cadence™

Encounter® User Guide

Product Version 7.1.2
May 2008

© 2002-2008 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

Cadence Design Systems, Inc., 2655 Seely Avenue, San Jose, CA 95134, USA

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

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

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

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

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. 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

About This Manual . . . 33

Audience . . . 33
How This Manual Is Organized . . . 33
Conventions Used in This Manual . . . 34
Related Documents . . . 35

1
Getting Started . . . 37

Product and Installation Information . . . 39
Setting the Run-Time Environment . . . 39
- Setting the 64-bit Mode Environment . . . 39
- Examples . . . 40

Configuring OpenAccess . . . 40
Product Licenses . . . 42
Starting the Encounter Software . . . 43
- Parameters . . . 43
- Checking Out an Encounter License . . . 45
- Starting Encounter X . . . 46
- Encounter Console . . . 46
- Command Name Completion . . . 47
- Command-Line Editing . . . 47

Running Multiple-CPU Processing . . . 49
- Licensing Considerations . . . 50
- Running Multi-Threading . . . 52
- Running Distributed Processing . . . 53
- Running Superthreading . . . 54

Setting Preferences . . . 55
- Initialization Files . . . 56

Online Documentation and Help . . . 57
- Other Sources . . . 57
- Using Form Help . . . 57

Using Text Command Help 58

2
Data Preparation 59

Generating a Technology File 60
Creating Technology Information Using LEF 60
Creating Technology Information Using OpenAccess 60
Preparing Physical Libraries 60
Using LEF to Create Physical Libraries 60
Creating OpenAccess Physical Libraries 61
Unsupported LEF and DEF Syntax 61
Unsupported LEF 5.6 Syntax 61
Unsupported DEF 5.6 Syntax 62
Generating the I/O Assignment File 64
Creating an I/O Assignment File 65
Creating a Rule-Based I/O Assignment File 75
I/O Pad and Pin Assignment Examples 76
Performing Area I/O Placement 79
Preparing Timing Libraries 81
Encrypting Libraries 81
Preparing Stamp Models 82
Preparing Timing Constraints 82
Preparing Capacitance Tables 83
Preparing Data for Delay Calculation 83
Preparing Data for Crosstalk Analysis 83
Checking Designs 83
Preparing Data in the Timing Closure Design Flow 84

3
Importing and Exporting Designs 85

Overview 86
Verifying Data before Importing a Design 86
Preparing the Design Netlist 86
Creating a Flat Verilog Netlist from a DEF File 87
Recommended DEF Import Commands 87

Alternative DEF Import Flow 88
Beginning Designs 90
Loading Previously Saved Configuration Files 91
Loading Configurations Files from the Command Line 91
Loading Configuration Files from the GUI 92
Selecting Files 93
Using Select Files 93
Working with OpenAccess Designs 95
Importing an OpenAccess Design 95
Saving an OpenAccess Design 95
Restoring an OpenAccess Design 95
Transferring OpenAccess Data between Encounter and Virtuoso Chip Editor for ECO . 96
Importing OpenAccess Physical Design Data Only 96
Removing Assign Nets from a Verilog Netlist 96
Saving and Restoring Designs 97
Saving Designs 97
Restoring Designs 98
Saving and Restoring OpenAccess Designs 98
Restoring Designs from Pre-3.2 Versions of Encounter 98
Importing and Exporting Design Data 100
Loading a Partition 100
Loading Floorplan Data 100
Placement File Requirement 101
Loading an I/O Assignment File 101
Loading an FSDB File 101
Saving a Partition 102
Saving Floorplan Data 102
Importing and Exporting TDF Files 103
Importing Tile Cell Design Data 103
Converting an Encounter Database to GDSII Stream or OASIS Format 105
Creating Cells and Instances 105
Renaming LEF Vias 106
Merging GDSII Stream or OASIS Files 107
Merge Examples 107
About the GDSII Stream or OASIS Map File 111

Map File Format . . . 111
Map File Columns . . . 112
Specifying Object Subtypes . . . 115
Using Multiple Layers and Data Types . . . 118
Updating Files during an Encounter Session . . . 119

4
Flip Chip Methodologies . . . 121
Overview . . . 122
Before You Begin . . . 123
Results . . . 123
Useful Tasks . . . 123
Flip Chip Flow in Encounter . . . 125
Flip Chip Flow Steps . . . 126
APD Bump Flow . . . 130
Bump Flow Steps . . . 130
Reducing Data Size for APD Import (Bypass Flow) . . . 136
Splitting Wires in Metal Layers . . . 136
Testing the Package Routing Feasibility . . . 137
Area I/O Flow . . . 138
Routing Bumps to I/O Driver Cells (Hierarchical Area I/O Flow) . . . 140
Peripheral I/O Flow . . . 141
Data Preparation . . . 141
Peripheral I/O Flow Steps . . . 142
RDL Planning and Routing . . . 144
Peripheral I/O Extraction . . . 153
SI and Timing Analysis . . . 154
Differentiating Area I/O and Peripheral I/O . . . 155
Swapping Signals . . . 156
Creating Differential Routing to Signal Bumps . . . 158
Specify Routing Nets . . . 159
Define Differential Pairs . . . 159
Define Differential Groups . . . 159
Define a Shield Net . . . 160
Route Multiple Nets with Different Widths . . . 160

Examples and Report Files 162
Routing and Placement Constraints 162
IO_FILE Example 164
LEF and CML Example Files 165
Route Feasibility Report 166

5
Using Interface Logic Models in Hierarchical Designs 169
Overview 170
Creating ILMs 171
Example ILM Creation 172
Specifying ILM Directories at the Top Level 173
Example Top-Level Design Flow with ILMs 173
Preserving Selected Instances in ILMs 174
Creating ILMs for Shared Modules 175

6
What-If Timing Analysis 177
Performing What-If Timing Analysis 177
Prerequisite 178
Timing Models Supported for What-If Timing Analysis 178
Using the What-If Timing Commands 181

7
Partitioning the Design 185
Overview 186
Flow Methodologies 186
Top-down Methodology 187
Bottom-up Methodology 192
Specifying Partitions and Blackboxes 195
Defining Partitions 196
Defining Partitions as Power Domains 198
Defining Blackboxes 198
Handling of Blackboxes with Non-R0 Orientation 201

Specifying Multiple Instantiated Partitions and Blackboxes 203
Changing Partition Clone Orientation 204
Specifying Rectilinear Partitions and Blackboxes 205
Specifying Core-to-I/O Distance for Partition Cuts 206
Specifying Nested Partitions 207
Assigning Pins 208
Assigning Partition and Blackbox Pins 210
Assigning I/O Pins 232
Assigning Clock Pins 236
Performing Congestion-aware Pin Assignment for Channel-based Designs 237
Assigning Pins on Rectilinear Edges 240
Swapping Partition Pins 241
Pin Alignment 241
Snapping Pins to the Grid 242
Pin Assignment Limitations 243
Inserting Feedthroughs 244
Inserting Feedthrough Buffers 246
Highlighting the Nets for which Feedthrough Buffers Have been Inserted 253
Utilizing Pre-defined Feedthrough Pins in Custom Macros 253
Inserting Routing Feedthroughs 259
Generating the Wire Crossing Report 261
Interpreting the Wire Crossing Report 262
Estimating the Routing Channel Width 264
Running the Partition Program 266
Pushing Down Signal Routes 267
How Top-level Stripes Are Pushed Down 268
How Bumps, Routes, and Area I/O Cells Are Affected 271
Limitations 277
How Objects Related to X Routing Are Pushed Down 282
Restoring the Top-Level Floorplan with Partition Data 285
Concatenating Netlist Files of a Partitioned Design 286
Saving Partitions 287
Loading Partitions 287
Unpartitioning with Routing Data 287
Working with OpenAccess Database 289
Parallel Job Processing 290

8
Floorplanning the Design . . . 291
Overview . . . 292
Common Floorplanning Sequence . . . 293
Viewing the Floorplan . . . 294
Module Constraint Types . . . 297
Target Utilization Display . . . 298
Effective Utilization Display . . . 300
Calculating Density . . . 301
Standard Row Spacing . . . 302
Grouping Instances . . . 303
Defining the Bounding Box . . . 304
Creating and Editing Rows . . . 305
Using Vertical Rows . . . 305
Using Multiple-height Rows . . . 307
Using Integer Multiple-height Rows . . . 307
Using Non-Integer Multiple-height Rows . . . 309
Working with User-defined DEF Files that Contain NIMH Rows or Unaligned Rows 312
Merging Hierarchical Floorplans from Partitions . . . 314
Using Blackblobs . . . 319
Specifying Blackblobs . . . 320
Blackblob Useflow . . . 322
Blackblob Display . . . 324
Saving and Restoring Blackblobs . . . 326
Editing Pins . . . 327
Pin Snapping on Resized Boundaries . . . 327
Moving Pins . . . 327
Swapping Pins . . . 328
Using the Pin Editor . . . 328
Creating Local Preferred Direction Regions . . . 338
Running Relative Floorplanning . . . 340
Orientation Key . . . 340
Instance Place Example . . . 341
Pre-Route Examples . . . 341
Saving and Restoring Relative Floorplan . . . 343

Saving and Loading Floorplan Data 343
Adding Logical Hierarchy Without Creating Additional Hierarchy 344
Logical Hierarchy Manipulation 344
Moving Instances to a New Top Module 345
Moving Instances to an Existing Module 346
Moving Instances to the Top Root Level 347

9
Power Planning and Routing 349

Overview 350
Before You Begin 351
Results 352
Loading, Saving, and Updating Special Route 352
Creating a Ring with User Defined Coordinates 352
Global Net Connections 353
globalNetConnect Command and Connections for Signal Pins and Power/Ground Pins 354
Connecting Ring Pins 355
Fixing LEF MINIMUMCUT Violations 355
Fixing LEF Minimum Spacing Violations 355
Adding Stripes to Power Domains 356
Automatic Power Planning (APP) 358
Creating a Template 359
Using the IP Block Page 359
Using the Design Page 360
Specifying Template Parameters 361
Instantiating a Template 362
Template Naming Conventions 362
Using the Synthesize Power Plan Functionality 363
Creating Differential Routing to Signal Bumps 365

10
Low Power Design 367

Overview 368
Power Domain Shutdown and Scaling 368

Support for the Common Power Format (CPF) 370
Encounter Commands Supporting CPF 370
Loading and Committing a CPF File 371
Saving a CPF Database 371
CPF Documentation 371
Multiple Supply Voltage Flat Flow 372
Data Preparation 373
Verify Power Domains 379
Load and Commit the CPF File 379
Floorplan the Design 380
Plan Power 381
Place Standard Cells and Macros 382
Synthesize Buffer Trees (Optional) 383
Add Tie High/Low cells 384
Highlight Power Domains (Optional) 384
Power Routing 385
Trial Route 386
Synthesize Clock Trees 387
Route the Design 388
Analyze Timing 388
Optimize Timing 388
Power Analysis 391
Finish the Design Flow 392
Conformal Low-Power Verification 392
Signal Integrity Analysis 392
Multiple Supply Voltage Top-Down Hierarchical Flow 393
Leakage Power Optimization Techniques 397
Multi-Vth Optimization 397
Substrate Biasing 398
Power Shutdown Techniques 402
Power Shutdown Commands 402
Data Preparation 403
Buffer Styles 404
Adding Column Switches 405
Attaching the Acknowledge Receiver Pin 406
Enable Chaining 407

Adding Power Switch Rings 411
Ring Conventions 413
Using Pitch Control and Offsets 419
Power Switch Optimization 429
Power Switch Reduction 429
Power Switch ECO 430

11
Placing the Design 433

Overview 434
Preparing for Placement 434
Loading a Design 435
Guiding Placement With Blockages 435
Placement Treatment of Preroutes 436
Adding Well-Tap Cells 437
Controlling the Distance Between Well-Tap Cells 438
Adding Well-Tap Cells to MSV Designs 438
Deleting Well-Tap Cells 439
Adding End-Cap Cells 439
Adding End Cap Cells to MSV Designs 439
Deleting End-Cap Cells 440
Placing Spare Cells and Spare Modules 440
Placing Spare Cells That Are Included in the Netlist 440
Placing Spare Cells That Are Not Included in the Netlist 441
Spare Cell Placement Behavior 441
Running Hierarchy-Aware Spare Cell Placement 443
Adding Padding 446
Adding Instance or Module Padding 447
Adding Cell Padding 448
Placing Standard Cells 450
Running Placement in Multi-CPU Mode 451
Multi-Threading Placement Steps 451
Running Placement in MSV Designs 453
Checking Placement 453
Using the Amoeba View 455

Using the checkPlace Command 455
Using the Violation Browser 455
Using the Density Map 455
Refining Placement 456
Adding Filler Cells 456
Adding Fillers to MSV Designs 457
Deleting Filler Cells 457
Placing Gate Array Style Filler Cells for Post-Mask ECO 457
Adding Decoupling Capacitance 459
Deleting Decoupling Capacitance 459
Adding Logical Tie-Off Cells 460
Running Incremental Placement 460
Running ECO Placement 460
Saving Placement Data 460
Using Netlist Clustering Mode 460
Netlist Clustering Command Sequence 461
Specifying and Placing JTAG and Other Cells Close to the I/Os 461
Optimizing and Reordering Scan Chains 463
Specifying Scan Cells 463
About Scan Chains 463
Reordering Scan Chains 464

12
Synthesizing Clock Trees 475

Before You Begin 476
Results 476
Understanding CTS Operation Modes 477
Manual CTS Mode 477
Automatic CTS Mode 478
How CTS Calculates Skew Values 482
Improving Postroute Correlation 484
Specifying Macro Model Delays 485
Grouping Clocks 486
Analyzing Hierarchical Clock Trees 487
Module Placement Utilization 489

Clock Designs with Tight Area 489
Balancing Pins for Macro Models 489
Timing Model Requirement for Cells 489
Delay Variation and OCV 489
Understanding Post-CTS Clock Tree Optimization 490
Using the ckECO Command for Post-CTS Clock Tree Optimization 490
Support for Local Skew Optimization 491
Command Modes for the ckECO Command 491
Using a SPEF File with the ckECO Command for RC Estimation 491
Running Post-CTS Optimization with the ckECO Command 492
Guidelines for Using the ckECO Command 493
Creating a Clock Tree Specification File 494
Example of a Clock Tree Specification File 494
Timing Constraint File Statement 497
Naming Attributes Section 498
NanoRoute Attribute Section 499
Macro Model Data Section 500
Clock Grouping Data Section 503
Clock-Tree Topology Section 503
Automatic Gated CTS Section 503
Log File Headings 522
CTS Report Descriptions 524
General Information 524
Macro Model Information 526
Power Information 526
AC Current Density Violations 527

13
Working with Clock Mesh Structures 529

Overview 530
Clock Meshes Versus Clock Trees 530
Creating Clock Meshes 533
Determining the Mesh Structure 533
Implementing the Clock Mesh 538
Analyzing the Clock Mesh 539

14
Editing Wires . . . 543

Overview . . . 545
Before You Begin . . . 546
Results . . . 546
Using Keyboard Shortcuts . . . 546
- Keyboard Shortcuts That Open Forms . . . 546
- Keyboard Shortcuts That Are Equivalent to Tool Widgets . . . 546
- Keyboard Shortcuts Used in Auto Query Mode . . . 547
- Keyboard Shortcuts Used in Add Wire Mode . . . 548
- Keyboard Shortcuts Used in Stretch Wire Mode . . . 548
- Keyboard Shortcuts Used to Change Vias . . . 548

Selecting Wires . . . 549
Deleting Wires . . . 549
Moving Wires . . . 549
- Using the Mouse to Move Wires . . . 549
- Using Arrow Keys to Move Wires . . . 550
- Moving Selected Wires or Vias . . . 551

Adding Wires . . . 551
- Adding a Wire for a Single Net . . . 551
- Adding Wires for Multiple Nets . . . 553
- Adding Wires that Automatically Extend to a Target . . . 554
- Using Override to Add Wire Groups with Multiple Widths and Spacing . . . 555

Cutting Shielding Wires . . . 556
Trimming Antennas on Selected Stripes . . . 556
Changing Wire Width . . . 557
Repairing Maximum Wire Width Violations . . . 558
Duplicating Wires . . . 558
Stretching Wires . . . 559
Changing Wire Layers . . . 559
Splitting and Merging Wires . . . 560
Adding Vias . . . 560
Changing Vias . . . 561
Moving Vias . . . 562
Reshaping Routes . . . 562

Controlling Cell Blockage Visibility 563
Editing Wires with Virtuoso Chip Assembly Router 564
Running Virtuoso CAR in Batch Mode 564
Running Virtuoso CAR in Interactive Mode 565

15
Using Trial Route for Congestion and Timing Analysis 567

Overview 568
Data Preparation 568
Routing A Flat Design 569
Routing a Partitioned Design 569
Routing Two-Metal Layer Designs 572
Loading and Saving Route Data 572
Analyzing Route Data 572
Congestion Markers in the Display 573
Congestion Distribution Report 575
Improving Route Congestion 581
Additional Information 582
Wire Overlap 582

16
Using the NanoRoute Router 585

About NanoRoute Routing Technology 588
Routing Phases 588
Global Routing 588
Detailed Routing 589
ECO Routing 589
NanoRoute Router in the Encounter Flow 589
Before You Begin 590
Checking Your LEF Files 590
Checking for Problems with Cells, Pins, and Vias 591
Generating Tracks 592
Specifying Preferred Routing Layers 592
Results 592
Use Models 594

Running the NanoRoute Router with Encounter Menu Commands and Forms 594
Running the NanoRoute Router with Encounter Text Commands 594
Running the NanoRoute Router in Standalone Mode 595
Using NanoRoute Parameters 596
Using Attributes and Options Together 597
Accelerating Routing with Multi-Threading and Superthreading 599
When to Accelerate Routing 600
Superthreading Log File Excerpts 601
Following a Basic Routing Strategy 603
Using the Encounter Text Commands 603
Using the Encounter GUI 604
Checking Congestion 607
Using the Congestion Analysis Table 607
Using the Congestion Map 608
Resolving Open Nets 612
Log File Examples 612
Diagnosing Problems Using verifyTracks 613
Resolving Additional Open Net Problems 613
Running Timing-Driven Routing 615
Input Files 615
Using the CTE and the NanoRoute Router in Native Mode 615
Using the CTE and Standalone NanoRoute 616
Using Another Timing Engine 617
Routing Clocks 618
Setting Attributes for Clock Nets 618
Routing the Clock Nets 619
Running Postroute Optimization 619
Preventing and Repairing Crosstalk Problems 620
Crosstalk Prevention Options 622
Running ECO Routing 624
ECO Limitations 624
ECO Flow 625
Evaluating Violations 626
Violations on Upper Metal Layers 630
Violations in Timing-Driven Routing 632
Deleting Violated Nets 633

Using Additional Strategies to Repair Violations 633
Concurrent Routing and Multi-Cut Via Insertion 634
Postroute Via Optimization 634
Optimizing Vias in Selected Nets 635
Via Optimization Options 635
Performing Shielded Routing 637
Shielding Option 638
Performing Shielded Routing Using the GUI 638
Performing Shielded Routing Using Text Commands 639
Interpreting the Shielding Statistics Report 639
Routing Wide Wires 642
Using Non-Default Rules 642
Repairing Process Antenna Violations 644
Repairing Violations on Multiple-Pin Nets 644
Changing Layers 645
Using Diodes 645
Deleting and Rerouting Nets with Violations 645
Repairing Violations on Cut Layers 645
Process Antenna Options 646
Examples 646
Using a Design Flow that Includes Astro or Apollo 648
Troubleshooting 650

17
Using the Encounter Software in Mixed Signal Designs 651
Overview 653
Using the Mixed Signal Router 654
Before You Begin 654
Results 654
Specialized Routing Techniques 655
Matched Nets 655
Differential Pair Nets 656
Shielded Nets 656
Bus Routes 660
Using Routing Constraints 661

Constraint File Format 661
Generic Constraints 662
Specialized Constraints and Keyword Descriptions 664
NETS 664
MATCH 666
DIFFPAIR 668
SHIELDING 670
BUS 679
Creating a Constraint File 681
Using the Mixed Signal Constraint Editor 681
Using a Text Editor 685
Loading a Constraint File 686
Using the Mixed Signal Constraint Editor Form 686
Using the Mixed Signal Router Form 686
Using the routeMixedSignal Command 688
Editing a Constraint File 689
Using the Mixed Signal Constraint Editor 689
Using a Text Editor 690
Sample Constraint File 691

18
Optimizing Metal Density 695
Overview 696
Before You Begin 696
Adding Metal Fill in Distributed Processing Mode 697
After You Complete Adding Via and Metal Fill 697
Metal Fill Features 697
Staggered Metal Fill Pattern 698
Connected and Floating Metal Fill 699
Timing-Aware Metal Fill 703
Specifying Metal Fill Parameters 704
Recommendations for Adding Timing-Aware Metal Fill 705
Timing-Aware Examples 707
Specifying the Active Spacing Value 708
Adding Metal Fill Over Macros 708

Recommendations for Power Strapping Mode 709
Adding Via Fill 710
Trimming Metal Fill 711
Verifying Metal Density 711
Adding Metal Fill Using the GUI 712

19
Timing Budgeting 713
Overview 714
Is My Design Ready for Budgeting? 716
Deriving Timing Budgets 717
Budgeting Using the GUI 717
Budgeting Using Text Commands 717
Top-Level Budgets Derived by Using Virtual Partitioning 718
Deriving Preliminary Budgets in Early Design Phase 719
Budgeting Output Files for MMMC Designs 721
Corner Cloning 721
Mode Cloning 722
Setup and View Handling for MMMC Designs 723
Constraints Adjustment 724
Analyzing Timing Budgets 726
Resolving Conflicts with Path-Based Exceptions 726
Budgeting Clock Latency in Propagated Mode 729
Calculating Timing Budgets 731
Customizing Budget Generation 734
Verifying Timing Budgets 736
Reading the Justify Budget Report 737
Design Example 739
SDC Constraints for Design Example 740
Generated Report for Design Example 740
Constraints Support in Budgeting 742
Warning Report 745
Pin Constraint Values Greater than Available Time 745
Warning Report Example 745

20
RC Extraction 747

Overview 748
Before You Begin 750
- Results 750
- Specifying Temporary File Locations 750

Native RC Extraction 751
Native Sign-Off RC Extraction 752
- Incremental Extraction Support for Design Changes 754

Generating a Capacitance Table 755
- Inputs for Generating a Capacitance Table 755
- Capacitance Table Generation Flow 756
- Output of Generating a Capacitance Table 757
- Generating Capacitance Table With Specified Scaling Factors 759

Reading a Capacitance Table 761
Correlating Native Extraction With Sign-Off Extraction 762
- Comparing the ICT File with the Technology File 763
- Correlating SPEF Files Using the Ostrich Utility 763
- Comparing SPEF Files Using a Perl Script 766
- Defining the Scaling Factor 769

Sign-Off Extraction Using QRC 770
- Inputs for QRC Sign-Off Extraction 771

21
Calculating Delay 773

Overview 774
Data Preparation 774
- Operating Conditions 774
- ECSM Libraries 775
- Using IPDB-based ECSM Libraries 775

Delay Calculation Modes and Related Controls 777
Choosing A Delay Calculation Engine 778
Running Delay Calculation 778

