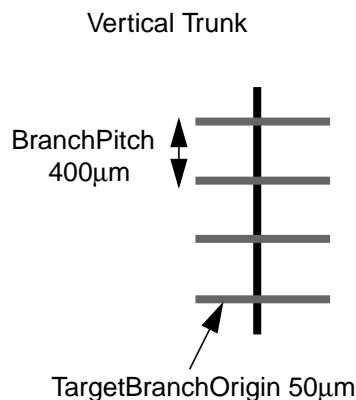
Specifies the absolute location, in microns, of the first branch for a clock mesh.

**Note:** The `TargetBranchOrigin` statement changes the branch origin only. It does not affect the number of branches or the branch pitch.

For a horizontal trunk, `coordinate_in_um` is the x-axis coordinate for the left-most branch, as shown in the following illustration:



For a vertical trunk, `coordinate_in_um` is the y-axis coordinate for the bottom branch, as shown in the following illustration:

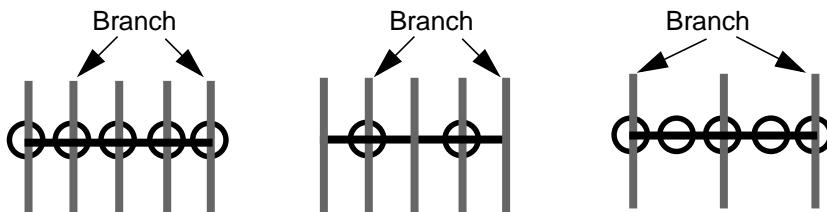


`BranchAttachFrequency number`

Controls the frequency with which buffers are attached to branches. You can specify a positive or negative whole number for *number*.

**Note:** The clock mesh tool honors the `BranchAttachFrequency` statement depending on the setting of the `TrunkDriveDist` statement. If you specify `TrunkDriveDist Uniform`, the clock mesh tool ignores the `BranchAttachFrequency` value.

In the following illustration, if `BranchAttachFrequency` is 1, each branch must have a driver attached to it, as shown in the diagram on the left. If `BranchAttachFrequency` is 2, there must be one branch between each driver that does not have a driver attached to it, as shown in the middle diagram. If `BranchAttachFrequency` is -2, there are drivers between the branches that are not attached to branches, as shown in the diagram on the right.



`TargetTrunkLocs +trunkLocation...`

Positions the trunks at the specified locations.

For example:

```
TargetTrunkLocs  
+100μm  
+200μm  
+300μm  
+550μm
```

You are not required to specify target trunk locations for every stage; however, if you specify a stage, you must list all trunks for that stage.

**Note:** The clock mesh tool considers the `TargetTrunkLocs` statement a soft constraint; it attempts to position the trunks at the specified locations, but might deviate a small distance to find a solution.

## Encounter User Guide

### Clock Mesh Specification File

---

`NonDefaultRule ruleName`

Specifies the LEF NONDEFAULTRULE statement for the router to use when routing the nets for a particular stage of the global mesh structure.

*Default:* The router uses the default routing rule.

`TopPreferLayer layerName`

Specifies the top preferred metal layer to use for routing for a particular stage of the global mesh structure. The name you specify must be a layer defined in the LEF file.

`BottomPreferLayer layerName`

Specifies the bottom preferred metal layer to use for routing for a particular stage of the global mesh structure. The name you specify must be a layer defined in the LEF file.

`PreferredExtraSpace [0-3]`

Specifies the extra space to add around clock wires, when routing the nets in the clockmesh stage.

*Default:* 0

`End`

Denotes the end of a particular stage definition. Each stage definition must begin with a `Stage` statement, and close with an `End` statement.

`End`

Denotes the end of the global mesh definition. The global mesh definition must begin with a `GlobalMesh` statement, and close with an `End` statement.

## Top Chain Section

This section defines how the top chain is to be synthesized. A top-level chain is a cascaded driver chain from the mesh root to the first level of mesh pre-driver drivers.

The following table describes the entries for the top chain section:

`TopChain`

Denotes the beginning of the top chain definition. The top chain definition must begin with a `TopChain` statement, and close with an `End` statement.

`Enabled [true | false]`

## Encounter User Guide

### Clock Mesh Specification File

---

Enables the implementation of the top chain.

*Default:* false

`DriveCell cellName` Specifies the type of buffer to be used for constructing the top-level chain. Only one cell type can be used for the chain.

`NumLevel number` Specifies the number of levels in the chain structure (which is equal to the number of buffers).

You must specify an even number of levels if you use an inverter for the drive cell (`DriveCell`).

`TargetLocs + driverLocationX driverLocationY ...`

Positions the top chain drivers at the specified locations.

For example:

```
TargetLocs  
+250μm 100μm  
+500μm 500μm
```

The number of locations you specify must match the number of levels. If they do not match, the clock mesh tool ignores them.

**Note:** The clock mesh tool considers the `TargetLocs` statement to be a soft constraint; it attempts to place the drivers at the specified locations, but might deviate a small distance in order to find a solution.

`NonDefaultRule ruleName`

Specifies the LEF NONDEFAULTRULE statement for the router to use when routing the nets in the top-level chain.

*Default:* The router uses the default routing rule.

`TopPreferLayer layerName`

Specifies the top preferred metal layer to use for routing the top-level chain buffers. The name you specify must be a layer defined in the LEF file.

`BottomPreferLayer layerName`

Specifies the bottom preferred metal layer to use for routing the top-level chain buffers. The name you specify must be a layer defined in the LEF file.

`PreferredExtraSpace [0-3]`

Specifies the extra space to add around clock wires, when routing the nets in the top-level chain.

*Default:* 0

End Denotes the end of the top chain definition. The top chain definition must begin with a `TopChain` statement, and close with an `End` statement.

## Local Tree Section

This section defines how the local tree is to be synthesized.

The following table describes the entries for the local tree section:

<code>LocalTree</code>	Denotes the beginning of the local tree definition. The local tree definition must begin with a <code>LocalTree</code> statement, and close with an <code>End</code> statement.
------------------------	---

<code>Enabled [true   false]</code>	Enables the implementation of the local tree.
-------------------------------------	---

*Default:* false

<code>RootPos {ClusterCenter   OnMesh   NearMeshInCluster}</code>	Specifies how to place the buffers.
---	-------------------------------------

*Default:* ClusterCenter

<code>ClusterCenter</code>	Places buffers at the cluster center of gravity.
----------------------------	--

<code>OnMesh</code>	Places buffers on or along the nearest mesh trunk or branch. Buffer placement can extend beyond the bounding box of the cluster.
---------------------	--

<code>NearMeshInCluster</code>	Places the buffers as close as possible to a mesh trunk or branch without going outside the bounding box of leaves driven by the local root.
--------------------------------	--

`DriveCells + cellName1 + cellName2...`

## Encounter User Guide

### Clock Mesh Specification File

---

Specifies the types of drive cells to use in the local tree. Multiple drive cells can be specified.

You must specify at least one buffer choice to drive non-gated leaves; the clock mesh tool does not automatically choose a buffer if none is specified.

For gated domains, the clock mesh tool chooses an appropriate cell from the list (that is, an LEQ cell to the original gating element). If no LEQ cells are specified, the tool attempts to find choices from the footprint of the original gating cell. If no footprint exists, the tool uses the original gating cell.

`NonDefaultRule ruleName`

Specifies the LEF NONDEFAULTRULE statement for the router to use when routing the local tree.

*Default:* The router uses the default routing rule.

`TopPreferLayer layerName`

Specifies the top preferred metal layer to use for routing the local tree. The name you specify must be a layer defined in the LEF file.

`BottomPreferLayer layerName`

Specifies the bottom preferred metal layer to use for routing the local tree. The name you specify must be a layer defined in the LEF file.

`PreferredExtraSpace [ 0 - 3 ]`

Specifies the extra space to add around clock wires when routing the local tree.

*Default:* 0

`Cluster`

Denotes the beginning of a cluster subsection within the local tree definition.

By default, the clock mesh tool performs automatic clustering during the local tree synthesis. You can use a cluster subsection to manually specify how a certain group of leaves can be driven by a driver or a clock gating cell. You can specify multiple clusters, as needed.

The cluster subsection must begin with a `Cluster` statement, and close with an `End` statement.

## Encounter User Guide

### Clock Mesh Specification File

---

**DriveCell *cellName*** Specifies the type of cell to use to drive the cluster. If you do not specify a cell type, the clock mesh tool automatically chooses one. If you specify a cell type, the cell must be consistent with the leaves of the cluster. For example, you cannot change the logic by specifying an AND gate for leaves that are originally ungated.

**TargetLoc +*driverLocationX* +*driverLocationY*...**

Positions the drivers at the specified locations.

**Note:** The clock mesh tool considers the TargetLoc statement to be a soft constraint; it attempts to place the drivers at the specified locations, but might deviate a small distance in order to find a solution.

**Default:** The clock mesh tool automatically chooses the local root location

**LeafPin +*instanceName/pinName***

Specifies the instance pins to use to form a cluster. The set of leaf pins must be consistent (that is, they must all be from the same original domain). Additionally, the same leaf cannot appear in multiple clusters.

**End**

Denotes the end of the cluster subsection. The cluster subsection must begin with a Cluster statement, and close with an End statement.

**End**

Denotes the end of the local tree definition. The local tree definition must begin with a LocalTree statement, and close with an End statement.

## Clock Mesh Specification File Example

#Comments in the clock mesh spec file should be preceded with the # character.

```
#Routing Type definition
#At least two routing types must be specified.

RouteTypeDef RT1
    Layer M5
    Width 2
    Spacing 0.5
```

## Encounter User Guide

### Clock Mesh Specification File

---

End

```
RouteTypeDef RT2
```

```
    Layer M6
```

```
    Width 2
```

```
    Spacing 0.5
```

End

```
#Cutout definition
```

```
#All target locations are absolute values.
```

```
Cutout
```

```
+480µm 400µm 860µm 560µm
```

```
+Inst/RAM1/ HALO 30µm
```

```
#Clock Mesh definition
```

```
ClockMesh clk
```

```
#Timing and Power Constraints definition
```

```
    Period 1000
```

```
    SupplyVoltage 1.0
```

```
    MaxPower 0
```

```
    RootTrans 400
```

```
    MinDelay 1000
```

```
    MaxDelay 1020
```

```
    MaxSkew 10
```

```
    MaxBufTrans 400
```

```
    MaxLeafTrans 400
```

```
#Tracing and Analysis Scope definition
```

```
RootPin clk
```

```
#LeafPins can be rising or falling.
```

```
LeafPin
```

```
+U1/A rising
```

```
#LeafCellPins can be rising or falling.
```

```
LeafCellPin
```

```
+NAND/A rising
```

```
#DefaultTrigger can be rising or falling.
```

## Encounter User Guide

### Clock Mesh Specification File

---

```
DefaultTrigger rising

#Set AllowGating to true to handle gated clock.
AllowGating true

#Mesh Structure definition
#UseMeshModule can be true or false
UseMeshModule true
MeshModule mesh_module

#GlobalMesh definition
GlobalMesh
    #MeshDrivePoint can be Center, Root, or X,Y.
    MeshDrivePoint Center

    #MeshType can be Fishbone or HTreeMesh
    MeshType HTreeMesh

    #TrunkOrientation can be Horizontal or Vertical
    TrunkOrientation Horizontal

    #Use H and V for HTreePattern. H* means HVHVHV (alternating pattern).
    HTreePattern H*

    #TrunkPlacement can be UniformPitch or LoadWeighted
    TrunkPlacement UniformPitch

    #TrunkDriveDist can be StrictAttach, Uniform, LoadWeighted, or
    #LoadWeightedMatch.
    TrunkDriveDist StrictAttach

#CTS pre-drive structure definition
PreDriveCTS
    #Set Enabled to true to implement top chain.
    Enabled true
    DriveCells
        + CLKBUFX20
        + CLKBUFX16
    NonDefaultRule NDR1
    TopPreferlayer M5
    BottomPreferLayer M4
```

## Encounter User Guide

### Clock Mesh Specification File

---

```
PreferredExtraSpace 1

#Markes the end of the CTS pre-drive structure definition
End

#Definition for mesh Stage 1
Stage
    NumDriver 1
    X 4
    DriveCell CLKBUFX20
    RouteTypePair RT1 RT2
End

#Definition for mesh Stage 2
Stage
    NumDriver 4
    X 2
    DriveCell CLKBUFX20
    RouteTypePair RT1 RT2
End

#Definition for mesh Stage 3 (final stage)
Stage
    NumDriver 8
    X 2
    DriveCell CLKBUFX20
    RouteTypePair RT1 RT2
    NumTrunk 2
    NumBranch 4
    #TrunkPitch 50
    #BranchPitch 20
    #TrunkAttachFrequency 2
    #BranchAttachFrequency 2
    #All target locations in clock mesh spec file are absolute values.
    #TargetTrunkLocs
    #+500µm
    #+1200µm
End

#Top Chain definition (this section is optional)
```

## Encounter User Guide

### Clock Mesh Specification File

---

```
TopChain
    #Set Enabled to true to implement top chain.
    Enabled true
    DriveCell CLKBUFX20
    NumLevel 2
    #NonDefaultRule NDR1
    #TargetLocs
    TopPreferlayer M5
    BottomPreferLayer M4
    PreferredExtraSpace 1
    #Marks the end of the Top Chain definition.
    End

#Local Tree definition (this section is optional)
LocalTree
    #Set Enabled to true to implement local tree synthesis
    Enabled true
    #RootPos can be ClusterCenter, OnMesh, or NearMeshInCluster.
    RootPos ClusterCenter
    DriveCells
    +CLKBUFX20
    +CLKBUFX16
    +CLKBUFX12
    #NonDefaultRule NDR2
    TopPreferLayer M4
    BottomPreferlayer M3
    PreferredExtraSpace 1

    #Manual Cluster definition (this subsection is optional)
    Cluster
        #DriveCell can be buffer or gated cell.
        DriveCell BUFX16
        #TargetLoc +300µm 600µm
        LeafPin
        +FF1/CK
        +FF2/CK
        +FF9/CK
    #Marks the end of the cluster definition
    End

#Marks the end of the local tree definition
```

## **Encounter User Guide**

### Clock Mesh Specification File

---

End

#Marks the end of the clock mesh defintion

End

**Encounter User Guide**  
Clock Mesh Specification File

---

---

## Supported CPF 1.0 Commands

---

**Note:** The following commands are supported unless otherwise noted.

Command Name	Option	Notes
		N/A = not available in this release
create_analysis_view		
	-name	
	-mode	
	-domain_corners	
create_bias_net		
	-net	
	-driver	N/A
	-user_attributes	Accessible by getCPFUserAttr
	-peak_ir_drop_limit	N/A
	-average_ir_drop_limit	N/A
create_global_connection		
	-net	
	-pins	
	-domain	
	-instances	
create_ground_nets		
	-nets	
	-voltage	N/A

**Encounter User Guide**  
Supported CPF 1.0 Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-internal	N/A
	-user_attributes	Accessible by getCPFUserAttr
	-peak_ir_drop_limit	N/A
	-average_ir_drop_limit	N/A
create_isolation_rule		
	-name	
	-isolation_condition	
	-pins	
	-from	
	-to	
	-isolation_target	N/A
	-isolation_output	
	-exclude	
create_level_shifter_rule		
	-name	
	-pins	
	-from	
	-to	
	-exclude	
create_mode_transition		N/A
create_nominal_condition		
	-name	
	-voltage	
	-pmos_bias_voltage	N/A
	-nmos_bias_voltage	N/A
create_operating_corner		
	-name	

**Encounter User Guide**  
Supported CPF 1.0 Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-voltage	
	-process	
	-temperature	
	-library_set	
create_power_domain		
	-name	
	-default	
	-instances	
	-boundary_ports	
	-shutoff_condition	
	-default_restore_edge	
	-default_save_edge	
	-power_up_states	N/A
create_power_mode		
	-name	
	-domain_conditions	
	-default	
create_power_nets		
	-nets	
	-voltage	
	-external_shutoff_condition	
	-internal	
	-user_attributes	Accessible by getCPFUserAttr
	-peak_ir_drop_limit	N/A
	-average_ir_drop_limit	N/A
create_power_switch_rule		
	-name	

**Encounter User Guide**  
Supported CPF 1.0 Commands

---

Command Name	Option	Notes
	-domain	
	-external_power_net	
	-external_ground_net	
create_state_retention_rule		
	-name	
	-domain	
	-instances	
	-restore_edge	
	-save_edge	
define_always_on_cell		N/A
define_isolation_cell		
	-cells	
	-library_set	
	-always_on_pin	
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
	-valid_location	
	-non_dedicated	N/A
	-enable	
define_level_shifter_cell		
	-cells	
	-library_set	
	-always_on_pin	N/A
	-input_voltage_range	
	-output_voltage_range	
	-direction	

**Encounter User Guide**  
Supported CPF 1.0 Commands

---

Command Name	Option	Notes
	-output_voltage_input_pin	N/A
	-input_power_pin	
	-output_power_pin	
	-ground	
	-valid_location	
define_open_source_input_pin		
	-cells	
	-pin	
	-library_set	
define_power_clamp_cell		N/A
define_power_switch_cell		
	-cells	
	-library_set	
	-stage_1_enable	
	-stage_1_output	
	-stage_2_enable	
	-stage_2_output	
	-type	
	-power_switchable	
	-power	
	-ground	
	-ground_switchable	
	-on_resistance	Accessible by ::CPF::getCpfPsoCell
	-stage_1_saturation_current	Accessible by ::CPF::getCpfPsoCell
	-stage_2_saturation_current	Accessible by ::CPF::getCpfPsoCell

**Encounter User Guide**  
Supported CPF 1.0 Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-leakage_current	Accessible by ::CPF::getCpfPsoCell
define_state_retention_cell		
	-cells	
	-library_set	
	-always_on_pin	N/A
	-clock_pin	N/A
	-restore_function	
	-restore_check	N/A
	-save_function	
	-save_check	N/A
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
define_library_set		
	-name	
	-libraries	
end_design		
identify_always_on_driver		N/A
identify_power_logic		
	-type	
	-instances	
set_array_naming_style		
set_cpf_version		
set_design		
	-ports	
set_hierarchy_separator		

**Encounter User Guide**  
Supported CPF 1.0 Commands

---

Command Name	Option	Notes
set_instance		
	-port_mapping	
	-merge_default_domains	
set_power_target		N/A
set_power_unit		N/A
set_register_naming_style		
set_switching_activity		
	-all	
	-pins	
	-instances	
	-hierarchical	
	-probability	
	-toggle_rate	
	-clock_pins	N/A
	-toggle_percentage	N/A
	-mode	N/A
set_time_unit		
update_isolation_rules		
	-names	'
	-location	
	-cells	
	-library_set	
	-prefix	
	-combine_level_shifting	N/A
	-open_source_pins_only	
-update_level_shifter_rules		
	-names	
	-location	

**Encounter User Guide**  
Supported CPF 1.0 Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-cells	
	-library_set	
	-prefix	
update_nominal_condition		
	-name	
	-library_set	
update_power_domain		
	-name	
	-internal_power_net	
	-internal_ground_net	
	-min_power_up_time	N/A
	-max_power_up_time	N/A
	-pmos_bias_net	N/A
	-nmos_bias_net	N/A
	-user_attributes	Accessible by ::CPF::getCpfUserAttr
	-rail_mapping	N/A
	-library_set	
update_power_mode		
	-name	
	-activity_file	N/A
	-activity_file_weight	N/A
	-sdc_files	
	-peak_ir_drop_limit	N/A
	-average_ir_dropt_limit	N/A
	-leakage_power_limit	N/A
	-dynamic_power_limit	N/A

**Encounter User Guide**  
Supported CPF 1.0 Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
update_power_switch_rule		
	-name	
	-enable_condition_1	
	-enable_condition_2	
	-acknowledge_receiver	
	-cells	
	-library_set	
	-prefix	
	-pead_ir_drop_limit	Accessible by ::CPF::getCpfUserAttr
	-average_ir_drop_limit	Accessible by ::CPF::getCpfUserAttr
update_state_retention_rule		
	-name	
	-cell_type	
	-cell	
	-library_set	

**Encounter User Guide**  
Supported CPF 1.0 Commands

---

---

## Supported CPF 1.0e Commands

---

**Note:** The following commands and options are supported unless otherwise noted.

Command Name	Option	Notes
create_always_on_rule		
	-name	
	-isolation_condition	
	-no_condition	
	-pins	
	-from	
	-to	
	-exclude	
	-isolation_target	
	-isolation_output	
	-secondary_domain	
create_analysis_view		
	-name	
	-mode	
	-domain_corners	
	-group_views	
create_assertion_control		
	-name	Unsupported
	-assertions	Unsupported
	-domains	Unsupported

**Encounter User Guide**  
**Supported CPF 1.0e Commands**

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-shutoff_condition	Unsupported
	-type	Unsupported
create_bias_net		
	-net	
	-driver	
	-user_attributes	Supported: query getCPFUserAttributes
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
create_global_connection		
	-net	
	-pins	
	-domain	
	-instances	
create_power_domain		
	-name	
	-instances	
	-boundary_ports	
	-default	
	-shutoff_condition	
	-external_controlled_shutoff	
	-default_isolation_condition	
	-default_restore_edge	
	-default_save_edge	
	-default_restore_level	Supported
	-default_save_level	Supported
	-power_up_states	Unsupported
	-active_state_condition	Unsupported