22
Timing Analysis . . . 779

Overview . . . 780
Timing Analysis Features . . . 781
Before You Begin . . . 782
Timing Analysis Results . . . 783
Reading Timing Libraries . . . 783
Specifying Different Timing Libraries for Setup and Hold Checks . . . 784
Resolving Discrepancies in Timing Libraries . . . 784
Reading Timing Constraints . . . 784
Constraints Quick Reference . . . 785
Setting Operating Conditions . . . 788
Calculating Clock Latency . . . 789
Defining RC Corners . . . 790
Specifying Timing Analysis Modes . . . 792
Definition of Early and Late Paths . . . 792
Single Timing Analysis Mode . . . 794
Best-Case Worst-Case (BC-WC) Timing Analysis Mode . . . 798
On-Chip Variation (OCV) Timing Analysis Mode . . . 809
Clock Path Pessimism Removal . . . 815
Analyzing Timing Problems . . . 821
Resolving Buffer-Related Problems . . . 822

23
Debugging Timing Results . . . 825

Overview . . . 826
Timing Debug Flow . . . 827
Generating Timing Debug Report . . . 828
Displaying Violation Report . . . 828
Analyzing Timing Results . . . 828
Creating Path Categories . . . 832
Creating Predefined Categories . . . 832
Creating New Categories . . . 833
Hiding path categories . . . 835

Reporting Path Categories 836
Using Categories to Analyze Timing Results 838
Analyzing MMMC Categories 839
Manual Slack Correction of Categories 842
Editing Table Columns 842
Cell Coloring 844
Viewing Schematics 846

24
Optimizing Timing 847

Overview 848
Before You Begin 848
Results 849
Performing Optimization Before Clock Tree Synthesis 851
Correcting Violations in Pre-CTS Mode for the First Time 852
Performing Rapid Timing Optimization for Design Prototyping 852
Using Additional Pre-CTS Timing Optimization Parameters 852
Performing Incremental Pre-CTS Optimization 854
Changing Default Settings in Pre-CTS Mode 855
Performing Post-CTS Optimization 856
Correcting Violations in Post-CTS Mode 856
Using Additional Post-CTS Timing Optimization Parameters 856
Performing Incremental Post-CTS Optimization 857
Changing Default Settings in Post-CTS Mode 858
Performing Postroute Optimization 859
About Postroute Optimization 859
Correcting Violations in Postroute Mode 861
Correcting Signal Integrity Violations 862
Changing Default Settings in Postroute Mode 864
Using Useful Skew 864
Using Useful Skew in Pre-CTS Mode 864
Using Useful Skew in Post-CTS Mode 865
Controlling Useful Skew Optimization 865
Using Virtual Partitioning for Chip-Level Interface Circuit Timing Closure 867
Optimizing Timing in On-Chip Variation Analysis Mode 868

Specifying the MMMC Environment 868
Optimizing Timing in OCV Mode Using the Default Delay Calculator 870
Optimizing Timing in OCV Mode Using the Sign-Off Delay Calculator 870
Using Conformal Constraint Designer During Timing Optimization 871
Post-Processing Approach 871
Integrated Approach 872
Optimizing Timing Using a Rule File 875
Optimizing Timing When the Constraint File Includes the set_case_analysis Constraint . . 875
Using the Footprintless Flow 875
Using Cell Footprints 876
Viewing Added Buffers, Instances, and Nets 877
Default Naming Conventions 877

25
Interactive ECO 879

Overview 880
Before You Begin 880
Results 880
Adding Buffers 880
Changing the Cell 883
Deleting Buffers 884
Displaying Buffer Trees 886
Running ECO Placement 888
Naming Conventions for Interactive ECO 889
Comparing Physical Design Data 889

26
Analyzing and Repairing Crosstalk 895

Overview 896
Before You Begin 898
Results 898
Performing Crosstalk Prevention 899
Data Preparation 899
Analyzing Crosstalk 900

Repairing Crosstalk Violations 901
Running the Incremental SI Flow 901
Running the Sign-Off SI Flow 903

27
Power Analysis 905

Overview 907
Power Analysis Modes 907
Before You Begin 908
Setting Up Power Analysis 909
Results 909
Displaying Macro Current Source Locations 910
Running Power Analysis with Encounter 911
Running Power Analysis in Simple Estimate Mode 913
Estimation Parameter File 914
Running Power Analysis with VoltageStorm 915
VoltageStorm Power Grid Analysis Flow 915
Running VoltageStorm 915
Performing Static Power Calculation 916
Inputs 917
Outputs 918
Power Analysis Flow — Early Stage 919
Power Analysis Flow — Late Stage 920
Supported CPE Text Commands 921
Supported Database Commands 921
Performing CPF-Driven Power-Up Analysis 923
Creating a Power Pad Location File 924
Using Auto Fetch 926
Simple Decoupling Capacitor Modeling 927
Viewing the Power Graph 927
Saving Rail Analysis Values 928
Power/Rail Summary Report 929
Power Graph Report 929
Instance Average Power Report 931
Net Average Power Report 931

Instance Power Data File . . . 932
Instance Voltage File . . . 932
Boundary Voltage File . . . 933
Display Setting File . . . 934
Calculating EM Values for Wires and Vias . . . 935
Log of Physical Connectivity to Power Grid . . . 935
Operating Voltage Recognition . . . 936
Removing Power Consumption for Specified Pins or Ports . . . 936

28
Verifying Violations . . . 937

Overview . . . 938
Verifying Connectivity . . . 941
  Before You Begin . . . 941
  Types of Connectivity Violations Reported . . . 941
  Results . . . 942
Verifying Metal Density . . . 943
  Before You Begin . . . 943
  Results . . . 943
Verifying Geometry . . . 944
  Before You Begin . . . 944
  Spacing Violation Checks . . . 944
  Types of Antenna Violations Reported . . . 945
  Support for Via Rules . . . 946
  Results . . . 946
Verifying Process Antennas . . . 947
  Before You Begin . . . 947
  Verifying PAE . . . 947
  Results . . . 947
  Sample Process Antenna Report . . . 948
Verifying Maximum Floating Area Violations . . . 950
Verifying AC Limit . . . 951
  Before You Begin . . . 951
  Results . . . 951
Viewing Violations With the Violation Browser . . . 952

Viewing Geometry or Metal Density Violations 952
Viewing Connectivity, Process Antenna, or AC Limit Violations 952
Viewing Violation Markers From Assura or Calibre 952
Violation Browser Features 953
Clearing Violations 955

29
Analyzing Yield 957
Overview 958
What Effects Does reportYield Consider? 958
Calculating Failure Probabilities 959
Critical Area Analysis 960
Defect Data and Cumulative Defect Data Functions 961
Before You Begin 961
Results 961
Interpreting the Yield Map 963
Displaying the Yield Map 963
Interpreting the Yield Report 966
Yield Report 966
Detailed Report 970
Understanding the Yield Technology File 972
File Format 972
File Sections and Keyword Statement Descriptions 974
Yield Technology File Example 984
Formulas and Calculations 987
Calculating the Probability of Failure for a Metal Layer 987
Calculating Defect and Cumulative Defect Data 987
Cost Formulas 990

30
Creating An Initial Floorplan Using Masterplan 991
Overview 992
Masterplan Automatic Floorplanner Flow 993
Data Preparation 994
Selecting Seeds 994

Importing the Design 1000
Setting Masterplan Global Parameters 1001
Creating an Initial Floorplan 1001
Creating a Hierarchical Floorplan 1002
Analyzing the Floorplan 1003
Guidelines For a Good Floorplan 1005
Adjusting Macro Placement 1005
Manual Macro Adjustment 1005
Masterplan Macro Adjustment 1006
Placing Cells and Generating Floorplan Guides 1011
Saving the Floorplan 1011

31
Performing Multi-Mode Multi-Corner Timing Analysis and Optimization 1013

Overview 1015
Configuring the Setup for Multi-Mode Multi-Corner Analysis 1016
Creating Library Sets 1017
Creating Virtual Operating Conditions 1019
Creating RC Corner Objects 1020
Creating Delay Calculation Corner Objects 1021
Adding A Power Domain Definition To A Delay Calculation Corner 1023
Creating Constraint Mode Objects 1024
Creating Analysis Views 1028
Setting Active Analysis Views 1029
Checking the Multi-Mode Multi-Corner Configuration 1030
Saving Multi-Mode Multi-Corner Configurations 1031
Controlling Multi-Mode Multi-Corner Analysis Through the Flow 1031
Performing Timing Analysis 1034
Generating Timing Reports 1034

32
Creating the ICT File 1037

Format 1038

Data 1038
Comments 1038
Case Sensitivity 1038
Warnings and Errors 1038
Invalid Layer Names 1038
Commands 1038
Sample ICT File 1051

33
ECO Flows 1063

Overview 1064
Assumptions 1064
Flows 1064
Pre-Mask ECO Changes from an ECO File 1066
Preparation 1066
Flow 1067
Steps 1067
Pre-Mask ECO Changes from a New Verilog File 1070
Preparation 1070
Flow 1070
Steps 1071
Pre-Mask ECO Changes from a New DEF File 1073
Preparation 1073
Flow 1074
Steps 1074
Post-Mask ECO Changes from a New Verilog Netlist 1077
Preparation 1077
Flow 1078
Steps 1078
Post-Mask Gate Array Style ECO from a New Verilog Netlist 1082
Preparation 1082
Steps 1084

ECO Directives 1087
ADDHIERINST 1089
ADDINST 1090
ADDNET 1092
ATTACHTERM 1093
CHANGECELL 1095
CHANGEINST 1096
DELETEBUFFER 1098
DELETEINST 1100
DELETENET 1101
DETACHTERM 1102
INSERTBUFFER 1103
SWITCHTERM 1106
Example ECO File 1107

Clock Mesh Specification File 1111
Overview 1111
Routing Type Definitions 1112
Cutout Definitions 1112
Clock Mesh Definitions 1113
Timing and Power Constraints Section 1114
Tracing and Analysis Scope Section 1115
Mesh Structure Section 1116
Mesh Stage Section 1119
Top Chain Section 1125
Local Tree Section 1126
Clock Mesh Specification File Example 1129

CPF Script Example 1135

Index 1147

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 1, “Getting Started”

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

- Chapter 2, “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}`

`{[ ] [ ]}`	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.
Command – Subcommand	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 *Floorplan* from the menu, then *Power Planning* from the submenu, and then *Add Rings* from the displayed list: *Floorplan – Power Planning – Add Rings* 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 Menu Reference Manual*

  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 Timing Closure Guide*

  Addresses issues related to timing closure for challenging designs. It presents a recommended task flow and provides tips on how to resolve or avoid common timing closure problems typically found in a design.

- *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 Library Development Guide*

  Describes library development guidelines for the independent tools that make up the Encounter family of products.

- *Encounter Text Command Reference*

  Describes the Encounter text commands, including syntax and examples.

- *What's New in Encounter*

  Provides information about new and changed features in this release of the Encounter family of products.

- `README` file

  Contains installation, compatibility, and other prerequisite information, including a list of CCRs that were resolved in this release. You can read this file online at downloads.cadence.com.

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

5/6/08

1

Getting Started

- Product and Installation Information on page 39
- Setting the Run-Time Environment on page 39
  - Setting the 64-bit Mode Environment on page 39
  - Examples on page 40
- Configuring OpenAccess on page 40
- Product Licenses on page 42
- Starting the Encounter Software on page 43
  - Parameters on page 43
  - Checking Out an Encounter License on page 45
  - Starting Encounter X on page 46
  - Encounter Console on page 46
  - Command Name Completion on page 47
  - Command-Line Editing on page 47
- Running Multiple-CPU Processing on page 49
  - Licensing Considerations on page 50
  - Running Multi-Threading on page 52
  - Running Distributed Processing on page 53
  - Running Superthreading on page 54
- Setting Preferences on page 55
  - Initialization Files on page 56
- Online Documentation and Help on page 57

- Other Sources on page 57
- Using Form Help on page 57
- Using Text Command Help on page 58

Product and Installation Information

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

- downloads.cadence.com, where you can review the README before you download the Encounter software.
- In the software installation, where it is also available when you are using or running the Encounter software.
- At the top level of your installation hierarchy.

Setting the Run-Time Environment

➤ To set the run-time environment, set the following installation directory in your path:

install_dir/tools/bin

Setting the 64-bit Mode Environment

Some Cadence® applications have both 32-bit and 64-bit versions. The 64-bit versions of applications are installed in the same tools hierarchy as the 32-bit versions. A “wrapper” for each application determines which version of the application is run.

To run a 64-bit version of Encounter software:

1. Verify that your operating system supports 64-bit applications.
2. Verify that a 64-bit version of the application is installed.

   The 64-bit version of an application is located in the `64bit` directory in the standard installation location of the application. For example, *your_install_dir*/tools/bin/64bit.
3. Set the following environment variable:

   ```
   CDS_AUTO_64BIT { ALL | NONE | list }
   ```

ALL	All applications are run as 64 bit.
NONE	All applications are run as 32 bit.

list	Only the applications specified are run as 64-bit. Specify *list* as a list of case-sensitive application names, separated by a colon, comma, or semicolon. If you use a semicolon, enclose the list in single quotation marks. For the Encounter family of products, the following executable names are valid entries for *list*: ■ `celtic` ■ `encounter` ■ `nanoroute` ■ `pks_shell` ■ `wroute`

Note: If you do not set the `ALL` environment variable, the Encounter software automatically starts in 32-bit mode.

Examples

The following examples show how to run the Encounter software in various 64-bit modes.

- All Encounter products are run in 64-bit mode:

  ```
  setenv CDS_AUTO_64BIT ALL
  ```

- The Encounter and CeltIC® products run in 64-bit mode. All other products are run in 32-bit mode:

  ```
  setenv CDS_AUTO_64BIT encounter:celtic
  ```

  or

  ```
  setenv CDS_AUTO_64BIT ‘encounter;celtic’
  ```

- Only `pks_shell` runs in 64-bit mode. All other products run in 32-bit mode:

  ```
  setenv CDS_AUTO_64BIT pks_shell
  ```

Configuring OpenAccess

Cadence products that require OpenAccess automatically install OpenAccess in the *cadence_install_dir/* directory. A symbolic link is automatically created from *cadence_install_dir/*`share/oa` to the OpenAccess installation directory.

Cadence tools report with which OpenAccess version and data model they were compiled. For example, when you start the Encounter software, the software displays a message similar to the following:

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

It is strongly recommended that you use the OpenAccess kit that comes with the Encounter software. However, you can change the OpenAccess kit that your sofware uses by setting the `OA_HOME` environment variable to override the default OpenAccess installation:

```
setenv OA_HOME oa_install_dir
```

Where *`oa_install_dir`* is the path to the OpenAccess installation to use.

Important

If you choose to use the `OA_HOME` environment variable:

- The version of the OpenAccess kit that 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.

For information on the version of OpenAccess supported with this release, see the README file in the *`cadence_install_dir/`* directory.

Product Licenses

The following table shows the mapping of license strings to product names.

License	Product Name
FE80	First Encounter® L silicon virtual prototype system
FE100 GPS	First Encounter® XL global physical synthesis system
FE800	First Encounter® GXL hierarchical RTL-to-GDSII physical implementation system
FE300 DBS	SoC Encounter™ L demand-based savings (DBS) system
FE200 GPS	SoC Encounter™ XL hierarchical RTL-to-GDSII physical implementation system
FE850	SoC Encounter™ GXL advanced hierarchical RTL-to-GDSII physical implementation system
FE150	NanoRoute Ultra® nanometer ultra router to GDSII system
3002	Virtuoso® Digital Implementation
X4ENC	SoC Encounter™ X Option

The Encounter family of products offers a wide range of features and functionality. The features to which you have access are determined by your product license.

- Go to http://www.cadence.com/products/product_index.aspx?platform=DigitalIC and click on either SoC Encounter or First Encounter Silicon Virtual Prototyping under “PRODUCTS”. Click on “Download PDF datasheet” under “SoC ENCOUNTER RTL-to-GDSII SYSTEM” or “FIRST ENCOUNTER SILICON VIRTUAL PROTYPING” and scroll down to the table which lists the features that are supported for the L, XL, and GXL platforms.
- For NanoRoute Ultra, go to http://www.cadence.com/products/product_index.aspx?platform=DigitalIC and click on “NanoRoute Ultra” under “PRODUCTS”. Click on “Download PDF datasheet” under “NANOROUTE ULTRA ROUTER” to view a description of the product.
- For the Virtuoso® Digital Implementation go to http://www.cadence.com/products/custom_ic/vdio/index.aspx and click on “Download PDF datasheet” to view a description of the Virtuoso® Digital Implementation.

Starting the Encounter Software

To start an Encounter session, type the `encounter` command with the appropriate options at the UNIX prompt. The syntax for the `encounter` command is as follows:

```
encounter
     [-ultra | -nanor | -fegps | -socegps | -vip | -vdi [-N{1 | 2}]
     | -fel | -fexl | -fegxl | -socel | -socexl | -socegxl | -Xmode]
     [-log logFileName]
     [-config configFileName]
     [-init tclFileName]
     [-accel_any_lic]
     [-win | -nowin]
     [-overwrite]
     [-version]
     [-accel_any_lic]
```

Parameters

`-accel_any_lic`

Searches for available licenses to use for multi-threading with the NanoRoute router or the Wroute router. After completion, the software returns the checked-out licenses.

If you do not specify this parameter, you must have available Route Accelerator licenses for additional threads.

The `-accel_any_lic` parameter is for backward compatibility. For information on the commands to use for accelerating the design process, see the "Multiple-CPU Processing Commands" chapter in the *Encounter Text Command Reference* or "Running Multi-Threading" on page 52.

`-config` *`configFileName`*

Specifies the design input configuration file.

`-init` *`tclFileName`*

Specifies the Tcl file to be read in during the start of the Encounter session. When the command finishes executing the Tcl file, type the `win` command in the Encounter console to display the Encounter main window.

`-log` *`logFileName`*

Represents the user-specified log filename.

`-nowin` Starts an Encounter session in a non-GUI environment. If you specify `-nowin`, you cannot use the `win` parameter to display the Encounter main window.

`-overwrite` Overwrites the existing file specified by *`logFileName`*. Neither the log nor command files are incremented, and both are overwritten.

`-replay` Specifies the Tcl file to be read in when starting Encounter.

```
-ultra | | -nedbs | -nanor | -fegps | -socegps | -vip |
-vdi [-N{1 | 2}]| -fel | -fexl | -fegxl | -socel | -socexl |
-socegxl | -Xmode
```

Checks out the specified license; license names are case-insensitive.

Note: If the license you specify is not available, an error message is generated and the Encounter software does not start.

`-ultra` First Encounter® Ultra

`-nanor` NanoRoute® Ultra

`-fegps` First Encounter® XL (previously named First Encounter® Global Physical Synthesis)

`-socegps` SoC Encounter™ XL (previously named SoC Encounter™ Global Physical Synthesis)

`-vip` First Encounter® L (previously named First Encounter® Virtual Prototype)

`-vdi -N{1 | 2}`

Virtuoso® Digital Implementation

Note: The VDI option is limited to designs with a maximum of 50,000 instances. To check out two VDI licenses, type the following command:

```
encounter -vdi -N2
```

`-fel` First Encounter® L

`-fexl` First Encounter® XL

`-fegxl` First Encounter® GXL

`-socel` SoC Encounter™ L

`-socexl` SoC Encounter™ XL

`-socegxl` SoC Encounter™ GXL

`-Xmode`	Encounter X *Important* This parameter is obsolete and has not been replaced.
`-version`	Displays the version of Encounter software you are running. (When you use this parameter, no Encounter license is checked for, nor does the Encounter software start.)
`-win`	Displays the Encounter main window if you have started the Encounter software using the `-init` parameter.

If you do not specify any parameters, the following actions occur:

- A log file and a command file are created in the form `encounter.log#` and `encounter.cmd#`.
- The first available license is checked out.
- The Encounter main window opens.

Checking Out an Encounter License

When you start the software, the system attempts to check out the most complete license, in the following sequence:

1. SoC Encounter GXL
2. First Encounter GXL
3. SoC Encounter XL SoC Encounter GPS
4. SoC Encounter
5. First Encounter XL First Encounter GPS
6. First Encounter Ultra
7. NanoRoute Ultra
8. SoC Encounter L Nano Encounter DBS
9. First Encounter L First Encounter VIP

Note: If you specify a license that is not available, the software does not start.

Note: You cannot run Encounter software version 6.2 without a license for 6.2 or higher.

- If you specify Encounter without any license-related options, such as `-fel | -fexl | .. | -socegxl`, Encounter will try to check out the license with the most functionality.
- If you specify Encounter with just *one* license-related option, such as `-fel`, the specified license will be checked out, if available.
- If you specify Encounter with more than one license-related option, such as `{ -o1 -o2 -o3 ... -oN }`, any option from `o1` to `oN` will be checked out, starting from `o1`.

For example, if you specify `encounter -fel -fexl -socegxl -init run.tcl,` only `First_Encounter_VIP`, `FE_GPS`, `SOC_Encounter_GXL` will be searched, starting from `First_Encounter_VIP`.

Starting Encounter X

Encounter X® runs only on the Linux operating system. Start the software with the following parameters:

```
encounter -Xmode {soce | socexl | socegxl}
```

Important

The `-Xmode` parameter is obsolete and has not been replaced.

Encounter Console

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

```
encounter>
```

If you use this 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.

Command Name Completion

You can use the `Tab` key within the Encounter console to complete text command names.

After you type a partial text command name and press the `Tab` key, the Encounter software displays either

- 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 partial text you typed

Caution

When the Encounter software displays multiple matches for a given string of partial text, the list of matches might include command names that are not formally supported in this release.* Only *those commands that appear in the* Encounter Text Command Reference *and in the Encounter console's command-line help and man pages have been validated and are supported.

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

Escape Sequences

`Esc ^H`	Delete previous word (`Backspace` key) [*n*]
`Esc Delete`	Delete previous word (`Delete` key) [*n*]
`Esc SP`	Set the mark (`Space` bar); see `^X^X` and `^Y` above
`Esc .`	Get the last (or [*n*]'th) word from previous line

`Esc <`	Move to start of history
`Esc >`	Move to end of history
`Esc b`	Move backward a word [*n*]
`Esc d`	Delete word under cursor [*n*]
`Esc f`	Move forward a word [*n*]
`Esc l`	Make word lowercase [*n*]
`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

Running Multiple-CPU Processing

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

- Multiple-threading

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

  The following Encounter features support multiple-threading:

  - Global placement in `-modulePlan` mode
  - Global and detailed routing with the NanoRoute router
  - Verify geometry

- Distributed processing

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

  The following Encounter features support distributed processing:

  - Verify geometry

- Masterplan
- Metal fill

- 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.

  The following Encounter feature supports Superthreading:

  - NanoRoute global routing

Licensing Considerations

The Encounter product you are using and the number of threads and computers required determine the number of additional licenses you need. The same license model is used for multi-threading, distributed processing, and Superthreading.