**Encounter User Guide**  
**Supported CPF 1.0e Commands**

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-secondary_domains	
create_ground_nets		
	-nets	
	-voltage	
	-external_shutoff_condition	
	-user_attributes	Supported: query getCPFUserAttributes
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
create_isolation_rule		
	-name	
	-isolation_condition	
	-no_condition	Unsupported
	-pins	
	-from	
	-to	
	-exclude	
	-isolation_target	
	-isolation_output	
	-secondary_domain	
create_level_shifter_rule		
	-name	
	-pins	
	-from	
	-to	
	-exclude	
create_mode_transition		
	-name	

**Encounter User Guide**  
**Supported CPF 1.0e Commands**

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-from	
	-to	
	-state_condition	
	-end_condition	
	-cycles	
	-clock_pin	
	-latency	
create_nominal_condition		
	-name	
	-voltage	
	-ground_voltage	
	-state	Unsupported
	-pmos_bias_voltage	Unsupported
	-nmos_bias_voltage	Unsupported
create_operating_corner		
	-name	
	-voltage	
	-ground_voltage	Unsupported
	-pmos_bias_voltage	Unsupported
	-nmos_bias_voltage	Unsupported
	-processes	
	-temperature	
	-library_set	
create_power_mode		
	-name	
	-default	
	-domain_conditions	
	-group_modes	

**Encounter User Guide**  
**Supported CPF 1.0e Commands**

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-domain_conditions	
create_power_nets		
	-nets	
	-voltage	
	-external_shutoff_condition	
	-user_attributes	Supported: query getCPFUserAttributes
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
create_power_switch_rule		
	-name	
	-domain	
	-external_power_net	
	-external_ground_net	
create_state_retention_rule		
	-name	
	-domain	
	-instances	
	-exclude	
	-restore_edge	
	-save_edge	
	-restore_precondition	
	-save_precondition	
	-target_type	
	-secondary_domain	
define_always_on_cell		
	-cells	
	-library_set	

**Encounter User Guide**  
**Supported CPF 1.0e Commands**

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
define_isolation_cell		
	-cells	
	-library_set	
	-always_on_pins	
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
	-valid_location	
	-enable	
	-no_enable	Unsupported
	-non_dedicated	
define_level_shifter_cell		
	-cells	
	-library_set	
	-always_on_pins	
	-input_voltage_range	
	-output_voltage_range	
	-ground_output_voltage_range	Unsupported
	-grouping_output_voltage_range	
	-direction	
	-input_power_pin	

**Encounter User Guide**  
Supported CPF 1.0e Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-output_power_pin	
	-input_ground_pin	Unsupported
	-output_ground_pin	Unsupported
	-ground	
	-power	Unsupported
	-enable	
	-valid_location	
define_library_set		
	-name	
	-libraries	
	-user_attributes	
define_power_clamp_cell		
	-cells	Unsupported
	-location	Unsupported
	-within_hierarchy	Unsupported
	-cells	Unsupported
	-prefix	Unsupported
define_power_switch_cell		
	-cells	
	-library_set	
	-stage_1_enable	
	-stage_1_output	
	-stage_2_enable	
	-stage_2_output	
	-type	
	-enable_pin_bias	Unsupported
	-gate_bias_pin	Unsupported
	-power_switchable	

**Encounter User Guide**  
Supported CPF 1.0e Commands

---

Command Name	Option	Notes
	-power	
	-ground_switchable	
	-ground	
	-on_resistance	Supported (for use with addPowerSwitch)
	-stage_1_saturation_current	Supported (for use with addPowerSwitch)
	-stage_2_saturation_current	Supported (for use with addPowerSwitch)
	-leakage_current	Supported (for use with addPowerSwitch)
define_state_retention_cell		
	-cells	
	-library_set	
	-cell_type	
	-always_on_pins	
	-clock_pin	
	-restore_function	
	-save_function	
	-restore_check	
	-save_check	
	-always_on_components	Unsupported
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
end_macro_model		
end_power_mode_control_group		
get_parameter		

**Encounter User Guide**  
Supported CPF 1.0e Commands

---

Command Name	Option	Notes
include		
identify_secondary_on_driver		
	-secondary_domain	
	-instances	
	-cells	
	-domain	
	-from	
	-to	
identify_power_logic		
	-type	Only “isolation” is supported for the -type
	-instances	Supported
	-module	Supported
set_array_naming_style		
set_cpf_version		
set_hierarchy_separator		
set_design		
	-ports	
	-honor_boundary_port_domain	
	-parameters	
	-end_design	
set_equivalent_control_pins		
	-master	Unsupported
	-pins	Unsupported
	-domain	Unsupported
	-from	Unsupported
	-to	Unsupported

**Encounter User Guide**  
Supported CPF 1.0e Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
set_floating_ports		Unsupported
set_input_voltage_tolerance		
	-ports	Unsupported
	-bias	Unsupported
set_instance		
	-design	
	-model	
	-of_macro	
	-port_mapping	
	-domain_mapping	
	-parameter_mapping	
set_macro_model		
set_power_mode_control_group		
	-name	
	-domains	
	-groups	Unsupported
	-domains	
set_power_target		
	-leakage	Unsupported
	-dynamic	Unsupported
set_power_unit		
set_register_naming_style		
set_switching_activity		
	-all	Supported
	-pins	Supported
	-instances	Supported
	-hierarchical	Supported
	-probability	Supported

**Encounter User Guide**  
Supported CPF 1.0e Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-toggle_rate	Supported
	-clock_pins	Unsupported
	-toggle_percentage	Unsupported
	-mode	Supported
set_time_unit		
set_wire_feedthrough_ports		
update_isolation_rules		
	-names	
	-location	
	-within_hierarchy	
	-cells	
	-prefix	
	-open_source_pins_only	Supported
update_level_shifter_rules		
	-names	
	-location	
	-within_hierarchy	
	-cells	
	-prefix	
update_power_domain		
	-name	
	-primary_power_net	
	-primary_ground_net	
	-pmos_bias_net	Unsupported
	-nmos_bias_net	Unsupported
	-user_attributes	Supported: query getCPFUserAttributes
	-transition_slope	Unsupported

**Encounter User Guide**  
Supported CPF 1.0e Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-transition_latency	Unsupported
	-transition_cycles	Unsupported
update_power_mode		
	-name	
	-activity_file	Unsupported
	-activity_file_weight	Unsupported
	-sdc_files	
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
	-leakage_power_limit	Unsupported
	-dynamic_power_limit	Unsupported
update_power_switch_rule		
	-name	
	-enable_condition_1	
	-enable_condition_2	
	-acknowledge_receiver	
	-cells	
	-gate_bias_net	Unsupported
	-prefix	
	-peak_ir_drop	
	-average_ir_drop_limit	
update_state_retention_rules		
	-names	
	-cell_type	
	-cells	
	-set_rest_condition	Unsupported

---

## Supported CPF 1.1 Commands

---

**Note:** The following commands and options are supported unless otherwise noted.

Command Name	Option	Notes
asset_illegal_domain_configurations		
	-domain_conditions	
	-group_modes	
create_always_on_rule		
	-exclude	
	-from	
	-isolation_condition	
	-isolation_output	
	-isolation_target	
	-name	
	-no_condition	
	-pins	
	-secondary_domain	
	-to	
create_analysis_view		
	-domain_corners	
	-group_views	
	-mode	
	-name	

**Encounter User Guide**  
Supported CPF 1.1 Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
create_assertion_control		
	-assertions	Unsupported
	-domains	Unsupported
	-exclude	
	-name	Unsupported
	-shutoff_condition	Unsupported
	-type	Unsupported
create_bias_net		
	-average_ir_drop_limit	
	-driver	
	-net	
	-peak_ir_drop_limit	
	-user_attributes	Supported: query getCPFUserAttributes
create_global_connection		
	-domain	
	-instances	
	-net	
	-pins	
create_ground_nets		
	-average_ir_drop_limit	
	-external_shutoff_condition	
	-nets	
	-peak_ir_drop_limit	
	-user_attributes	Supported: query getCPFUserAttributes
	-voltage	
create_isolation_rule		

**Encounter User Guide**  
Supported CPF 1.1 Commands

---

Command Name	Option	Notes
	-exclude	
	-from	
	-isolation_condition	
	-isolation_output	
	-isolation_target	
	-name	
	-no_condition	Unsupported
	-pins	
	-secondary_domain	
	-to	
create_level_shifter_rule		
	-exclude	
	-from	
	-name	
	-pins	
	-to	
create_mode_transition		
	-clock_pin	
	-cycles	
	-end_condition	
	-from	
	-latency	
	-name	
	-state_condition	
	-to	
create_nominal_condition		
	-ground_voltage	
	-name	

**Encounter User Guide**  
Supported CPF 1.1 Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-nmos_bias_voltage	Unsupported
	-pmos_bias_voltage	Unsupported
	-state	Unsupported
	-voltage	
create_operating_corner		
	-ground_voltage	Unsupported
	-library_set	
	-name	
	-nmos_bias_voltage	Unsupported
	-pmos_bias_voltage	Unsupported
	-processes	
	-temperature	
	-voltage	
create_power_domain		
	-active_state_condition	Unsupported
	-base_domains	
	-boundary_ports	
	-default	
	-default_isolation_condition	
	-default_restore_edge	
	-default_restore_level	Supported
	-default_save_edge	
	-default_save_level	Supported
	-external_controlled_shutoff	
	-instances	
	-name	
	-power_up_states	Unsupported
	-shutoff_condition	

**Encounter User Guide**  
Supported CPF 1.1 Commands

---

Command Name	Option	Notes
create_power_mode		
	-default	
	-domain_conditions	
	-domain_conditions	
	-group_modes	
	-name	
create_power_nets		
	-average_ir_drop_limit	
	-external_shutoff_condition	
	-nets	
	-peak_ir_drop_limit	
	-user_attributes	Supported: query getCPFUserAttributes
	-voltage	
create_power_switch_rule		
	-domain	
	-external_ground_net	
	-external_power_net	
	-name	
create_state_retention_rule		
	-domain	
	-exclude	
	-instances	
	-name	
	-restore_edge	
	-restore_precondition	
	-save_edge	
	-save_precondition	

**Encounter User Guide**  
Supported CPF 1.1 Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-secondary_domain	
	-target_type	
define_always_on_cell		
	-cells	
	-ground	
	-ground_switchable	
	-library_set	
	-power	
	-power_switchable	
define_isolation_cell		
	-always_on_pins	
	-cells	
	-enable	
	-ground	
	-ground_switchable	
	-library_set	
	-no_enable	Unsupported
	-non_dedicated	
	-power	
	-power_switchable	
	-valid_location	
define_level_shifter_cell		
	-always_on_pins	
	-cells	
	-direction	
	-enable	
	-ground	
	-ground_input_voltage_range	

**Encounter User Guide**  
Supported CPF 1.1 Commands

---

Command Name	Option	Notes
	-ground_output_voltage_range	
	-input_ground_pin	
	-input_power_pin	
	-input_voltage_range	
	-library_set	
	-output_ground_pin	
	-output_power_pin	
	-output_voltage_range	
	-power	
	-valid_location	
define_library_set		
	-libraries	
	-name	
	-user_attributes	
define_power_clamp_cell		
	-cells	Unsupported
	-cells	Unsupported
	-location	Unsupported
	-prefix	Unsupported
	-within_hierarchy	Unsupported
define_power_switch_cell		
	-cells	
	-enable_pin_bias	Unsupported
	-gate_bias_pin	Unsupported
	-ground	
	-ground_switchable	

**Encounter User Guide**  
Supported CPF 1.1 Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-leakage_current	Supported (for use with addPowerSwitch)
	-library_set	
	-power	
	-power_switchable	
	-stage_1_on_resistance	
	-stage_2_on_resistance	
	-stage_1_enable	
	-stage_1_output	
	-stage_1_saturation_current	Supported (for use with addPowerSwitch)
	-stage_2_enable	
	-stage_2_output	
	-stage_2_saturation_current	Supported (for use with addPowerSwitch)
	-type	
define_state_retention_cell		
	-always_on_components	Unsupported
	-always_on_pins	
	-cell_type	
	-cells	
	-clock_pin	
	-ground	
	-ground_switchable	
	-library_set	
	-power	
	-power_switchable	
	-restore_check	
	-restore_function	

**Encounter User Guide**  
Supported CPF 1.1 Commands

---

Command Name	Option	Notes
	-save_check	
	-save_function	
end_design		
end_macro_model		
end_power_mode_control_group		
get_parameter		
include		
identify_secondary_on_driver		
	-cells	
	-domain	
	-from	
	-instances	
	-secondary_domain	
	-to	
identify_power_logic		
	-instances	Supported
	-module	Supported
	-type	Only “isolation” is supported for the -type
set_array_naming_style		
set_cpf_version		
set_hierarchy_separator		
set_design		
	-end_design	
	-honor_boundary_port_domain	
	-parameters	
	-ports	

**Encounter User Guide**  
Supported CPF 1.1 Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
set_equivalent_control_pins		
	-domain	Unsupported
	-from	Unsupported
	-master	Unsupported
	-pins	Unsupported
	-to	Unsupported
set_floating_ports		Unsupported
set_input_voltage_tolerance		
	-bias	Unsupported
	-ports	Unsupported
set_instance		
	-design	
	-domain_mapping	
	-model	
	-of_macro	
	-parameter_mapping	
	-port_mapping	
set_macro_model		
set_power_mode_control_group		
	-domains	
	-domains	
	-groups	Unsupported
	-name	
set_power_target		
	-dynamic	Unsupported
	-leakage	Unsupported
set_power_unit		
set_register_naming_style		

**Encounter User Guide**  
Supported CPF 1.1 Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
set_switching_activity		
	-all	Supported
	-clock_pins	Unsupported
	-hierarchical	Supported
	-instances	Supported
	-mode	Supported
	-pins	Supported
	-probability	Supported
	-toggle_percentage	Unsupported
	-toggle_rate	Supported
set_time_unit		
set_wire_feedthrough_ports		
update_isolation_rules		
	-cells	
	-location	
	-names	
	-open_source_pins_only	Supported
	-prefix	
	-within_hierarchy	
update_level_shifter_rules		
	-cells	
	-location	
	-names	
	-prefix	
	-within_hierarchy	
update_power_domain		
	-equivalent_ground_nets	
	-equivalent_power_nets	

**Encounter User Guide**  
Supported CPF 1.1 Commands

---

<b>Command Name</b>	<b>Option</b>	<b>Notes</b>
	-name	
	-nmos_bias_net	Unsupported
	-pmos_bias_net	Unsupported
	-primary_ground_net	
	-primary_power_net	
	-transition_cycles	Unsupported
	-transition_latency	Unsupported
	-transition_slope	Unsupported
	-user_attributes	Supported: query getCPFUserAttributes
update_power_mode		
	-activity_file	Unsupported
	-activity_file_weight	Unsupported
	-average_ir_drop_limit	
	-dynamic_power_limit	Unsupported
	-leakage_power_limit	Unsupported
	-name	
	-peak_ir_drop_limit	
	-sdc_files	
update_power_switch_rule		
	-acknowledge_reciever_1	
	-acknowledge_reciever_2	
	-average_ir_drop_limit	
	-cells	
	-enable_condition_1	
	-enable_condition_2	
	-gate_bias_net	Unsupported
	-name	

**Encounter User Guide**  
Supported CPF 1.1 Commands

---

Command Name	Option	Notes
	-peak_ir_drop	
	-prefix	
update_state_retention_rules		
	-cell_type	
	-cells	
	-names	
	-set_rest_condition	Unsupported

**Encounter User Guide**  
Supported CPF 1.1 Commands

---

---

## CPF 1.0 Script Example

---

The following section contains an example of the CPF 1.0 file using a sample design and library.

For list of supported CPF commands and options within Encounter product family, see [Appendix C, “Supported CPF 1.0 Commands.”](#)

```
set_cpf_version 1.0

#####
#                                     #
# Technology portion of the CPF:      #
# Defining the special cells for low-power designs #
#                                     #
#####

#####
#### High-to-Low level shifters
#####

define_level_shifter_cell -cells LVLH2L* \
    -input_voltage_range 0.8:1.0:0.1 \
    -output_voltage_range 0.8:1.0:0.1 \
    -direction down \
    -output_power_pin VDD \
    -ground VSS \
    -valid_location to

#####
#### Always-on High-to-low level shifters
#####
```

## Encounter User Guide

### CPF 1.0 Script Example

---

```
define_level_shifter_cell -cells AOLVLH2L* \
    -input_voltage_range 0.8:1.0:0.1 \
    -output_voltage_range 0.8:1.0:0.1 \
    -direction down \
    -output_power_pin TVDD \
    -ground VSS \
    -valid_location to

#####
##### Low-to-High Level Shifters
#####
define_level_shifter_cell -cells LVLL2H* \
    -input_voltage_range 0.8:1.0:0.1 \
    -output_voltage_range 0.8:1.0:0.1 \
    -input_power_pin VDDI \
    -output_power_pin VDD \
    -direction up \
    -ground VSS \
    -valid_location to

#####
##### Low-to-High level shifting plus isolation combo cells
#####
define_level_shifter_cell -cells LVLCIL2H* \
    -input_voltage_range 0.8:1.0:0.1 \
    -output_voltage_range 0.8:1.0:0.1 \
    -output_voltage_input_pin ISO \
    -input_power_pin VDDI \
    -output_power_pin VDD \
    -direction up \
    -ground VSS \
    -valid_location to

#####
##### Isolation cells
#####

define_isolation_cell -cells LVLCIL2H* \
    -power VDD \
    -ground VSS \
```

## Encounter User Guide

### CPF 1.0 Script Example

---

```
-enable ISO \
-valid_location to

#####
##### Power switch cells: headers
#####

define_power_switch_cell -cells {HEADERHVT HEADERAOPHVT} \
    -power_switchable VDD -power TVDD \
    -stage_1_enable !ISOIN1 \
    -stage_1_output ISOOUT1 \
    -stage_2_enable !ISOIN2 \
    -stage_2_output ISOOUT2 \
    -type header

#####
##### SRPG cells
#####

define_state_retention_cell -cells { SRPG2Y } \
    -clock_pin CLK \
    -power TVDD \
    -power_switchable VDD \
    -ground VSS \
    -save_function "SAVE" \
    -restore_function "!NRESTORE"

#####
##### Always-on cells: buffers and level shifters
#####

define_always_on_cell -cells {AOBUFF2Y AOLVLH2L*} \
    -power_switchable VDD -power TVDD -ground VSS
#####

#          #
#      Design part of the CPF          #
#          #

set_design top
```

## Encounter User Guide

### CPF 1.0 Script Example

---

```
set_hierarchy_separator "/"  
  
set constraintDir ../CONSTRAINTS  
  
set libdir ../LIBS  
  
#####  
#### create the power and ground nets in this design  
#####  
  
### VDD will connect the power follow-pin of the instances in the always-on  
#power domain  
### VDD_core_SW will connect the power follow-pin of the instances in the  
#switchable power domain and is the power net that can be shut-off  
### VDD_core_AO is the always-on power net for the switchable power domain  
  
create_power_nets -nets VDD -voltage {0.8:1.0:0.1}  
create_power_nets -nets VDD_core_AO -voltage 0.8  
create_power_nets -nets VDD_core_SW -internal -voltage 0.8  
create_power_nets -nets AVDD -voltage 1.0  
  
create_ground_nets -nets VSS  
create_ground_nets -nets AVSS  
  
#####  
#### Creating three power domains:  
#### AO is the default always-on power domain  
#### CORE is the switchable power domain  
#### PLL is another always-on power domain  
## Also specifying the power net-pin connection in each power domain  
#####  
  
#####  
### For power domain "AO"  
#####  
  
create_power_domain -name AO -default  
update_power_domain -name AO -internal_power_net VDD  
  
create_global_connection -domain AO -net VDD -pins VDD  
create_global_connection -domain AO -net VSS -pins VSS
```

## Encounter User Guide

### CPF 1.0 Script Example

---

```
create_global_connection -domain AO -net VDD_core_SW -pins VDDI

#####
### For power domain "CORE"
#####

create_power_domain -name core -instances CORE_INST \
    -shutoff_condition {PWR_CONTROL/power_switch_enable}
update_power_domain -name core -internal_power_net VDD_core_SW

create_global_connection -domain CORE -net VSS -pins VSS
create_global_connection -domain CORE -net VDD_core_AO -pins TVDD
create_global_connection -domain CORE -net VDD_core_SW -pins VDD

#####
### For power domain "PLL"
#####

### PLL contains a single PLL macro and five top-level boundary ports which
#connect to the PLL macro directly

create_power_domain -name PLL -instances PLLCLK_INST -boundary_ports \
{refclk vcom vcop ibias pllrst}
update_power_domain -name PLL -internal_power_net AVDD

create_global_connection -domain PLL -net AVDD -pins avdd!
create_global_connection -domain PLL -net AVSS -pins agnd!
create_global_connection -domain PLL -net VDD -pins VDDI
create_global_connection -domain PLL -net AVDD -pins VDD
create_global_connection -domain PLL -net AVSS -pins VSS

#####
#####

set lib_1p1_wc "$libdir/technology45_std_1p1.lib"
set lib_1p3_bc "$libdir/technology45_std_1p3.lib"
set lib_1p0_bc "$libdir/technology45_std_1p0.lib"
set lib_0p8_wc "$libdir/technology45_std_0p8.lib"

set lib_ao_wc_extra "\
```

## Encounter User Guide

### CPF 1.0 Script Example

---

```
$libdir/technology45_lv112h_1p1.lib \
"
set lib_ao_bc_extra "\$libdir/technology45_lv112h_1p3.lib \
"
set lib_core_wc_extra "\$libdir/technology45_lvh21_0p8.lib \
\$libdir/technology45_headers_0p8.lib \
\$libdir/technology45_sprg_ao_0p8.lib \
"
set lib_core_bc_extra "\$libdir/technology45_lvh21_1p0.lib \
\$libdir/technology45_headers_1p0.lib \
\$libdir/technology45_srpq_ao_1p0.lib \
"
"
set lib_pll_wc "\$libdir/pll_slow.lib \
\$libdir/ram_256x16_slow.lib \
\$libdir/rom_512x16_slow.lib \
"
set lib_pll_bc "\$libdir/pll_fast.lib \
\$libdir/ram_256x16_fast.lib \
\$libdir/rom_512x16_fast.lib \
"
#####
#### Define library sets
#####
define_library_set -name ao_wc_0p8 -libraries "$lib_0p8_wc \$lib_ao_wc_extra"
define_library_set -name ao_bc_1p0 -libraries "$lib_1p0_bc \$lib_ao_bc_extra"

define_library_set -name ao_wc_1p1 -libraries "$lib_1p1_wc_base \$lib_ao_wc_extra"
define_library_set -name ao_bc_1p3 -libraries "$lib_1p3_bc_base \$lib_ao_bc_extra"

define_library_set -name core_wc_0p8 -libraries "$lib_0p8_wc \$lib_core_wc_extra"
define_library_set -name core_bc_1p0 -libraries "$lib_1p0_bc \$lib_core_bc_extra"

define_library_set -name pll_wc_1p1 -libraries "$lib_pll_wc"
define_library_set -name pll_bc_1p3 -libraries "$lib_pll_bc"
```

## Encounter User Guide

### CPF 1.0 Script Example

---

```
#####
##### Create operating corners
#####

create_operating_corner -name BC_PVT_AO_L \
    -process 1 -temperature 0 -voltage 1.0 \
    -library_set ao_bc_1p0

create_operating_corner -name WC_PVT_AO_L \
    -process 1 -temperature 125 -voltage 0.8 \
    -library_set ao_wc_0p8

create_operating_corner -name BC_PVT_AO_H \
    -process 1 -temperature 0 -voltage 1.3 \
    -library_set ao_bc_1p3

create_operating_corner -name WC_PVT_AO_H \
    -process 1 -temperature 125 -voltage 1.1 \
    -library_set ao_wc_1p1

create_operating_corner -name BC_PVT_CORE \
    -process 1 -temperature 0 -voltage 1.0 \
    -library_set core_bc_1p0

create_operating_corner -name WC_PVT_CORE \
    -process 1 -temperature 125 -voltage 0.8 \
    -library_set tdsp_wc_0p8

create_operating_corner -name BC_PVT_PLL \
    -process 1 -temperature 0 -voltage 1.3 \
    -library_set core_bc_1p3

create_operating_corner -name WC_PVT_PLL \
    -process 1 -temperature 125 -voltage 1.1 \
    -library_set tdsp_wc_1p1

#####
##### Create and update nominal conditions
#####
```

## Encounter User Guide

### CPF 1.0 Script Example

---

```
create_nominal_condition -name high_ao -voltage 1.1
update_nominal_condition -name high_ao -library_set ao_wc_1p1

create_nominal_condition -name low_ao -voltage 0.8
update_nominal_condition -name low_ao -library_set ao_wc_0p8

create_nominal_condition -name low_core -voltage 0.8
update_nominal_condition -name low_core -library_set core_wc_0p8

create_nominal_condition -name high_pll -voltage 1.1
update_nominal_condition -name high_pll -library_set pll_wc_1p1

create_nominal_condition -name off -voltage 0

#####
#### Create and upDate four power modes
#####

create_power_mode -name PM_HL_FUNC \
    -domain_conditions {AO@high_ao CORE@low_core PLL@high_pll} \
    -default
update_power_mode -name PM_HL_FUNC -sdc_files ${constraintDir}/top_func.sdc

create_power_mode -name PM_HL_TEST \
    -domain_conditions {AO@high_ao CORE@low_core PLL@high_pll}
update_power_mode -name PM_HL_TEST -sdc_files ${constraintDir}/top_test.sdc

create_power_mode -name PM_HO_FUNC \
    -domain_conditions {AO@high_ao CORE@off PLL@high_pll}
update_power_mode -name PM_HO_FUNC -sdc_files ${constraintDir}/top_func.sdc

create_power_mode -name PM_LO_FUNC \
    -domain_conditions {AO@low_ao CORE@off PLL@high_pll}
update_power_mode -name PM_LO_FUNC -sdc_files ${constraintDir}/top_slow.sdc

#####
#### Creating ten analysis views
#####

create_analysis_view -name AV_HL_FUNC_MIN_RC1 -mode PM_HL_FUNC \

```

## Encounter User Guide

### CPF 1.0 Script Example

---

```
-domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_HL_FUNC_MIN_RC2 -mode PM_HL_FUNC \
    -domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}

create_analysis_view -name AV_HL_FUNC_MAX_RC1 -mode PM_HL_FUNC \
    -domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}
create_analysis_view -name AV_HL_FUNC_MAX_RC2 -mode PM_HL_FUNC \
    -domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

create_analysis_view -name AV_HL_SCAN_MIN_RC1 -mode PM_HL_TEST \
    -domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_HL_SCAN_MAX_RC1 -mode PM_HL_TEST \
    -domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

create_analysis_view -name AV_HO_FUNC_MIN_RC1 -mode PM_HO_FUNC \
    -domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_HO_FUNC_MAX_RC1 -mode PM_HO_FUNC \
    -domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

create_analysis_view -name AV_LO_FUNC_MIN_RC1 -mode PM_LO_FUNC \
    -domain_corners {AO@BC_PVT_AO_L CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_LO_FUNC_MAX_RC1 -mode PM_LO_FUNC \
    -domain_corners {AO@WC_PVT_AO_L CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

#####
##### Creating and updating the rules for the insertion
##### of power switch, level shifter, isolation cell
#####

#####
##### One power switch rule
#####

create_power_switch_rule -name PWRSW_CORE -domain CORE \
-external_power_net VDD_core_AO

update_power_switch_rule -name PWRSW_CORE \
    -cells HEADERHVT \
    -prefix CDN_SW_ \
    -acknowledge_receiver SIWTCH_ENOUT
#####

September 2009          1353          Product Version 8.1.3
```

## Encounter User Guide

### CPF 1.0 Script Example

---

```
##### One isolation rule using level-shifting and isolation combo cells
#####
create_isolation_rule -name ISORULE -from CORE \
-isolation_condition "!PWR_CONTROL/isolation_enable" \
-isolation_output high

update_isolation_rules -names ISORULE -location to -cells LVLCIL2H2Y

#####
##### Three level shifting rules
#####

##### For signals from AO to CORE

create_level_shifter_rule -name LSRULE_H2L -from AO -to CORE \
-exclude {PWR_CONTROL/power_switch_enable PWR_CONTROL \
/state_retention_enable PWR_CONTROL/state_retention_restore}
update_level_shifter_rules -names LSRULE_H2L -cells LVLH2L2Y -location to

##### Only for the control signals from AO to CORE

create_level_shifter_rule -name LSRULE_H2L_AO -from AO -to CORE \
-pins {PWR_CONTROL/power_switch_enable PWR_CONTROL/state_retention_enable\
PWR_CONTROL/state_retention_restore}
update_level_shifter_rules -names LSRULE_H2L_AO -cells AOLVLH2L2Y -location to

##### For signals from PLL to AO

create_level_shifter_rule -name LSRULE_H2L_PLL -from PLL -to AO
update_level_shifter_rules -names LSRULE_H2L_PLL -cells LVLH2L2Y -location to

#####
##### One SRPG rule
#####

create_state_retention_rule -name SRPG_CORE \
-domain CORE \
-restore_edge {!PWR_CONTROL/state_retention_restore} \
-save_edge {PWR_CONTROL/state_retention_enable}
```

## **Encounter User Guide**

### CPF 1.0 Script Example

---

```
update_state_retention_rules -names SRPG_CORE \
    -cell SRPG2Y \
    -library_set tdsp_wc_0v792

end_design
```

## **Encounter User Guide**

### CPF 1.0 Script Example

---

---

## CPF 1.0e Script Example

---

The following section contains an example of the CPF 1.0e file using a sample design and library.

For list of supported CPF commands and options within Encounter product family, see [Appendix D, “Supported CPF 1.0e Commands.”](#)

```
#-----
#   setting
#-----
set_cpf_version 1.0e
set_hierarchy_separator /

#-----
#   define library_set/cells
#-----
define_library_set -name wc_0v81 -libraries { \
    ../LIBS/timing/library_wc_0v81.lib }
define_library_set -name bc_0v81 -libraries { \
    ../LIBS/timing/library_bc_0v81.lib }
define_library_set -name wc_0v72 -libraries { \
    ../LIBS/timing/library_wc_0v72.lib }
define_library_set -name bc_0v72 -libraries { \
    ../LIBS/timing/library_bc_0v72.lib }

define_always_on_cell -cells {PTLVLHLD* AOBUFF*} -power_switchable \
VDD -power TVDD -ground VSS

define_isolation_cell -cells { LVLLH* } -power VDD -ground VSS -enable \
NSLEEP -valid_location to
define_isolation_cell -cells { ISOHID* ISOLOD* } -power VDD -ground VSS \
-enable ISO -valid_location to

define_level_shifter_cell -cells { LVLHLD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells { PTLVLHLD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin TVDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHCD* } -input_voltage_range \
```

## Encounter User Guide

### CPF 1.0e Script Example

```
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 \
-output_voltage_input_pin NSLEEP -direction up -input_power_pin VDDL \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction up \
-input_power_pin VDDL -output_power_pin VDD -ground VSS -valid_location to

define_power_switch_cell -cells { HEADERHVT1 HEADERHVT2 } \
-stage_1_enable NSLEEPIN1 -stage_1_output NSLEEPOUT1 -stage_2_enable \
NSLEEPIN2 -stage_2_output NSLEEPOUT2 -type header -power_switchable VDD \
-power TVDD

define_state_retention_cell -cells { MSSRPG* } -cell_type \
master_slave -clock_pin CP -restore_check !CP -save_function !CP \
-always_on_components { DFF_inst } -power_switchable VDD -power TVDD \
-ground VSS
define_state_retention_cell -cells { BLSPRG* } -cell_type ballon_latch \
-clock_pin CP -restore_function !NRESTORE -save_function SAVE \
-always_on_components { save_data } -power_switchable VDD -power TVDD \
-ground VSS

#-----
#   macro models
#-----

#-----
#   top design
#-----
set_design top

create_operating_corner -name PMdvfs2_bc -voltage 0.88 -process 1 -temperature \
0 -library_set bc_0v72
create_operating_corner -name PMdvfs1_bc -voltage 0.99 -process 1 -temperature \
0 -library_set bc_0v81
create_operating_corner -name PMdvfs1_wc -voltage 0.81 -process 1 -temperature \
125 -library_set wc_0v81
create_operating_corner -name PMdvfs2_wc -voltage 0.72 -process 1 -temperature \
125 -library_set wc_0v72

create_power_nets -nets VDD -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_sw -voltage { 0.72:0.81:0.09 } -internal \
-peak_ir_drop_limit 0 -average_ir_drop_limit 0
create_power_nets -nets VDDL -voltage 0.72 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDDL_sw -voltage 0.72 -internal -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets Avdd -voltage 0.81 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_IO -voltage { 0.72:0.81:0.09 } \
-external_shutoff_condition { io_shutoff_ack } -peak_ir_drop_limit 0 \
```

## Encounter User Guide

### CPF 1.0e Script Example

```
-average_ir_drop_limit 0

create_ground_nets -nets Avss -voltage 0.00 -peak_ir_drop_limit 0 \
    -average_ir_drop_limit 0
create_ground_nets -nets VSS -voltage 0.00 -peak_ir_drop_limit 0 \
    -average_ir_drop_limit 0

create_nominal_condition -name nom_0v81 -voltage 0.81
create_nominal_condition -name nom_0v72 -voltage 0.72

#-----
#   create power domains
#-----
create_power_domain -name PDdefault -default
create_power_domain -name PDshutoff_io -instances { IOPADS_INST/Pspifsip \
    IOPADS_INST/Pspidip } -boundary_ports { spi_fs spi_data } \
    -external_controlled_shutoff -shutoff_condition io_shutoff_ack
create_power_domain -name PDpll -instances { INST/PLLCLK_INST \
    IOPADS_INST/Pbiasip IOPADS_INST/Ppllrstip IOPADS_INST/Prefclkip \
    IOPADS_INST/Presetip IOPADS_INST/Pvcomop IOPADS_INST/Pvcopop } - \
    boundary_ports { ibias reset \
        refclk vcom vcop pllrst }
create_power_domain -name PDram_virtual
create_power_domain -name PDram -instances { INST/RAM_128x16_TEST_INST } \
    -shutoff_condition !INST/PM_INST/power_switch_enable \
    -secondary_domains { PDram_virtual }
create_power_domain -name PDTdsp -instances { INST/RAM_128x16_TEST_INST1 \
    INST/DSP_CORE_INST0 INST/DSP_CORE_INST1 } -shutoff_condition \
    !INST/PM_INST/power_switch_enable -secondary_domains { PDdefault }

#-----
#   set instances
#-----
set_instance INST/RAM_128x16_TEST_INST1/RAM_128x16_INST -domain_mapping \
    { {RAM_DEFAULT PDTdsp} }

set_macro_model ram_256x16A

create_power_domain -name RAM_DEFAULT -boundary_ports { A* D* CLK CEN WEN Q* } \
    -default -external_controlled_shutoff

create_state_retention_rule -name RAM_ret -instances { mem* } -save_edge !CLK

update_power_domain -name RAM_DEFAULT -primary_power_net VDD \
    -primary_ground_net VSS

end_macro_model

#-----
#   create power modes
#-----
```

## Encounter User Guide

### CPF 1.0e Script Example

---

```
create_power_mode -name PMdvfs1 -default -domain_conditions { PDpll@nom_0v81 \
    PDdefault@nom_0v81 PDTdsp@nom_0v81 PDram@nom_0v72 PDshutoff_io@nom_0v81 \
    PDram_virtual@nom_0v72 }
create_power_mode -name PMdvfs1_off -domain_conditions { PDpll@nom_0v81 \
    PDdefault@nom_0v81 PDshutoff_io@nom_0v81 PDram_virtual@nom_0v72 }
create_power_mode -name PMdvfs1_shutoffio_off -domain_conditions { \
    PDpll@nom_0v81 PDdefault@nom_0v81 PDram_virtual@nom_0v72 }
create_power_mode -name PMdvfs2 -domain_conditions { PDpll@nom_0v81 \
    PDdefault@nom_0v72 PDTdsp@nom_0v72 PDram@nom_0v72 PDshutoff_io@nom_0v72 \
    PDram_virtual@nom_0v72 }
create_power_mode -name PMdvfs2_off -domain_conditions { PDpll@nom_0v81 \
    PDdefault@nom_0v72 PDshutoff_io@nom_0v72 PDram_virtual@nom_0v72 }
create_power_mode -name PMdvfs2_shutoffio_off -domain_conditions { \
    PDpll@nom_0v81 PDdefault@nom_0v72 PDram_virtual@nom_0v72 }
create_power_mode -name PMscan -domain_conditions { PDpll@nom_0v81 \
    PDdefault@nom_0v81 PDTdsp@nom_0v81 PDram@nom_0v72 PDshutoff_io@nom_0v81 \
    PDram_virtual@nom_0v72 }

create_analysis_view -name AV_dvfs1_BC -mode PMdvfs1 -domain_corners { \
    PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDTdsp@PMdvfs1_bc PDram@PMdvfs2_bc \
    PDshutoff_io@PMdvfs1_bc }
create_analysis_view -name AV_dvfs1_WC -mode PMdvfs1 -domain_corners { \
    PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDTdsp@PMdvfs1_wc PDram@PMdvfs2_wc \
    PDshutoff_io@PMdvfs1_wc }
create_analysis_view -name AV_dvfs1_off_BC -mode PMdvfs1_off -domain_corners { \
    PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDshutoff_io@PMdvfs1_bc }
create_analysis_view -name AV_dvfs1_off_WC -mode PMdvfs1_off -domain_corners { \
    PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDshutoff_io@PMdvfs1_wc }
create_analysis_view -name AV_dvfs1_shutoffio_off_BC -mode \
    PMdvfs1_shutoffio_off -domain_corners { PDpll@PMdvfs1_bc \
    PDdefault@PMdvfs1_bc }
create_analysis_view -name AV_dvfs1_shutoffio_off_WC -mode \
    PMdvfs1_shutoffio_off -domain_corners { PDpll@PMdvfs1_wc \
    PDdefault@PMdvfs1_wc }
create_analysis_view -name AV_dvfs2_BC -mode PMdvfs2 -domain_corners { \
    PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc PDTdsp@PMdvfs2_bc PDram@PMdvfs2_bc \
    PDshutoff_io@PMdvfs2_bc }
create_analysis_view -name AV_dvfs2_WC -mode PMdvfs2 -domain_corners { \
    PDpll@PMdvfs1_wc PDdefault@PMdvfs2_wc PDTdsp@PMdvfs2_wc PDram@PMdvfs2_wc \
    PDshutoff_io@PMdvfs2_wc }
create_analysis_view -name AV_PMdvfs2_off_BC -mode PMdvfs2_off -domain_corners \
    { PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc PDshutoff_io@PMdvfs2_bc }
create_analysis_view -name AV_PMdvfs2_off_WC -mode PMdvfs2_off -domain_corners \
    { PDpll@PMdvfs1_wc PDdefault@PMdvfs2_wc PDshutoff_io@PMdvfs2_wc }
create_analysis_view -name AV_dvfs2_shutoffio_off_BC -mode \
    PMdvfs2_shutoffio_off -domain_corners { PDpll@PMdvfs1_bc \
    PDdefault@PMdvfs2_bc }
create_analysis_view -name AV_dvfs2_shutoffio_off_WC -mode \
    PMdvfs2_shutoffio_off -domain_corners { PDpll@PMdvfs1_wc \
    PDdefault@PMdvfs2_wc }
create_analysis_view -name AV_scan_BC -mode PMscan -domain_corners { \
```

## Encounter User Guide

### CPF 1.0e Script Example

```
PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDTdsp@PMdvfs1_bc PDram@PMdvfs2_bc \
PDshutoff_io@PMdvfs1_bc }
create_analysis_view -name AV_scan_WC -mode PMscan -domain_corners { \
PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDTdsp@PMdvfs1_wc PDram@PMdvfs2_wc \
PDshutoff_io@PMdvfs1_wc }

#-----
#   create rules
#-----
create_power_switch_rule -name PDram_SW -domain PDram -external_power_net VDDL
create_power_switch_rule -name PDTdsp_SW -domain PDTdsp -external_power_net \
VDD

create_isolation_rule -name ISORULE1 -isolation_condition { \
!INST/PM_INST/isolation_enable } -from { PDTdsp } -to { PDdefault } \
-isolation_target from -isolation_output high
create_isolation_rule -name ISORULE3 -isolation_condition { \
!INST/PM_INST/isolation_enable } -from { PDram } -to { PDdefault } \
-isolation_target from -isolation_output high
create_isolation_rule -name ISORULE4 -isolation_condition { \
!INST/PM_INST/spi_ip_isolate } -from { PDshutoff_io } \
-isolation_target from -isolation_output low

create_level_shifter_rule -name LSRULE_H2L3 -from { PDdefault } -to { PDram } \
-exclude { INST/PM_INST/power_switch_enable }
create_level_shifter_rule -name LSRULE_H2L_PLL -from { PDpll }
create_level_shifter_rule -name LSRULE_L2H2 -from { PDram } -to { PDdefault }

create_state_retention_rule -name \
INST/DSP_CORE_INST0/PDTdsp_retention_rule -instances { \
INST/DSP_CORE_INST0 } -save_edge !INST/DSP_CORE_INST0/clk
create_state_retention_rule -name \
INST/DSP_CORE_INST1/PDTdsp_retention_rule -instances { \
INST/DSP_CORE_INST1 } -restore_edge \
!INST/PM_INST/state_retention_restore -save_edge \
INST/PM_INST/state_retention_save

#-----
#   update domains/modes
#-----
update_nominal_condition -name nom_0v81 -library_set wc_0v81
update_nominal_condition -name nom_0v72 -library_set wc_0v72

update_power_domain -name PDdefault -primary_power_net VDD -primary_ground_net \
VSS
update_power_domain -name PDshutoff_io -primary_power_net VDD_IO \
-primary_ground_net VSS
update_power_domain -name PDpll -primary_power_net Avdd -primary_ground_net \
Avss
update_power_domain -name PDram_virtual -primary_power_net VDDL \
-primary_ground_net VSS
```

## Encounter User Guide

### CPF 1.0e Script Example

```
update_power_domain -name PDram -primary_power_net VDDL_sw -primary_ground_net \
VSS
update_power_domain -name PDtdsp -primary_power_net VDD_sw -primary_ground_net \
VSS

update_power_mode -name PMdvfs1 -sdc_files ../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs1_off -sdc_files ../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs1_shutoffio_off -sdc_files \
../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs2 -sdc_files ../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMdvfs2_off -sdc_files ../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMdvfs2_shutoffio_off -sdc_files \
../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMscan -sdc_files ../RELEASE/mmmc/scan.sdc

#-----
#   update rules
#-----
update_power_switch_rule -name PDram_SW -cells { HEADERHVT1 } -prefix \
CDN_SW_RAM -peak_ir_drop_limit 0 -average_ir_drop_limit 0
update_power_switch_rule -name PDtdsp_SW -cells { HEADERHVT2 } -prefix \
CDN_SW_TDSP -peak_ir_drop_limit 0 -average_ir_drop_limit 0

update_isolation_rules -names ISORULE1 -location to -prefix CPF_ISO_
update_isolation_rules -names ISORULE3 -location to -prefix CPF_ISO_
update_isolation_rules -names ISORULE4 -location to -prefix CPF_ISO_

update_level_shifter_rules -names LSRULE_H2L3 -location to -cells { LVLHLD* \
} -prefix CPF_LS_
update_level_shifter_rules -names LSRULE_H2L_PLL -location to -prefix CPF_LS_
update_level_shifter_rules -names LSRULE_L2H2 -location to -prefix CPF_LS_

update_state_retention_rules -names \
INST/DSP_CORE_INST0/PDtdsp_retention_rule -cell_type master_slave
update_state_retention_rules -names \
INST/DSP_CORE_INST1/PDtdsp_retention_rule -cell_type ballon_latch

#-----
#   end
#-----
end_design
```

---

## CPF 1.1 Script Example

---

The following section contains an example of the CPF 1.1 file using a sample design and library.