- The license used for the initial invocation of the software is called the “base license.” The product being invoked is called the “base product.”
- The base license determines the feature set available for the multiple-CPU session.
- The base license enables one CPU.
- Additional licenses used for multithreading, distributed processing, and superthreading are called “multi-CPU” licenses.
- For First Encounter, NanoRoute, and the Encounter Timing System, the multi-CPU licenses must be in the same product family as the base license. For example, a First Encounter base product can use only `fel`, `fexl`, and `fegxl` multi-CPU licenses; an ETS base product can use only `etsgxl`, `etsxl` and `etsl` multi-CPU licenses.
- An SOC-E product license can be used as a multi-CPU license for any Encounter product, including First Encounter, NanoRoute and the Encounter Timing System. The SOC-E product licenses are `socegxl`, `socexl`, `socel`.
- The software checks out licenses starting from the base product family (L, XL, GXL) and then SoC-E (L, XL, GXL). You can change this behavior by using the following command: `getMultiCpuLicense`.
- The software holds the additional licenses for the duration of the Encounter session. You can change this behavior by using one of the following commands:

  - `releaseMultiCpuLicense`

- ❑ `setReleaseMultiCpuLicense`

- Each multi-CPU license for an XL or GXL product enables three additional CPUs.
- Each multi-CPU license for an L product enables one additional CPU per 300 K of logical instances in the design.
- If you request more CPUs than are available (based on the number of available licenses), the software issues a warning and runs with the number of CPUs that are available
- In distributed processing and Superthreading, if there are not enough machines, or CPUs, available, most applications will wait for all machines, or CPUs, are available.

The following table shows the number of additional CPUs that are enabled when the base license is for an SOC-E product and the multi-CPU licenses are also for SoC-E products:

Base License		**Multi-CPU Licenses**		
SOC-E L/XL/GXL		SoC-E L	SoC-E XL	SoC-E GXL
CPU	1	+1 per 300 K logical instances	+3	+3

The following table shows the number of additional CPUs that are enabled when the base license is for a First Encounter product and the multi-CPU licenses are for First Encounter or SoC-E products:

Base License		**Multi-CPU Licenses**		
FE L/XL/GXL		FE L SoC-E L	FE XL SoC-E XL	FE GXL SoC-E GXL
CPU	1	+1	+3	+3

The following table shows the number of additional CPUs that are enabled when the base license is for a NanoRoute product and the multi-CPU licenses are for NanoRoute or SoC-E products. (There is no NanoRoute L product.):

Base License		**Multi-CPU Licenses**		
NR XL/GXL		SoC-E L	NR XL SoC-E XL	NR GXL SoC-E GXL
CPU	1		+3	+3

Base License		Multi-CPU Licenses		
NR XL/GXL	1	+1	+3	+3

Running Multi-Threading

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

```
setMultiCpuUsage
```

Use this command to specify the maximum number of threads to use.

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
```

For more information, see `setMultiCpuUsage` in the “Multiple-CPU Processing Commands” chapter of the *Encounter Text Command Reference*.

Important

Verify Geometry does not use this command. To run Verify Geometry with four threads, specify the following command:

```
verifyGeometry -cpu 4
```

For more information, see `verifyGeometry` in the “Verification Commands” chapter of the *Encounter Text Command Reference*.

Changing the License Search Order

To change the license search order, use one of the following commands:

- `getMultiCpuLicense`
- `encounter -accel_any_license`

Using getMultiCpuLicense

Use the following command before running the multiple-CPU application (for example, if the base license is for First Encounter GXL and you do not want to consume any SOC-E GXL licenses for multi-CPU applications):

```
getMultiCpuLicense -licOptions "fel fexl soce socel socexl"
```

For more information, see `getMultiCpuLicense` in the “Multiple-CPU Processing Commands” chapter of the *Encounter Text Command Reference*.

Using encounter -accel_any_lic

The `-accel_any_lic` parameter is valid only for multi-threading with the NanoRoute router or the Wroute router.

When you specify this parameter, the software uses other available licenses, such as SoC-E GXL and SoC-E XL license, to enable multi-threading.

For more information, see “Starting the Encounter Software” on page 43.

Releasing Licenses Before the Session Ends

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
  ```

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

  ```
  releaseMultiCpuLicense
  ```

For more information, see the following commands in the “Multiple-CPU Processing Commands” chapter in the *Encounter Text Command Reference*:

- `releaseMultiCpuLicense`
- `setReleaseMultiCpuLicense`

Running Distributed Processing

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

- `setDistributeHost`

  Use this command to specify a configuration file, or create the configuration for the remote shell, secure shell, or load-sharing facility queue.

- `setMultiCpuUsage`

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

  The software uses the same number of threads on each computer. If you specify a maximum number of threads that is not an even multiple of the maximum number of hosts, the software uses the maximum number for an even multiple.

For example, to verify geometry on four machines with a remote shell configuration, specify the following commands:

```
setDistributeHost -rsh -add host1 host2 host3 host4
setMultiCpuUsage -numThreads 4
verifyGeometry
```

You can change the license search order or release the licenses before the session ends, as described in the Running Multi-Threading section.

For more information, see the following topics:

- Changing the License Search Order on page 52
- Releasing Licenses Before the Session Ends on page 53
- Multiple-CPU Commands chapter in the *Encounter Text Command Reference*
  - `setDistributeHost`
  - `setMultiCpuUsage`

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.

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
setMultiCpuUsage -superThreadsNumHosts 2 -superThreadsNumThreads 6
globalRoute
```

Important

The software still accepts the following `setNanoRouteMode` parameters in this release, but they are overwritten by user-specified values for the multi-CPU commands:

- `-envNumberProcessor`
- `-envSuperthreading`
- `-envDontUseLicsForThreadings`

For more information, see the following commands in the *Encounter Text Command Reference*:

- Multiple-CPU Processing Commands chapter
  - `setDistributeHost`
  - `setMultiCpuUsage`
- Route Commands chapter
  - `setNanoRouteMode`

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. **Note:** Usage of this file is no longer recommended, but is allowed for backward compatibility. You should use `enc.tcl` 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. **Note:** 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 (*`encounter_installation_path/`*`tools/fe/etc`), 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 Design – Preferences in the *Encounter Menu Reference*). **Note:** By default, Encounter saves changes that you make to your preferences to the `enc.pref.tcl` file in the working directory.
`.enc`	Contains design preferences set using the Windows tab in the Preferences form in the GUI (see Design – Preferences in the *Encounter Menu Reference*).

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
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.

Online Documentation and Help

You can access Encounter documentation by using the Cadence Help online documentation system:

- From the `<installation_dir>\tools\bin` directory, type `cdnshelp` at the command prompt.
- From the Encounter Timing System, click the *Help* button on any forms and dialog boxes.
- From the main *Help* menu.

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

Other Sources

You can also access the following Cadence web sites by clicking *Help – Customer Support* on the upper-right side of the main window:

- SourceLink®

  Opens SourceLink in your browser.

- Web Collaboration

  Opens SpaceCruiser in your browser.

Using Form Help

To see information about a specific form, click the form's *Help* button. This takes you to the form's information in the *Encounter Menu Reference*.

Using Text Command Help

To see syntax information for an Encounter command, type the following in the Encounter console:

```
help command_name
```

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

```
help
```

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

```
man command_name
```

2

Data Preparation

- Generating a Technology File on page 60
- Preparing Physical Libraries on page 60
- Unsupported LEF and DEF Syntax on page 61
- Generating the I/O Assignment File on page 64
- Preparing Timing Libraries on page 81
- Encrypting Libraries on page 81
- Preparing Stamp Models on page 82
- Preparing Timing Constraints on page 82
- Preparing Capacitance Tables on page 83
- Preparing Data for Delay Calculation on page 83
- Preparing Data for Crosstalk Analysis on page 83
- Checking Designs on page 83
- Preparing Data in the Timing Closure Design Flow on page 84

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 `MACRO`s manually.

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

Creating OpenAccess Physical Libraries

You can translate the LEF `MACRO`s 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 *diagWidth* ;] [DIAGSPACING *diagSpacing* ;] [DIAGMINEDGELENGTH *diagLength* ;] [SLOTWIREWIDTH *minWidth* ;] [SLOTWIRELENGTH *minLength* ;] [SLOTWIDTH *minWidth* ;] [SLOTLENGTH *minLength* ;] [MAXADJACENTSLOTSPACING *spacing* ;] [MAXCOAXIALSLOTSPACING *spacing* ;] [MAXEDGESLOTSPACING *spacing* ;] [SPLITWIREWIDTH *minWidth* ;] [HEIGHT *distance* ;] [SHRINKAGE *distance* ;] [CAPMULTIPLIER *value* ;]
Macro Pin	[TAPERRULE *ruleName* ;] [NETEXPR *“netExprPropName defaultNetName”* ;] [SUPPLYSENSITIVITY *powerPinName* ;] [GROUNDSENSITIVITY *groundPinName* ;]
Nondefault Rule	[DIAGWIDTH *diagWidth* ;] [HARDSPACING ;] [MINCUTS *cutLayer numCuts* ;]

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 viaRuleName ;]`
Via	`POLYGON pt pt pt ... ;`
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 {propName propValue}...]`
Extensions	All `BEGINEXT` syntax
History	All `HISTORY` syntax

Nets	`[+ SYNTHESIZED]`
	`[+ VPIN vpinName [LAYER layerName] pt pt`
	`[PLACED pt orient | FIXED pt orient | COVER pt orient]]`
	`[+ SUBNET subNetName`
	`[ ( {compName pinName | PIN pinName | VPIN vpinName} ) ]`
	`[NONDEFAULTRULE ruleName]]`
	Note: `SUBNET NONDEFAULTRULE` is ignored; routing uses rule for `NET`.
	`[+ USE {RESET | SCAN | TIEOFF}]`
	Note: Supports `ANALOG`, `CLOCK`, `GROUND`, `POWER`, and `SIGNAL`.
	`[+ PATTERN {STEINER | WIREDLOGIC}`
	`[+ ESTCAP wireCapacitance]`
	`[+ SOURCE {DIST | NETLIST | TEST | USER}`

DEF Statement	Unsupported Syntax

Pins	`[+ USE {TIEOFF | SCAN | RESET}`
	Note: Supports `SIGNAL`, `POWER`, `GROUND`, `ANALOG`, and `CLOCK`.
	`[+ DIRECTION FEEDTHRU]`
	`[+ NETEXPR "netExprPropName defaultNetName"]`
	`[+ SUPPLYSENSITIVITY powerPinName]`
	`[+ GROUNDSENSITIVITY groundPinName]`
	`[+ COVER pt orient]`
	Note: `+ COVER` is ignored and converted to `+ FIXED`

Pin Properties	All `PINPROPERTIES` syntax
Property Definitions	The object types: `GROUP`, `REGION`, and `ROW`
Regions	`[+ PROPERTY {propName propVal}...]`
Rows	`[+ PROPERTY {propName propVal}...]`
Slots	All `SLOTS` syntax

Special Nets	`[+ SYNTHESIZED]`
	`[+ VOLTAGE volts]`
	`[+ SOURCE {DIST | NETLIST | USER}]`
	`[+ USE {RESET | SCAN | TIEOFF}]`
	Note: Supports `ANALOG`, `CLOCK`, `GROUND`, `POWER`, and `SIGNAL`.
	`[+ PATTERN {STEINER | WIREDLOGIC}]`
	`[+ ESTCAP wireCapacitance]`
	`[+ WEIGHT weight]`
	Note: `+ WEIGHT` only supported in `NETS` section.
	Special Wiring Statement:
	`[+ STYLE styleNum]`
	Note: 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.

Styles	All `STYLES` syntax

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

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

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. 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 65
- Creating a Rule-Based I/O Assignment File on page 75
- I/O Pad and Pin Assignment Examples on page 76
- Performing Area I/O Placement on page 79

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
    (locals
        space = 1.2
        ring_number = 1
    )

    (inst
        name = ioinst_1
        skip = 2.2
        space = 1.2
        offset = 10.2
        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)
    )
    (topright
    (locals
        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
        )

        (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 = bc1
        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
        name = lvdsov33v12_ca_sref_i42_r1c1
        cell = bc1
        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.
`io_order`	Specifies the order of the I/O pads and pins. This can be: ■ clockwise ■ counterclockwise ■ default **Note:** 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.
`total_edge`	Specifies the number of edges for the rectilinear block design. The edges are numbered starting from 0. For example, if the *`total_edge`* is 4, then the edges are numbered as edge 0, edge 1, edge 2, and edge 3. **Note:** You must verify that the total number of edges that you specify matches with the value in the destination design.
`space`	Specifies the global I/O pin spacing, in µmeters.

`iopad locals`

`space`	Specifies the local I/O pad spacing, in µmeters. **Note:** This space setting is honored by the first cell on one edge, when xy or offset is not specified.

	ring_number	Specifies the ring number in which the I/O pad is placed.
`iopad instance`		
	name	Specifies the name of the I/O instance.
	x, y	Specifies the absolute `x,` y location of the I/O pad instance, starting from the lower left corner. **Note:** Specifying x,y location for sides and edges of I/O pads is not supported in the I/O file.
	skip	Specifies the distance, in µmeters, of the I/O pad from the previously defined I/O pad. The value that you specify here is valid only for this cell.
	space	Specifies the spacing, in µmeters, 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 µmeters. 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.

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 µmeters, 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.

keepclear

Specifies an area on the chip where you cannot place pins or pads. Specify a range between *begin* and *end*, in µmeters, 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 µmeters, between the corner pad and the last I/O pad for the specified side of the design.

`iopin locals`

`side` Specifies the side of the I/O pin. This can be:

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

`edge num = 0` Specifies the edge number of the I/O pin, with `edge num = 0` starting from the left side of the lowest y coordinate and the left most corner, in the clockwise direction.

`space` Specifies the spacing, in µmeters, between the previously defined pin and the pin being defined.

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

`iopin`

`pin name` Specifies the name of a pin. Specify I/Os as pins for block designs.

`layer` Specifies the metal layer on which the pin must be placed.

`width` Specifies the width of the pin in µmeters. It is the length of the edge that is centered at the reference point.

depth	Specifies the length of the pin in µmeters.
#	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

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

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
```

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
```

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 65.

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.
`in_file`	Specifies the name of library file to be encrypted.
`-ogz`	Creates a gzip file of the encrypted output library file.
`out_file`	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 21, “Calculating Delay” for information about preparing ECSM libraries.

Preparing Data for Crosstalk Analysis

For information on preparing data for crosstalk analysis, see Chapter 26, “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*.

3

Importing and Exporting Designs

- Overview on page 86
- Verifying Data before Importing a Design on page 86
- Preparing the Design Netlist on page 86
- Creating a Flat Verilog Netlist from a DEF File on page 87
- Beginning Designs on page 90
- Loading Previously Saved Configuration Files on page 91
- Selecting Files on page 93
- Working with OpenAccess Designs on page 95
- Removing Assign Nets from a Verilog Netlist on page 96n
- Saving and Restoring Designs on page 97
- Importing and Exporting Design Data on page 100
- Converting an Encounter Database to GDSII Stream or OASIS Format on page 105
- About the GDSII Stream or OASIS Map File on page 111
- Updating Files during an Encounter Session on page 119

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:

`uniquifyNetlist`

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

There is no equivalent command for uniquifying 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.

Caution

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

Important

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*.

Alternative DEF Import Flow

As an alternative to the recommended commands, you can use the following procedures.

- Creating a Verilog Netlist from a DEF File on page 88
- Reconciling the Object Names and Creating New DEF File That Can Be Used With the Normal Encounter Flows on page 88

Creating a Verilog Netlist from a DEF File

The following procedure creates a Verilog netlist from a DEF file, which must contain connectivity information.

1. Start Encounter

   ```
   encounter
   ```

2. Load the library information.

   ```
   loadLefFile example.lef
   ```

3. Load the DEF file that contains connectivity information.

   ```
   loadDefNetlist example.def
   ```

4. Write the Verilog file created by the `loadDefNetlist` command.

   ```
   saveNetlist output.v
   ```

Important

You cannot continue to use Encounter with the netlist you have created. You must exit at this point, then reconcile the object names.

5. Exit the current session.

   ```
   exit
   ```

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 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*.

To begin a design, complete the following steps:

1. Select *Design – Design Import.*
2. Select the *Design* tab if it is not already selected.
3. Enter the netlist type. Choose one of the following options:
   - *Verilog*

     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.

- *OA Database*

  In this case, the software reads the netlist information from the OpenAccess database you specify.

4. (Optional) Specify RTL or ILM files.
5. Specify technology and physical information, either in LEF or in OpenAccess format.
   - If you have LEF files, enter the names of the files. 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.
   - If you have OpenAccess data, specify the following information:
     - OpenAccess reference libraries
     - Abstract view names
     - Layout view names
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.

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]
```

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

```
commitConfig
```

Loading Configuration Files from the GUI

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

1. Select *Design – Design Import*.
2. Select the *Design* 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* page 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. From the *Design* page of the Design Import form, click the browse (...) button next to the text field of the file type you are interested in.

   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:

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 95
- Saving an OpenAccess Design on page 95
- Restoring an OpenAccess Design on page 95
- Transferring OpenAccess Data between Encounter and Virtuoso Chip Editor for ECO on page 96
- Importing OpenAccess Physical Design Data Only on page 96

Importing an OpenAccess Design

For information on importing OpenAccess designs, see "Beginning Designs" on page 90.

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 – Design form.

Then, after you run *Design – Restore OA Design* in a new session, the Encounter software restores the design state to the same as state it was in when you used *Design – Save OA Design* 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 – Design Import*. Specify the information needed in the *OA Database Library/Cell/View* fields.
- After you have imported the design, if the OpenAccess database contains physical information, select *Load – OA Database*.
- To restore a design that you saved previously by using *Save OA Design*, select *Restore OA Design*.

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.

   Note: Later, when you start a new Encounter session, you must use `oaCopyRestoreFiles` if you use *Save As* when you save the design in VCE.

5. Exit the VCE tool.

6. Start an Encounter session.

7. (Optional) Copy Encounter tool-specific files from the old design to the new design.

   `oaCopyRestoreFiles` *`fromLib fromCell fromView toLib toCell toView`*

   You only need to do this if *`fromLib fromCell fromView`* differs from *`toLib toCell toView`*.

8. Restore the OpenAccess design.

   `restoreOaDesign` *`lib cell view`*

Importing OpenAccess Physical Design Data Only

Before you load an OpenAccess database, you must run the `lef2oa` command if you have LEF files.

After you run the `lef2oa` command, you can use the *Design – Load – OA Database* form or the `oaIn` command to restore floorplan, placement, and routing information in the OpenAccess format.

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 97
- Restoring Designs on page 98
- Saving and Restoring OpenAccess Designs on page 98
- Restoring Designs from Pre-3.2 Versions of Encounter on page 98

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

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.

Important

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

Saving and Restoring OpenAccess Designs

For information on saving and restoring OpenAccess designs, see "Working with OpenAccess Designs" on page 95

Restoring Designs from Pre-3.2 Versions of Encounter

If your current software release is SoC 3.2 or later, restoring a routing file can cause errors if the original data was saved using Encounter software versions earlier than 3.2.

In these earlier versions, settings for `setDoAssign` and `setImportMode -keepEmptyModule` are not saved in the configuration file.

To restore a route file using the same `setDoAssign` and `setImportMode -keepEmptyModule` settings used in the previous session, complete the following steps:

1. Start Encounter.

   ```
   encounter
   ```

2. Load the configuration file.

   ```
   loadConfig fileName 0
   ```

3. Specify `setDoAssign` and/or `setImportMode` settings.

   ```
   setDoAssign [same value as in previous session]
   setImportMode -keepEmptyModule [same value as in previous session]
   ```

4. Apply the settings and import the design.

   ```
   commitConfig
   ```

You can then restore the route file successfully.

To restore a design and change the `setDoAssign` and/or `setImportMode` settings, complete the following steps:

1. Write routing and other information from the existing design to a DEF file.

   ```
   defOut -routing [other options...] defFileName
   ```

2. Exit encounter or free the design.

   ```
   exit
   ```

3. Restart Encounter to begin a new session.

   ```
   encounter
   ```

4. Load the configuration file.

   ```
   loadConfig fileName 0
   ```

5. Specify new `setDoAssign` and/or `setImportMode` settings.

   ```
   setDoAssign new_value
   setImportMode -keepEmptyModule new_value
   ```

6. Apply the settings and import the design.

   ```
   commitConfig
   ```

7. Read the DEF file.

   ```
   defIn defFileName
   ```

The routing information is restored.

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.

Important

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

Important

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.

Important

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 via1 2
VIA via2 3
VIA via3 4
VIA via4 5
VIARULE via1_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*
- For OASIS format:
  - `setOasisOutMode`
  - `oasisOut`

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.

For more information, see "Merging GDSII Stream or OASIS Files" on page 107.

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 ON`
- `setOasisOutMode -SEVianames ON`

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

`topSructureName``_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 107.

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:

- 2contains 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 109.

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 uniquification, 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 uniquification, 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 Files Using the GUI

Use the GDS Export form.

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

For more information, see Save – GDS 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.

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#
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

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

`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 115.

`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`.

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 addCustomText and addCustomBox.
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 "Fill Subtype" on page 116.
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 "Fill Subtype" on page 116. **Note:** 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 streamOut or oasisOut.)
LEFPIN	Equivalent to LEF PIN. (Applies only when the -outPutMacros parameter is specified. For more information, see streamOut or oasisOut.)
NET	Equivalent to DEF NETS wiring. For more information, see "Net Name Subtype" on page 116.
PIN	Equivalent to DEF PINS.

	`SPNET`	Equivalent to DEF `SPECIALNETS` without `+ SHAPE FILLWIRE` or `+ SHAPE FILLWIREOPC`. For more information, see "Net Name Subtype" on page 116.
	`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 "Fill Subtype" on page 116.
	`VIAFILLOPC`	You can separate `VIAFILLOPC` into floating and connected fill by specifying the `FLOATING` subtype. For more information, see "Fill Subtype" on page 116. **Note:** DEF 5.6 does not support this object type."Fill Syntax" on page 107
`layerNumber`	Specifies the GDSII Stream/OASIS layer number or numbers. The number must be an integer between 1 and 65535.	
`dataType`	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.

Important

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 116.

- Net names

  For more information, see "Net Name Subtype" on page 116.

- Voltage levels

  For more information, see "Voltage Subtype" on page 117.

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
```

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

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

4

Flip Chip Methodologies

- Overview on page 122
- Flip Chip Flow in Encounter on page 125
- APD Bump Flow on page 130
- Area I/O Flow on page 138
- Peripheral I/O Flow on page 141
- Differentiating Area I/O and Peripheral I/O on page 155
- Swapping Signals on page 156
- Creating Differential Routing to Signal Bumps on page 158
- Examples and Report Files on page 162

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.

Bump Array

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 620 (APD) is a related packaging tool that interfaces with flip chip. You must have a separate license to run APD. The documentation for APD is provided in the *Cadence Allegro® Package Designer 620 User Guide*.

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 125.
- For information on a specific type of flow, see one of the following sections:
  - APD Bump Flow on page 130
  - Area I/O Flow on page 138
  - Peripheral I/O Flow on page 141
- For a list of tasks contained within the flows, see "Useful Tasks" on page 123.

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 FlipChip menu forms, see the “FlipChip Menu” chapter of the *Encounter Menu Reference*.

Before You Begin

Before using flip chip, the following information is required:

- Parameter data for:
  - Bumps
  - I/O drivers
- APD interface files

Note: You must have a license for APD.

Results

Once all parameters have been assigned, a routing feasibility test is performed using the APD interface with Encounter. A report is generated that determines whether the routing package is feasible. For more information, see Testing the Package Routing Feasibility on page 137.

Useful Tasks

The following list of tasks provides links to procedures embedded in the flows which are described in this chapter:

- Assign signals to bumps. on page 132
- Assign power and ground to bumps. on page 135
- Connect the power and ground bumps. on page 128
- Do a route feasibility analysis. on page 135 (Bump flow)
- Edit the bumps. on page 127

- Place driver cells and assign bumps. on page 128
- Swapping Signals on page 156
- Set up the I/O driver rows. on page 131

Flip Chip Flow in Encounter

The following figure shows the general Encounter flip chip flow including sub flows. For information on the APD tool, see the *Cadence Chip I/O Planner User Guide*.

[1] License Required

[2] Bypasses *FlipChip* menu (see Reducing Data Size for APD Import (Bypass Flow) on page 136)

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 APD 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 155.

     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 164.

       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 "Bump Flow Steps" on page 130.

     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 6.
     - 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.
     - Define Tiles
       Define a tile (CLASS COVER BUMP) that contains pins representing bumps.
     - 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 and menu commands to edit bumps:

   - Unassign Bump
   - Swap Signals

4. Check the routing feasibility using APD.

   The APD packaging tool interfaces to Encounter via LEF/DEF files. Once the bumps are assigned, use the `chipioRoutability` command to send the bump matrix to APD to test the routing feasibility of the bumps to the package. APD will return a report and Encounter will highlight any bumps that cannot be routed to the package. Two APD files are required:

   - `*.cio`
   - `*.ldf`

   For more information, see the *Cadence Chip I/O Planner User Guide*.

5. Add stripes.

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

6. Place driver cells and assign bumps.

   Use the `placeAIO -assignBump` command and option to place the I/O driver cells into the rows/sites closest to the corresponding bumps. If the bumps are not assigned at this time, this command can also be used to assign the bumps and also place all of the standard cells, if requested.

7. Connect the power and ground bumps.

   Use the `fcroute` command to connect the power and ground bumps to the stripes, and connect the signal bumps to the I/O driver cell specified in the netlist.

   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.

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

8. Partition the design.

   Bumps, bump routing, power routing, and I/O driver cells can be pushed down as blockages into the partition.

   *Text Command*s:

   - `specifyPartition`
   - `handlePtnAreaIo`

9. Place the design.

   Place the design using the `placeAIO` command.

10. Route the design.

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

11. Verify the connectivity

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

12. Run extraction.

    Extract the RC data using the `runQRC` command or the `extractRC` command and then generate a SPEF file. The `runQX` 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 `runQX` 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.

*Text Command*s:

- `runQRC`
- `saveNetlist -phys -includeBumpCell`

13. Do a timing analysis.

The timing analysis report is the same as in the normal Encounter flow.

Text Command:

- `report_timing`

14. Update the power.

Update the power. In a flip chip design there will be multiple power sources.

*Text Command*s:

- `autoFetchDCSources`
- `savePadLocation`
- `updatePower`

15. 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 APD Import (Bypass Flow)" on page 136.

*Text Command*s:

- `defOut -noCoreCells`
- `saveNetlist`
- `oaOut`
- `rcOut -spef`
- `streamOut`.

APD Bump Flow

The following figure shows the bump flow, which is a sub flow of the Flip Chip Flow in Encounter on page 125.

Bump Flow Steps

1. Import the design.

   Load the Verilog netlist and LEF files into Encounter. For more information about importing a design, see the Design Import form in the *Encounter Menu Reference*.

2. Set up the bumps.

Use the *Setup Chip IO (Bump Array)* flip chip form to set up the bump array. The following figure shows the use of this form and the resulting bump array.

3. Set up the I/O driver rows.

Use the *Setup Chip IO (IO Driver Row)* flip chip form to set up the driver rows. Figures 5-1 and 5-2 show the use of this form and the resulting driver rows.

The base unit for *Number of Sites* is determined by the value of *Total* in the *Site to be Created* panel. This information comes from the LEF file. Thus, selecting 2 for *Number*

of Sites creates a site twice as large as the base unit. In the figure, the *Number of Sites* for the lower box is set to 2 while the value for the upper box is set to 1.

4. Assign signals to bumps.

Use the Assign Signals flip chip form to assign the signals to the bumps. The following figures show the use of this form:

- "Before Signal Assignment" on page 133
- "After Signal Assignment" on page 134

The selection of the *In Select Set* allows you to select a particular bump (outlined box) for signal assignment. Bumps with signals assigned to them change to blue-filled squares.

Note: You can also select *Closest* to automatically assign nets to the closest bump.

Figure 4-1 Before Signal Assignment

Figure 4-2 After Signal Assignment

5. Assign power and ground to bumps.

Use the Assign Power/Ground Bumps flip chip form to assign power and ground to bumps. Power bumps are red-filled squares. Ground bumps are yellow-filled squares. The following figure shows the use of this form to set ground bumps.

6. Do a route feasibility analysis.

Use the `chipioRoutability` command to test the routability of the bumps. APD creates two files from this analysis:

- `route_feasibility.rpt`

  The Encounter GUI highlights the non-routable bumps contained in the report file with yellow squares. See "Route Feasibility Report" on page 166 for an example.

- `route_feasibility.cio`

  A package binary file created by APD.

If there are problems with routing, you can make adjustments to your design and begin this flow again. Otherwise, the routing is feasible, and you can proceed with the design.

Note: APD requires a separate license.

Reducing Data Size for APD Import (Bypass Flow)

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

```
defOut -noCoreCells
```

This flow bypasses the bump flow (see Flip Chip Flow in Encounter on page 125).

Important

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

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.

Note: A separate license is required for APD.

Use the `chipioRoutability` command to test the routing feasibility.

Two APD input files are required:

- `*.cio`
- `*.ldf`

For more information, see the *Cadence Chip I/O Planner User Guide*.

Two files are created via APD:

- `route_feasibility.rpt`
  This is an ASCII report file which describes non-routable bumps. For an example report file, see Route Feasibility Report on page 166.
- `route_feasibility.cio`
  This is the routing feasibility file which interfaces with (is loaded into) APD.

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 *Setup Chip IO (Bump Array)* form to set up the bump array.

3. 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.

4. Create I/O driver rows.

   Use the *Setup Chip IO (IO Driver Row)* form to set up the driver rows.

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

   Use the Place Area I/O form to place I/O driver cells.

6. Check the route feasibility to ensure that the bumps can be routed to the package.

   Use the `chipioRoutability` command to test the routability of the bumps.

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. Route I/O cells.

   Use the Route Flip Chip Signal form to connect I/O cells to the bumps.

9. Route power to the bumps.

Use the Route Flip Chip Power form to connect the power stripes and rings to the bumps.

Note: The remainder of this flow is similar to the typical Encounter flow.

10. Route signal nets.

Use the NanoRoute form to route the signal and clock nets in the design.

11. Write DEF.

Use the Save DEF form to write out the DEF information.

Note: You only need to perform this step if you plan to use runQX to extract parasitics in the next step.

12. Extract parasitics.

Use the Extract RC form to extract parasitics.

13. Analyze timing.

Use the Timing Analysis form to analyze timing.

14. Perform power analysis.

Use the Power Analysis Statistical Mode form to estimate power consumption.

15. Add metal fill.

Use the Add Metal Fill form to add metal so that the design meets metal density requirements.

16. Verify the design.

- ❑ Use the Verify Connectivity form to verify that there are no open nets or loops in the design.
- ❑ Use the Verify Metal Density form to verify that metal density is within acceptable limits.
- ❑ Use the Verify Geometry form to verify that the physical layout of the design is within acceptable limits.

17. Write out the design data.

You can write the design in one of the following formats:

- ❑ Use the Save OA Design form to save the data in OpenAccess format.
- ❑ Use the Save DEF form to save the DEF information to a file.

- Use the GDS Export form to create a GDSII Stream file version of the database.

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.

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, but does not require 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 bumps to power 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, there are two methods to define the connection of the I/O pads to bumps.

- It can be automatically be done with the command:
  `fcroute -connectPowerCellToBump`
- Or, the following property definition can be used.

  ```
  PROPERTYDEFINITIONS
  PIN BUMP_PORT STRING ;
  END PROPERTYDEFINITIONS
  ```

  (Insert the following LEF statement after the `MACRO PIN` statement and before `END` *`pin_name`*.)

  ```
  PROPERTY BUMP_PORT "layer x y" ;
  ```

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 and 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.

The following flow diagram shows the major flow components for implementing an RDL design.

Netlist
SDC
.lib
LEF
Captbl
CDB
Initial Floorplanning
Bump Creation and Assignment
Bump and IO Optimization
RDL Planning and Routing
Package Design and Analysis with APD
LEF and DEF
RDL Routing
RCLG Model
Incremental SDF
Power Planning and Routing
Placement, CTS, IPO, Routing
DEF, Verilog, VCD/TWF, Power Location
GDS
RDL Extraction with extractRC or Assura RCX for signoff
Coupled SPEF
IR-Drop Analysis with Voltage Storm
Timing Analysis
SI Analysis
Instance Voltage
TWF; Incremental SDF
Violation Fixing
Violation Fixing

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 147.

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. (Optional) Check the route feasibility to ensure that the bumps can be routed to the package.

   You must send the floorplan DEF file to APD for RDL routing and route feasibility. Use the `chipioRoutability` command to test the routability of the bumps. The DEF file that is created contains the 45-degree routes.

7. 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 151.

8. 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 150.

9. 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.

10. 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.

11. Route power to the bumps.

Use the Route Flip Chip Power form to connect the power stripes and rings to the bumps.

Note: The remainder of this flow is similar to the typical Encounter flow.

12. Place standard cells.

Use the Place form to complete the placement of the entire design after floorplanning.

13. Route signal nets.

Use the NanoRoute form to route the signal and clock nets in the design.

14. Write DEF.

Use the Save DEF form to write out the DEF information.

Note: You only need to perform this step if you plan to use `runQX` to extract parasitics in the next step. You must run QX if you are using 45-degree routes.

15. Extract parasitics.

- ❑ Use the Extract RC form to extract parasitics if you are not using 45-degree routes.
- ❑ Use Fire &Ice QX to extract parasitics if you are using 45-degree routes. For more information, see the *Fire & Ice QX Gate-Level Extraction* manual.

16. Read SPEF data into the design.

Use the Load Spef File form to read in SPEF information.

17. Perform power analysis.

Use the Power Analysis Statistical Mode form to estimate power consumption.

18. Analyze timing.

Use the Timing Analysis form to analyze timing.

19. Add metal fill.

Use the Add Metal Fill form to add metal so that the design meets metal density requirements.

20. Use the wire editor to finish incomplete routes or to modify routes that have DRC violations.

For information on using the wire editor, see Chapter 13, Editing Wires.

21. Use Virtuoso Chip Editor to perform chip finishing tasks.

For more information, see the *Virtuoso Chip Editor User Guide*.

22. Verify the design.

- ❑ Use the Verify Connectivity form to verify that there are no open nets or loops in the design.
- ❑ Use the Verify Metal Density form to verify that metal density is within acceptable limits.
- ❑ Use the Verify Geometry form to verify that the physical layout of the design is within acceptable limits.

23. Write out the design data.

You can write the design in one of the following formats:

- ❑ Use the Save OA Design form to save the data in OpenAccess format.
- ❑ Use the Save DEF form to save the DEF information to a file.

Use the GDS Export form to create a GDSII Stream file version of the database.

The following diagram shows the peripheral I/O task flow.

Peripheral I/O RDL 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).

From the Encounter user interface, select *Place -> Flip-Chip -> Place Peripheral IO …*

The syntax for the `placePIO` command is

```
placePIO [-rows numOfRows] [-assignBump] [-optIOs] [-maxIOHeight] [-ioFile
fileName] [-rdlConstraintFile fileName] [-noRandomPlacement] [-extraConfig
filename]
```

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.

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 162.

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. From the Encounter user interface, select Route -> FlipChip -> Signal
2. Click on the Advanced tab and select Routing Style: 45-degree routing
3. Select the pio routing style from the form, or use the command `fcroute -style pio.`

   This option calls both the detail and global routers to route the bumps.
4. Set the minimum escape distance from the bump to where the route can proceed at an angle. Use either the Minimum Escape Distance option on the form or use the command `fcroute -minEscapeDistance` *`unit`* to specify the distance.
5. Select the Connect Power Cell to Bump option for routing from a power bump to a power cell. Alternately, you can use the command `fcroute -connectPowerCellToBump`. You can also use a LEF property to specify the connection. See Data Preparation for property details.
6. 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.
7. 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 162.

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.

SoCE RDL Extraction 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_coupled.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_coupled.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

  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 79 in the Data Preparation chapter for more information and and example.

- Peripheral I/O

  CLASS 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.

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.

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 157
- Signals Swapped on page 158

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 Signal form to specify differential routing parameters.

You can create a constraint file to define differential pairs, shield nets, and differential groups. The following information provides the syntax and examples for creating a constraint file.

Specify Routing Nets

Syntax

```
ROUTINGNETS
    { net_name }
END ROUTINGNETS
```

Example

```
ROUTINGNETS
    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 ROUTINGNETS
```

Define Differential Pairs

Syntax

```
DIFFPAIR
    { net1 net2 [SHIELDNET net3] }
END DIFFPAIR
```

Example

```
DIFFPAIR
    out[10] out[11] [SHIELDNET VDD]
    out[14] out[19]
    out[8] out[9]
END DIFFPAIR
```

Define Differential Groups

Syntax

```
DIFFGROUP group1
    out[15] out[16] out[17] out[18] resetn
END DIFFGROUP
```

Example

```
DIFFGROUP [group_name]
    { net }
END DIFFGROUP
DIFFGROUP group2
    clk out[12] out[12] out[13]
END DIFFGROUP
```

Define a Shield Net

Syntax 1

```
netname1 SHIELDNET netname2
```

Example 1

```
out[3] SHIELDNET VSS
```

Syntax 2

```
SHIELDNETS shield_net_name
    { net }
END SHIELDNETS
```

Example 2

```
SHIELDNETS vdd
    in1 net2
    out1 out2
END SHIELDNETS
```

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
    TOPLAYER 7
    BOTTOMLAYER 7
## Net Definition ##
    VDDPST                #apply two nets only
    VSSPST
END NETS
NETS
    WIDTH 20.0
    TOPLAYER 8
    BOTTOMLAYER 7
## Net Definition ##
    ~VDDPST                #negation - for all other nets
    ~VSSPST                #negation
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.

```
************************************************************************
Routing constraints: fcroute -designStyle aio | pio
************************************************************************
#One constraint file is used for fcroute and placePIO
VERSION 2
    WIDTH 10              ; global constraint
    MAXLENGTH 800         ; 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
    MAXLENGTH 1000         ; 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 DIFFGROUP results maybe different in -designStyle aio | pio
DIFFPAIR
    out[10] out[11] SHIELDNET VDD
    out[14] out[19]
END DIFFPAIR
DIFFGROUP
    out[15] out[16] out[17] out[18] resetn
END DIFFGROUP
```

```
#SHIELDING only works with fcroute -designStyle aio
SHIELDING
    SHIELDNET VDD (width spacing)
        in1  net2
        out1 out2
END SHIELDING
#NETS or ROUTINGNETS only works with fcroute -designStyle pio
NETS
    WIDTH 24.0
    TOPLAYER 8
    BOTTOMLAYER 7
    VDDPT                 # apply two nets only
    VSSPT
END NETS
NETS
    WIDTH 20.0
    TOPLAYER 8
    BOTTOMLAYER 7
    ~VDDPT              # negation - for all other nets.
    ~VSSPT              # negation
END NETS
**********************************************************************
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
GROUP
    pad_name_list
END GROUP
FIXBUMP
```

```
    net1 net2 net3
END FIXBUMP
```

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
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 **H**orizontal or a **V**ertical 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.

For more information, see the `addAIORow` command in the “Flip Chip Commands” chapter of the *Encounter Text Command Reference*.

LEF and CML Example Files

The following file examples are for the `chipioRoutability` command, which is used to test routing feasibility.

LEF File Example

This file points to the LEF file locations.

```
DEFINE abccdr
  LEFFILE ../lef/atechfile.lef
  LEFFILE ../lef/abumpfile.lef
  LEFFILE ../lef/xyzcdr.lef
```

CML File Example

The CML file is required for APD. It contains information similar to that found in the LEF file.

```
VERSION 5.20
NAMESCASESENSITIVE OFF
```

```
MACRO BUMPP
  MACROCLASSNAME COVER BUMP
  EXTENTS (102.00,102.00) DEFINESDRIVER
  PIN PAD UNSPEC PAD ORIGIN(51.00,51.00) DIM(102.00,102.00)
END BUMPP

MACRO BUMPG
  MACROCLASSNAME COVER BUMP
  EXTENTS (102.00,102.00) DEFINESDRIVER
  PIN PAD UNSPEC PAD ORIGIN(51.00,51.00) DIM(102.00,102.00)
END BUMPG

MACRO BUMPCELL
  MACROCLASSNAME COVER BUMP
  EXTENTS (102.00,102.00) DEFINESDRIVER
  PIN PAD UNSPEC PAD ORIGIN(51.00,51.00) DIM(102.00,102.00)
END BUMPCELL
```

Route Feasibility Report

The `route_feasibility.rpt` file is an ASCII text file which is created with the `chipioRoutability` command.

The following sample is a partial report file showing routing errors. A graphical view of this report is shown in the following figure. In the figure, non-routable bumps have yellow squares surrounding them which correspond to the report file.

```
#################
# Route Results #
#################

20

ABC1CK "Fail: Net ABC1CK not scheduled due tounresolvable cross near x 14605. y -6238."
SPARE3 "Fail: Net SPARE3 not scheduled due tounresolvable cross near x 13335. y -4464.889"
QRATE_R "Fail: Net QRATE_R not scheduled due to overloaded channel near x 4416.804 y -1258.1"
XYZOUT_X "Fail: Net XYZOUT_X not scheduled due to overloaded channel near x 3192.657 y
  -3900."
XYZOUT_Y "Fail: Net XYZOUT_Y not scheduled due to overloaded channel near x 2540.726 y
  -3900."
LD_4 "Fail: Net LD_4 not scheduled due to overloaded channel near x 132.931 y -3900."
LD_5 "Fail: Net LD_5 not scheduled due to overloaded channel near x -851.278 y -3900."
CNTRTP "Fail: Net CNTRTP not scheduled due to overloaded channel near x -4400. y -3498.382"

ETOGGLE_R "Fail: Net ETOGGLE_R not scheduled due to overloaded channel near x -4400. y 1401.133"
FIX10 "Fail: Net FIX10 not scheduled due to overloaded channel near x -4400. y 1812.881"
FORCE_1 "Fail: Net FORCE_1 not scheduled due to overloaded channel near x -4400. y 2208.791"

HDIN_XN "Fail: Net HDIN_XN not scheduled due to overloaded channel near x -2533.542 y 3700."

HDIN_VN1 "Fail: Net HDIN_VN1 not scheduled due to overloaded channel near x -2282.605 y 3700."

HDOUT_VN2 "Fail: Net HDOUT_VN2 not scheduled due to overloaded channel near x 1961.415 y 3700."
HDOUT_VN3 "Fail: Net HDOUT_VN3 not scheduled due to overloaded channel near x 2319.069 y 3700."
HDOUT_VN4 "Fail: Net HDOUT_VN4 not scheduled due to overloaded channel near x 3034.379 y 3700."
MRESET_R "Fail: Net MRESET_R not scheduled due to overloaded channel near x 4434.049 y 1453.288"
TSTLVDS_VN "Fail: Net TSTLVDS_VN not scheduled due to overloaded channel near x -359.91 y
  -3900."
TSTMUX_UX1 "Fail: Net TSTMUX_UX1 not scheduled due to overloaded channel near x -4400. y
  -1850.419"
TSTMUX_VX "Fail: Net TSTMUX_VX not scheduled due to overloaded channel near x -4400. y
  -1377.986"
TSTPECL_RN "Fail: Net TSTPECL_RN not scheduled due to overloaded channel near x -4400. y
  -330.36"
SYSCLK_RP "Fail: Net SYSCLK_Z not scheduled due to overloaded channel near x -1676.35 y 3700."
```

FlipChip
Partition
Floorplan
Place
Clock
Route
Timing
SI
Power
Verify
Tools
Help
Design is: In Memory
All Colors
V S
Module
Black Box
Fence
Guide
Obstruct
Region
Screen
Instance
Net
PG Net
Pin
Ruler
VCongest
HCongest
Text
21(3145.851, 2011.714)
Non-routable bumps
(yellow squares)

5

Using Interface Logic Models in Hierarchical Designs

- Overview on page 170
- Creating ILMs on page 171
  - Example ILM Creation on page 172
- Specifying ILM Directories at the Top Level on page 173
  - Example Top-Level Design Flow with ILMs on page 173
- Preserving Selected Instances in ILMs on page 174
- Creating ILMs for Shared Modules on page 175

Overview

Models are compact and accurate representations of timing characteristics of a block. An Interface Logic Model (ILM) is a structural representation of a model. (ILMs are not abstracted timing models.)

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:

- Uses the original constraint file for top-level analysis
  - No abstraction for timing exceptions
- No need to characterize blocks
  - Works on a actual design data
- Allows you to modify only top-level data
  - Fully preserves implemented partitions
- More accurate analysis than a black box flow
- Can be used in the initial prototyping stage for very big designs. when loading full design data is not feasible.

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.

Important

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 circuitry leading from the I/O ports to interface registers (edge-triggered registers), and from interface registers 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
  - Intenal paths controlled by different clock, or clocks connected to the ILM module through different ports (interclaock paths)/
  - Logic between the internal registers

  If the logic between the I/O ports is pure combinational, it is preserved in an ILM. A transparent latch is treated as a combinational device, and path tracing continues through the latch.

- ILM data for SI

  The model includes all of the above, plus nets that are affected by SI effects.

- ILM data for CTS

  The model includes all flip-flops and clock net path information.

Example ILM Creation

The following method creates a model that can be used in the top-level design flow by both `timeDesign` and `optDesign` for both setup and hold efforts:

```
createInterfaceLogic -hold -dir dirName
```

Specifying ILM Directories at the Top Level

Use `specifyILM` to specify ILM directories at the top level. 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 only if the ILMs are unflattened (`unflattenIlm`) or the design is in black box view. You cannot change ILM settings in flattened or ILM view.

Use `unSpecifyILM` to unspecify the ILM directory for the block.

- The following form enables you to specify and unspecify ILM directories:
  - *Design Import – Advanced – Specify ILM*

Example Top-Level Design 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, the `partition` command automatically creates a Verilog file at the top level that includes dummy modules for the blocks in the design. You do not need to change the Verilog file.

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, `.lib` (optional), and `.lef` for the blocks, top-level constraints, and chip-level constraints.

```
loadConfig fileName
specifyIlm -cell block_A -dir $blocks_ilm
specifyIlm -cell block_B -dir $blocks_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.