For list of supported CPF commands and options within Encounter product family, see [Appendix E, “Supported CPF 1.1 Commands.”](#)

```
#-----
#   setting
#-----
set_cpf_version 1.1
set_hierarchy_separator /

#-----
#   define library_set/cells
#-----
define_library_set -name wc_0v81 -libraries { \
    ../LIBS/timing/library_wc_0v81.lib }
define_library_set -name bc_0v81 -libraries { \
    ../LIBS/timing/library_bc_0v81.lib }
define_library_set -name wc_0v72 -libraries { \
    ../LIBS/timing/library_wc_0v72.lib }
define_library_set -name bc_0v72 -libraries { \
    ../LIBS/timing/library_bc_0v72.lib }

define_always_on_cell -cells {PTLVLHLD* AOBUFF*} -power_switchable \
VDD -power TVDD -ground VSS

define_isolation_cell -cells { LVLLH* } -power VDD -ground VSS -enable \
NSLEEP -valid_location to
define_isolation_cell -cells { ISOHID* ISOLOD* } -power VDD -ground VSS \
-enable ISO -valid_location to

define_level_shifter_cell -cells { LVLHLD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells { PTLVLHLD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin TVDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHCD* } -input_voltage_range \
```

## Encounter User Guide

### CPF 1.1 Script Example

```
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 \
-output_voltage_input_pin NSLEEP -direction up -input_power_pin VDDL \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction up \
-input_power_pin VDDL -output_power_pin VDD -ground VSS -valid_location to

define_power_switch_cell -cells { HEADERHVT1 HEADERHVT2 } \
-stage_1_enable NSLEEPIN1 -stage_1_output NSLEEPOUT1 -stage_2_enable \
NSLEEPIN2 -stage_2_output NSLEEPOUT2 -type header -power_switchable VDD \
-power TVDD -stage_1_on_resistance 10 - stage_2_on_resistance 10

define_state_retention_cell -cells { MSSRPG* } -cell_type \
master_slave -clock_pin CP -restore_check !CP -save_function !CP \
-always_on_components { DFF_inst } -power_switchable VDD -power TVDD \
-ground VSS
define_state_retention_cell -cells { BLSPRG* } -cell_type ballon_latch \
-clock_pin CP -restore_function !NRESTORE -save_function SAVE \
-always_on_components { save_data } -power_switchable VDD -power TVDD \
-ground VSS

#-----
#   macro models
#-----

#-----
#   top design
#-----
set_design top

create_operating_corner -name PMdvfs2_bc -voltage 0.88 -process 1 -temperature \
0 -library_set bc_0v72
create_operating_corner -name PMdvfs1_bc -voltage 0.99 -process 1 -temperature \
0 -library_set bc_0v81
create_operating_corner -name PMdvfs1_wc -voltage 0.81 -process 1 -temperature \
125 -library_set wc_0v81
create_operating_corner -name PMdvfs2_wc -voltage 0.72 -process 1 -temperature \
125 -library_set wc_0v72

create_power_nets -nets VDD -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_EQ -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_sw -voltage { 0.72:0.81:0.09 } -internal \
-peak_ir_drop_limit 0 -average_ir_drop_limit 0
create_power_nets -nets VDDL -voltage 0.72 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDDL_sw -voltage 0.72 -internal -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets Avdd -voltage 0.81 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
```

## Encounter User Guide

### CPF 1.1 Script Example

```
create_power_nets -nets VDD_IO -voltage { 0.72:0.81:0.09 } \
    -external_shutoff_condition { io_shutoff_ack } -peak_ir_drop_limit 0 \
    -average_ir_drop_limit 0

create_ground_nets -nets Avss -voltage 0.00 -peak_ir_drop_limit 0 \
    -average_ir_drop_limit 0
create_ground_nets -nets VSS -voltage 0.00 -peak_ir_drop_limit 0 \
    -average_ir_drop_limit 0

create_nominal_condition -name nom_0v81 -voltage 0.81
create_nominal_condition -name nom_0v72 -voltage 0.72

#-----
#   create power domains
#-----
create_power_domain -name PDdefault -default
create_power_domain -name PDshutoff_io -instances { IOPADS_INST/Pspifspip \
    IOPADS_INST/Pspidip } -boundary_ports { spi_fs spi_data } \
    -external_controlled_shutoff -shutoff_condition io_shutoff_ack
create_power_domain -name PDpll -instances { INST/PLLCLK_INST \
    IOPADS_INST/Pibiasip IOPADS_INST/Ppllrstip IOPADS_INST/Prefclkip \
    IOPADS_INST/Presetip IOPADS_INST/Pvcomop IOPADS_INST/Pvcopop } - \
    boundary_ports { ibias reset \
        refclk vcom vcop pllrst }
create_power_domain -name PDram_virtual
create_power_domain -name PDram -instances { INST/RAM_128x16_TEST_INST } \
    -shutoff_condition !INST/PM_INST/power_switch_enable \
    -base_domains { PDram_virtual }
create_power_domain -name PDtdsp -instances { INST/RAM_128x16_TEST_INST1 \
    INST/DSP_CORE_INST0 INST/DSP_CORE_INST1 } -shutoff_condition \
    !INST/PM_INST/power_switch_enable -base_domains { PDdefault }

#-----
#   set instances
#-----
set_instance INST/RAM_128x16_TEST_INST1/RAM_128x16_INST -domain_mapping \
    { {RAM_DEFAULT PDtdsp} }

set_macro_model ram_256x16A

create_power_domain -name RAM_DEFAULT -boundary_ports { A* D* CLK CEN WEN Q* } \
    -default -external_controlled_shutoff

create_state_retention_rule -name RAM_ret -instances { mem* } -save_edge !CLK

update_power_domain -name RAM_DEFAULT -primary_power_net VDD \
    -primary_ground_net VSS

end_macro_model ram 256x16A

#-----
```

## Encounter User Guide

### CPF 1.1 Script Example

```
#  create power modes
#-----
create_power_mode -name PMdvfs1 -default -domain_conditions { PDpll@nom_0v81 \
    PDdefault@nom_0v81 PDTdsp@nom_0v81 PDram@nom_0v72 PDshutoff_io@nom_0v81 \
    PDram_virtual@nom_0v72 }
create_power_mode -name PMdvfs1_off -domain_conditions { PDpll@nom_0v81 \
    PDdefault@nom_0v81 PDshutoff_io@nom_0v81 PDram_virtual@nom_0v72 }
create_power_mode -name PMdvfs1_shutoffio_off -domain_conditions { \
    PDpll@nom_0v81 PDdefault@nom_0v81 PDram_virtual@nom_0v72 }
create_power_mode -name PMdvfs2 -domain_conditions { PDpll@nom_0v81 \
    PDdefault@nom_0v72 PDTdsp@nom_0v72 PDram@nom_0v72 PDshutoff_io@nom_0v72 \
    PDram_virtual@nom_0v72 }
create_power_mode -name PMdvfs2_off -domain_conditions { PDpll@nom_0v81 \
    PDdefault@nom_0v72 PDshutoff_io@nom_0v72 PDram_virtual@nom_0v72 }
create_power_mode -name PMdvfs2_shutoffio_off -domain_conditions { \
    PDpll@nom_0v81 PDdefault@nom_0v72 PDram_virtual@nom_0v72 }
create_power_mode -name PMscan -domain_conditions { PDpll@nom_0v81 \
    PDdefault@nom_0v81 PDTdsp@nom_0v81 PDram@nom_0v72 PDshutoff_io@nom_0v81 \
    PDram_virtual@nom_0v72 }

create_analysis_view -name AV_dvfs1_BC -mode PMdvfs1 -domain_corners { \
    PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDTdsp@PMdvfs1_bc PDram@PMdvfs2_bc \
    PDshutoff_io@PMdvfs1_bc }
create_analysis_view -name AV_dvfs1_WC -mode PMdvfs1 -domain_corners { \
    PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDTdsp@PMdvfs1_wc PDram@PMdvfs2_wc \
    PDshutoff_io@PMdvfs1_wc }
create_analysis_view -name AV_dvfs1_off_BC -mode PMdvfs1_off -domain_corners { \
    PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDshutoff_io@PMdvfs1_bc }
create_analysis_view -name AV_dvfs1_off_WC -mode PMdvfs1_off -domain_corners { \
    PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDshutoff_io@PMdvfs1_wc }
create_analysis_view -name AV_dvfs1_shutoffio_off_BC -mode \
    PMdvfs1_shutoffio_off -domain_corners { PDpll@PMdvfs1_bc \
    PDdefault@PMdvfs1_bc }
create_analysis_view -name AV_dvfs1_shutoffio_off_WC -mode \
    PMdvfs1_shutoffio_off -domain_corners { PDpll@PMdvfs1_wc \
    PDdefault@PMdvfs1_wc }
create_analysis_view -name AV_dvfs2_BC -mode PMdvfs2 -domain_corners { \
    PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc PDTdsp@PMdvfs2_bc PDram@PMdvfs2_bc \
    PDshutoff_io@PMdvfs2_bc }
create_analysis_view -name AV_dvfs2_WC -mode PMdvfs2 -domain_corners { \
    PDpll@PMdvfs1_wc PDdefault@PMdvfs2_wc PDTdsp@PMdvfs2_wc PDram@PMdvfs2_wc \
    PDshutoff_io@PMdvfs2_wc }
create_analysis_view -name AV_PMdvfs2_off_BC -mode PMdvfs2_off -domain_corners \
    { PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc PDshutoff_io@PMdvfs2_bc }
create_analysis_view -name AV_PMdvfs2_off_WC -mode PMdvfs2_off -domain_corners \
    { PDpll@PMdvfs1_wc PDdefault@PMdvfs2_wc PDshutoff_io@PMdvfs2_wc }
create_analysis_view -name AV_dvfs2_shutoffio_off_BC -mode \
    PMdvfs2_shutoffio_off -domain_corners { PDpll@PMdvfs1_bc \
    PDdefault@PMdvfs2_bc }
create_analysis_view -name AV_dvfs2_shutoffio_off_WC -mode \
    PMdvfs2_shutoffio_off -domain_corners { PDpll@PMdvfs1_wc \
```

## Encounter User Guide

### CPF 1.1 Script Example

```
PDdefault@PMdvfs2_wc }
create_analysis_view -name AV_scan_BC -mode PMscan -domain_corners { \
    PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDTdsp@PMdvfs1_bc PDram@PMdvfs2_bc \
    PDshutoff_io@PMdvfs1_bc }
create_analysis_view -name AV_scan_WC -mode PMscan -domain_corners { \
    PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDTdsp@PMdvfs1_wc PDram@PMdvfs2_wc \
    PDshutoff_io@PMdvfs1_wc }

#-----
#   create rules
#-----
create_power_switch_rule -name PDram_SW -domain PDram -external_power_net VDDL
create_power_switch_rule -name PDTdsp_SW -domain PDTdsp -external_power_net \
    VDD

create_isolation_rule -name ISORULE1 -isolation_condition { \
    !INST/PM_INST/isolation_enable } -from { PDTdsp } -to { PDdefault } \
    -isolation_target from -isolation_output high
create_isolation_rule -name ISORULE3 -isolation_condition { \
    !INST/PM_INST/isolation_enable } -from { PDram } -to { PDdefault } \
    -isolation_target from -isolation_output high
create_isolation_rule -name ISORULE4 -isolation_condition { \
    !INST/PM_INST/spi_ip_isolate } -from { PDshutoff_io } \
    -isolation_target from -isolation_output low

create_level_shifter_rule -name LSRULE_H2L3 -from { PDdefault } -to { PDram } \
    -exclude { INST/PM_INST/power_switch_enable }
create_level_shifter_rule -name LSRULE_H2L_PLL -from { PDpll }
create_level_shifter_rule -name LSRULE_L2H2 -from { PDram } -to { PDdefault }

create_state_retention_rule -name \
    INST/DSP_CORE_INST0/PDtdsp_retention_rule -instances { \
    INST/DSP_CORE_INST0 } -save_edge !INST/DSP_CORE_INST0/clk
create_state_retention_rule -name \
    INST/DSP_CORE_INST1/PDtdsp_retention_rule -instances { \
    INST/DSP_CORE_INST1 } -restore_edge \
    !INST/PM_INST/state_retention_restore -save_edge \
    INST/PM_INST/state_retention_save

#-----
#   update domains/modes
#-----
update_nominal_condition -name nom_0v81 -library_set wc_0v81
update_nominal_condition -name nom_0v72 -library_set wc_0v72

update_power_domain -name PDdefault -primary_power_net VDD -primary_ground_net \
    VSS -equivalent_power_nets VDD_EQ
update_power_domain -name PDshutoff_io -primary_power_net VDD_IO \
    -primary_ground_net VSS
update_power_domain -name PDpll -primary_power_net Avdd -primary_ground_net \
    Avss
```

## Encounter User Guide

### CPF 1.1 Script Example

```
update_power_domain -name PDram_virtual -primary_power_net VDDL \
    -primary_ground_net VSS
update_power_domain -name PDram -primary_power_net VDDL_sw -primary_ground_net \
    VSS
update_power_domain -name PDtdsp -primary_power_net VDD_sw -primary_ground_net \
    VSS

update_power_mode -name PMdvfs1 -sdc_files ../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs1_off -sdc_files ../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs1_shutoffio_off -sdc_files \
    ../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs2 -sdc_files ../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMdvfs2_off -sdc_files ../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMdvfs2_shutoffio_off -sdc_files \
    ../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMscan -sdc_files ../RELEASE/mmmc/scan.sdc

#-----
#   update rules
#-----
update_power_switch_rule -name PDram_SW -cells { HEADERHVT1 } -prefix \
    CDN_SW_RAM -peak_ir_drop_limit 0 -average_ir_drop_limit 0
update_power_switch_rule -name PDtdsp_SW -cells { HEADERHVT2 } -prefix \
    CDN_SW_TDSP -peak_ir_drop_limit 0 -average_ir_drop_limit 0

update_isolation_rules -names ISORULE1 -location to -prefix CPF_ISO_
update_isolation_rules -names ISORULE3 -location to -prefix CPF_ISO_
update_isolation_rules -names ISORULE4 -location to -prefix CPF_ISO_

update_level_shifter_rules -names LSRULE_H2L3 -location to -cells { LVLHLD* \
    } -prefix CPF_LS_
update_level_shifter_rules -names LSRULE_H2L_PLL -location to -prefix CPF_LS_
update_level_shifter_rules -names LSRULE_L2H2 -location to -prefix CPF_LS_

update_state_retention_rules -names \
    INST/DSP_CORE_INST0/PDtdsp_retention_rule -cell_type master_slave
update_state_retention_rules -names \
    INST/DSP_CORE_INST1/PDtdsp_retention_rule -cell_type balloon_latch

#-----
#   end
#-----
end_design
```



---

# Cadence-Specific Liberty Extensions

---

This appendix describes Cadence-specific extensions to the Synopsys Liberty (.lib) library file. These extensions allow the storage of the data required by SignalStorm® nanometer delay calculator's effective current source model (ECSM) voltage and current profiles.

This appendix contains information on the following topics:

- [Overview](#) on page 1369
- [Guidelines For Adding ECSM Extensions](#) on page 1370
- [Representing ECSM Information in a Library](#) on page 1370
- [Defining ECSM Extensions in a Library](#) on page 1371
  - [ecsm waveform Group](#) on page 1373
  - [ecsm waveform set Group](#) on page 1376
  - [ecsm capacitance Group](#) on page 1378
- [Example](#) on page 1382

## Overview

SignalStorm nanometer delay calculator uses an advanced cell driver model to represent the effect of non-linear switching waveforms on cell-based interconnect delay calculation and signal integrity. This ECSM model provides an efficient mechanism for storing output current or voltage profiles during active transitions on the circuit.

ECSM models are typically generated by the Encounter library characterizer, and stored directly in a SignalStorm database. However, you can also derive ECSM information from a generic characterization flow and store it in a traditional library file using the extensions described in this appendix. These extensions allow the contents of ECSM models to be stored in library files in a way that is compatible with existing data and with other new signal integrity constructs.

**Note:** The presence of ECSM data does not interfere with other uses of the library file.

## Guidelines For Adding ECSM Extensions

- Extensions must be compatible with existing or new library model constructs. They cannot be stored as comments, which could potentially be stripped out by internal or commercial tools that read the library models.
- Extensions must resemble similar library constructs.
- Extensions must use the same context as the equivalent library construct. For example, units must be consistent. If an extension uses a current value and the current unit is defined as 1 milliampere, the values placed in the attribute must be in milliamperes.
- Syntactically correct extensions added to the libraries models must pass through the Synopsys Library Compiler, and should not cause any change in behavior in any tool that uses the compiled models.
- Extensions should be easily extracted by tools that correctly parse library models.
- Storage of the waveforms must be efficient. Appropriate reduction must be performed on the waveforms to a user-specified level of accuracy. The controls provided to adjust the accuracy should enable you to achieve a compromise between the accuracy of the analysis and the size of the model.

## Representing ECSM Information in a Library

ECSM information is stored in the form of output voltage waveforms, which enables the library to include more information about the transition, and to improve the driver model accuracy. Given that the library transition table is based on two types of indexes (input slew rate and output loading capacitance), and the delay model uses a lumped capacitance to represent the observed loading, the following equation computes the output current ( $I_{out}$ ):

$$ECSM(t_1, t_2) = \frac{\int_{t_1}^{t_2} I_{out}(t) dt}{t_2 - t_1} = C_{load} \times \frac{(V_{out}(t_2) - V_{out}(t_1))}{t_2 - t_1}$$

You can use this equation to convert the output voltage waveform to ECSM. To support multiple supply voltage corners, normalize the output voltage numbers from 0.0 to 1.0 of the supply voltage. The effective current should also be normalized to the supply voltage.

Using the output voltage waveform approach, an ECSM extension becomes a table containing voltage over transition times for every input slew and output loading capacitance index combination. Add this table to the output transition table group using the following format:

```
rise_transition ( template_of_slew_load ) { // fall_transition
(template_of_slew_load) {
    index_1 : // redefine of slew rate index
    index_2 : // redefine of loading index

    ecsm_waveform( name0 ) {
        index_1 : "..."; // output voltage sample points
        values : "..."; // output voltage sample times
    }
    ecsm_waveform(name1 ) {
        index_1 : "..."; // output voltage sample points
        values : "..."; // output voltage sample times
        ...
    }
}
```

## Defining ECSM Extensions in a Library

Three user-defined groups can be added to a library to define an ECSM extension:

- `ecsm_waveform`
- `ecsm_waveform_set`
- `ecsm_capacitance`

The `ecsm_waveform` and `ecsm_waveform_set` groups are alternative ways of describing the output rise or fall voltage waveform during a transition. These groups allow the creation of output voltage waveforms as a function of time. The `ecsm_capacitance` group describes the input capacitance during the rise or fall transition. The input capacitance can be specified as a function of input transition and output load.

**Note:** In accordance with ECSM specification version 1.2, the `ecsm_capacitance` group also allows you to represent the input pin capacitance as a function of input transition and output load.

All three groups are defined within the `rise_transition` or `fall_transition` group within a `timing` group for regular delay arcs, and within the `retain_rise_slew` or `retain_fall_slew` group within a `timing` group for retain delay arcs. The

## Encounter User Guide

### Cadence-Specific Liberty Extensions

---

`rise_transition, fall_transition, retain_rise_slew, and retain_fall_slew` groups specify the output slew or transition time as a function of the input net transition and total output capacitance.