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
   ```

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
    ```

Preserving Selected Instances in ILMs

You can preserve all selected instances and nets when you create ILMs. Selected instances are those instances that drive the selected nets. Use the `-keepSelected` and `-noKeepSelected` parameters to `createInterfaceLogic`.

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 only the enabled parts of a module 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.

6

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*.

Important

The what-if timing analysis commands do not support the Multi-Mode Multi-Corner (MMMC) feature.

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 6-1 shows the intrinsic timing model.

Figure 6-1 Intrinsic Timing Model

The data types associated with the numbers in the Figure 6-1 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	`setWhatIfCombDelay`
2	Delay from the clock input port to the data input port	`setWhatIfTimingCheck`
3	Sequential delay from the clock input port to the input of the driver	`setWhatIfSeqDelay`
4	Type of Driver	`setWhatIfDriveType`
5	Driver input slew	
7	Clock insertion delay to internal registers	`setWhatIfClockLatency`

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 6-2 shows the normalized timing model.

Figure 6-2 Normalized Timing Model

The data types associated with the numbers in Figure 6-2, 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	`setWhatIfCombDelay`
2	Delay from the clock input port to the data input port	`setWhatIfTimingCheck`
3	Sequential delay from the clock input port to the data output port. It includes the driver delay	`setWhatIfSeqDelay`
4	Driver type	`setWhatIfDriveType`
5	Driver input slew	
6	Total driver output net capacitance	
7	Clock insertion delay to internal registers	`setWhatIfClockLatency`

A normalized timing model uses the following formula for timing arcs ending on output ports:

`Delay = constant delay - driver delay* + driver delay (look-up table)`

Where,

`constant delay` = Timing arc delay including driver delay

`driver delay*` = Constant delay considering an input slew and an output capacitance

`constant delay - clock latency` must be greater than *`driver delay*`*

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`

- 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`

■ 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`

7

Partitioning the Design

- Overview on page 186
- Flow Methodologies on page 186
- Specifying Partitions and Blackboxes on page 195
- Inserting Feedthroughs on page 244
- Generating the Wire Crossing Report on page 261
- Estimating the Routing Channel Width on page 264
- Running the Partition Program on page 266
- Restoring the Top-Level Floorplan with Partition Data on page 285
- Concatenating Netlist Files of a Partitioned Design on page 286
- Saving Partitions on page 287
- Loading Partitions on page 287
- Working with OpenAccess Database on page 289
- Parallel Job Processing on page 290

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. Designs 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 187
- Bottom-up Methodology on page 192

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 200.

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 placement, and 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 195.

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 244.

   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 design that have multiple partitions, are congested, and/or have thin channels, use the `ptnAwareRouteForPA` command. 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 203.

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 `dtmf` 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 `dtmf_chip`
- `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.

The syntax of the command is as follows:

```
assembleDesign
     {topDesignDir blockDesign
          [{-topFP | -chipFP} FPFile]
          [-fe]} |
     {-topDesign {oaTopLib oaTopCell oaTopView}
          {-block {oaBlockLib oaBlockCell oaBlockView}}...}
     [-timingCon timingConFile]
     [-row]
```

where:

- *topDesignDir* specifies the path to the top-level design created by saveDesign -def. This design directory should contain the Verilog netlist and the DEF files.

  The LEF abstract files of the block-level designs that are referenced in the topDesignDir configuration file must have the same names as the block-level design names. For example, if the design name for a block-level design is tdsp_core, the corresponding LEF abstract file must have the name tdsp_core.lef.

- *blockDesign* specifies the path to block-level design(s). An entry is required for each partition or block to be assembled.

  This directory should be generated by the saveDesign -def command. If it is not, make sure that the DEF (.def) file and the Verilog netlist (.v) file have the same name as the design name (module netlist name). For example, if the design name for the tdsp partition is tdsp_core, the DEF file and the Verilog file must have the names tdsp_core.def and tdsp_core.v respectively.

- *topFPFile* specifies the chip-level floorplan file. By default, the command reads the floorplan file from the provided top-level design. This option is mandatory for the top-down hierarchical flow.

- -topDesign {*oaTopLib oaTopCell oaTopView*} specifies the top-level library, cell, and view names respectively for a design that was saved into an OpenAccess database with the savePartition command. This parameter is used when bringing back block data for the partition from an OpenAccess database.

- -block *oaBlockLib oaBlockCell oaBlockView* specifies the block-level library, cell, and view names respectively for a design that was saved into an OpenAccess database. This parameter is used when bringing back block data for the partition from an OpenAccess database.

- *timingConFile* specifies the timing constraints file.

- -fe specifies that Encounter data should be used. By default, data from DEF files is used.

- -row specifies that the command should bring back row information.

For this example, you would run the assembleDesign command as follows:

```
assembleDesign PTN/dtmf_chip/dtmf_chip.enc.dat PTN/arb/arb.enc.dat 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.

Important

For the `assembleDesign` command to work correctly with X designs, the routing—from both inside and outside—must be completely over the partition pins.

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 192
- Top-level Implementation on page 194

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 module.
3. Left click and drag over the area.
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. The I/O constraint file format currently does not support rectilinear block-level designs.

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 `-noassignIoPins` 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 189, 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 196
- Defining Partitions as Power Domains on page 198
- Defining Blackboxes on page 198
- Saving Blackboxes on page 200
- Handling of Blackboxes with Non-R0 Orientation on page 201
- Specifying Multiple Instantiated Partitions and Blackboxes on page 203
- Changing Partition Clone Orientation on page 204
- Specifying Rectilinear Partitions and Blackboxes on page 205
- Specifying Core-to-I/O Distance for Partition Cuts on page 206
- Specifying Nested Partitions on page 207
- Assigning Pins on page 208

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 203.

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.

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 196).

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:

- Specify a module or hard macro as blackbox after the design import step, using the `specifyBlackBox` command or the *Partition – Specify Black Box* menu command.

  Note: Converting a hard macro into a blackbox will not update the blockage definitions when you change the blackbox size.
- Use the `setImportMode -treatUndefinedCellAsBbox true -keepEmptyModule false` command before importing a design to automatically convert all undefined modules or cells into blackboxes. For this method, the initial

blackbox size is zero. After reading a netlist into the Encounter software, use the `specifyBlackBox` text command to change a 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.

Caution

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 constrains.

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 201
- Performing R0 Transformation on page 203

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:

```
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 `saveBlackBox` 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 are 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 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.

The following example shows a design that has R90, R180, and R270 orientation clones:

Floorplan View

Physical View after Unpartitioning
(does not show the top-level connection)

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 240 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/cdump) 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 is 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 µm for the top and bottom, and 2 µm for left and right sides:

The core to I/O distance for the edge A (facing east) should be 2 µm. The core to I/O distance for the edge B (faced to north) should be 5 µm, 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
- Repeated partitions and black boxes. Both master and clones are considered when assigning their pins.
- Non-uniform tracks.
- X-routing. For X-pin assignment, the pins are aligned either horizontally or vertically but not diagonally.

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.

Important

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 210
- Assigning I/O Pins
- Assigning Clock Pins
- Performing Congestion-aware Pin Assignment for Channel-based Designs
- Assigning Pins on Rectilinear Edges
- Swapping Partition Pins

- Snapping Pins to the Grid
- Pin Assignment Limitations

Assigning Partition and Blackbox Pins

Assigning pins for partitions and blackboxes includes the following steps:

- Setting Pin Constraints on page 210
- Assigning Pins on page 219
- Validating Pin Placement Results on page 223
- Refining Pin Assignment and Fixing Pin Violations on page 226
- ECO Pin Assignment on page 229

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 group.

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
```

As the `-layer` option is *not* specified, the pin guide will be created on all the reserved layers of this partition or blackbox of cell `ALU`. For example, if the reserved layers for this partition are M1, M2, M3, M4, M5, and M6, the pin guide will be created on M2, M3, M4, M5, and M6 layers. However, pins will only be assigned on the preferred routing layers. As the specified area overlaps with the left edge of the partition, the INT[]0], INT[2], and INT[3] pins will be placed on either M3 or M5 layer.

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}
```

```
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 the *Min Space* field in the partition pin guide's Attribute Editor. 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 Attribute Editor, or the `createNetGroup` command, to create a net group name. Use the `addNetToNetGroup` command to assign nets to the net 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 layer width. 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.

```
setLayerPinWdith -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.

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 commands are `setAllowedPinLayers` and `setAllowedPinLayersOnSide`.

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.

- 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 a 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 partition-aware routing (`ptnAwareRouteForPA`) instead of Trial Route. 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 (`ptnAwareRouteForPA` 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:

  ```
  ptnAwareRouteForPA
  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.

Note: To assign clock pins, see Assigning Clock Pins on page 236.

Validating Pin Placement Results

You can perform the following steps to validate and correct pin placement results:

- Checking the Pin Legality on page 223
- Reporting QoR of Pin Assignment on page 224
- Adjusting Pins on page 227
- Aligning Partition Pins on page 227
- Running incremental Pin Assignment on page 228
- Adjusting Floorplan or Floorplanning the Design Again on page 229
- Performing Pin Assignment Again on page 229

Checking the Pin Legality

Use the `checkPinAssigment` 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 `checkPinAssigment` command to print detailed pin-specific information for each reported violation. You cans also exclude certain checks, for example, checks related to pin spacing violation or pin layer violation. For more information, see the description of the `checkPinAssigment` 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.

Important

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* reefers 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.

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:

----------------------------------------------------------------------------------
|                    Partiton |  Total | Internal | Unplaced |  Legal | Illegal |
|                        Name |  Pins  | Pins     | Pins     |  Pins  | Pins    |
----------------------------------------------------------------------------------
|                results_conv |    39  |     0    |    39    |     0  |     0   |
----------------------------------------------------------------------------------
|                   tdsp_core |   114  |     0    |   114    |     0  |     0   |
----------------------------------------------------------------------------------
|                   dtmf_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 |
--------------------------------------------------------------------------------
| Horizontal  |        0.000%         |        0.032%         |         NA       |
| Congestion  |                       |                       |                  |
--------------------------------------------------------------------------------
|  Vertical   |        0.000%         |        0.697%         |         NA       |
| Congestion  |                       |                       |                  |
--------------------------------------------------------------------------------
|    Total    |   6.089e+05um         |   6.150e+05um         |       1.011%     |
| Net-Length  |                       |                       |                  |
--------------------------------------------------------------------------------
|    Total    |         48300         |         54036         |      11.876%     |
| Via-Count   |                       |                       |                  |
--------------------------------------------------------------------------------
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 (`deriveTmingBudget`).
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 232
- Performing Initial Pin Assignment on page 232
- Refining Pin Placement on page 233
- Validating Pin Placement on page 234

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 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 210 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 `loadIoFile` 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: Specify the `-noassignIoPins` option of the `setPlaceMode` command 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 `checkPinAssigment` 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.

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.

Assigning Clock Pins

To improve the Quality of Results (QoR) of clock pin placement, clock pins can be assigned based on analysis done by the clock pin estimator, using the `estimatePartition` command.

The clock tree estimator analyzes and suggests clock pin locations. Suggested clock pin locations are legalized to the nearest valid location while honoring user-specified pin constraints. The status of the pins is changed to *Fixed*.

The clock tree estimator updates clock buffer locations based on the new, legalized, clock pin locations. This step should be performed before legalizing the rest of the partition pins. The following is the general flow for clock pin assignment:

1. Import the design.
2. Floorplan the design.
3. Run the `estimatePartition` command.
4. Optionally run Trial Route.
5. Assign partition pins with the `assignPtnPin` command.
6. Derive timing budget using the `deriveTimingBudget` command.
7. Partition the design using the `partition` command.
8. Save the partition using the `savePartition` command.

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. 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.

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     tdsp_core(DTMF_INST/TDSP_CORE_INST)

2       PtnAware         42(223)    ram_256x16_test(DTMF_INST/
                                    RAM_256x16_TEST_INST)

3       PtnAware         32(223)    G1(DTMF_INST/G1_PH

4       PtnAware         20(223)    results_conv(DTMF_INST/RESULTS_CONV_INST)
                                    tdsp_core(DTMF_INST/TDSP_CORE_INST)
                                    ram_128x16_test(DTMF_INST/
                                    RAM_128x16_TEST_INST)

5       Normal           18(223)    ram_128x16_test(DTMF_INST/
                                    RAM_128x16_TEST_INST)

6       Normal           15(223)    results_conv(DTMF_INST/RESULTS_CONV_INST)

7       Normal           3(223)     tdsp_core(DTMF_INST/TDSP_CORE_INST)
                                    ram_128x16_test(DTMF_INST/
                                    RAM_128x16_TEST_INST)

8       Normal           2(223)     G1(DTMF_INST/G1_PH)
                                    results_conv(DTMF_INST/RESULTS_CONV_INST)
                                    tdsp_core(DTMF_INST/TDSP_CORE_INST)

9       Normal           1(223)     G1(DTMF_INST/G1_PH)
                                    tdsp_core(DTMF_INST/TDSP_CORE_INST)

10      Normal           1(223)     G1(DTMF_INST/G1_PH)
                                    results_conv(DTMF_INST/RESULTS_CONV_INST)

11      Normal           1(223)     G1(DTMF_INST/G1_PH)
                                    tdsp_core(DTMF_INST/TDSP_CORE_INST)

12      Normal           1(223)     G1(DTMF_INST/G1_PH)
                                    results_conv(DTMF_INST/RESULTS_CONV_INST)
                                    ram_256x16_test(DTMF_INST/
                                    RAM_256x16_TEST_INST)
                                    ram_128x16_test(DTMF_INST/
                                    RAM_128x16_TEST_INST
```

- `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`

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 201.
- 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.

Important

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.

- 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 249.
- 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.

■ `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.

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 261).

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.

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.

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 257.

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.

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.

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 *Metal6* will generate a *Metal6* routing blockage for the partition, and an opening in the *Metal6* 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.

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
```

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
```

Horizontal segment net length
X1 = 15
X2 = 5
X3 = 18

Vertical segment net length
Y1 = 10
Y2 = 12
Y3 = 25

In the above example, the 38 length in the X direction is calculated for the X direction net segments (X1 + X2 + X3), and the 47 in the Y direction is calculated for the Y direction net segments (Y1 + Y2 + Y3).

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

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:

Block1	Block2	Current	Required
bot-boundary	INST1	24.6	28.8
bot-boundary	INST2	54.3	46.9
bot-boundary	HB2	25.0	31.2
lft-boundary	INST1	38.2	45.5
lft-boundary	INST3	43.2	37.8
lft-boundary	HB1	46.8	33.5
INST1	INST3	64.8	39.4
INST1	INST2	72.1	55.7
HB1	top-boundary	57.2	10.9
HB1	BB1	44.5	69.1
INST4	top-boundary	59.5	51.7
INST4	rht-boundary	53.0	50.5
...	...	...	...
...	...	...	...

Partition Hard Macro Blackbox

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.

Important

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.

Important

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.

The Encounter software provides support for X routing. For details on how objects from X routing are pushed down during partitioning, see How Objects Related to X Routing Are Pushed Down on page 282.

You can save the partition data in an OpenAccess database. For more information, see Working with OpenAccess Database on page 289.

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.

Important

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.
  - Intra-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.

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 268
- Behavior with the -stripStayOnTop Option on page 269

The Default Behavior

The following table summarizes the default behavior.

Stripe Position	How Stripe Is Pushed Down
Stripe is completely inside partition boundary	■ Top-level: The stripe is removed from the top. ■ Block-level: The stripe is pushed down as two pins and one stripe. ■ Block Abstract: ❑ On layers reserved for partition, two pins are created on the boundary. ❑ On layers not reserved for partition, one big LEF pin is created.
Stripe is partially inside partition boundary.	■ Top-level: The stripe is retained at the top. ■ Block-level: The stripe is pushed down as two pins and one stripe with +COVER status. ■ Block Abstract: The stripe is pushed down as a big LEF pin.

Stripe Position	How Stripe Is Pushed Down
Stripe is outside but close to partition boundary	■ Top-level: The stripe is retained at the top. ■ Block-level: The stripe is retained at the top and is copied as a routing blockage (same size as stripe) with a +PUSHDOWN attribute. ■ Block-abstract: No effect.

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.

Stripe Position	How Stripes Are Pushed Down
Stripe completely inside partition boundary	■ Top-level: ❑ On layers not reserved for partition, the stripe is retained at the top and is copied to the block-level design. ❑ On layers reserved for partition, the stripe is pushed down to the block-level design. ■ Block-level: ❑ On layers not reserved for partition, the stripe is retained at the top and is copied as two pins and one stripe with +COVER status. ❑ On layers reserved for partition, the stripe is pushed down as two pins and one stripe. ■ Block Abstract: ❑ On layers not reserved for partition, two pins are created at the edges. ❑ On layers reserved for partition except the topmost layer, two pins are created at the edges. ❑ On the topmost layer reserved for partition, one big LEF pin is created.
Stripe is partially inside Partition boundary.	■ Top-level: The stripe is retained on the top and is copied to the block-level design. ■ Block-level: The stripe is retained at the top and is copied as two pins and one stripe with +COVER status. ■ Block Abstract: The stripe is retained at the top and is copied as a big LEF pin.
Stripe is outside but close to boundary	■ Top-level: Stripe is retained at the top. ■ Block-level: Stripe is retained at the top and is copied as a routing blockage (same size as wire) with a +PUSHDOWN attribute. ■ Block-abstract: No effect.

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.

	Top Level	Partition Level
Area I/O cell	Area I/O cell	Placement and Routing Blockage

	Top Level	Partition Level
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

Important

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 268.

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.	■ 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. ■ If the bump overlaps with the partition but is *not* connected to the partition: The signal bump is pushed down as a routing blockage.

Object Type	Top Level	Partition Level
Signal Routes	Signal Routes routed on the top routing layers stays at top.	■ 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. ■ 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.

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

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 How Top-level Stripes Are Pushed Down on page 268.

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.

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.

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.

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

How Objects Related to X Routing Are Pushed Down

The Encounter software supports X routing. The following sections describe how the objects created with X Routing are pushed down.

How LPDRs Are Pushed Down

A Local Preferred Direction Region (LPDR) is treated in a way similar to the floorplan object. Region. It is copied inside the partition and is retained at the top. During chip assembly, LPDRs ar the block-level are discarded.

How Special Nets Are Pushed Down

Special nets for X routing are pushed down as follows

Note: The `-stripStayOnTop` parameter is not supported for X routing.

- Power stripes on layers reserved for the top
- Power stripes on layers reserved for the top level design will be retained at the top and will be copied to the block design with +COVER status.
- Power stripes on layers reserved for the partition will be cut from the top and pushed down to the block design with pins at the boundary.
- Stripes on Top layers and Top Partition layer will also be copied as FAT pin shapes in the block abstract.

The following figures show the details of the pin shape created at the partition boundary

The following diagram shows the pin shape at the partition boundary.

The following diagram shows the view inside the partition.

The following diagram shows the view at the top when the stripe is on the reserved layer.

Signal Routes Handling During Chip Assembly

For X routing, the signal routes will be merged during chip assembly.

Note: The routes should cross the partition boundary orthogonally.

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 286).

Important

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*.

Tip

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 285).

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.

Caution

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 289

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*.

8

Floorplanning the Design

- Overview on page 292
- Common Floorplanning Sequence on page 293
- Viewing the Floorplan on page 294
- Module Constraint Types on page 297
- Creating and Editing Rows on page 305
- Using Vertical Rows on page 305
- Using Multiple-height Rows on page 307
- Using Blackblobs on page 319
- Editing Pins on page 327
- Creating Local Preferred Direction Regions on page 338
- Running Relative Floorplanning on page 340
- Saving and Loading Floorplan Data on page 343
- Adding Logical Hierarchy Without Creating Additional Hierarchy on page 344
- Logical Hierarchy Manipulation on page 344

Overview

The effort required for floorplanning depends on the prototyping level of the design. 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.

Floorplanning usually starts by preplacing 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 Encounter software 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

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.
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 modules and 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.
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

The placer completes the placement of the entire design of all modules and blocks that were not preplaced during floorplanning.

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.

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

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:

Core Size = (standard cell area/core utilization) + (macro area + 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:

Core Size = (standard cell area /standard cell utilization) + (macro area + 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 `setFPlanRowSpacingAndType` 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
```

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.

`createRow`	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.
`cutRow`	Cuts site rows that intersect with the specified area or object.
`deleteRow`	Deletes the specified row(s).
`stretchRows`	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.*

Note: Only symmetrical I/O rows are supported. For example, if the I/O rows on one side are double-height rows, all the I/O rows on the other sides should also be double-height rows.

Using Vertical Rows

Caution

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
```

```
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 305

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 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,

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.

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.

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*.

The following figure shows a full-chip view with the partition halos specified.

The following figure illustrates the result when you use the `flattenPartition` command *without* the `-bringbackRow` parameter

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.

Using 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 black blob 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.
- You can define a timing model for a blackblob immediately after you have created the blackblob—you do not need to floorplan the design first.
- 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 cannot, however, stretch a blob.

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 Black Blob 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 point apply to the specification of blackblobs:

- Blob 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 blob area.
- The shape of the blackblob is automatically determined during blob placement.
- Pins need not be assigned for a blackblob because the external connections or pins are represented by super cell pins.
- Timing for a black blob 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.

Important

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 black blob:

- Area

  Initial blackblob area can be specified in any of the following manner:

  - Total blob 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 account these macro and/or cell areas

- Target utilization

  Similar to a module guide target utilization. This utilization is mainly used for deriving the initial blob area.

- Placement porosity

  The placement porosity specifies the percentage of cell area that will be added to the blob area during placement.This is used by the Encounter placer for cell padding during the placement step.

- Routing porosity

  Specifies the routing resources, per layer, that will be reserved for the blackblob (this porosity is for standard cell area only).

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 complete the virtual connections for them.

The `elaborateBlackBlob` command is run automatically when you restore a design that contains blackblobs.

If you specify blackblobs through the Specify Blackblob GUI form, clicking the Apply button does not run the `elaborateBlackBlob` command, but clicking the OK button does.

You can unspecify a blackblob through the `unspecifyBlackBlob` command.

Blackblob Useflow

This section describes the blackblob useflow for the following scenarios:

- General Black Blob Flow on page 322
- ECO Black Blob Flow on page 323

General Black Blob 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 black blobs.
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 blobs in the floorplan view to guide blob 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 blob shapes.
   - If blob placement porosity information is specified, placer adds cell pads during blob placement to honor user-specified placement utilization.
   - Blobs placement can be overlapping.
   - You run either non-timing-driven or timing-driven placement.
7. if you re-specify a blob specification or manually change location of a blob 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 blob routing porosity information to derive blob routing resources on the fly. Maximum blob porosity value will be used for calculating overlapping blob area(s).

10. Save the design.
11. Optionally run What-If timing analysis.
12. Continue with the normal flow.

ECO Black Blob Flow

The flow when you have updated blob netlists is as follows:

1. Restore a blob design. Or, perform the following steps:
   a. Import 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 blob netlist,
5. If you want to retain the blackblobs as blob objects:
   a. Re-specify the blob 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 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 blob 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-hand side of the design in the floorplan view. By default, blackblob objects are displayed in green as shown here:

Blackblob

You can perform the following actions on the blackblob:

- Select the blackblob
- Move the black blob 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, it is not recommended because the actual area remains the same after resizing. If you want to change the blob area, specify the blackblob again with the new area.

After specifying a blackblob, you need to run placement to place the design and obtain the blob shape in the amoeba and physical views. Black blob 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 blob:

In the floorplan view, the instantiated hard macros and the user-specified hard macros will be highlighted.

In the amoeba and physical view, blob instance will be highlighted.

The following figure shows the amoeba view after placement.

The following figure shows the physical view after placement.

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. Blob 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.

Size Increase Size Decrease

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 329
- 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 µm apart. You want to spread the pins to the right with 3.0 µm 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 µm

Desired spacing: 3.0 µm, 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 µm 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:

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:

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 (*A1*, *A2*, *A3*, and *A4*). 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:

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.

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:

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.

In the following case, the specified side is `bottom`, and the pins are, therefore, spread along the edges of the bottom side.

Pins spread from edge 19 to edge 11
(19, 17, 15, 11)

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)

Creating Local Preferred Direction Regions

Important

Creating Local Preferred Direction Regions is obsolete and will be removed in the next major release of the software.

For designs that support X routing, you can create Local Preferred Direction Regions (LPD regions) to guide the routes in a direction different from the default direction.