1. To include these groups in a library, add the following `define_group` statements to the library.

```
define_group( ecsm_waveform, rise_transition);
define_group( ecsm_waveform, fall_transition);
define_group( ecsm_waveform_set, rise_transition);
define_group( ecsm_waveform_set, fall_transition);
define_group( ecsm_capacitance, rise_transition);
define_group( ecsm_capacitance, fall_transition);
define_group( ecsm_capacitance, pin);
```

or:

```
define_group( ecsm_waveform, retain_rise_slew);
define_group( ecsm_waveform, retain_fall_slew);
define_group( ecsm_waveform_set, retain_rise_slew);
define_group( ecsm_waveform_set, retain_fall_slew);
define_group( ecsm_capacitance, retain_rise_slew);
define_group( ecsm_capacitance, retain_fall_slew);
define_group( ecsm_capacitance, pin);
```

2. Add the following `define` statements to the library to define attributes for the groups:

```
define( index_1, ecsm_waveform, string );
define( values, ecsm_waveform, string );
define( values, ecsm_waveform_set, string );
define( values, ecsm_capacitance, string );
```

**Note:** The `ecsm_waveform_set` group must be named using a reference to a lookup table template defined with the `ecsm_lut_template` construct. Add the following `define_group` and `define` statements to the library to describe the lookup template table:

```
define_group( ecsm_lut_template, library );
define( variable_1, ecsm_lut_template, string );
define( index_1, ecsm_lut_template, string );
```

You must also include a version statement to enable the exact processing of ECSM constructs. Add the following version statement in the library:

```
define( ecsm_version, library, float );
```

The current version is 1.2.

## **ecsm\_waveform Group**

The `ecsm_waveform` group enables the output voltage waveforms to be specified separately for each index permutation suggested by the slew and load indexes. It also enables each waveform to be sampled at different points so that waveforms that are predominantly linear, can be represented with fewer sample points. The number of `ecsm_waveform` groups must match the number of entries in the `values` attribute of the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group.

The name of the `ecsm_waveform` group must be an integer enclosed within quotation marks. The minimum value of the integer must be 0, and the maximum value must be a number that is one less than the number of entries in the `values` attribute of the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group.

Including the `ecsm_waveform` group in the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group specifies that all information related to the transition arc also applies to the `ecsm_waveform` group. If not explicitly specified, the lookup template name and the index overrides are inherited from the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group. Attributes associated with the timing group in which the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group resides are also associated with the `ecsm_waveform` group, including `related_pin`, `timing_sense`, and `timing_type`.

[Figure I-1](#) on page 1374 shows four sampled output voltage waveforms that begin a transition at 1 nanosecond. The following example represents an `ecsm_waveform` group (for a regular delay arc) that specifies the points on each waveform shown in the figure:

```
rise_transition( temp_1x4 ) {
    index_1 : "0.1n";
    index_2 : "0.1p 0.2p 0.3p 0.4p";
    values ( "0.01n, 0.02n, 0.026n, 0.45n" );
}

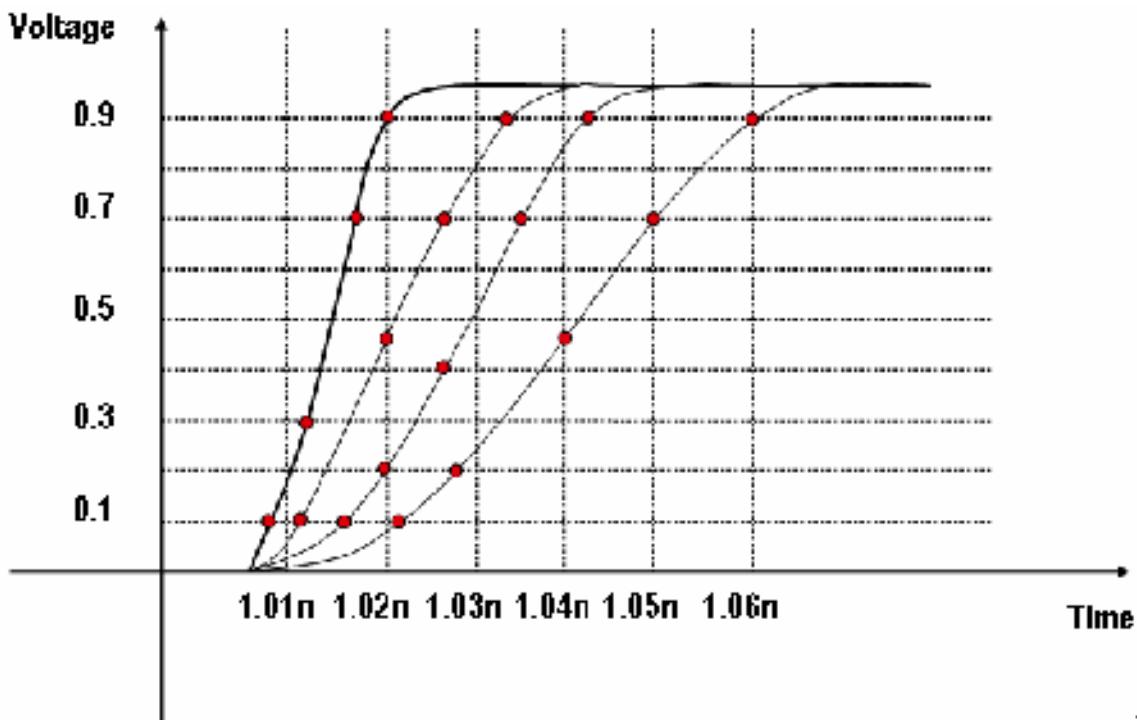
ecsm_waveform( "0" ) {
    index_1 : "0.1, .3, .7,.9";
    values : "1.005n, 1.012n, 1.018n, 1.02n";
}
ecsm_waveform( "1" ) {
    index_1 : "0.1, .48, .7,.9";
    values : "1.011n, 1.02n, 1.027n, 1.032n";
}
ecsm_waveform( "2" ) {
    index_1 : "0.1, .2, .4, .7,.9";
```

## Encounter User Guide

### Cadence-Specific Liberty Extensions

```
    values : "1.015n, 1.02n, 1.028n, 1.035n, 1.07n";
}
ecsm_waveform( "3" ) {
    index_1 : "0.1, .2, .45, .7,.9";
    values : "1.021n, 1.029n, 1.04n, 1.06n, 1.07n";
}
}
```

**Figure I-1 Sampled Waveform**



The following example represents an `ecsm_waveform` group for a retain delay arc:

```
retain_rise_slew(delay_template_6x6) {
    index_1 ("0.001, 0.0105, 0.02, 0.039, 0.077, 0.152");
    index_2 ("0.012007, 0.09354, 0.189187, 0.373938, 0.757224, 1.50616");
    values ( \
        "0.026141, 0.027748, 0.031751, 0.038769, 0.051329, 0.070025", \
        "0.136231, 0.135178, 0.137198, 0.137161, 0.145185, 0.160992", \
        "0.247332, 0.247016, 0.244592, 0.244943, 0.251245, 0.260942", \
        "0.469122, 0.469234, 0.463448, 0.467459, 0.464259, 0.478051", \
        "0.912834, 0.903097, 0.907181, 0.907186, 0.907485, 0.902419", \
        "1.78894, 1.77071, 1.78538, 1.78471, 1.76567, 1.781");
```

## Encounter User Guide

### Cadence-Specific Liberty Extensions

---

```
ecsm_waveform("0") {
    index_1 : "0.02, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.98" ;
    values : "0.100926, 0.104628, 0.108441, 0.112042, 0.11568, 0.119465,
0.12354, 0.128418, 0.134582, 0.144439, 0.160979" ;
}
ecsm_waveform("1") {
    index_1 : "0.02, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.98" ;
    values : "0.101614, 0.108072, 0.113352, 0.117633, 0.121708, 0.125782,
0.129998, 0.13488, 0.141099, 0.15062, 0.164307" ;
}
ecsm_waveform("2") {
    index_1 : "0.02, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.98" ;
    values : "0.1019, 0.109498, 0.116861, 0.122593, 0.127486, 0.132252,
0.136939, 0.142209, 0.148612, 0.158116, 0.178427" ;
}
ecsm_waveform("3") {
    index_1 : "0.02, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.98" ;
    values : "0.103866, 0.119331, 0.129292, 0.137018, 0.143679, 0.149411,
0.155144, 0.161135, 0.168061, 0.178948, 0.198529" ;
}
...
}

ecsm_waveform("35") {
    index_1 : "0.02, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.98" ;
    values : "0.148963, 0.344816, 0.57925, 0.813683, 1.0576, 1.3144, 1.59557,
1.93059, 2.36025, 2.98918, 4.13212" ;
}
```

}

### **ecsm\_waveform Attributes**

Two simple attributes are allowed in an `ecsm_waveform` group:

- `index_1`

The `index_1` attribute is a comma-separated list of floating-point numbers representing normalized output voltage sample points. These values are normalized and must be between 0.0 and 1.0.

- `values`

The `values` attribute is a comma-separated list of floating-point numbers representing the times at which the output voltages are sampled. The number of floating-point numbers must be equal to the number of entries in the `rise_transition`,

`fall_transition, retain_rise_slew or retain_fall_slew` table multiplied by the number of entries in the `index_1` attribute. The time entries in this attribute must monotonically increase, but it is not necessary to start from a time reference of 0.

Each value represents the time at which the corresponding sampled point in the `index_1` list is crossed for the first time. The `index_1` and `values` attributes must have the same number of floating-point entries. Each `ecsm_waveform` group can have a different number of entries for this attribute; however, the number of entries for a group must match the number of points in the corresponding `index_1` attribute. The time units for this attribute use the value of the library-level `time_unit` attribute.

## Waveform Order and Size

It is not necessary for `ecsm_waveform` groups to appear in order. However, the integer number of `ecsm_waveform` groups must correspond exactly to the number of entries in the `values` attribute of the `rise_transition` or `fall_transition` group. The parser uses the integer number to determine the value of `index_1` and `index_2`.

The `ecsm_waveform` group provides more accurate voltage waveforms using a smaller number of sampling points. You can use any kind of waveform reduction method to obtain the minimum number of points, and linear interpolation to eliminate those points between two voltage points. Set the interpolation error to less than 0.001.

## **`ecsm_waveform_set` Group**

The `ecsm_waveform_set` group enables output voltage waveforms for all index permutations to be stored in the same group. It creates more compact tables; however, each waveform must use the same sampling voltages.

For example, the following `ecsm_waveform_set` group uses five sampling voltages to describe the waveforms in [Figure I-1](#) on page 1374:

```
rise_transition( temp_1X4 ) {
    index_1 : "0.1n";
    index_2 : "0.1p 0.2p 0.3p 0.4p";

    ecmp_waveform_set(template_5_ecsm_table) {
        index_1 : "0.1, 0.3, 0.5, 0.7, 0.9"
        values : "1.005n, 1.012n, 1.014n, 1.017n, 1.02n, \
                  1.011n, 1.017n, 1.021n, 1.027n, 1.033n, \
                  1.017n, 1.023n, 1.03n, 1.035n, 1.042n, \
                  1.021n, 1.033n, 1.041n, 1.05n, 1.06n"
    }
}
```

}

The name of the `ecsm_waveform_set` group must be the name of a lookup table template defined with the `ecsm_lut_template` construct. The `ecsm_lut_template` table defines the points at which the output voltage is sampled. The sample points can be overridden within the `ecsm_waveform_set` group, if required.

Attributes associated with the `timing` group in which the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group resides are also associated with the `ecsm_waveform_set` group, including `related_pin`, `timing_sense`, and `timing_type`.

Including the `ecsm_waveform_set` group in the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group specifies that all information related to the transition arc also applies to the `ecsm_waveform_set` group. If not explicitly specified, the lookup table template name and the index overrides are inherited from the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group.

### **`ecsm_waveform_set` Attributes**

Two simple attributes are allowed in an `ecsm_waveform_set` group:

- **`index_1`**

The `index_1` attribute is a comma-separated list of floating-point numbers representing normalized output voltage sample points. These values are normalized and must be between 0.0 and 1.0.

- **`values`**

The `values` attribute is a comma-separated list of floating-point numbers representing the times at which the output voltages are sampled. All waveforms corresponding to the entries in the `values` attribute within the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group are compactly represented. The number of floating-point numbers must be equal to the number of entries in the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` table multiplied by the number of entries in the `index_1` attribute. The time entries in this attribute must monotonically increase, but it is not necessary to start from a time reference of 0.

Each value represents the time at which the corresponding sampled point in the `index_1` list is crossed for the first time. The time units for this attribute use the value of the library-level `time_unit` attribute.

## Waveform Order and Size

Waveforms are processed in an `ecsm_waveform_set` group in the order specified by the rules used to analyze the values within a `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group. The indexes are assumed to change outwardly from the innermost row. The sample points specified in the `index_1` attribute within the `ecsm_waveform_set`, or in the referenced template, are assumed to be in the innermost loop.

The `ecsm_waveform_set` group outputs the most compact table, although you can have more points for waveforms that switch very fast. You should have at least five points to represent the waveform. You usually need to add more points after the 50 percent period.

- Use the following eight sampling voltage points for rising transition time:

0.05, 0.1, 0.3, 0.5, 0.6, 0.8, 0.9, 0.95

- Use the following eight points for a falling transition:

0.95, 0.9, 0.7, 0.5, 0.4, 0.2, 0.1, 0.05

You can add sampling points to obtain more accurate models.

## **`ecsm_capacitance` Group**

The `ecsm_capacitance` group describes the input capacitance during the rise or fall transition. It specifies the input capacitance for each input transition and capacitive load defined by the specified `lu_table_template` or `index_1` and `index_2` overrides. The indexes are inherited from the specified `lu_table_template` or `index_1` and `index_2` overrides within the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group, and cannot be overridden. The `ecsm_capacitance` group requires the name `rise` or `fall`, and must match the transition direction of the parent group.

Including the `ecsm_capacitance` group in the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group specifies that the capacitance values are processed in the order specified by the rules used to analyze the values within a `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group.

The following example represents an `ecsm_capacitance` group in a library for a regular delay arc:

```
rise_transition( temp_1x4 ) {  
    index_1 : "0.1n";  
    index_2 : "0.1p 0.2p 0.3p 0.4p";  
    values ( "0.01n, 0.02n, 0.026n, 0.45n" );
```

## Encounter User Guide

### Cadence-Specific Liberty Extensions

---

```
        ecssm_capacitance( rise ) {
            values : "0.01, 0.02, 0.03, 0.04";
        }
    }
```

The following example represents an `ecsm_capacitance` group in a library for a retain delay arc:

```
retain_rise_slew(delay_template_6x6) {
    index_1 ("0.001, 0.0105, 0.02, 0.039, 0.077, 0.152");
    index_2 ("0.012007, 0.09354, 0.189187, 0.373938, 0.757224, 1.50616");
    values ( \
        "0.026141, 0.027748, 0.031751, 0.038769, 0.051329, 0.070025", \
        "0.136231, 0.135178, 0.137198, 0.137161, 0.145185, 0.160992", \
        "0.247332, 0.247016, 0.244592, 0.244943, 0.251245, 0.260942", \
        "0.469122, 0.469234, 0.463448, 0.467459, 0.464259, 0.478051", \
        "0.912834, 0.903097, 0.907181, 0.907186, 0.907485, 0.902419", \
        "1.78894, 1.77071, 1.78538, 1.78471, 1.76567, 1.781");
}

ecsm_capacitance(rise) {
    values : "0.000967, 0.001051, 0.001088, 0.001122, 0.001151, 0.001176, \
              0.000967, 0.001051, 0.001088, 0.001122, 0.001151, 0.001176, \
              0.000967, 0.001051, 0.001088, 0.001122, 0.001151, 0.001176, \
              0.000967, 0.001051, 0.001088, 0.001122, 0.001151, 0.001176, \
              0.000967, 0.001051, 0.001088, 0.001122, 0.001151, 0.001176, \
              0.000967, 0.001051, 0.001088, 0.001122, 0.001151, 0.001176" ;
}
```

Including the `ecsm_capacitance` group in the `pin` group allows you to represent the input pin capacitance as a function of input transition and output load. For example:

```
cell (cellname) {
    pin (pinname) {
        ecssm_capacitance(rise) {
            index_1 : "0.1n 0.2n";
            values   : "0.01, 0.02";
        }
        ecssm_capacitance(fall) {
            index_1 : "0.1n 0.2n";
            values   : "0.01, 0.02 ";
        }
    }
}
```

```
}
```

Attributes associated with the `timing` group in which the `pin` group resides are also associated with the `ecsm_capacitance` group, including `timing_sense`, and `timing_type`.

### **ecsm\_capacitance Attributes**

One simple attribute is allowed in an `ecsm_capacitance` group:

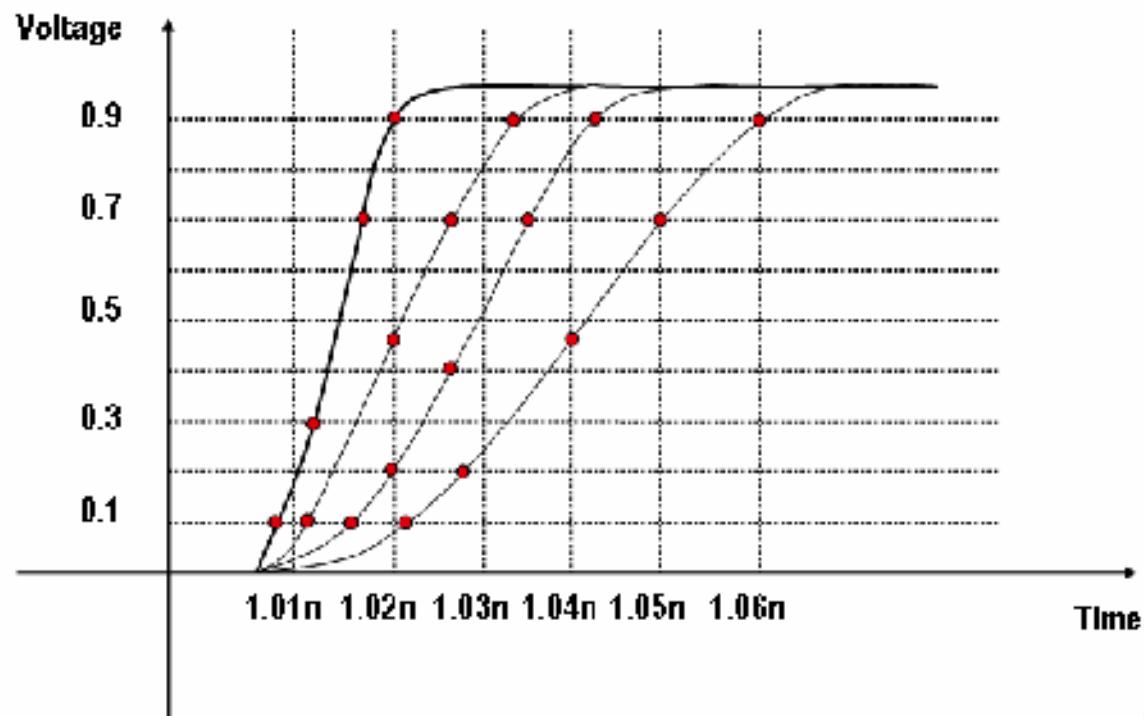
■ **values**

The `values` attribute is a comma-separated list of floating-point numbers representing the input capacitance at each specified index point. The values for `index_1` and `index_2` cannot be overridden and are inherited from the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group. The number of floating point numbers must be equal to the number of entries in the `values` attribute in the `rise_transition`, `fall_transition`, `retain_rise_slew` or `retain_fall_slew` group. The capacitive units for this attribute use the same value as the library-level `capacitive_load_unit` attribute.

Figure I-2 on page 1381 shows four sampled output voltage waveforms that begin a transition at 1 nanosecond.