The LPD regions are created on diagonal layers, and the regions are assigned either horizontal or vertical routing directions. These LPD regions are snapped to the manufacturing grid.

The Encounter GUI includes an icon you can use to create a new LPD region. Double-click on the icon to invoke the Attribute Editor, where you can edit the LPDR properties.

The following table lists the commands that support LPDR in Encounter.

`autoCreateLPDR`	Automatically creates LPD regions. All areas of the design that have a Manhattan routing style are covered by LPD regions. In the areas of the design that have X routing style, the macros are covered by LPD regions. *Important* The `autoCreateLPDR` command is obsolete. It will continue to be supported in this release, but will be removed in the next major release of the software.
`createLPDR`	Creates an LPD Region for use with X routing and returns an object id that can be used with the `setLPDRLayer` and the `unsetLPDRLayer` commands.You can use an LPD region to guide the routes in a particular direction on the respective layers of the LPD region. You can also set the preferred routing direction for the layers of the LPD region with this command. *Important* The `createLPDR` command is obsolete. It will continue to be supported in this release, but will be removed in the next major release of the software.
`deleteLPDR`	Deletes an LPD region that was created earlier.
`setLPDRLayer`	Sets the preferred routing direction for the layer of an LPD region.
`unsetLPDRLayer`	Removes the preferred routing direction for the layer(s) of an LPD region set with the `setLPDRLayer` command, and resets the preferred routing direction to the default value.

For more information on using these commands, see the *Encounter Text Command Reference*.

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:

Value	Definition	
`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
```

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.

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 345
- Moving Instances to an Existing Module on page 346
- Moving Instances to the Top Root Level on page 347

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
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.

Before Restructuring

After Restructuring

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
```

Note: The leading / (slash) is required for the top root level.

Before Restructuring

After Restructuring

9

Power Planning and Routing

- Overview on page 350
- Before You Begin on page 351
- Results on page 352
- Loading, Saving, and Updating Special Route on page 352
- Creating a Ring with User Defined Coordinates on page 352
- Global Net Connections on page 353
- Connecting Ring Pins on page 355
- Fixing LEF MINIMUMCUT Violations on page 355
- Fixing LEF Minimum Spacing Violations on page 355
- Adding Stripes to Power Domains on page 356
- Automatic Power Planning (APP) on page 358
- Creating a Template on page 359
- Specifying Template Parameters on page 361
- Instantiating a Template on page 362
- Using the Synthesize Power Plan Functionality on page 363
- Creating Differential Routing to Signal Bumps on page 365

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. Connect ring pins using the SRoute form.
8. 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.

Connecting Ring Pins

You can use the SRoute form to connect ring pins to pad rings and to targets outside the block.

The Connect Ring Pins form is primarily designed to route straight from ring pins to the first target outside the block (stripe/ring/design boundary) when the ring pin has many obstructions. Using Connect Ring Pins finishes stripe connections without the jogging that SRoute might add.

Using the Connect Ring Pin form is the preferred method for routing from ring pins to the pad ring. SRoute can do this only if you use an extra configuration file.

Using the Connect Ring Pin form is also the preferred method for routing from the pin of one block to pin of another block if there are no rows in between the blocks. SRoute can do this only if you use an extra configuration 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**
Without template			
–	–	`appD`	`appD_appR`

Input template for APP		Output template from APP	
Design template	**IP template**	**Design template**	**IP template**
With user's template			
uD	–	appD_uD	appD_uD_appR_region*N*
–	uIP	appL_uIP	appL_uIP_appR
With system template			
appD	–	appD	appD_appR
appD_uD	–	appD_uD	appD_uD_appR_region*N*
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 anlysis 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 defintition
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]
```

10

Low Power Design

- Overview on page 368
- Power Domain Shutdown and Scaling on page 368
- Support for the Common Power Format (CPF) on page 370
- Multiple Supply Voltage Flat Flow on page 372
- Multiple Supply Voltage Top-Down Hierarchical Flow on page 393
- Leakage Power Optimization Techniques on page 397
- Power Shutdown Techniques on page 402
- Power Switch Optimization on page 429

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 dynamic power problems 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 402

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

savings, and can provide leakage power savings, depending the 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. Isolation cells 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
- Specifying timing libraries
- Specifying power/ground connections
- Defining operating conditions
- Adding level shifters
- Adding isolation cells

Encounter Commands Supporting CPF

- `loadCPF`
- `commitCPF`
- `saveCPF`
- `saveDesign`
- `runCLP`
- `getCpfUserAttributes`
- `repairPowerDomain`

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`.

Use the following commands to save a CPF database:

- `saveCPF`
- `loadCPF`
- `commitCPF`
- `saveDesign`

CPF Documentation

For more information about CPF, see the following documents:

- *Encounter - Common Power Format Migration Guide for Low Power Commands.*

  Contains side-by-side comparisons between Encounter and CPF commands, and examples to describe how to accomplish design tasks in the low power design flow. The document also provides a table of supported CPF commands and options.

- *Common Power Format User Guide*

  Provides step-by-step instructions on how to create a CPF file for MSV designs.

- *Common Power Format Language Reference*

  Describes CPF commands and provides syntax.

- *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 script in the "CPF Script Example" chapter.

Multiple Supply Voltage Flat Flow

You can perform low power functional verification, SI and static timing analysis, and static and dynamic power analysis throughout the flat low power design flow:

The following section describes the following key steps in the flow:

- Data Preparation on page 373
- Verify Power Domains on page 379
- Load and Commit the CPF File on page 379
- Floorplan the Design on page 380
- Plan Power on page 381
- Place Standard Cells and Macros on page 382
- Synthesize Buffer Trees (Optional) on page 383
- Add Tie High/Low cells on page 384
- Highlight Power Domains (Optional) on page 384
- Power Routing on page 385
- Trial Route on page 386
- Synthesize Clock Trees on page 387
- Route the Design on page 388
- Analyze Timing on page 388
- Optimize Timing on page 388
- Power Analysis on page 391
- Finish the Design Flow on page 392
- Conformal Low-Power Verification on page 392
- Signal Integrity Analysis on page 392

Data Preparation

You must prepare data as follows before you begin an MSV design:

- Preparing the Netlist on page 374
- Preparing Libraries on page 374
- Defining Level Shifter Cells on page 375
- Removing Assign Nets on page 377

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.

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.

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.

Removing Assign Nets

To remove assign statements with a unique buffer for the chip and a different buffer for each power domain, complete the following procedure:

1. Replace assign nets with the specified buffer.

   `setDoAssign on –buffer` *`buf_name`*

2. Load the configuration file.

   `loadConfig`

3. Load the floorplan.

   `loadFPlan`

After you have created the power domains (see Create Power Domains), use the following command to replace the assign buffers for each power domain.

`replaceAssignBuffer` *`powerDomainName replacementBufferName`*

For example, the `setDoAssign` command replaces assign nets from the Verilog sample with three instances of cell `buf1`:

After you create the power domains, these buffers are included in three different power domains: `PD1`, `PD2`, and `PD3`:

You can then replace `buf1` in `PD2` with `buf2`:

You can replace buf1 in `PD3` with `buf3`:

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.

Load and Commit 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

Floorplan 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: PDSPCore 0.7000 0
+14 ss_hvt_0v70_125c(common) ss_g_0v70_124c(common) ...
+p 30.0000 30.0000 30.0000 30.0000 50.0000 50.0000 50.0000 50.0000
+r 3 TSM130NMMETROSITE 2 0.0000 0.0000 0.0000 0.0000
```

```
+m LVLHLX8 +P (vdd_lp_s:VDD_ +G (vss_VSS)
+m DTMF_INST/TDSP_CORE_INST +P (vdd_lp_s:VDD) (vdd_lp:VDDG) +G (vss:VSS)
+i IOPADS_INST/Pvss2 +P - +G (vss:VSS)
END_PD
PowerDomain: PLL 1.6200 1
...
END_PD
```

Plan 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 349 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.

Place Standard Cells and Macros

1. Place standard cells that have not already been placed.

```
setPlaceMode -timingDriven
placeDesign
```

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
```

Synthesize Buffer Trees (Optional)

➤ To synthesize buffer trees, use the following command:

```
bufferTreeSynthesis
```

Add Tie High/Low cells

➤ Add tie high/low cells; for example, for PD1:

```
addTieHiLo -cell TIEONE_PD1 -tiHiPin Z -powerDomain PD1
```

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.

Power Routing

The Encounter software uses the `sroute` command to route the followpin connections and the additional power pins contained in level shifters, just is 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 } -noPadPins -noStripes -noPadRings
-shifterPinConnection { vdd } -noBlockPins -jogControl { preferWithChanges
differentLayer } -nets { vdd_lp vss }
```

```
sroute -powerDomains { DEFAULT } -noPadPins -noStripes -noPadRings -noBlockPins
-shifterPinConnection { vdd_lp } -jogControl { preferWithChanges
differentLayer } -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 or SRPG pins, specify the following command before you run sroute:

`setPGPinUseSignalRoute`

Trial Route

By default, the `trialRoute` command is not MSV-aware, so you must use the following options to `setTrialRouteMode`:

- `-handleEachPD true`
- `-handlePDComplex true`

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

Synthesize 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

Route 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`.

Analyze 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.

Optimize Timing

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.

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.

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.

Power Analysis

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.

Finish the Design Flow

The remainder of the design flow tasks are, for the most part, unchanged from the standard design flow.

1. Add spare cells.
2. Add filler cells; for example:

   ```
   addFill -cell FILL_PD1 … -powerdomain PD1
   addFill -cell FILL_PD2 … -powerdomain PD2
   addFill -cell FILL_PD3 … -powerdomain PD3
   ```

   This command honors rectilinear domain fences and automatically connects power/ground pins to the domain’s power/ground nets.
3. Verify power domains again:

   ```
   verifyPowerDomains
   ```
4. Export the netlist, parasitics, SDF delays, and GDSII.

   To preserve the power/ground pin/net connections when you export the netlist, you must use the following command:

   ```
   saveNetlist -phys
   ```

Conformal Low-Power Verification

➤ Run the Conformal Low-Power Verification tool:

```
runCLP -cpf cpfFile
```

Signal Integrity Analysis

1. Analyze signal integrity:

   ```
   runVStorm
   ```

Multiple Supply Voltage Top-Down Hierarchical Flow

The Encounter tool supports a top-down hierarchical flow based on CPF, which can handle power domains inside a partition.

In this flow, the tool pushes-down the following information into the block level:

- Power domain definition and its power/ground connections
- Power mode, nominal condition, library set, and view definition
- Block-level boundary port power domain marking
- Isolation, level shifter, state retention, and power switch rules
- Special power cell definition such as always-on, isolation and level shifter

This flow generates the top-level CPF by reusing the block-level CPF. The hierarchical top-level CPF and block-level CPF can be assembled without losing the original information.

Use the following steps to implement the top-level hierarchical MSV flow:

Example flow:

- `loadConfig chip.config`
- `loadFPlan chip.fp`
- `loadCPF chip.cpf`
- `commitCPF`
- `set_analysis_view`
- Power planning
- `definePartition ....`
- Placement
- `trialRoute` (with `setTrialRouteMode -handlePD true -handlePartition true`)
- `assignPtnPin`

- `deriveTimingBudget`
- `saveTimingBudget`
- `partition`
- `savePartition`
- Block level flow
  - `loadConfig block.config`
  - `loadFPlan block.fp`
  - `loadCPF block.cpf`
  - `commitCPF`
  - Placement
  - `trialRoute`
  - Power Routing
  - `optDesign -preCTS`
  - `clockDesign`
  - `optDesign -postCTS`
  - `globalDetailRoute`
  - `saveDesign block.enc -def`
- Top level flow
  - `loadConfig top.config`
  - `loadFPlan top.fp`
  - `loadCPF top.cpf`
  - `commitCPF`
  - `trialRoute`
  - Placement
  - Power routing
  - `optDesign -preCTS`
  - `clockDesign`

- ❑ `optDesign -postCTS`
- ❑ `globalDetailRoute`
- ❑ `optDesign -postRoute`
- ❑ `saveDesign toplevel.enc -def`

■ Assemble the design

- ❑ `assembleDesign block.enc top.enc`
- ❑ `loadCPF chip.cpf`
- ❑ `commitCPF`
- ❑ `verifyPowerDomain`
- ❑ `timeDesign`
- ❑ DRC verification commands

This following figure shows power domains PD1, PD2, and the default power domain:

The following chip-level CPF file creates the power information such as power domains, level shifter logics, and so on:

```
chip-level.cpf

set_design top
create_power_domain -name PD2 -instances {dma}
```

```
create_power_domain -name soc -default
create_power_domain -name PD1 -instances {mem}
create_level_shifter_logic -from default -to PD2
...
create_level_shifter_logic -to PD1
...
end_design
```

Partitioning creates the following CPF files:

```
top.cpf
```

```
set_design top
create_power_domain -name PD2 -instances {dma}
create_power_domain -name soc -default
set_instance partition1 -domain_mapping {{default soc}{PD2 PD2}}
source mem_unit.cpf
create_level_shifter_logic -from default -to-PD2
```

```
mem_unit.cpf
```

```
set_design mem_unit
create_power_domain -name PD1 instances {mem}
create_power_domain -name default -default
create_level_shifter_logic -to PD1
create_power_domain -name PD2 -boundary_ports {B}
end_design
```

Note: In `mem_unit.cpf`, `-instances` is ignored because `partition1` is a black box, and `-boundary_ports` becomes a specification for the top level.

Leakage Power Optimization Techniques

- Multi-Vth Optimization on page 397
- Substrate Biasing on page 398

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.

   ```
   reportLeakagePower
   ```

   If you want to obtain a report file, run `reportLeakagePower` 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 -postRoute -highEffort
   ```

   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`.

3. After running the `optLeakagePower` command, you can create a new leakage power report to view results.

   ```
   reportLeakagePower -outfile fileName
   ```

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
  ```

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.

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.

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:

- 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 -ring -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`.

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 PSO1_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 pso1_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.

Important

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 a 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.

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 \
-topDown \
-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
```

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:

```
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 – Power Switch Insertion*, 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 \
```

```
-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
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0}
```

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.

```
addPowerSwitchRing -ring -specifySides {1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0} -startEnableChainAtCorner 4
```

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.

```
addPowerSwitch -ring -specifySides {1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0} -counterclockwise
```

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.

```
addPowerSwitch -ring -specifySides {1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0} -counterclockwise -startEnableCorner 4
```

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
```

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
```

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
```

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
```

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
```

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
```

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`

- `-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:

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.

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:

- `-startOffset*`: Places the first switch 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:

Start Offset	End Offset	Switch Pitch	Applies to
-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 switches abut. The software starts placing switches 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
    ```

- 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 switch placement or 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`

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.

- 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`

- `-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.

The following figures shows the power switch optimization ECO flow with VSDG:

11

Placing the Design

- Overview on page 434
- Preparing for Placement on page 434
- Loading a Design on page 435
- Guiding Placement With Blockages on page 435
- Adding Well-Tap Cells on page 437
- Adding End-Cap Cells on page 439
- Placing Spare Cells and Spare Modules on page 440
- Adding Padding on page 446
- Placing Standard Cells on page 450
- Running Placement in Multi-CPU Mode on page 451
- Checking Placement on page 453
- Refining Placement on page 456
- Adding Filler Cells on page 456
- Placing Gate Array Style Filler Cells for Post-Mask ECO on page 457
- Adding Decoupling Capacitance on page 459
- Adding Logical Tie-Off Cells on page 460
- Running Incremental Placement on page 460
- Saving Placement Data on page 460
- Using Netlist Clustering Mode on page 460
- Specifying and Placing JTAG and Other Cells Close to the I/Os on page 461
- Optimizing and Reordering Scan Chains on page 463

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 297.

After the cells are placed and resulting violations corrected, run pre-CTS optimization.

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 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 435 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*.
  - Specify the following command before running `placeDesign`:

    ```
    setPlaceMode -modulePlan false
    ```

> **Note:** Cadence recommends setting the `-modulePlan` parameter to `false` only for designs with more than 1,000 floorplan constraints or other types of complex floorplan constraints.

- Run the following commands to run placement in floorplan mode and quickly gauge the feasibility of the netlist. For more information, see `setPlaceMode` in the "Placement Commands" chapter of the *Encounter Text Command Reference*.

```
setPlaceMode -fp true
placeDesign
```

For more information on preparing the design for placement, see "Data Preparation" on page 59.

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
- Netlist
- 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 85
- `restoreDesign` in the "Import and Export Commands" chapter of the *Encounter Text Command Reference*.

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. 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.

Use the `addWellTap -minGap` parameter to increase minimum distance between the cells. In the previous example, if you specified `-minGap 0.55`, the minimum distance between the cells would be 26.4 microns (55 percent of 48 microns).

The following figure shows how `-cellInterval`, `-maxGap`, and `-minGap` are measured.

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.

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 443.

   Note: In the GUI, select the *Hierarchy Aware Spare Cell Placement* option on the Design – Mode Setup – Placement form.

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 443.

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.

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.`

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.`

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.`

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 *metal2* pitch dimension.

You can add padding to instances, leaf cells, and hierarchical modules.

Tip

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
   ```

Tip

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.

Tip

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
```

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.

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 to 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.

For more information, see the following commands in the *Encounter Text Command Reference:*

- `placeDesign` and `setPlaceMode` in the “Place Commands” chapter
- `specifyClockTree` in the “Clock Tree Synthesis Commands” chapter

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. For more information, see “Licensing Considerations” on page 50.

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`

Tip

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.
   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
```

Important

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.

For more information, see

- "Running Multi-Threading" on page 52
- "Multiple Processing Commands" chapter in the *Encounter Text Command Reference*.
- "Multiple CPU Processing" in the *Encounter Menu Reference*

Running Placement in MSV Designs

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 checkPlace Command

Using the Violation Browser

Using the Density Map

Use the density map to

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.

Refining Placement

Adding Filler Cells

The software uses filler cells to fill the gaps between standard cell instances. They 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.

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.

Running Incremental Placement

Running ECO Placement

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.

Using Netlist Clustering Mode

Use netlist clustering for full-chip prototyping of large designs. A cluster is a set of standard cells with strong connectivity that belong to the same intermediate module in the hierarchical netlist. Running placement in clustering mode is a quick way to evaluate the netlist to get a quick assessment on floorplan congestion. You can use the floorplan generated by clustering mode placement in a partitioned design.

With netlist clustering, you can create a smaller clustered netlist than the original netlist with the same logic hierarchy. For example, a netlist with 2 million instances can be reduced to 500,000 instances if the average cluster size is set to 4. This reduction results in increased system capacity.

Important

When running clustering mode placement, be aware of the following:

- Congestion and wirelength will not be as good as with normal flat placement.

- There is no guarantee that the placement will be free of overlaps.
- It is not intended to be used as final placement.

Do not generate a final placement by running `refinePlace` directly on the placement obtained from clustering mode placement.

Netlist Clustering Command Sequence

Use the netlist clustering commands in the following order:

1. `designImport`
2. `loadFPlan`
3. `netlistClustering` or `restoreClustering`
4. `placeDesign` or `restorePlace` if you previously ran netlist clustering and saved the placement data.

   Note: As an alternative to steps 3 and 4, you can use the `clusteringPlace` command. This command bundles the `netlistClustering` and placement commands as one.
5. `netlistUnclustering`
6. `savePlace flat.place`

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`

Tip

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.

For more information, see the following commands in the “Placement Commands” chapter of the *Encounter User Guide:*

- `specifyJtag`
- `unspecifyJtag`
- `placeJtag`
- `reportJtagInst`
- `unplaceJtag`
- `traceJtag`

What about tracing JTAG cells? When do you do this? In the training course, it sounds like specifyJtag and traceJTAG are interchangeable--is this true? When would you use one, and when would you use the other?

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.

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 uniquified, 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.

Note: If the netlist was generated using BuildGates Synthesis or Cadence PKS, you can remove the assignment statements prior to saving the netlist by running the following command:

```
do_xform_fix_multiport_nets
```

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

  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

  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_2` 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`

■ 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.

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.
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
```

Note: The Encounter software reads and parses DEF 5.5 syntax. However, scan reordering does not utilize the scan reorder DEF 5.5 syntax, such as:

```
        [+ PARTITION partitionName [MAXBITS maxBits] ]
```

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 BuildGates/PKS command to create a scanDEF file:

```
write_def –scan_only fileName.scan.def
```

or 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
  ```

  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.

12

Synthesizing Clock Trees

- Before You Begin on page 476
- Results on page 476
- Understanding CTS Operation Modes on page 477
- How CTS Calculates Skew Values on page 482
- Improving Postroute Correlation on page 484
- Specifying Macro Model Delays on page 485
- Grouping Clocks on page 486
- Analyzing Hierarchical Clock Trees on page 487
- Module Placement Utilization on page 489
- Clock Designs with Tight Area on page 489
- Balancing Pins for Macro Models on page 489
- Timing Model Requirement for Cells on page 489
- Delay Variation and OCV on page 489
- Understanding Post-CTS Clock Tree Optimization on page 490
- Creating a Clock Tree Specification File on page 494
- CTS Report Descriptions on page 524

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™ technology file, or Encounter capacitance table

  For information on RC extraction in Encounter, see RC Extraction on page 747 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

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 (for clock tree preroutes to be used during trial routing) 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.

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 28ff
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

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.

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 del1
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 ns - 3.4 ns = 0.2 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 ns - 3.0 ns = 1.0 ns.

The following figure illustrates how trigger-edge skew can be smaller than either rise skew or fall skew:

Rise skew (**C - A**): 500 ps - 100 ps = 400 ps
Fall skew (**D - B**): 800 ps - 300 ps = 500 ps
Trigger-edge skew (**C - B**): 500 ps - 300 ps = 200 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
  ```

  where *`cellName`* is the cell type name and the *`portName`* is the port name. For example:

  ```
  MacroModel port ram256x64/clk 10ns 80ns 110ns 7ns
  ```

- Pin instance delay specification—This specification can supersede a cell delay or port delay specification.

  ```
  MacroModel pin leafPinName maxRiseDelay minRiseDelay
       maxFallDelay minFallDelay
  ```

  where the *`leafPinName`* is the leaf pin instance name. For example:

  ```
  MacroModel pin mem_pin/clk 20ps 18ps 20ps 18ps
  ```

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 478 and Automatic CTS for Gated Clocks on page 479.

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:

```
ClkGroup
+ SH1/I23/Z1
+ SH22/I6/Z2
```

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.

Important

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
```

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}t`

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.

Important

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.

Example of a Clock Tree Specification File

The following example illustrates the content of a clock tree specification file:

```
TimingConstraintFile dtmf_cts.tc
UseSingleDelim YES
NameDelimiter |
MacroModel pin freg/mod004048/CLK 20ps 18ps 20ps 18ps
```

```
ClkGroup
+ CGEN_1
+ CGEN_2

UseCTSRouteGuide NO

#Start RouteTypeName section
RouteTypeName CK1
NonDefaultRule rule1
PreferredExtraSpace 1
TopPreferredLayer 5
BottomPreferredLayer 4
#End RouteTypeName section

Shielding VSS
End

#Example of manual CTS specification information
ClockNetName CK
LevelNumber 2
LevelSpec 1 1 BUFX2
LevelSpec 2 2 BUFX2
PostOpt YES
OptAddBuffer YES
End

AutoCTSRootPin cgen/i_5/Y
MaxDelay 5.0ns
MinDelay 0ns
MaxFanout 2
MaxSkew 250ps
SinkMaxTran 550ps
BufMaxTran 550ps
RootInputTran XYZ
NoGating NO
DetailReport YES
Obstruction YES
RouteType CK1
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
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*
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
+ clk_and1 clk_neg1
+ clk_and2 clk_neg2
```

Timing Constraint File Statement

You can optionally define a timing constraint file for use during CTS.

➤ To specify a timing constraint file, the `TimingConstraintFile` *`fileName`* statement must be the first statement in the clock tree specification file.

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 characters you can use for name delimiters. 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:

`MacroModel pin` *`pinName delay_and_capacitance_values`*

	Specifies the name of the macro model pin.
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 global tracing directive section.

Important

Global tracing 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.

`falling`	CTS treats the input pin as a falling-edge-triggered flip-flop clock pin.

`GlobalLeafPort`
`+` *`portName`* `rising | falling`
`+ …`

Marks the port as a “leaf” port for non-clock-type instances 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`
`+` *`portName`*
`+ …`

Treats the port as a non-leaf port 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.

Clock Grouping Data Section

The following table describes the entries for the clock grouping data section:

`ClkGroup` `+` *`clockRootPinName`* `+` *`clockRootPinName`* …	Specifies two or more clock domains for which you want CTS to balance the skew.

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` *`netName`*

Specifies the name of the clock net.

`LevelNumber` *`totalNumberOfClockTreeLevels`*

Specifies the total number of clock tree levels.

`LevelSpec` *`levelNumber numberOfBuffers bufferType`*

Provides the specifications for an individual clock level.

Specify all the following information:

`levelNumber` Sets the level number in the clock tree.

`numberOfBuffers`

Specifies the total number of buffers CTS should allow on the specified level of the clock tree.

`bufferType` 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` *`clockRootPinName`*

	Specifies the name of the clock root pin name from which to start tracing.
`MaxDelay` *`number`*`{ns|ps}`	Specifies the maximum phase delay constraint. *Default*: If you do not use this statement, CTS uses a maximum phase delay constraint of 10 ns.
`MinDelay` *`number`*`{ns|ps}`	Specifies the minimum phase delay constraint. *Default*: If you do not use this statement, CTS uses a minimum phase delay constraint of 0.0 ns.
`SinkMaxTran` *`number`*`{ns|ps}`	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. *Default*: If you do not use this statement, CTS uses a maximum sink transition time constraint of 400 ps.
`BufMaxTran` *`number`*`{ns|ps}`	Specifies the maximum input transition time constraint for buffers. CTS does not allow you to specify a value greater than 10,000 ns. *Default*: If you do not use this statement, CTS uses a maximum buffer transition time constraint of 400 ps.
`MaxGateRatio` *`ratio`*	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) *Default:* `1`
`MinGateRatio` *`ratio`*	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) *Default:* `0`
`MaxDeltaGateRatio` *`delta_ratio`*	

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

`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:

$$\mathtt{SrcLatency}_{\mathtt{clock1}} + \mathtt{MinDelay}_{\mathtt{clock1}} = \mathtt{SrcLatency}_{\mathtt{clock2}} + \mathtt{MinDelay}_{\mathtt{clock2}} = \mathtt{SrcLatency}_{\mathtt{clock}n} + \mathtt{MinDelay}_{\mathtt{clock}n}$$

$$\mathtt{SrcLatency}_{\mathtt{clock1}} + \mathtt{MaxDelay}_{\mathtt{clock1}} = \mathtt{SrcLatency}_{\mathtt{clock2}} + \mathtt{MaxDelay}_{\mathtt{clock2}} = \mathtt{SrcLatency}_{\mathtt{clock}n} + \mathtt{MaxDelay}_{\mathtt{clock}n}$$

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.

`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:

(*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*

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.

Important

You cannot use this statement for gated clocks.

`Period` *`value`*

Specifies the clock period of a root pin.

Default: 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:

`rising`	Stops tracing through a gate (including buffers and inverters) and treats the gate as a rising-edge-triggered flip-flop clock pin.
`falling`	Stops tracing through a gate (including buffers and inverters) and treats the gate as a falling-edge-triggered flip-flop clock pin.
`NO`	Allows CTS to trace through clock gating logic. *Default*: This is the default behavior for gated-clock designs. (If you omit the `NoGating` statement, CTS traces through clock gating logic.

`AddDriverCell` *`driver_cell_name`*

Places a driver cell name at the closest possible location to the clock port location for block-level CTS.

`MaxDepth` *`number`*

Sets the maximum depth of clock tree tracing.

Default: If you do not use this statement, CTS limits the number of levels of clock tree tracing to 1024.

`RouteType` *`routeTypeName`*

Specifies the name of the clock for which you are defining routing attributes.

Note: CTS uses the `RouteType` statement *only* if you specify `RouteClkNet YES`.

Note: If you use a `RouteType` statement, you must also use a corresponding *`routeTypeName`*.

`LeafRouteType` *`leafRouteTypeName`*

	Specifies the leaf 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.
`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.

`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`.

`PostOpt YES | NO`

Specifies whether CTS resizes buffers or inverters, refines placement, and corrects routing for signal and clock wires.

Default: YES

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.

Note: If you specify `PostOpt NO`, CTS does *not* resize gating components. If you specify `PostOpt YES`, CTS *attempts* to resize gating components, though it may not do so.

`OptAddBuffer {YES | NO}`

Controls whether CTS adds buffers during optimization.

Important

The `OptAddBuffer` statement is effective *only* if you specify `PostOpt YES`.

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.

`OptAddBufferLimit` *`positive_integer`*

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.

Default: If you do not use this statement, CTS inserts no more than 50 buffers.

`PadBufAfterGate YES | FIRST | NO`

Controls whether CTS moves padding cells within a chain of gating cells.

Padding cells are buffers (with a fanout of 1) that CTS has inserted before a gating cell in the clock tree. A chain of padding cells can contain one or more buffers that CTS has inserted.

Gating cells are buffers that exist in a clock tree before you run CTS. A chain of gating cells can contain one or more pre-existing buffers (each with a fanout of 1).

The `PadBufAfterGate` statement helps CTS reduce the power used in a gated clock tree.

Default: If you do not include this statement in the clock tree specification file, CTS behaves as if you had specified `PadBufAfterGate NO`.

YES	Instructs CTS to move padding cells from the input of the first gating cell in a chain to the output of the last gating cell in the chain.

Padding cell chain and gating cell chain

Following `PadBufAfterGate YES`

`FIRST` Instructs CTS to move padding cells from the input of the first gating cell in a chain to the output of the *first* gating cell in a chain.

Padding cell chain and gating cell chain

Following `PadBufAfterGate FIRST`

`NO` CTS does not move any cells to reduce power in a gated clock tree.

`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`

`LeafBuffer` *`buffer_list`*

Specifies a list of buffers for CTS to use when inserting buffers at the leaf level of the clock tree. You can specify one or more buffers, but you cannot specify inverters.

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 it 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 *only* with input pins. CTS ignores `LeafPin` 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.
`falling`	CTS treats the input pin as a falling-edge-triggered flip-flop clock pin.

```
LeafPort
+ portName rising | falling
+ …
```

Marks the port as a “leaf” port 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.

```
ExcludedPort
+ portName
+ …
```

Treats the port as a non-leaf port, and prevents tracing and skew analysis of the pin.

```
ThroughPin
+ pinName [outputPinNames]
+ …
```

Traces through the pin, 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.

`ThroughPort`
`+` *`portName`*
`+ …`
	Traces through the port, even if the pin is a clock pin.
	
	When you use this statement, CTS applies the `ThroughPort` statement to all ports with the same footprint as *`portName`*. CTS does so because it may resize the gates associated with these ports.

`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: ■ Data pins ■ Enable pins ■ Scan-in pins ■ Scan-enable pins ■ Synchronous set and reset pins This parameter does not control tri-state control pins. By default, the software treats them as synchronous pins. *Default*: 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 tri-state control pins. By default, the software treats them as synchronous pins. *Default*: If you omit this statement, CTS treats I/O pins as excluded pins.

`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.
`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 `?`.

```
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. **Note:** `ckCloneGate` 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` `+` *instance1* `+` *instance2* `+` *instance3 instance4* `+` …	Specifies the instances to be decloned. `ckDeCloneGate` merges the instances you specify into one instance. To declone groups of gating cells, specify more than one instance on one line.
`End`	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:** `Clock` *clockName* **plus—**

`RouteClkNet NO`	`ckSynthesis`
	`globalDetailRoute`
	`reportClkTree -clk` *clock*
		`Pre-Route Timing Analysis`

`RouteClkNet YES`	`ckSynthesis`
	`globalDetailRoute`
	`reportClkTree -clk` *clock* `-clkRouteOnly`
		`Clk-Route-Only Timing Analysis`

Clock tree specification file statements	Example clock tree text command sequences	Encounter log file section heading: `Clock` *`clockName`* **plus—**

`RouteClkNet YES`
`PostOpt YES`
	`ckSynthesis`
	`globalDetailRoute`
	`reportClkTree -clk` *`clock`* `-postRoute`
		`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)
```

```
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
-------------------------------------------------------
```

```
[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
PadBufAfterGate YES
```

```
    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)
```

13

Working with Clock Mesh Structures

- Overview on page 530
- Clock Meshes Versus Clock Trees on page 530
- Creating Clock Meshes on page 533
  - Determining the Mesh Structure on page 533
    - Supported Mesh Styles on page 533
    - Clock Mesh Structure Characteristics on page 535
    - Multilevel Structure of a Mesh on page 537
  - Implementing the Clock Mesh on page 538
  - Analyzing the Clock Mesh on page 539
    - Pre-Route Wire Estimation on page 539
    - RC Extraction on page 539
    - Computing Mesh Delays on page 540

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

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

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 533
2. Implementing the Clock Mesh on page 538
3. Analyzing the Clock Mesh on page 539

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 13-1 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 13-2 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 µm 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 13-3 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 13-4 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 13-5 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:

- 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.

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.

14

Editing Wires

- Overview on page 545
- Before You Begin on page 546
- Results on page 546
- Using Keyboard Shortcuts on page 546
- Moving Wires on page 549
- Adding Wires on page 551
- Deleting Wires on page 549
- Cutting Shielding Wires on page 556
- Trimming Antennas on Selected Stripes on page 556
- Changing Wire Width on page 557
- Repairing Maximum Wire Width Violations on page 558
- Duplicating Wires on page 558
- Stretching Wires on page 559
- Changing Wire Layers on page 559
- Splitting and Merging Wires on page 560
- Adding Vias on page 560
- Changing Vias on page 561
- Moving Vias on page 562
- Reshaping Routes on page 562
- Controlling Cell Blockage Visibility on page 563
- Editing Wires with Virtuoso Chip Assembly Router on page 564

- Running Virtuoso CAR in Batch Mode on page 564
- Running Virtuoso CAR in Interactive Mode on page 565

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.

Tip

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*

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 559.

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 561.

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
```

Tip

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, 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.

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 554 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 555 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 555 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 connected 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 554 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.
5. Use the `w` keyboard shortcut to enter *select* mode.

   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 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 549.

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 551.

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 slidebar 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.

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.

15

Using Trial Route for Congestion and Timing Analysis

- Overview on page 568
- Data Preparation on page 568
- Routing A Flat Design on page 569
- Routing a Partitioned Design on page 569
- Routing Two-Metal Layer Designs on page 572
- Loading and Saving Route Data on page 572
- Analyzing Route Data on page 572
- Improving Route Congestion on page 581
- Additional Information on page 582

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. If you specify a routing guide file, Trial Route divides the design into routing regions and routes within these regions.

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.

Data Preparation

- The design must be successfully placed.

Important

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
- 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
  ```

Important

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 15-1 on page 571 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.

Figure 15-1

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 *metal4* and *metal5* 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 573 for more information.

2. In the log file, inspect the Trial Route contents in the congestion distribution table.

Ensure the last routing overflow percent values do not exceed 1.0 percent for both horizontal (H) and vertical (V) tracks. For a design that uses five or more layers of metal interconnect, the 1.0 percent limit is optimal. For a design that uses three layers of metal interconnect, a 0.5 percent limit is optimal.

See "Congestion Distribution Report" on page 575 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 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.

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 573.

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)
```

```
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` µm, and the total vertical wire length used equals `1.314e+07` µm.

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
---------------------------------------------
  -6:      0       0.00%    1      0.00%
  -5:      2       0.00%    0      0.00%
  -3:      10      0.00%    26     0.00%
  -2:      510     0.03%    830    0.05%
  -1:      8100    0.47%    17618  1.05%
-----------------------------------------------
   0:      78504   4.59%   178501  10.63%
   1:      102934  6.02%   214588  12.78%
   2:      76165   4.46%   185168  11.03%
   3:      72832   4.26%   179080  10.67%
   4:      81555   4.77%   180443  10.75%
   5:      92704   5.42%   158498   9.44%
   6:      106167  6.21%   137707   8.20%
```

The following table defines the columns in the congestion report:

Column	Definition
`Remain`	The track supply minus the track demand.
`cntH`	When `Remain` is positive, the number and percentage of gcells where the horizontal track supply exceeds the horizontal track demand. When `Remain` is negative, the number and percentage of gcells where the horizontal track demand exceeds the horizontal track supply. When `Remain` is 0 (zero), the number and percentage of gcells where the horizontal track demand is equal to the horizontal track supply.
`cntV`	When `Remain` is positive, the number and percentage of gcells where the vertical track supply exceeds the vertical track demand. When `Remain` is negative, the number and percentage of gcells where the vertical track demand exceeds the vertical track supply. When `Remain` 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.

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%
```

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  ...
-------------------------------------------------------------------------
-6:   0        0.00%  0        0.00%   0        0.00%  0         0.00%
-5:   0        0.00%  0        0.00%   0        0.00%  3         0.00%
-4:   0        0.00%  0        0.00%   0        0.00%  5         0.00%
-3:   0        0.00%  0        0.00%   0        0.00%  37        0.00%
-2:   10       0.00%  28       0.00%   6        0.00%  264       0.00%
-1:   355      0.00%  476      0.00%   229      0.00%  1254      0.01%
---------------------------------------------------------------------------
 0:   8855290 65.73%  8858305 65.76%   70715    0.52%  127176    0.94%
 1:   91903    0.68%  269941   2.00%   124321   0.92%  460099    3.42%
 2:   166878   1.24%  441388   3.28%   192300   1.43%  1052688   7.81%
 3:   250040   1.86%  559047   4.15%   347255   2.58%  1109275   8.23%
 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

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 *metal2*, 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 *metal3*.

```
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 *metal2*, and 269,941 gcells where the track supply exceeds the track demand on layer *metal3*.

```
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%  0         0.00%
-5:   0        0.00%  0        0.00%   0        0.00%  3         0.00%
-4:   0        0.00%  0        0.00%   0        0.00%  5         0.00%
-3:   0        0.00%  0        0.00%   0        0.00%  37        0.00%
-2:   10       0.00%  28       0.00%   6        0.00%  264       0.00%
-1:   355      0.00%  476      0.00%   229      0.00%  1214      0.01%
-------------------------------------------------------------------------
 0:   34536    0.74%  83715    1.78%   18240    0.14%  101470    0.75%
 1:   91903    1.98%  269941   5.75%   124321   0.93%  460099    3.42%
 2:   166878   3.59%  441388   9.40%   192300   1.43%  1052688   7.83%
 3:   250040   5.38%  559047  11.90%   347255   2.59%  1109275   8.25%
```

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:          3.2       1.7      8.5        5.0
-----------------------------------------------------------------------

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:

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."

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.

16

Using the NanoRoute Router

- About NanoRoute Routing Technology on page 588
- Routing Phases on page 588
  - Global Routing on page 588
  - Detailed Routing on page 589
  - ECO Routing on page 589
- NanoRoute Router in the Encounter Flow on page 589
- Before You Begin on page 590
  - Checking Your LEF Files on page 590
  - Checking for Problems with Cells, Pins, and Vias on page 591
  - Generating Tracks on page 592
  - Specifying Preferred Routing Layers on page 592
- Results on page 592
- Running the NanoRoute Router with Encounter Menu Commands and Forms on page 594
  - Running the NanoRoute Router with Encounter Menu Commands and Forms on page 594
  - Running the NanoRoute Router with Encounter Text Commands on page 594
  - Running the NanoRoute Router in Standalone Mode on page 595
- Using NanoRoute Parameters on page 596
  - Using Attributes and Options Together on page 597
- Accelerating Routing with Multi-Threading and Superthreading on page 599
  - When to Accelerate Routing on page 600

- Superthreading Log File Excerpts on page 601
- Following a Basic Routing Strategy on page 603
  - Using the Encounter Text Commands on page 603
  - Using the Encounter GUI on page 604
- Checking Congestion on page 607
  - Using the Congestion Analysis Table on page 607
  - Using the Congestion Map on page 608
- Resolving Open Nets on page 612
  - Log File Examples on page 612
  - Diagnosing Problems Using verifyTracks on page 613
  - Resolving Additional Open Net Problems on page 613
- Running Timing-Driven Routing on page 615
  - Input Files on page 615
  - Using the CTE and the NanoRoute Router in Native Mode on page 615
  - Using the CTE and Standalone NanoRoute on page 616
  - Using Another Timing Engine on page 617
- Routing Clocks on page 618
  - Setting Attributes for Clock Nets on page 618
  - Routing the Clock Nets on page 619
  - Running Postroute Optimization on page 619
- Preventing and Repairing Crosstalk Problems on page 620
  - Crosstalk Prevention Options on page 622
- Running ECO Routing on page 624
  - ECO Limitations on page 624
  - ECO Flow on page 625
- Evaluating Violations on page 626
  - Violations on Upper Metal Layers on page 630

- Violations in Timing-Driven Routing on page 632
- Deleting Violated Nets on page 633
- Using Additional Strategies to Repair Violations on page 633

- Concurrent Routing and Multi-Cut Via Insertion on page 634
- Postroute Via Optimization on page 634
- Optimizing Vias in Selected Nets on page 635
- Via Optimization Options on page 635
- Performing Shielded Routing on page 637
  - Shielding Option on page 638
  - Performing Shielded Routing Using the GUI on page 638
  - Performing Shielded Routing Using Text Commands on page 639
  - Interpreting the Shielding Statistics Report on page 639
- Routing Wide Wires on page 642
  - Using Non-Default Rules on page 642
- Repairing Process Antenna Violations on page 644
  - Changing Layers on page 645
  - Using Diodes on page 645
  - Process Antenna Options on page 646
  - Examples on page 646
- Using a Design Flow that Includes Astro or Apollo on page 648
- Troubleshooting on page 650

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*.

Routing Phases

Full routing consists of global and detailed routing. ECO routing consists of incremental global and detailed routing passes on a routed design.

Global Routing

During this phase, the NanoRoute 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 it 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 read 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.

- 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 607.

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.

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.

ECO Routing

The NanoRoute router runs both global and detailed routing during ECO routing. The router makes minimal changes to the topology of fully routed nets during ECO routing.

For more information, see "Running ECO Routing" on page 624.

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.

Cadence® recommends you run a global and detailed route with the router after each of the major routing steps in the design flow, check for localized and global congestion, and resolve any problems before going to the next step.

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 unroutable, stop and resolve problems before continuing.

For information on the major routing steps, see "Following a Basic Routing Strategy" on page 603.

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.
- Power is routed.

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
  ```

  For information on this command, see "Import and Export Commands" in the *Encounter Text Command Reference*.

- `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.

  For more information, see "Unsupported LEF and DEF Syntax" in the *Encounter User Guide* and the "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 using one of the following commands:

  - `verifyGeometry -wireOverlap`

    For more information, see `verifyGeometry` in the *Encounter Text Command Reference*.

  - `checkPlace`

    For more information, see `checkPlace` in the *Encounter Text Command Reference*.

- Pins underneath power routes

  Pins that are underneath power routes are inaccessible and cause violations on *metal1* and *metal2*. Check for pins underneath power routes 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 *metal1* 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 NanoRoute 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.

- 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 Preferred Routing Layers

You can route a particular net to a specific layer range by adding a `PREFERLAYERRANGE` property to the net definition in your DEF file. For example, if you want the NanoRoute router to route a particular net on metal layers 4, 5, and 6, add the following property to the net definition:

```
+ PROPERTY PREFERLAYERRANGE "4:6" ;
```

In the example above, 4 is the preferred bottom layer. It corresponds to specifying the following command:

```
setAttribute -bottom_preferred_routing_layer 4
```

In the example above, 6 is the preferred top routing layer. It corresponds to specifying the following command:

```
setAttribute -top_preferred_routing_layer 6
```

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 85.

  For information on outputting an SDF or SPEF file, see "Timing Analysis" on page 779.

- 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 Statistics Report" on page 639.

  - Timing analysis

    For information, see "Timing Analysis" on page 779.

  - Capacitance

    For information, see "RC Extraction" on page 747.

  - 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 937.

  - Signal integrity

For information on signal integrity reports, see "Analyzing and Repairing Crosstalk" on page 895.

Use Models

Running the NanoRoute Router with Encounter Menu Commands and Forms

The NanoRoute router can perform all basic functions using the following Encounter forms:

- Design – Mode Setup – NanoRoute

- NanoRoute/Attributes

  Use this form to set attributes for nets.

- NanoRoute

  Use this form to set run-time options for the NanoRoute router.

  For more information on using the NanoRoute forms, see "Route Menu" in the *Encounter Menu Command Reference*.

Running the NanoRoute Router with Encounter Text Commands

Within the Encounter products, the following commands set NanoRoute attributes and options, and run global and detailed routing with the NanoRoute router:

- `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).

- `getAttribute`
  `setAttribute`

  Display and set the current net attributes.

- `getNanoRouteMode`
  `setNanoRouteMode`

  Display and set the current run-time options.

- `globalRoute`
  `detailRoute`
  `globalDetailRoute`
  `routeDesign`

  Route the design.

The text commands include some NanoRoute options that are not included on the forms. For more information on the commands and options, see "Route Commands" in the *Encounter Text Command Reference*.

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. For more information, see the *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	■ NanoRoute/Attributes form ■ `setAttribute` command ■ Some attributes can only be specified by `setAttribute`.	■ NanoRoute form ■ `setNanoRouteMode` command ■ Some options can only be specified by `setNanoRouteMode`.
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.

Characteristic	Attributes	Options
Format	*-attribute_name* ■ Example: `-avoid_detour` ■ Case sensitive (all lower case) ■ Mandatory underscores separate words ■ Native and standalone NanoRoute formats are the same	*-optionName* ■ Example: `-drouteAutoStop` ■ Case sensitive (all mixed case) ■ No underscores. Each word starts with an uppercase letter ■ 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
See settings for this run …	Use the `getAttribute` command	Use the `getNanoRouteMode` command
More information available at …	`setAttribute` in the *Encounter Text Command Reference*	`setNanoRouteMode` in the *Encounter Text Command Reference*

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
```

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

Important

In this release, Encounter features use a common multiple-CPU processing interface for multi-threading and Superthreading. You can still use the NanoRoute options and forms, but they will be overwritten by the values for the multiple-CPU processing commands and form, if specified. For more information, see the following documents:

- "Running Multiple-CPU Processing" on page 49
- Multiple-CPU Commands chapter in the *Encounter Text Commands Reference*
- "Multiple CPU Processing" form in the Design Menu chapter of the *Encounter Menu Reference*

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" on page 52.

The NanoRoute 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.