**Encounter User Guide**  
Cadence-Specific Liberty Extensions

**Figure I-2 Sampled Waveform**



## Example

The following example of a Liberty library includes ECSM extensions:

```
library (typ) {
    delay_model : table_lookup ;
    date : "Tue Mar 25 14:54:48 CST 2003" ;
    time_unit : 1ns ;
    voltage_unit : 1V ;
    current_unit : 1mA ;
    capacitive_load_unit(1, pf);
    pulling_resistance_unit : 1kohm ;
    leakage_power_unit : 1mW ;
    input_threshold_pct_fall : 50.0 ;
    input_threshold_pct_rise : 50.0 ;
    output_threshold_pct_fall : 50.0 ;
    output_threshold_pct_rise : 50.0 ;
    slew_lower_threshold_pct_fall : 10.0 ;
    slew_lower_threshold_pct_rise : 10.0 ;
    slew_upper_threshold_pct_fall : 90.0 ;
    slew_upper_threshold_pct_rise : 90.0 ;
    nom_process : 1.0 ;
    nom_temperature : 25 ;
    nom_voltage : 1.5 ;
    default_cell_leakage_power : 0.0 ;
    default_fanout_load : 1.0 ;
    default inout_pin_cap : 1.0 ;
    default input_pin_cap : 1.0 ;
    default_leakage_power_density : 0.0 ;
    default output_pin_cap : 0.0 ;
    define_group( ecsm_lut_template, library );
    define_group( ecsm_waveform_set, rise_transition);
    define_group( ecsm_waveform_set, fall_transition);
    define_group( ecsm_waveform, rise_transition);
    define_group( ecsm_waveform, fall_transition);
    define_group( ecsm_capacitance, rise_transition);
    define_group( ecsm_capacitance, fall_transition);
    define( ecsm_version, library, float)
    define( variable_1, ecsm_lut_template, string );
    define( index_1, ecsm_lut_template, string );
    define( index_1, ecsm_waveform_set, string );
```

## Encounter User Guide

### Cadence-Specific Liberty Extensions

---

```
define( values, ecsm_waveform_set, string );
define( index_1, ecsm_waveform, string );
define( values, ecsm_waveform, string );
define( index_1, ecsm_capacitance, string );
ecsm_version : 1.2

input_voltage(default) {
    vil : 0.0 ;
    vih : 1.5 ;
    vimin : 0.0 ;
    vimax : 1.5 ;
}

operating_conditions(typ) {
    process : 1.0 ;
    temperature : 25 ;
    voltage : 1.5 ;
}

output_voltage(default) {
    vol : 0.0 ;
    voh : 1.5 ;
    vomin : 0.0 ;
    vomax : 1.5 ;
}

lu_table_template(tmg_ntin_oload_5x5) {
    variable_1 : input_net_transition ;
    variable_2 : total_output_net_capacitance ;
    index_1("1.0, 2.0, 3.0, 4.0, 5.0");
    index_2("1.0, 2.0, 3.0, 4.0, 5.0");
}

power_lut_template(pwr_tin_oload_3x3) {
    variable_1 : input_transition_time ;
    variable_2 : total_output_net_capacitance ;
    index_1("1.0, 2.0, 3.0");
    index_2("1.0, 2.0, 3.0");
}

ecsm_lut_template(ecsm_nvolt_5) {
```

## Encounter User Guide

### Cadence-Specific Liberty Extensions

---

```
variable_1 : normalized_voltage;
index_1 : "1.0, 2.0, 3.0, 4.0, 5.0";
}

cell(INV1) {
    area : 1.0 ;
    cell_leakage_power : 4.467483e-06 ;

    pin(A) {
        capacitance : 0.00230301522 ;
        direction : input ;
    }

    pin(Z) {
        direction : output ;
        function : "!A" ;

        internal_power()
        related_pin : "A" ;

        power(pwr_tin_oload_3x3) {
            index_1("0.074822, 0.249940, 0.828130");
            index_2("0.000010, 0.051277, 0.412740");
            values("1.673300, 1.478500, 1.323100", \
                   "1.746800, 1.541300, 1.372000", \
                   "2.187000, 1.934600, 1.669400");
        }
    }
}

timing() {
    related_pin : "A" ;
    timing_sense : negative_unate ;
    timing_type : combinational ;

    cell_fall(tmg_ntin_oload_5x5) {
        index_1("0.01, 0.06494796, 0.2578274, 0.6261584, 1.2");
        index_2("0.01, 0.030465, 0.1023016, 0.239484, 0.4532074");
        values("0.0176577, 0.0450122, 0.1403913, 0.3230411, 0.6076288", \
               "0.0273645, 0.055584, 0.1511069, 0.3335839, 0.6181163", \
               "0.0310345, 0.0807605, 0.1886131, 0.3696548, 0.6535805", \
               "0.0133436, 0.0864669, 0.2438067, 0.4415566, 0.7228604", \
               "0.0200000, 0.0500000, 0.1000000, 0.2000000, 0.4000000");
    }
}
```

## Encounter User Guide

### Cadence-Specific Liberty Extensions

---

```
        "0.000844, 0.0666426, 0.2756392, 0.5369333, 0.8356729");
    }

cell_rise(tmg_ntin_oload_5x5) {
    index_1("0.01, 0.06494796, 0.2578274, 0.6261584, 1.2");
    index_2("0.01, 0.01745396, 0.043619, 0.09358491, 0.1714293");
    values("0.0403989, 0.0630895, 0.1434606, 0.2971998, 0.5368189", \
           "0.0520428, 0.0741513, 0.1547805, 0.3085111, 0.5480488", \
           "0.0903201, 0.1192166, 0.2002351, 0.3506313, 0.5883372", \
           "0.1425557, 0.1785485, 0.2836617, 0.4403199, 0.6725487", \
           "0.2117557, 0.2558067, 0.379942, 0.5725538, 0.8146103");
}

fall_transition(tmg_ntin_oload_5x5) {
    index_1("0.01, 0.06494796, 0.2578274, 0.6261584, 1.2");
    index_2("0.01, 0.030465, 0.1023016, 0.239484, 0.4532074");
    values("0.0331679, 0.0861214, 0.2739402, 0.6353564, 1.198358", \
           "0.0484462, 0.095128, 0.274244, 0.6354434, 1.198372", \
           "0.1031723, 0.1519032, 0.3070776, 0.6442821, 1.198351", \
           "0.1955602, 0.2593558, 0.4179065, 0.7180122, 1.235083", \
           "0.3278351, 0.4087576, 0.6007222, 0.8868572, 1.358997");

    ecsm_waveform("0") {
        index_1 : "0.1, 0.3, 0.5, 0.7, 0.9" ;
        values : "0.0, 0.02, 0.025, 0.45, 0.85" ;
    }

    ecsm_waveform("1") {
        index_1 : "0.1, 0.3, 0.5, 0.7, 0.9" ;
        values : "0.0, 0.02, 0.025, 0.45, 0.85" ;
    }

    ecsm_capacitance(fall) {
        values : "0.00, 0.01, 0.02, 0.03, 0.04, \
                  0.05, 0.06, 0.07, 0.08, 0.09, \
                  0.10, 0.11, 0.12, 0.13, 0.14, \
                  0.15, 0.16, 0.17, 0.18, 0.19, \
                  0.20, 0.21, 0.22, 0.23, 0.24" ;
    }
}
```

## Encounter User Guide

### Cadence-Specific Liberty Extensions

---

```
rise_transition(tmg_ntin_oload_5x5) {
    index_1("0.01, 0.06494796, 0.2578274, 0.6261584, 1.2");
    index_2("0.01, 0.01745396, 0.043619, 0.09358491, 0.1714293");
    values("0.0802415, 0.132298, 0.3133739, 0.6588676, 1.197433", \
           "0.0894187, 0.1360771, 0.3130822, 0.6588872, 1.197413", \
           "0.1337757, 0.1777775, 0.3401834, 0.6683502, 1.197639", \
           "0.2099556, 0.261308, 0.4215237, 0.7312425, 1.236211", \
           "0.3291244, 0.3794321, 0.5553791, 0.8574848, 1.341265");

    ecssm_waveform("1") {
        index_1 : "0.1, 0.3, 0.5, 0.7, 0.9" ;
        values : "0.0, 0.02, 0.025, 0.45, 0.85" ;
    }

    ecssm_waveform("0") {
        index_1 : "0.1, 0.3, 0.5, 0.7, 0.9" ;
        values : "0.0, 0.02, 0.025, 0.45, 0.85" ;
    }

    ecssm_capacitance(rise) {
        values : "0.00, 0.01, 0.02, 0.03, 0.04, \
                  0.05, 0.06, 0.07, 0.08, 0.09, \
                  0.10, 0.11, 0.12, 0.13, 0.14, \
                  0.15, 0.16, 0.17, 0.18, 0.19, \
                  0.20, 0.21, 0.22, 0.23, 0.24" ;
    }
}
}
}
}

cell(INV2) {
    area : 1.0 ;
    cell_leakage_power : 4.467483e-06 ;

    pin(A) {
        capacitance : 0.00230301522 ;
        direction : input ;
    }

    pin(z) {
```

## Encounter User Guide

### Cadence-Specific Liberty Extensions

---

```
direction : output ;
function : "!A" ;

internal_power() {
    related_pin : "A" ;

    power(pwr_tin_oload_3x3) {
        index_1("0.074822, 0.249940, 0.828130") ;
        index_2("0.000010, 0.051277, 0.412740");
        values("1.673300, 1.478500, 1.323100", \
               "1.746800, 1.541300, 1.372000", \
               "2.187000, 1.934600, 1.669400");
    }
}

timing() {
    related_pin : "A" ;
    timing_sense : negative_unate ;
    timing_type : combinational ;

    cell_fall(tmg_ntin_oload_5x5) {
        index_1("0.01, 0.06494796, 0.2578274, 0.6261584, 1.2");
        index_2("0.01, 0.030465, 0.1023016, 0.239484, 0.4532074");
        values("0.0176577, 0.0450122, 0.1403913, 0.3230411, 0.6076288", \
               "0.0273645, 0.055584, 0.1511069, 0.3335839, 0.6181163", \
               "0.0310345, 0.0807605, 0.1886131, 0.3696548, 0.6535805", \
               "0.0133436, 0.0864669, 0.2438067, 0.4415566, 0.7228604", \
               "0.000844, 0.0666426, 0.2756392, 0.5369333, 0.8356729");
    }
}

cell_rise(tmg_ntin_oload_5x5) {
    index_1("0.01, 0.06494796, 0.2578274, 0.6261584, 1.2");
    index_2("0.01, 0.01745396, 0.043619, 0.09358491, 0.1714293");
    values("0.0403989, 0.0630895, 0.1434606, 0.2971998, 0.5368189", \
           "0.0520428, 0.0741513, 0.1547805, 0.3085111, 0.5480488", \
           "0.0903201, 0.1192166, 0.2002351, 0.3506313, 0.5883372", \
           "0.1425557, 0.1785485, 0.2836617, 0.4403199, 0.6725487", \
           "0.2117557, 0.2558067, 0.379942, 0.5725538, 0.8146103");
}

fall_transition(tmg_ntin_oload_5x5) {
```

## Encounter User Guide

### Cadence-Specific Liberty Extensions

---

```
index_1("0.01, 0.06494796, 0.2578274, 0.6261584, 1.2");
index_2("0.01, 0.030465, 0.1023016, 0.239484, 0.4532074");
values("0.0331679, 0.0861214, 0.2739402, 0.6353564, 1.198358", \
       "0.0484462, 0.095128, 0.274244, 0.6354434, 1.198372", \
       "0.1031723, 0.1519032, 0.3070776, 0.6442821, 1.198351", \
       "0.1955602, 0.2593558, 0.4179065, 0.7180122, 1.235083", \
       "0.3278351, 0.4087576, 0.6007222, 0.8868572, 1.358997");

ecsm_waveform_set(ecsm_nvolt_5) {
    index_1 : "0.1, 0.3, 0.5, 0.7, 0.9" ;
    values : "0.000, 0.02, 0.025, 0.45, 0.85, \
              "0.001, 0.02, 0.025, 0.45, 0.85, \
              "0.002, 0.02, 0.025, 0.45, 0.85, \
              "0.003, 0.02, 0.025, 0.45, 0.85, \
              "0.004, 0.02, 0.025, 0.45, 0.85, \
              "0.005, 0.02, 0.025, 0.45, 0.85, \
              "0.006, 0.02, 0.025, 0.45, 0.85, \
              "0.007, 0.02, 0.025, 0.45, 0.85, \
              "0.008, 0.02, 0.025, 0.45, 0.85, \
              "0.009, 0.02, 0.025, 0.45, 0.85, \
              "0.010, 0.02, 0.025, 0.45, 0.85, \
              "0.011, 0.02, 0.025, 0.45, 0.85, \
              "0.012, 0.02, 0.025, 0.45, 0.85, \
              "0.013, 0.02, 0.025, 0.45, 0.85, \
              "0.014, 0.02, 0.025, 0.45, 0.85, \
              "0.015, 0.02, 0.025, 0.45, 0.85, \
              "0.016, 0.02, 0.025, 0.45, 0.85, \
              "0.017, 0.02, 0.025, 0.45, 0.85, \
              "0.018, 0.02, 0.025, 0.45, 0.85, \
              "0.019, 0.02, 0.025, 0.45, 0.85, \
              "0.020, 0.03, 0.035, 0.45, 0.85, \
              "0.021, 0.03, 0.035, 0.45, 0.85, \
              "0.022, 0.03, 0.035, 0.45, 0.85, \
              "0.023, 0.03, 0.035, 0.45, 0.85, \
              "0.024, 0.03, 0.035, 0.45, 0.85" ;
}

ecsm_capacitance(fall) {
    values : "0.00, 0.01, 0.02, 0.03, 0.04, \
              0.05, 0.06, 0.07, 0.08, 0.09, \
              0.10, 0.11, 0.12, 0.13, 0.14, \

```

## Encounter User Guide

### Cadence-Specific Liberty Extensions

---

```
    0.15, 0.16, 0.17, 0.18, 0.19, \
    0.20, 0.21, 0.22, 0.23, 0.24" ;
}

rise_transition(tmg_ntin_oload_5x5) {
    index_1("0.01, 0.06494796, 0.2578274, 0.6261584, 1.2");
    index_2("0.01, 0.01745396, 0.043619, 0.09358491, 0.1714293");
    values("0.0802415, 0.132298, 0.3133739, 0.6588676, 1.197433", \
           "0.0894187, 0.1360771, 0.3130822, 0.6588872, 1.197413", \
           "0.1337757, 0.1777775, 0.3401834, 0.6683502, 1.197639", \
           "0.2099556, 0.261308, 0.4215237, 0.7312425, 1.236211", \
           "0.3291244, 0.3794321, 0.5553791, 0.8574848, 1.341265");

    ecsm_waveform_set(ecsm_nvolt_5) {
        index_1 : "0.0, 0.2, 0.4, 0.6, 0.8" ;
        values : "0.000, 0.02, 0.025, 0.45, 0.85, \
                  "0.001, 0.02, 0.025, 0.45, 0.85, \
                  "0.002, 0.02, 0.025, 0.45, 0.85, \
                  "0.003, 0.02, 0.025, 0.45, 0.85, \
                  "0.004, 0.02, 0.025, 0.45, 0.85, \
                  "0.005, 0.02, 0.025, 0.45, 0.85, \
                  "0.006, 0.02, 0.025, 0.45, 0.85, \
                  "0.007, 0.02, 0.025, 0.45, 0.85, \
                  "0.008, 0.02, 0.025, 0.45, 0.85, \
                  "0.009, 0.02, 0.025, 0.45, 0.85, \
                  "0.010, 0.02, 0.025, 0.45, 0.85, \
                  "0.011, 0.02, 0.025, 0.45, 0.85, \
                  "0.012, 0.02, 0.025, 0.45, 0.85, \
                  "0.013, 0.02, 0.025, 0.45, 0.85, \
                  "0.014, 0.02, 0.025, 0.45, 0.85, \
                  "0.015, 0.02, 0.025, 0.45, 0.85, \
                  "0.016, 0.02, 0.025, 0.45, 0.85, \
                  "0.017, 0.02, 0.025, 0.45, 0.85, \
                  "0.018, 0.02, 0.025, 0.45, 0.85, \
                  "0.019, 0.02, 0.025, 0.45, 0.85, \
                  "0.020, 0.03, 0.035, 0.45, 0.85, \
                  "0.021, 0.03, 0.035, 0.45, 0.85, \
                  "0.022, 0.03, 0.035, 0.45, 0.85, \
                  "0.023, 0.03, 0.035, 0.45, 0.85, \
                  "0.024, 0.03, 0.035, 0.45, 0.85" ;
```

## Encounter User Guide

### Cadence-Specific Liberty Extensions

---

```
    }
```

```
    ecssm_capacitance(rise) {
        values : "0.00, 0.01, 0.02, 0.03, 0.04, \
                  0.05, 0.06, 0.07, 0.08, 0.09, \
                  0.10, 0.11, 0.12, 0.13, 0.14, \
                  0.15, 0.16, 0.17, 0.18, 0.19, \
                  0.20, 0.21, 0.22, 0.23, 0.24" ;
    }
}
```

```
}
```

```
}
```

```
}
```

# Index

## Symbols

.enc files [60](#)

## Numerics

32-bit mode, availability of [54](#)  
64-bit mode, starting [54](#), [72](#)

## A

AC current violations, checking [1118](#)  
AC limit  
    verifying [1118](#)  
actual value, in Violation Browser [1120](#)  
ADDHIERINST [1259](#)  
Adding logical hierarchy without creating  
    additional hierarchy [368](#)  
ADDINST [1260](#)  
ADDNET [1264](#)  
Amoeba view  
    using during placement [559](#)  
antenna cells  
    ensuring connectivity after inserting with  
        NanoRoute router [753](#)  
    highlighting after inserting with  
        NanoRoute router [754](#)  
    See also process antenna  
antenna violations  
    as distinguished from process antenna  
        violations by verifyConnectivity  
        and verifyGeometry [1112](#)  
antennas  
    types of connectivity and geometry  
        violations reported [1112](#)  
antennas, trimming [662](#)  
area I/O  
    See also flip chip  
    placement, and I/O assignment  
        files [106](#)  
array vias  
    in yield report, definition of [1133](#)  
    in yield technology file [1148](#)

assign nets, removing from Verilog  
    netlist [130](#)  
Astro or Apollo, using with NanoRoute  
    router [757](#)  
ATTACHTERM [1266](#)  
attributes  
    NanoRoute router [705](#)  
        persistency of [705](#)  
    automatic mode of CTS [580](#)  
Automatic Power Planning (APP),  
    described [443](#)

## B

back-end designer, and approach to basic  
    floorplanning [355](#)  
base license, definition of [50](#)  
bindkeys, for wire editing [652](#)  
Blackbox Timing Analysis, described [221](#)  
blackboxes  
    deleting [262](#)  
    flows [261](#)  
    pins [269](#)  
    R0 transformation [264](#)  
    specifying, methods of [260](#)  
block pins  
    orientation of, changing, to resolve route  
        congestion [688](#)  
block rings  
    creating [438](#)  
    specifying coordinates with [439](#)  
block-level partitions, creating [332](#)  
blocks  
    changing module status from fence [331](#)  
bounding box, defining [367](#)  
buffer trees, inspecting routing topology of in  
    interactive ECO [1052](#)  
buffers  
    added during timing optimization,  
        viewing [1042](#)  
    attaching to I/O pins in interactive  
        ECO [1049](#)  
    hole punch [324](#)  
        inserting [325](#)  
    problems with, in timing analysis [934](#)

bump arrays [154](#)  
bumps  
    assigning [158](#)  
    I/O assignment files and [107](#)  
    routing [159](#)  
    viewing flight lines [159](#), [165](#), [186](#)  
bus names, and partition pin guides [274](#)  
bus routing  
    constraint for [782](#)  
    description of [769](#)

## C

CAA. See critical area analysis  
Cadence Help, launching [77](#)  
capacitance calculations of the mixed signal router [775](#)  
capacitance comparison, graph and report [888](#)  
capacitance table  
    basic [876](#)  
    extended [876](#)  
    generating [874](#)  
    reading [882](#)  
CAR, editing wires with [670](#)  
CDS\_AUTO\_64BIT environment variable, setting [55](#), [72](#)  
cell blockage visibility, controlling [669](#)  
cell delay specification, and CTS [587](#)  
cell padding  
    and clock buffers [550](#)  
    and highly localized clocks [550](#)  
    and placement space [550](#)  
cell probability, section in yield technology file [1148](#)  
cell timing model, and CTS [593](#)  
cells  
    placing [554](#)  
CHANGEINST [1268](#)  
channel estimation, reporting example [330](#)  
channelless designs, and feedthroughs [307](#)  
clock buffers, and cell padding [550](#)  
clock fanout buffers, limiting [614](#)  
clock grouping, and CTS [590](#)  
clock mesh [1022](#)  
clock nets  
    routing guide file in CTS for NanoRoute [605](#)

routing with NanoRoute router [728](#), [729](#), [736](#)  
setting attributes for with NanoRoute router [728](#)  
clock tree specification file  
    automatic gated CTS section [610](#)  
    clock grouping data section [610](#)  
    clock tree topology section [610](#)  
    contents [598](#)  
    creating [598](#) to [628](#)  
    example [600](#)  
    global directive section [607](#)  
    macro model data section [606](#)  
    and maximum skew value [590](#)  
    NanoRoute attribute section [605](#)  
    order of items in [598](#)  
clock tree synthesis. See CTS  
clones, orientation of, in partitions [266](#)  
command names, completion [56](#)  
Common Timing Engine. See CTE  
configuration files  
    loading [125](#)  
congestion  
    checking with NanoRoute router [716](#)  
    resolving with density screens [688](#)  
    resolving by reorienting block pins [688](#)  
congestion analysis table, generated by NanoRoute router [716](#)  
congestion distribution report  
    in Trial Route [681](#)  
    usage and routing overflow percentage values in [681](#)  
congestion distribution table  
    acceptable values in [678](#)  
    valued described [683](#)  
congestion map  
    used by NanoRoute router [695](#), [718](#)  
congestion markers, in Trial Route, described [678](#)  
connectivity  
    loops, detecting [1107](#)  
    types of violations reported [1107](#)  
    verifying [1107](#)  
console, Encounter, described [56](#)  
constraint file  
    creating for mixed signal router [797](#)  
    editing for mixed signal router [805](#)  
    format for mixed signal router [773](#)  
    loading for mixed signal router [802](#)  
    sample of mixed signal router [807](#)  
constraint files, with timing budgets for

partitions	<a href="#">835</a>	cell timing model requirement	<a href="#">593</a>
constraints		clock buffers, and cell padding	<a href="#">550</a>
bus	<a href="#">782</a>	clock designs with tight areas	<a href="#">593</a>
differential pair	<a href="#">780</a>	clock grouping and automatic mode	<a href="#">590</a>
keywords used by mixed signal router	<a href="#">776</a>	clock tree specification file	
matched nets	<a href="#">778</a>	automatic gated CTS section	<a href="#">610</a>
nets	<a href="#">776</a>	clock grouping data section	<a href="#">610</a>
shielding	<a href="#">784</a>	clock tree topology section	<a href="#">610</a>
control characters, using	<a href="#">57</a>	contents	<a href="#">598</a>
conventions, syntax and typographic	<a href="#">38</a>	creating	<a href="#">598</a> to <a href="#">628</a>
coordinates, using to specify block rings or core rings	<a href="#">439</a>	example	<a href="#">600</a>
core dimensions		global directive section	<a href="#">607</a>
margin, described	<a href="#">359</a>	macro model data section	<a href="#">606</a>
size, described	<a href="#">359</a>	NanoRoute attribute section	<a href="#">605</a>
core rings		order of items in	<a href="#">598</a>
creating	<a href="#">438</a>	files required for	<a href="#">578</a>
specifying coordinates with	<a href="#">439</a>	hierarchical CTS analysis	<a href="#">591</a>
core size, determining	<a href="#">364</a>	highly localized clock, and cell padding	<a href="#">550</a>
core-to-I/O distance, specifying in partitioning	<a href="#">268</a>	and INSERTION_DELAY statement	<a href="#">587</a>
costs		log file headings	<a href="#">628</a>
definition of yield costs	<a href="#">1132</a>	macro model delay specification	<a href="#">587</a>
formulas for yield costs	<a href="#">1156</a>	manual mode	<a href="#">579</a>
CPF		modes	<a href="#">579</a>
block-level	<a href="#">487</a>	and module placement utilization	<a href="#">593</a>
bottom-up hierarchical flow	<a href="#">494</a>	and pin balancing for macro models	<a href="#">593</a>
top-down hierarchical flow	<a href="#">479</a>	and pin instance delay specification	<a href="#">587</a>
top-level	<a href="#">490</a>	and port delay specification	<a href="#">587</a>
creating sub-categories	<a href="#">950</a>	preparatory procedure	<a href="#">578</a>
critical area analysis		and recommended placement utilization	<a href="#">551</a>
definition of	<a href="#">1125</a>	tracing clock trees	<a href="#">581</a>
cross-probing, in Violation Browser	<a href="#">1120</a>	cumulative defect data function described	<a href="#">1127</a> , <a href="#">1154</a>
crosstalk		custom macros	
correcting violations with NanoRoute router	<a href="#">730</a>	utilizing pre-defined feedthroughs	<a href="#">318</a>
effect on incremental delay	<a href="#">1062</a>	cut areas, and rectilinear partitions	<a href="#">267</a>
overview	<a href="#">1062</a>	cut layers, using NanoRoute router to repair PAE violations on	<a href="#">754</a>
preventing violations with NanoRoute router	<a href="#">730</a>	cuts	
CTE		creating for power domains	<a href="#">464</a>
using for timing-driven routing with NanoRoute router	<a href="#">725</a>	cutting shielding wires	<a href="#">662</a>
Ctrl-C			
using	<a href="#">73</a> , <a href="#">700</a> , <a href="#">736</a> , <a href="#">1015</a>	<b>D</b>	
CTS		data	
automatic mode	<a href="#">580</a>		
and gated clocks	<a href="#">581</a>		
and nets	<a href="#">580</a>		
and cell delay specification	<a href="#">587</a>		

- 
- checking before importing a design [120](#)
  - exporting [132](#)
  - importing [132](#)
  - database
    - saving, with NanoRoute router [695](#)
  - database units (DBU), adjusting the value of for NanoRoute router [697](#)
  - DBU. See database units
  - debugging
    - NanoRoute router [759](#)
  - Debugging Timing Results [938](#)
  - decoupling capacitance
    - adding [562](#)
    - defining cell candidates for [562](#)
    - removing [563](#)
  - DEF files
    - comparing with database [1055](#)
    - comparison file format [1055](#)
  - DEF syntax, unsupported [93](#)
  - defect data function
    - described [1127, 1153](#)
  - defect-limited yield
    - definition of [1124](#)
  - DELETEBUFFER** [1269](#)
  - DELETEINST** [1271](#)
  - DELETENET** [1273](#)
  - demand, defined in Trial Route [682](#)
  - density screens, using to resolve route congestion [688](#)
  - density, calculating [364](#)
  - design
    - checking [116](#)
    - checking data before importing [120](#)
    - display area, described [357](#)
    - importing [121](#)
    - saving files [131](#)
  - DETACHTERM** [1275](#)
  - detailed routing
    - description of [695](#)
    - running in NanoRoute router [713](#)
    - violations, troubleshooting [713, 759](#)
  - die size
    - described [359](#)
  - dielectric layer
    - syntax [1215](#)
  - differential pair routing
    - constraint for [780](#)
    - description of [768](#)
    - shielding [772, 789](#)
  - differential signal routing [187, 451](#)
  - diodes
    - See also antenna cells, process antennas
    - diodes, highlighting after inserting with NanoRoute router [754](#)
  - disambiguating command names [56](#)
  - display area, for floorplanning [357](#)
  - distributed processing
    - definition of [84](#)
    - features that support [84](#)
    - running [86](#)
  - distributing jobs [352](#)
  - DLY. See defect-limited yield
  - documentation, accessing [77](#)
  - documents, related, list of [39](#)
  - DRC violations
    - recognized by NanoRoute router [736](#)
  - dynamic licenses
    - checking out [51](#)
    - definition of [51](#)

## E

### ECO

- naming conventions, table of [1055](#)
- netlist, updating [1054](#)
- placement [1054](#)
- general principles [1054](#)

### ECO File

- ADDHIERINST** [1259](#)
- ADDINST** [1260](#)
- ADDNET** [1264](#)
- ATTACHTERM** [1266](#)
- CHANGEINST** [1268](#)
- DELETEBUFFER** [1269](#)
- DELETEINST** [1271](#)
- DELETENET** [1273](#)
- DETACHTERM** [1275](#)
- Example [1279](#)
- INSERTBUFFER** [1276](#)

### ECO flows

- alternative pre-mask [1234](#)
- description [1232](#)
- GACORE cells [1252](#)
- post-mask from new netlist [1246](#)
- post-mask gate-array [1252](#)
- pre-mask [1242](#)
- pre-mask from new DEF [1238](#)

### Edit Route form

- populating [653](#)

### editing wires. See wire editing, wires

effective resistance, description of [775](#)  
effective utilization (EU value) [363](#)  
electrostatic discharge cells [154](#)  
encounter command, syntax of [68](#)  
Encounter console, described [56](#)  
Encounter Digital Implementation System, description of products in [44](#)  
Encounter software  
    interrupting [73](#)  
    starting [62](#)  
    stopping [73](#)  
encrypting libraries [114](#)  
end-cap cells  
    placing [542](#)  
    removing [543](#)  
environment, run-time, setting [54](#)  
escape sequences, using [58](#)  
EU value [363](#)  
exporting  
    GDSII Stream or OASIS files [137](#)  
    TDF files [135](#)  
extraction  
    and capacitance comparison graph and report [888](#)  
capacitance table  
    basic [876](#)  
    extended [876](#)  
    generating [874](#)  
    reading [882](#)  
    overview [866](#)  
    preparatory procedure [867](#)

**F**

fabrication process information [1205](#)  
fanouts, clock buffer, limiting [614](#)  
fat via, definition of [744](#)  
feedback buffer [307](#)  
feedthrough insertion, and partitioning [307](#)  
feedthroughs [305](#) to [326](#)  
    buffers vs. routing [305](#)  
    and channelless designs [307](#)  
    channel-type [325](#)  
    inserting [305](#), [306](#)  
    pre-defined [318](#)  
    replicating across ECO netlists [314](#)  
    routing [324](#)  
fences  
    changing module status to block [331](#)  
constraints [360](#)

module constraint types [360](#)  
restrictions [360](#)  
file selection, general description [126](#)  
files  
    required for CTS [578](#)  
    routing blockage, generating [258](#)  
    TDF, importing and exporting [135](#)  
    timing constraints [835](#)  
    updating during a session [152](#)  
    updating incrementally [152](#)  
filler cells  
    adding to MSV designs [560](#)  
    implant layers [560](#)  
    placing [560](#)  
    placing gate-array style cells in ECO [561](#)  
    removing [561](#)  
    replacing during process antenna repair with NanoRoute router [754](#)  
Flight [358](#)  
flight lines  
    colors used by wire editor [657](#)  
    definition [358](#)  
    for bumps [159](#), [165](#), [186](#)  
flip chip  
    defined [154](#)  
flows  
    bump flow [161](#)  
methodology [154](#)  
routing bumps to I/O driver cells [164](#)  
testing package routing feasibility [162](#)  
floorplan  
    data, types [133](#), [134](#)  
    display, described [357](#)  
    loading [426](#)  
    saving [426](#)  
floorplan files, containing routing blockage data [258](#)  
floorplan objects  
    and partition feedthroughs [324](#)  
floorplanning  
    overview [354](#)  
    relative [423](#) to [426](#)  
    row spacing [365](#)  
    sequence [355](#)  
    specifying spare cells during [544](#)  
    steps [355](#)  
footprint  
    using in timing optimization [1042](#)  
footprintless flow, using for timing optimization [1040](#)

form help, using [79](#)  
formula  
  used for yield costs [1156](#)  
  used to calculate the probability of failure [1153](#)  
  used to derive defect data and cumulative defect data functions from each other [1155](#)  
front-end designer, and approach to basic floorplanning [355](#)

## G

GACORE cells  
  ECO flows [1252](#)  
gated clocks, and CTS automatic mode [581](#)  
gcells  
  definition of [694](#)  
gcells, obstructed and overflowed, summarized in Trial Route congestion distribution report [682](#)  
GDSII  
  and FOREIGN statements [138](#)  
  exporting files to [137](#)  
  map file format [143](#)  
  merging files in [139](#)  
  renaming LEF vias in [138](#)  
  specifying floating fill in [148](#)  
  specifying layers for nets in [149](#)  
  specifying voltage levels in [149](#)  
  support for GZIP format [137](#)  
  suppressing text labels in [145](#)  
generic parameters, using [72](#)  
geometry  
  spacing violation checks during verification of [1111](#)  
  verifying [1110](#)  
global net connections [439](#)  
global routing  
  description of [694](#)  
  running in NanoRoute router [713](#)  
Global Timing Debug [938, 950](#)  
guide constraints [360](#)  
guides, module constraint types [360](#)

## H

hard blocks

  placing [539](#)  
hard blocks, changing status of partition from [331](#)  
Help menu, using [78](#)  
help, command-line, using [79](#)  
help, GUI, using [79](#)  
help, launching Cadence Help [77](#)  
hierarchical clock tree analysis, and CTS [591](#) to [592](#)  
hierarchical designs  
  timing budgeting, methods for [832](#)  
hierarchy icon, picture of [798](#)  
hole punch buffers, inserting [325](#)  
horizontal and vertical tracks, summarized in Trial Route congestion distribution report [682](#)

## I

I/O assignment files  
  multiple I/O rows in [106](#)  
ICT file  
  case sensitivity [1206](#)  
  commands [1206](#)  
  contents [1205](#)  
  example [1219](#)  
  format [1206](#)  
ILM [208](#)  
  Creating [209](#)  
  Interactive Uses [217](#)  
  SI Support [217](#)  
  Specifying ILM Directories [213](#)  
implant layers, for filler cells [560](#)  
implementation  
  block-level [253](#)  
  top-level [255](#)  
importing  
  TDF files [135](#)  
  tile cell design data [136](#)  
incremental delay  
  crosstalk effect on [1062](#)  
initialization files [60](#)  
insert [305](#)  
INSERTBUFFER [1276](#)  
INSERTION\_DELAY statement, and macro models in CTS [587](#)  
inserts [306](#)  
instance groups  
  adding an instance to [366](#)  
  deleting [366](#)

saving to netlist [367](#)  
specifying [363](#)  
interactive ECO  
naming conventions, table of [1055](#)  
overview [1046](#)  
interconnect capacitance modeling  
creating ICT file [1205](#) to [1229](#)  
interrupting  
batch script [74](#)  
Encounter software [73](#)  
NanoRoute [700](#)  
optDesign [1015](#)  
I/O assignment files  
and area I/O placement [106](#)  
bumps and [107](#)  
creating [97](#)  
described [96](#)  
examples of I/O pads in [108](#)  
module pins in [110](#)  
rule-based, creating [107](#)  
I/O pads  
core dimension size [359](#)  
and I/O assignment files [108](#)  
I/O pins, adding buffers to, in interactive  
ECO [1049](#)  
island, and routing feedthroughs [325](#)

## J

JTAG cells  
placing [564](#)

## K

keyboard shortcuts, for wire editing [652](#)  
keywords  
mixed signal router constraint [776](#)

## L

layers  
changing during wire editing [665](#)  
routing, checking metal density of [1109](#)  
LEF files  
checking for availability before  
importing [120](#)  
checking for compatibility with  
NanoRoute router [696](#)

LEF statements  
for metal density [1109](#)  
for process antenna calculations [1114](#)  
for verifying geometry [1110](#)  
LEF syntax, unsupported [93](#)  
libraries  
checking for availability before  
importing [120](#)  
timing [114](#)  
licenses  
base license, definition of [50](#)  
changing check-out order of multi-  
CPU [88](#)  
checking out [51](#)  
checking out dynamically [51](#)  
corresponding to product product  
names [48](#)  
dynamic license, definition of [51](#)  
finding more information on multi-  
CPU [89](#)  
limiting search order of multi-CPU [88](#)  
multi-CPU license, definition of [51](#)  
releasing multi-CPU [89](#)  
strings (keys), table of Encounter [50](#)  
log file, congestion analysis table in [716](#)  
log files, viewing [75](#)  
Logical hierarchy manipulation [369](#)

## M

macro model delay, and CTS [587](#)  
macro models, pin banancing for [593](#)  
MACRO, creating cells in GDSII Stream or  
OASIS format from [138](#)  
macros  
placing [539](#)  
man command, using [80](#)  
manual mode of CTS [579](#)  
map file format, GDSII or OASIS,  
described [143](#)  
master partitions, defined [265](#)  
Masterplan Automatic Floorplanner  
adjusting macro placement  
adjusting a specific macro  
pack [1176](#)  
adjusting within a specified  
area [1178](#)  
interactive commands, using [1176](#)  
constraint file  
creating a seed section [1163](#)

seed section, definition of [1163](#)  
 data preparation [1161](#)  
 examining modules for seed  
     selecion [1164](#)  
 floorplan analysis  
     checking macro placement [1175](#)  
     checking the seed report [1175](#)  
     visual check [1174](#)  
 generating an initial floorplan [1168](#)  
 guidelines for hierarchical  
     floorplanning [1171](#)  
 ideal designs for use with [1158](#)  
 input files for [1161](#)  
 macro pack, definition of [1176](#)  
 marking refinement steps [1180](#)  
 overview [1158](#)  
 replacing cells with I/O  
     connections [1161](#)  
 restoring design to previous state during  
     refinement [1181](#)  
 seed selection  
     automatic selection [1162](#)  
     definition [1161](#)  
     how seeds are used [1161](#)  
     optimal seed number [1162](#)  
     user defined selection [1162](#)  
 setting global parameters for [1168](#)  
 task flow [1159](#)  
 use model [1168](#)  
 matched net routing  
     constraint for [778](#)  
     description of [764](#)  
 maximum fanouts, limiting for clock  
     buffers [614](#)  
 maximum floating area, verifying [1117](#)  
 maximum skew value, and clock tree  
     specification file [590](#)  
 maximum wire width, repairing violations  
     of [664](#)  
 merging GDSII or OASIS files [139](#)  
 metal density  
     LEF statements for [1109](#)  
     verifying [1109](#)  
 metal fill  
     adding over macros [824](#)  
     connected, definition of [815](#)  
     cross-vias not used in [815](#)  
     definition of [812](#)  
     floating [823](#)  
     LEF statements for [821](#)  
     macro density considerations of [825](#)  
         power strapping recommendations  
             for [825](#)  
         preparatory procedure [813](#)  
         recommendations for [821](#)  
         setting parameters for [820](#)  
         staggering [814](#)  
         tied-off [823](#)  
         tied-off, definition of [815](#)  
         timing aware [819](#)  
         trimming [827](#)  
             problems caused by vias [815](#)  
             visibility of [813](#)  
         Milkyway technology file [757](#)  
         minimum cut violations, fixing [441](#)  
         minimum spacing violations, fixing [441](#)  
         mixed signal router  
             capacitance calculations of [775](#)  
         constraint file  
             creating [797](#)  
             editing [805](#)  
             format of [773](#)  
             keywords used in [776](#)  
             loading [802](#)  
             sample of [807](#)  
         constraints  
             bus [782](#)  
             diffpair [780](#)  
             matched nets [778](#)  
             nets [776](#)  
             shielding [784](#)  
             resistance calculations of [775](#)  
             specialized routing performed by [763](#)  
         MMMC  
             specifying for timing optimization [1034](#)  
         mode  
             standalone mode, NanoRoute  
                 router [704](#)  
         modes  
             auto query [653](#)  
             CTS, automatic [580](#)  
             CTS, manual [579](#)  
             timing-aware metal fill [819](#)  
         module constraint types  
             described [360](#)  
             fence [360](#)  
             guide [360](#)  
             and physical design size [361](#)  
             region [361](#)  
             soft guide [361](#)  
         module fences, restrictions [360](#)  
         module guides [360](#)

module pins, and I/O assignment files [110](#)  
module placement utilization, and  
    CTS [593](#)  
module status, changing from fence to  
    block [331](#)  
-modulePlan  
    limitations of [539, 554](#)  
    -modulePadding and [552](#)  
    multi-threading and [555](#)  
modules  
    placement utilization and CTS [593](#)  
MSV (multiple supply voltage)  
    power analysis [477](#)  
MSV designs  
    adding end-cap cells to [543](#)  
    adding filler cells to [560](#)  
    adding well-tap cells to [542](#)  
multi-CPU licenses, definition of [51](#)  
multi-CPU processing  
    features that support [84](#)  
multi-cut vias, using [744](#)  
multiple I/O rows, specifying in I/O  
    assignment file [106](#)  
multiple supply voltage  
    flat flow [459](#)  
multiple-CPU processing  
    features that support using [84](#)  
    placement [555](#)  
multi-threading  
    definition of [84](#)  
    features that support [84](#)  
    placement [555](#)  
    running [87](#)

## N

naming conventions  
    for ECO [1055](#)  
    for timing optimization [1042](#)  
NanoRoute  
    product number of [48](#)  
NanoRoute router  
    [753](#)  
        adjusting the database units (DBU) value  
            for [697](#)  
    Astro or Apollo, using with the  
        router [757](#)  
    attributes  
        characteristics of [705](#)  
        compared with options [705](#)

listed. See `setAttribute` in the  
*Encounter Text Command Reference*.

clock nets  
    routing [729](#)  
    setting attributes for [728](#)  
commands to run within Encounter  
    environment [703](#)  
congestion analysis table, using [716](#)  
congestion map [695, 718](#)  
congestion, checking [716](#)  
connectivity, power and ground  
    pins [753](#)  
crosstalk  
    correcting violations [730](#)  
    preventing violations [730](#)  
CTE, using [725](#)  
database, saving [695](#)  
debugging [759](#)  
detailed routing [713](#)  
evaluating violations in [736](#)  
generating tracks for [698](#)  
global routing [713](#)  
hard spacing, forcing [751](#)  
LEF files, checking for problems in [696](#)  
nets, shielding [748](#)  
nondefault rule routing [751](#)  
nondefault spacing in [751](#)  
open net messages, meaning of [738](#)  
options  
    compared with attributes [705](#)  
    crosstalk prevention [732](#)  
    listed. See `setNanoRouteMode` in  
        the *Encounter Text Command Reference*.  
process antenna repair [755](#)  
    via optimization [746](#)  
output files from [702](#)  
overlapping cells, checking for [697](#)  
overview [694](#)  
pin access, improving [697](#)  
pins, checking for under power  
    routes [698](#)  
postroute optimization in [715](#)  
preparatory procedure [696](#)  
prerouted nets, skipping during  
    routing [735, 748](#)  
routing  
    accelerating [708](#)  
    clock nets [728, 736](#)  
    detailed [695](#)

distributed [708](#)  
ECO [734](#)  
global [694](#)  
shielded [747](#)  
signal-integrity driven [732](#)  
standalone mode [704](#)  
statistics report [702](#)  
strategy [712](#)  
wide wires [750](#)  
shielded routing [747, 748](#)  
signal integrity, preventing problems  
with [730](#)  
soft spacing [751](#)  
soft spacing routing [751](#)  
table of violations [738](#)  
tapering wires in [750](#)  
timing graph, using with standalone  
NanoRoute router [726](#)  
timing-driven routing [742](#)  
troubleshooting design problems [759](#)  
vias  
    checking for rotated [698](#)  
    defining TOPOFSTACKONLY [697](#)  
    fat, definition of [744](#)  
    mapping Astro scheme format  
        of [757](#)  
    minimizing the number of [746](#)  
    optimizing [744](#)  
    optimizing in selected nets [745](#)  
    using multi-cut [744](#)  
violations  
    on upper layers [740](#)  
    horizontal [736](#)  
    marker placement for [736](#)  
    process antenna, repairing [753](#)  
    table for evaluating [738](#)  
    types of [736](#)  
    vertical [736](#)  
    via [736](#)  
native extraction  
    default mode [868](#)  
    defined [868](#)  
    detailed mode [868](#)  
    flow, illustrated [869](#)  
See also extraction, sign-off extraction  
net attributes  
    characteristics of in NanoRoute  
        router [705](#)  
net group names, and partition pin  
    guides [274](#)  
net names, and partition pin guides [274](#)

netlists  
    ECO, replicating feedthrough  
        insertions [314](#)  
    ECO, updating [1054](#)  
    and feedthroughs [305](#)  
    preparing [120](#)  
    saving instance groups to [367](#)  
    Verilog, concatenating for a partitioned  
        design [348](#)  
    Verilog, removing Assign nets from [130](#)  
nets  
    and CTS automatic mode [581](#)  
    deleting violated nets [743](#)  
    names, and partition pin guides [274](#)  
    shielding, with NanoRoute router [748](#)  
nets routing constraint [776](#)  
nondefault rule routing, NanoRoute  
    router [751](#)  
nonrectangular partitions [267](#)

## O

OASIS  
    and FOREIGN statements [138](#)  
    exporting files to [137](#)  
    map file format [143](#)  
    merging files in [139](#)  
    renaming LEF vias in [138](#)  
    specifying floating fill in [148](#)  
    specifying layers for nets in [149](#)  
    specifying voltage levels in [149](#)  
    support for GZIP format [137](#)  
    suppressing text labels in [145](#)  
obstructed gcell information, summarized in  
    Trial Route congestion distribution  
    report [682](#)  
OCV [922](#)  
on-chip variation [922, 1033](#)  
open net messages, generated by  
    NanoRoute router [738](#)  
open point defect, section in yield technology  
    file [1140](#)  
OpenAccess  
    restoring a design [129](#)  
    saving a design [129](#)  
OpenAccess, configuring [55](#)  
optimizing timing. See timing optimization  
optimizing vias [744](#)  
options  
    NanoRoute router

characteristics and persistency of [705](#)  
compared with attributes [705](#)  
crosstalk prevention [732](#)  
process antenna [755](#)  
via optimization [746](#)

options, descriptions of product options [45](#)  
orientation, of blocks, changing [688](#)  
orientation key [423](#)  
Ostrich  
    parasitics correlation utility [884](#)  
OverCon #Gcell, definition of [716](#)  
overflow percentage values, in Trial Route congestion distribution report [681](#)  
overflowed gcells, summarized in Trial Route congestion distribution report [682](#)  
overlapped objects  
    navigating in Violation Browser [1120](#)  
overlapping cells, checking for with NanoRoute router [697](#)  
overlapping objects  
    selecting [658](#)

**P**

package routing feasibility testing, using APD [162](#)  
padding  
    placing [544](#)  
parallel job runs [352](#)  
partition feedthroughs, defining [324](#)  
partition pin guides  
    and bus names [274](#)  
    and net group names [274](#)  
partition pins  
    guide objects [258](#)  
partition pins  
    guide names, changing [274](#)  
    guides, adding [274](#)  
    ranges [302](#)  
    snapping [302](#)  
partitioning  
    block-level [332](#)  
    channel estimates, reporting example [330](#)  
    concatenating netlists of [348](#)  
    core-to-I/O distance, specifying [268](#)  
    feedthrough insertion [307](#)  
    loading [349](#)

overview [248](#)  
restoring [349](#)  
running [331](#)  
saving [349](#)

partitions  
    adding cut areas to rectilinear [268](#)  
    changing status from hard block [331](#)  
    master, defined [265](#)  
    multiple instantiations [265](#)  
    nonrectangular [267](#)  
    pin assignments for [271](#)  
    pin blockage [279](#)  
    pins [269](#)  
    rectilinear [300](#)  
        creating [267](#)  
    repeated [265](#)  
    specifying [257](#)  
        and timing budgets for [835](#)

passivation layer  
    example [1217](#)

path groups [1021](#)

peripheral I/O, and flip chip [154](#)

physical design size, and module constraint types [361](#)

physical layout, verifying [1110](#)

physical libraries  
    creating [92](#)

pin access, improving for NanoRoute router [697](#)

pin alignment, illustrated [301](#)

pin assignments  
    rectilinear [300](#)  
    and rectilinear edges [267](#)

pin balancing, and macro models [593](#)

pin blockage [279](#)

pin guides  
    location [274](#)

pin instance delay specification, and CTS [587](#)

pins  
    blackbox  
        assigning [269](#)  
    partition  
        assigning [269](#)  
        preventing creation of [258](#)  
        when created in absence of partition pin guide objects [258](#)

underneath power routes, checking for with NanoRoute router [698](#)

visibility of [657](#)

placement

blockages  
 assigning attributes to [539](#)  
 creating [539](#)  
 hard, description of [539](#)  
 partial  
     behavior of [540](#)  
     description of [540](#)  
 soft  
     description of [540](#)  
 types of [539](#)  
 checking [558](#)  
 clock buffers, reserving space for [551](#)  
 files required for [538](#)  
 multi-threading and [555](#)  
 overview of [538](#)  
 padding  
     adding [550](#)  
 preparing for [538](#)  
 preroute treatment during [540](#)  
 saving data from [564](#)  
 tuning [554](#)

placing  
 end-cap cells [542](#)  
 filler cells [560](#)  
 gate-array style filler cells [561](#)  
 hard blocks [539](#)  
 hierarchical spare cells [547](#)  
 JTAG cells [564](#)  
 padding [544](#)  
 scan cells [565](#)  
 spare cells [543](#)  
 spare modules [544](#)  
 standard cells [554](#)  
 tie-off cells [563](#)  
 well-tap cells [541](#)

planar dielectric  
 syntax [1215](#)

Point-To-Point Routing [182](#)

port delay specification, and CTS [587](#)

post-CTS optimization [1020](#)

postroute optimization [1023](#)

postroute optimization, in NanoRoute  
 router [715](#)

power analysis  
 for MSV designs [477](#)

power domain as a partition [260](#)

power domains  
 adding power stripes to [441](#)  
 adding stripes to [467](#)  
 creating cuts [464](#)  
 creating rings around [467](#)

timing libraries [477](#)

power plan template  
 block rings [446](#)  
 creating [445](#)  
 design [446](#)  
 IP blocks [445](#)  
 power analysis data [448](#)  
 stripes [446](#)

power planning, overview [436](#)

power routing, overview [437](#)

power structures, creating, preparatory  
 procedure [436](#)

pre-CTS optimization [1016](#)

pre-defined feedthroughs [318](#)

preferences  
 initialization files for [59](#)  
 session, setting [59](#)

prerouted nets, skipping routing on [735](#),  
[748](#)

PrimeTime  
 format, and timing constraint file  
 creation [835](#)

process antenna violations  
 verifying [1114](#)

process antennas  
 ensuring antenna cell connectivity [753](#)  
 methods NanoRoute router uses to  
 repair violations from [753](#)  
 NanoRoute router options for repairing  
 violations from [755](#)  
 repairing by changing routing  
 layers [754](#)  
 repairing by diode insertion [754](#)  
 violations, repairing on nets with multiple  
 pins [753](#)

product numbers, listed [44](#)

product options, description of [45](#)

## R

R0 transformation [264](#)

RC extraction. See extraction, native  
 extraction, sign-off extraction

RC scaling factors  
 generating [883](#)  
 methods for specifying [890](#)

README file, where to find [54](#)

reclaiming area [1022](#)

rectilinear boundaries, identification of [300](#)

rectilinear pin assignments [300](#)

rectilinear power domains [464](#)  
 rectilinear shape, defining [367](#)  
 rectilinear shapes, and pin assignments [300](#)  
 redundant vias, using [744](#)  
 regions, module constraint types [361](#)  
 related documents, list of [39](#)  
 relative floorplanning [423](#)  
     orientation  
         Encounter [423](#)  
     overview [423](#)  
     saving commands for [426](#)  
 relative floorplans  
     loading [426](#)  
     reshaping [424](#), [425](#), ?? to [425](#)  
     restoring [426](#)  
 repeated partitions. See partitions  
 reports  
     capacitance comparison [889](#)  
     NanoRoute router [702](#)  
     routing statistics [702](#)  
     used by NanoRoute router [702](#)  
     verification [1104](#)  
     wire length [702](#)  
 reportYield  
     yield effects considered by [1124](#)  
 resistance  
     calculations of mixed signal router [775](#)  
 Resizing Rectilinear Blocks in the Floorplan [392](#)  
 restoring  
     OpenAccess designs [129](#)  
 rings  
     around power domains [467](#)  
 rotated vias, checking for with NanoRoute router [698](#)  
 route congestion, resolving  
     by inserting density screens [688](#)  
     by reorienting block pins [688](#)  
 routers  
     choosing NanoRoute or WRoute [694](#)  
 routes  
     removing while editing [654](#)  
     reshaping [668](#)  
 routing  
     See also NanoRoute router  
     choosing NanoRoute router or WRoute router [694](#)  
     clock nets with NanoRoute router [728](#), [736](#)  
     differential [187](#), [451](#)

feedthroughs [324](#)  
 routing blockage file, generating [258](#)  
 routing bumps to I/O driver cells [164](#)  
 routing feedthroughs, inserting [325](#)  
 routing guide file, from CTS [605](#)  
 routing layers, checking metal density of [1109](#)  
 routing resources, and channelless designs [307](#)  
 routing strategy, NanoRoute router [712](#)  
 row configurations, types supported [365](#)  
 row spacing, standard [365](#)  
 rule-based I/O assignment files, creating [107](#)  
 run parallel [352](#)  
 run-time environment, setting [54](#)

## S

Sbox. See switch box  
 scaling factors, RC  
     specifying, methods for [890](#)  
 scan cells  
     placing [565](#)  
 scan chains  
     reordering [566](#)  
     flows [567](#), [568](#)  
     native approach [566](#), [567](#) to [571](#)  
     scanDEF approach [566](#), [572](#) to [576](#)  
 scan files  
     loading [576](#)  
     saving [576](#)  
 scan functionality, principles of [565](#)  
 scan nets, reordering [566](#)  
 scheme format, mapping LEF vias and layers to [757](#)  
 search and repair  
     phases of, in NanoRoute router [714](#)  
 seed, definition of [1161](#)  
 selecting wire shapes [658](#)  
 session preferences, setting [59](#)  
 set\_dont\_touch constraint, implications of for timing analysis [905](#)  
 setLicenseCheck command, using [51](#), [52](#)  
 shielded net routing  
     coaxial [786](#)  
     coaxial, description of [771](#)  
     constraint for [784](#)  
     description of [769](#)  
     differential pairs in [772](#)

shared shields in [771](#)  
shielded routing  
  with NanoRoute router [747, 748](#)  
shielding wires  
  cutting [662](#)  
short point defect, section in yield technology  
  file [1144](#)  
SI. See signal integrity  
signal integrity  
  crosstalk effect on incremental  
    delay [1062](#)  
  NanoRoute router, using to prevent  
    problems [730](#)  
  preventing problems using NanoRoute  
    router [730](#)  
  setting NanoRoute router options  
    for [732](#)  
signal routers  
  choosing NanoRoute or WRoute [694](#)  
*See also* extraction, native extraction  
sign-off extraction  
  described [891](#)  
  requirements [892](#)  
slot. See routing feedthrough  
SoC Encounter RTL-to-GDSII System,  
  description of products in [48](#)  
soft guides, module constraint types [361](#)  
spare cells  
  distribution of [545](#)  
  placing [543](#)  
  placing when hierarchy is an issue [547](#)  
  specifying during floorplanning  
    session [544](#)  
spare modules  
  placing [544](#)  
special route  
  loading [438](#)  
  saving [438](#)  
specification file, for CTS. See clock tree  
  specification file  
SPEF files  
  comparing files from runQX and  
    extractDetailRC commands [887](#)  
SRoute  
  overview [437](#)  
Stamp models  
  defined [115](#)  
  preparing [115](#)  
standalone NanoRoute router  
  using a timing graph with [726](#)  
standard cell density, calculating [364](#)  
standard cells  
  placing [158, 554](#)  
starting  
  Encounter software [62](#)  
stopping  
  batch script [74](#)  
  Encounter software [73, 74](#)  
  NanoRoute [700](#)  
  optDesign [1015](#)  
stripes  
  adding to power domains [467](#)  
stripes, adding to power domains [441](#)  
Superthreading  
  definition of [84](#)  
  features that support [84](#)  
  running [88](#)  
superthreading  
  definition of [708](#)  
  features of [708](#)  
supply, defined, in Trial Route [682](#)  
switch box, definition of [695](#)  
syntax conventions [38](#)  
syntax, of commands, using help and man  
  commands to see [79](#)

## T

tapering wires, NanoRoute router [750](#)  
target utilization [361](#)  
target value, in Violation Browser [1120](#)  
TDF files, importing and exporting [135](#)  
technology file [92](#)  
template, for power plans [445](#)  
threshold, as used by mixed signal  
  router [781](#)  
tie-off cells  
  adding [563](#)  
  removing [563](#)  
tile cell  
  importing design data [136](#)  
Timing  
  Debug Results [938](#)  
timing analysis  
  overview [900](#)  
  preparatory procedure [902](#)  
  and buffer problems [934](#)  
timing budget  
  analyzing [844](#)  
  calculating [849](#)  
  constraints [860](#)

customizing [852](#)  
 deriving [835](#)  
 deriving preliminary budget [837](#)  
 justifying [855](#)  
 verifying [854](#)  
 warning report [863](#)  
 timing budgeting  
     and hierarchical designs [832](#)  
     methods for, in hierarchical  
         designs [832](#)  
 timing budgets  
     constraint files for partitions,  
         generating [835](#)  
     for partitions [835](#)  
     values for, analyzing [854](#)  
 timing closure design flow, preparing data  
     for [117](#)  
 timing constraints  
     file, and set\_dont\_touch constraint [905](#)  
     file [835](#)  
     importing [115](#)  
     preparing [115](#)  
**Timing Debug**  
     Global [938](#)  
 timing graph  
     commands to generate for NanoRoute  
         router [726](#)  
     generating with CTE [726](#)  
     using with standalone NanoRoute  
         router [726](#)  
 timing libraries  
     for power domains [477](#)  
     preparing [114](#)  
 timing models, for cells in CTS [593](#)  
 timing optimization  
     added buffers, viewing [1042](#)  
     after routing [1025](#)  
     clock domains [1017, 1018, 1021](#)  
     correcting signal integrity violations  
         during [1027](#)  
     hold violation repair [1025](#)  
     incremental [1018, 1021, 1022](#)  
     incremental, path groups [1018](#)  
     incremental, useful skew [1018](#)  
     inputting a SPEF file [1028](#)  
     location of timing reports generated  
         by [1014](#)  
     low-effort mode [1017](#)  
     naming conventions [1042](#)  
     optimizations performed during [1012](#)  
     path groups [1017, 1021](#)  
     post-CTS [1020](#)  
     postroute [1023](#)  
     pre-CTS [1016](#)  
     reclaiming area [1018](#)  
     requirements for [1012](#)  
     results of [1013](#)  
     and set\_case\_analysis timing  
         constraint [1040](#)  
     setup and hold violation repair [1026](#)  
     specifying MMMC environment  
         for [1034](#)  
     useful skew [1017, 1021](#)  
     using footprintless flow in [1040](#)  
     using on-chip variation (OCV) analysis  
         for [1033](#)  
     viewing added buffers [1042](#)  
 timing reports, commands for  
     generating [933](#)  
 timing-driven routing, NanoRoute  
     router [742](#)  
 tolerance, as used by mixed signal  
     router [779](#)  
 total density, calculating [364](#)  
 tracks  
     generating for NanoRoute router [698](#)  
 tracks, horizontal and vertical, summarized  
     in Trial Route congestion distribution  
         report [682](#)  
**Trial Route**  
     analyzing values in congestion  
         distribution table [678](#)  
     congestion distribution report [681](#)  
         obstructed and overflowed gcells  
             in [682](#)  
     congestion distribution table  
         acceptable values in [678](#)  
         values described [683](#)  
     congestion markers, overflow values  
         and [679](#)  
     data for, deleted during partitioning [331](#)  
     demand, defined [682](#)  
     design analysis requirements [678](#)  
     loading [678](#)  
     overview [674](#)  
     preparatory procedure [674](#)  
     route congestion, resolving [687](#)  
     saving [678](#)  
     supply, defined [682](#)  
     two layer designs [678](#)  
 troubleshooting  
     NanoRoute router violations (table) [738](#)

problems with NanoRoute router [759](#)  
TU (target utilization) value [361](#)  
typographic conventions [38](#)

## U

upper layer violations, NanoRoute  
    router [740](#)  
useful skew [1022](#)  
    controlling optimization [1031](#)  
    described [1030](#)  
    post-CTS [1021](#), [1022](#), [1031](#)  
    pre-CTS [1017](#), [1018](#), [1030](#)  
user-defined ring [438](#)

## V

velocity command, syntax of [63](#)  
verifying  
    AC limit [1118](#)  
    connectivity [1107](#)  
    geometry [1110](#)  
    maximum floating area [1117](#)  
    metal density [1109](#)  
    physical layout [1110](#)  
    process antenna violations [1114](#)  
Verilog  
    checking for availability before  
        importing [120](#)  
Verilog netlists  
    concatenating for a partitioned  
        design [348](#)  
    creating for entire design [348](#)  
    creating from a DEF file [121](#)  
    removing Assign nets from [130](#)  
    unique for use in CTS, scan, and IPO  
        features [120](#)  
vertical and horizontal tracks, summarized in  
    Trial Route congestion distribution  
        report [682](#)  
via fill, adding [826](#)  
via probability, section in yield technology  
    file [1145](#)  
via ratio, increasing multiple-cut to single-cut  
    with NanoRoute router [744](#)  
via rules, as supported by  
    verifyGeometry [1113](#)  
via syntax [1218](#)  
vias

adding [666](#)  
changing [667](#)  
concurrent insertion of multiple-cut and  
    routing [744](#)  
deleting [653](#)  
example [1218](#)  
fat, definition of [744](#)  
giving unique names after streamOut or  
    oasisOut [138](#)  
inserting multiple-cut with NanoRoute  
    router [744](#)  
mapping Astro scheme format by  
    NanoRoute router [757](#)  
minimizing the number of [746](#)  
moving selected [657](#)  
optimizing in selected nets with  
    NanoRoute router [745](#)  
optimizing with NanoRoute router [744](#)  
postroute optimization of [744](#)  
rotated, checking for with NanoRoute  
    router [698](#)  
TOPOFSTACKONLY, defining for  
    NanoRoute [697](#)  
using multi-cut (redundant) [744](#)  
ViewICT  
    errors and warnings [1206](#)  
viewing violations [1119](#)  
viewing, Ecounter log file [75](#)  
viewlog command [76](#)  
Violation Browser  
    actual value [1120](#)  
    features of [1120](#)  
    navigating overlapped objects in [1120](#)  
    target value [1120](#)  
violation markers  
    Assura or Calibre [1119](#)  
    clearing from design [1122](#)  
    highlighting [1119](#)  
    imported from NanoRoute router to  
        Encounter [736](#)  
    incremental updates to [1119](#)  
    NanoRoute router placement of [736](#)  
    overwriting [1119](#)  
    and verifying connectivity [1107](#)  
    viewing [1119](#)  
violation report  
    database impact of [1107](#)  
violations  
    clearing from design [1122](#)  
    deleting violated nets [743](#)  
    evaluating in NanoRoute router [736](#)

fixing minimum cut [441](#)  
fixing minimum spacing [441](#)  
in NanoRoute router, evaluation  
table [738](#)  
repair strategies for [743](#)  
types recognized by NanoRoute  
router [736](#)  
viewing [1119](#)

Virtuoso Digital Implementation, description  
of [45, 48](#)

voltage level shifters  
and MSV [468](#)

## W

well-tap cells  
adding to MSV designs [542](#)  
controlling the distance between [541](#)  
placing [541](#)  
removing [542](#)

wide wires, routing with NanoRoute  
router [750](#)

wire editing  
bindkeys [652](#)  
controlling visibility of cell blockages  
during [669](#)  
using CAR for [670](#)  
*See also* wires

wire groups, adding with wire editor [661](#)

wire length report [702](#)

wire tapering, with NanoRoute router [750](#)

wire width  
changing [663](#)  
repairing violations of [664](#)

wires  
adding [657](#)  
changing layers of [665](#)  
changing width of [663](#)  
correcting maximum width violations  
of [663](#)  
cutting [662](#)  
deleting [655](#)  
duplicating [664](#)  
editing with CAR [670](#)  
extending automatically [660](#)  
merging [666](#)  
moving  
selected [657](#)  
with arrow keys [656](#)  
with mouse [655](#)

reshaping routes of [668](#)  
selecting [655](#)  
splitting [666](#)  
stretching [665](#)  
trimming [662](#)

WRoute router, when to use [694](#)

## X

XML  
comment syntax [1139](#)  
reference for [1139](#)  
statement syntax [1138](#)

## Y

yield analysis  
effects considered [1124](#)  
prerequisites for [1127](#)  
results of [1127](#)  
support for 45-degree shapes [1124](#)

yield costs  
definition of [1132](#)

yield map  
changing bin size of [1130](#)  
changing colors in [1131](#)  
changing object types in [1131](#)  
displaying [1129](#)

yield report  
probabilities in [1132](#)  
sections in [1132](#)

yield technology file  
cell probability section in [1148](#)  
contents of [1138](#)  
conventions used in [1139](#)  
example of [1150](#)  
header section in [1140](#)  
open point defect section in [1140](#)  
short point defect section in [1144](#)  
using template for [1138](#)  
via probability section in [1145](#)

## **Encounter User Guide**

---