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)
. ...
```

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 626 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 626.

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.

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 588.

Following is an example of a Congestion Analysis table:

```
Congestion Analysis:
                OverCon       OverCon      OverCon      OverCon
                 #Gcell        #Gcell       #Gcell       #Gcell     %Gcell
Layer             (1-2)         (3-4)        (5-6)       (7-12)    OverCon
-----------------------------------------------------------------------------
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.
White	High congestion: You probably have at least four tracks that are over-used.

In the congestion map shown below, there is a congested area (a hot spot) in the lower left quadrant.

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 613.
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 613.
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 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

- 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

You can choose the timing engine to use for timing-driven routing. In the Encounter environment, running the router in timing-driven mode uses the Common Timing Engine (CTE) by default, but you can specify another timing engine, such as PrimeTime.

When you use the CTE, 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 632.

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 59.
- 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 16-1 on page 616 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 16-1

Native NanoRoute-CTE timing-driven routing flow

Using the CTE and Standalone NanoRoute

Figure 16-2 on page 617 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:

- Outside of the Encounter environment

```
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:

```
pdi set_option timingEngine design.tif
```

3. Extracting capacitance and analyzing timing

Figure 16-2

Standalone NanoRoute-CTE timing-driven routing flow

Using Another Timing Engine

To use another timing engine, such as PrimeTime, issue the following commands:

- Outside of the Encounter environment

```
pdi set_option timing_engine external_timing_graph
pdi set_option route_with_timing_driven true
pdi global_detail_route
```

Routing Clocks

Route clock nets before routing the rest of the signal nets. By default, the NanoRoute router does not move clock nets during postroute optimization.

This section gives information on routing clocks manually. For information on routing clocks automatically with the Encounter software, see "Synthesizing Clock Trees" on page 475.

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 *metal1* 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*.

Tip

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 the Clock Nets

➤ 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`.

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.

- 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.

  For more information on shielded routing see "Performing Shielded Routing" on page 637.

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 895.

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
```

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 879.

Evaluating Violations

After you run several search-and-repair iterations, look at the number and distribution of violations remaining, and determine whether they are problems with congestion or are design or library problems. If you see many violations (more than 1,000) or the number of violations does not decrease with each iteration, stop search and repair and check congestion graphically.

NanoRoute's violation types are limited to the following four types. It does not maintain detailed reasons for finding a violation, but it does leaves a violation marker of one of the following types:

- Horizontal

  If a violation falls on a horizontal wire, it is called a horizontal violation.

- Vertical

  If a violation falls on a vertical wire, it is called a vertical violation.

- Via

  If a violation falls on a via, it is called 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

  A minimum area rule violation indicates the AREA rule for the layer has not been satisfied.

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 16-3 on page 627 illustrates this stage in the design. Violations on different layers are shown by different-colored markers.

Figure 16-3

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.

Use the guidelines in the following table to help evaluate violations:

Violations/Warnings	Check for …
Many violations on *metal1*	■ Improper setup of routing tracks ■ Overlapping cells ■ Insufficient via rotation, causing inaccessible pins For information on fixing these problems, see the *Encounter Library Development Guide*.
Many violations on *metal1* and *metal2*	■ 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*.
Open net warning messages An open net is one that the router cannot route because it cannot complete the connection of a net.	■ Pin modelling, track definition, data, floorplanning problems ■ Conflicts between option settings or options and library specifications For information on fixing these problems, see "Resolving Open Nets" on page 612.

Violations/Warnings	Check for …
Violations not decreasing after first iteration of detailed routing	■ 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 630. 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 433. For information on floorplanning, see "Floorplanning the Design" on page 291
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 16-4 on page 630, Figure 16-5 on page 631, and the LEF and DEF file excerpts that follow show a design with many violations on *metal6*.

Figure 16-4

Area expanded in Figure 16-5 on page 631 to show violations

Figure 16-5

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:

```
Line to via metal6 = 1/2 Width + Spacing + 1/2 Via
                   = 0.1 + 0.21 + 0.19
                   = 0.5
```

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 615.

Figure 16-6 on page 632 and Figure 16-7 on page 633 illustrate non-timing-driven and timing-driven routing results for the same design.

Figure 16-6

Figure 16-7

Deleting Violated Nets

To delete violated nets, use the `editDeleteViolations` command. After deleting the nets, use area routing or ECO routing to re-route the design.

For information, see "Route Commands" in the *Encounter Text Command Reference*.

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 947.
- For information on process antenna options, see Repairing Process Antenna Violations.
- For information on wire editing, see "Editing Wires" on page 543.

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.
- 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 -drouteMinimizeViaCount 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
   ```

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`
- `-drouteMinimizeViaCount`
- `-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 -drouteMinimizeViaCount 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 Statistics Report" on page 639.

Figure 16-8 on page 637 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 16-8

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.

Tip

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 Statistics Report

The NanoRoute router automatically generates a shielding statistics report named *`design_name.`*`shield.rpt` in the current directory when there are shielded nets in the design. The Encounter log file contains a summary of the statistics from the report.

Following is a shielding statistics report for a design:

```
Name     : Shielded net name
Length   : Shielded net length
Noshield : Total length of shielded net segments denoted by NOSHIELD in DEF
Floating : Total length of floating shielding wire
Shield   : Total length of shielding wire not floating
ratio    : Average shielding ratio

Floating shielding wire removed
-------------------------------------------------------------------------------
Name        Length       Noshield   Floating    Shield   (ratio)
-------------------------------------------------------------------------------
my_net_1:   185.5        75.8       181.8        34.6    0.095
my_net_2:   214.9        54.6       241.7        79.1    0.184
my_net_3:   241.7        47.3       245.6       175.6    0.335
my_net_4:   270.0        66.5       229.0       195.2    0.347
my_net_5:   285.7        66.0       245.1       204.5    0.350
```

```
my_net_6:   310.7        43.2       228.7       330.6    0.509
my_net_7:   277.7        47.6       108.9       362.4    0.637
average:    255.2        57.3       211.5       197.4    0.374
```

Figure 16-9 on page 640 shows a section of the design in the report.

Figure 16-9

In the figure,

f	Represents floating shielding wire. For `my_net_1`, the length of *f* is `181.8`.
l	Represents a shielded net. For `my_net_1`, the length of *l* is `185.5`.
n	Represents the portion of the shielded net that is not shielded. For `my_net_1`, the length of *n* is `75.8`.
s	Represents the portion of the shielding net that is not floating. For `my_net_1`, the length of *s* is `34.6`.

Note that two wire segments, one on either side of net *l*, are needed to make up *s* and *f*.

The ratio in the report is calculated using the following formula:

$$ratio = W \times (1 - F)$$

where

$$W = \frac{l - n}{l}$$

$$F = \frac{f}{s + f}$$

The calculations for the ratio for `my_net_1` in the report are

$$W = \frac{185.5 - 75.8}{185.5} = 0.59$$

$$F = \frac{181.8}{34.6 + 181.8} = 0.84$$

$$ratio = 0.59 \times (1 - 0.84) = 0.095$$

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 16-10 on page 643 illustrates nondefault spacing (“soft spacing”) routing.

Figure 16-10

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
```

```
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.

Caution

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.

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 the `-drouteAutoStop` parameter should be set to `false` to continue routing. For more information on `-drouteAutoStop`, see the *Encounter Text Command Reference*.

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.

17

Using the Encounter Software in Mixed Signal Designs

- Overview on page 653
- Before You Begin on page 654
- Results on page 654
- Specialized Routing Techniques on page 655
  - Matched Nets on page 655
  - Differential Pair Nets on page 656
  - Shielded Nets on page 656
  - Bus Routes on page 660
- Using Routing Constraints on page 661
- Constraint File Format on page 661
- Generic Constraints on page 662
- Specialized Constraints and Keyword Descriptions on page 664
  - NETS on page 664
  - MATCH on page 666
  - DIFFPAIR on page 668
  - SHIELDING on page 670
  - BUS on page 679
- Creating a Constraint File on page 681
- Editing a Constraint File on page 689
- Loading a Constraint File on page 686

- Sample Constraint File on page 691

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.

  For a description of the AOT design flow, see “Analog-Driven Mixed Signal Design Implementation Using Encounter® and Virtuoso®,” posted on SourceLink. The document provides an example of a top-level physical design, including digital implementation of a standard cell-based block and integration of the block with an analog-driven design. It describes the steps to complete the design using both Virtuoso and Encounter software.

  In addition, 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.

  In addition, see the following Encounter documentation for information on this flow:

  - *Encounter Text Command Reference,* “Mixed Signal Commands” chapter
  - *Encounter Menu Reference,* “Route Menu” (Mixed Signal Router), “Tools Menu” (Mixed Signal Constraint Editor and UltraSim), and “Verify Menu” (Verify Mixed Signal Constraints) chapters

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
- Shielded routing, including coaxial shielded routing and differential pair shielded routing
- Bus routing

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
- 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 663.

The following figure shows a pair of matched nets.

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.

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.
- 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, at one end of the shield, or at both ends.
- 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 at one end.

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.

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 only.

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.

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 681

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 662.
- 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 664.
- 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
- Width and maximum width
- Spacing
- Routing layers
- Tapering

- 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`.

- 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 664
- MATCH on page 666
- DIFFPAIR on page 668
- SHIELDING on page 670
- BUS on page 679

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 width
  MAXWIDTH maxWidth
  MAXRES maxResistance
  MAXCAP maxCapacitance
  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
```

`MAXWIDTH` *`maxWidth`*

Specifies the maximum width of the constrained nets, 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.

`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 notmeet this requirement, the router adjusts the values to increse 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.

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
  MAXWIDTH maxWidth
  TOLERANCE value
  MAXRES maxResistance
  MAXCAP maxCapacitance
  MINRES minResistance
  MINCUT minCuts
  TAPERING numberTaperSteps taperEndWidth
  SPACING spacing
  ROUTELAYERS {bottom_layer_num:top_layer_num | list_of_layer_numbers ...}
    list_of_signal_nets
END MATCH
```

`MAXWIDTH` *`maxWidth`*

Specifies the maximum width of the constrained nets, 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.

`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 notmeet this requirement, the router adjusts the values to increse 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 655.

`WIDTH` *`minWidth`*

Specifies the minimum width of the constrained nets, in microns.

`list_of_signal_nets`

Specifies the list of constrained nets.

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.

> **Note:** For information on shielding differential pair nets, see "Differential Pair Shielding" on page 675.

```
DIFFPAIR
  WIDTH width
  MAXWIDTH maxWidth
  PAIRGAP spacing
  THRESHOLD value
  MAXRES maxResistance
  MAXCAP maxCap
  MINRES minResistance
  MINCUT minCuts
  TAPERING numberTaperSteps taperEndWidth
  SPACING spacing
  ROUTELAYERS {bottom_layer_num:top_layer_num | list_of_layer_numbers ...}
    {signal_net_1 | signal_net_2}
END DIFFPAIR
```

`MAXWIDTH` *`maxWidth`*

Specifies the maximum width of the constrained nets, 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.

`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.

`SPACING` *`spacing`*
: Specifies the distance between the constrained nets and other 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.

`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 increse 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 width of constrained nets, measured in microns.

`signal_net_1` `|` *`signal_net_2`*
: Specifies the constrained nets.

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
  SHIELDGAP spacing
  SHIELDWIDTH width
  NOSHARESHIELD {0 | 1}
  MAXRES maxResistance
  MAXCAP maxCapacitance
  MINRES minResistance
  ROUTELAYERS {bottom_layer_num:top_layer_num | list_of_layer_numbers}
  MINCUT minCuts
  TAPERING numberTaperSteps taperEndWidth
  SPACING spacing
  WIDTH minWidth
  CONNECTSUPPLY {FLOAT | ONEEND | BOTHEND | ANYPOINT}
  SHIELDNET special_net_name
    {signal_net_name | list_of_signal_nets}
END SHIELDING
```

`CONNECTSUPPLY {FLOAT | ONEEND | BOTHEND | 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.
`ONEEND`	Connects the shield to the supply at one end.
`BOTHEND`	Connects the shield to the supply at both ends.
`ANYPOINT`	Connects the shield to the supply at any point along the wire.

`MAXWIDTH` *`maxWidth`*

Specifies the maximum width of the constrained nets, 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.

`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.

`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 notmeet this requirement, the router adjusts the values to increse 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 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
  WIDTH minWidth
  MAXWIDTH maxWidth
  SHIELDGAP spacing
  SHIELDWIDTH width
  SHIELDLAYERS integer
  OVERHANG width
  MAXRES maxResistance
  MAXCAP maxCapacitance
  MINRES minResistance
  ROUTELAYERS {bottom_layer_num:top_layer_num | list_if_layer_numbers}
  MINCUT minCuts
  TAPERING numberTaperSteps taperEndWidth
  SPACING spacing
  CONNECTSUPPLY {FLOAT | ONEEND | BOTHEND | ANYPOINT}
  SHIELDNET special_net_name
```

```
    {signal_net_name | list_of_signal_nets}
END SHIELDING
```

`CONNECTSUPPLY {FLOAT | ONEEND | BOTHEND | 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.
`ONEEND`	Connects the shield to the supply at one end.
`BOTHEND`	Connects the shield to the supply at both ends.
`ANYPOINT`	Connects the shield to the supply at any point along the wire.

`MAXWIDTH` *`maxWidth`*

Specifies the maximum width of the constrained nets, 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.

`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.

`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

`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 notmeet this requirement, the router adjusts the values to increse 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 width of the constrained (signal) nets, in microns.

`signal_net_name | list_of_signal_nets`

Specifies a constrained net or a list of constrained nets to shield.

`special_net_name` Specifies a special net to use to shield the constrained signal nets.

Differential Pair Shielding

Differential pair shielding uses the following syntax (the shielding statements are included in the `DIFFPAIR` constraint and are shown in bold type below):

```
DIFFPAIR
  WIDTH width
  MAXWIDTH maxWidth
  PAIRGAP spacing
  THRESHOLD value
  MAXRES maxResistance
  MAXCAP maxCapacitance
  MINRES minResistance
  ROUTELAYERS {bottom_layer_num:top_layer_num | list_of_layer_numbers}
  MINCUT integer
  TAPERING numberTaperSteps taperEndWidth
  SPACING spacing
  SHIELDGAP spacing
  SHIELDWIDTH width
  CONNECTSUPPLY {FLOAT | ONEEND | BOTHEND | ANYPOINT}
```

```
  SHIELDNET special_net_name
    signal_net_1 signal_net_2
END DIFFPAIR
```

Important

In this constraint, the `SHIELDGAP` must be defined on the line immediately preceding the `SHIELDWIDTH` definition.

`CONNECTSUPPLY {FLOAT | ONEEND | BOTHEND | 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.
`ONEEND`	Connects the shield to the supply at one end.
`BOTHEND`	Connects the shield to the supply at both ends.
`ANYPOINT`	Connects the shield to the supply at any point along the wire.

`MAXWIDTH` *`maxWidth`*

Specifies the maximum width of the constrained nets, 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.

`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 658.

`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

`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 notmeet this requirement, the router adjusts the values to increse 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 width of the constrained (signal) nets, in microns.

`signal_net_name | list_of_signal_nets`

Specifies a constrained net or a list of constrained nets to shield.

`special_net_name`

Specifies a special net to use to shield the constrained signal nets.

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
  MAXWIDTH maxWidth
  BUSGAP spacing
  SAMELAYER {0 | 1}
  MAXRES maxResistance
  MAXCAP maxCapacitance
  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.

`MAXWIDTH` *`maxWidth`*
: Specifies the maximum width of the constrained nets, 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.

`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.

`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`

`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 notmeet this requirement, the router adjusts the values to increse 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 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]`.

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 664.
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 <u>step 5</u> until you have added all the nets for the constraint to the *Constraint Members List*.

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.

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 681 and "Entering Values for Constraint Parameters" on page 683 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.

   The following figure shows the left-hand portion of the Display All form.

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.msroutenumber.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 664.

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*.

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.

➤ 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 686.
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 681, "Entering Values for Constraint Parameters" on page 683, and "Saving the Constraint to the Constraint File" on page 685.

Using a Text Editor

Open the constraint file in a text editor and edit it, following the syntax in "Constraint File Format" on page 661. 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
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
```

```
    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
```

```
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
```

18

Optimizing Metal Density

- Overview on page 696
- Before You Begin on page 696
- After You Complete Adding Via and Metal Fill on page 697
- Metal Fill Features on page 697
- Specifying Metal Fill Parameters on page 704
- Recommendations for Adding Timing-Aware Metal Fill on page 705
- Adding Metal Fill Over Macros on page 708
- Recommendations for Power Strapping Mode on page 709
- Adding Via Fill on page 710
- Trimming Metal Fill on page 711
- Verifying Metal Density on page 711
- Adding Metal Fill Using the GUI on page 712

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 697.
- 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 703.

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 710.

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.

You 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.

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 Distributed Processing Mode

Use one of the following methods to add metal fill in distributed processing mode:

- Use the following multiple-CPU processing commands before running `addMetalFill`:
  - `setDistributeHost`
  - `setMultiCpuUsage`

  For more information on these and other multiple-CPU commands, see the "Multiple-CPU Processing" commands chapter in the *Encounter Text Command Reference*.
- 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 R unning Multiple-CPU Processing in the "Getting Started" chapter of the *Encounter User Guide*.

After You Complete Adding Via and Metal Fill

After adding via and metal fill and trimming the excess 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:

addMetalFill -stagger diag
(horizontal)

addMetalFill -stagger diag
(vertical)

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:

- 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 711.

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.

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.

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 779.

  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.

Caution

Use caution when overriding the values in the LEF file or the DRC manual—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.

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	`FILLACTIVESPACING`	`-activeSpacing`	*Active Spacing*
Minimum density allowed in the design	`MINIMUMDENSITY`	`-minDensity`	*Metal Density % Min*
Maximum density allowed in the design	`MAXIMUMDENSITY`	`-maxDensity`	*Metal Density % Max*
Area the Encounter software uses to examine metal density	`DENSITYCHECKWINDOW`	`-windowSize`	*Step Size X* *Step Size Y*
Distance the window moves for each metal fill iteration	`DENSITYCHECKSTEP`	`-windowStep`	*Window Size X* *Window Size Y*

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 µm to 10 µm
  - 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 µm to 1.0 µm for thin layers
    - 0.8 µm to 2.0 µm 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 µm for thin layers
  - 0.8 µm for thick layers

  Alternatively, for lower capacitance at the expense of more density variation, use values like 0.8 µm for thin layers and 1.6 µm 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 `FILLACTIVESPACING` value in the LEF file for the active spacing. If `FILLACTIVESPACING` 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
- 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 µm to 3 µm 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 En*counter 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.

    Floating via fill looks like a floating net to the Encounter database.

DEF syntax does not support floating metal fill vias in the `FILLS` section, so the software uses a `_FILLS_RESERVED` net with `+ SHAPE FILLWIRE` in the `SPECIALNETS` section to represent floating via fill.

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:

```
setViaFill
setMetalFill
addViaFill
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.

  For more information, see `trimMetalFill`.

- To remove connected metal fill, complete the following steps:

  a. Delete all metal fill by running the following command:

  ```
  editDelete -shapes {FILLWIRE FILLWIREOPC}
  ```

  For more information, see `editDelete`.

  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.

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, and preferred 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.

19

Timing Budgeting

- Overview on page 714
- Is My Design Ready for Budgeting? on page 716
- Deriving Timing Budgets on page 717
  - Budgeting Using the GUI on page 717
  - Budgeting Using Text Commands on page 717
  - Top-Level Budgets Derived by Using Virtual Partitioning on page 718
  - Deriving Preliminary Budgets in Early Design Phase on page 719
- Budgeting Output Files for MMMC Designs on page 721
- Constraints Adjustment on page 724
- Analyzing Timing Budgets on page 726
  - Resolving Conflicts with Path-Based Exceptions on page 726
  - Budgeting Clock Latency in Propagated Mode on page 729
- Calculating Timing Budgets on page 731
- Customizing Budget Generation on page 734
- Verifying Timing Budgets on page 736
- Reading the Justify Budget Report on page 737
- Constraints Support in Budgeting on page 742
- Warning Report on page 745

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 lPO (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 MMMC mode, timing budgeting is run per-view, and generates view files for partition implementation and top-level implementation (not chip assembly).

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 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 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 Virtual Partitioning

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 -ignoreScan
   placeDesign
   ```

   Note: Steps 1, 2, and 3 are optional. You could source a routed design instead.

3. Use the `createVirtualPartition` command to identify interface logic for all the partitions.

   ```
   createVirtualPartition -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.

4. (Optional) Set `trialRoute` mode to honor virtual partitions. Instead of routing the whole chip, the software routes only the nets related to interface paths between the partitions that are masked by virtual partitioning.

```
setTrialRouteMode -honorVirtualPartition true
```

The `createVirtualPartition` command automatically sets this option, so you do not need to set it again. If you want to unset it, specify `-honorVirtualParition false`.

5. Derive timing budgets.

```
deriveTimingBudget -trialIPO
```

The command rebuilds the timing graph as part of its operation.

6. (Optional) Evaluate the budgeting results.

```
justifyBudget -pin pinABC Partition1
```

7. Save the timing budgets.

```
saveTimingBudget
```

8. Clear the virtual partition marking after you save the timing budgets.

```
clearVirtualPartition
```

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 `setThresholdBudgetRatio` 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 `setThresholdBudgetRatio` 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 `setThresholdBudgetRatio -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.

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.

   ```
   setThresholdBudgetRatio -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
```

```
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
Prn_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: Mode1_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:

- 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:

```
setThresholdBudgetRatio -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_multicyle_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_mulicycle_path 3 -setup -from FF1 -to my_partition/FF3/D
set_mulicycle_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_mulicycle_path 2 -from clk_v1 -through in1 -setup -to FF4/D
```

Case 4

Two virtual clocks are created:

- Chip-level exceptions:

  ```
  set_mulicycle_path 2 -setup -from FF1 -to my_partition/FF4/D
  set_false_path -from FF1 -to my_partition/FF3/D
  set_mulicycle_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_mulicycle_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.

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:

$SD/TD * AT = BB$(neg)

For paths with negative slack, the proportioning formula for a hold check (min budgeting) is:

$SD/TD * (AT + HT) = BB$(neg)

Note: If $AT + HT$ is less then 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:

$SD + SD/TD * PS = BB$ (pos)

For paths with positive slack, the proportioning formula for a hold check (min budgeting) is:

$SD - SD/TD * PS = BB$ (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.

Example 19-1 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:

Budget for block A = 3/6 * 5 = 2.5 ns

Budget for block B = 2/6 * 5 = 1.67

Budget for net C = 0.83

Output delay at A = Budget for block B + Budget for net C

Input delay at B = Budgets for blocks A + Budget for net C

Example 19-2 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:

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 `setThresholdBudgetRatio` 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. `setThresholdBudgetRatio 0.05`

  b. `deriveTimingBudget SH17 SH25`

  c. `saveTimingBudget -snapBudget -dir` *`ptn`* `-pt`

Example 19-3 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:

Budget for block B = 0.1/8.1 * 5 = 0.06

Budget for block A = 6/8.1 * 5 = 3.70

Budget for net C = 1.24

The software saves the following budgets when you use the `setThresholdBudgetRatio` command with a ratio of `0.05`, and the `saveTimingBudget` command with the `-snapBudget` parameter.

Budget for block B = snapped to 5% of 5 ns = 0.25 ns

The remaining allowed time = 5 – 0.25 = 4.75 ns

Budget for block A = 6/8 * 4.75 = 3.56

Budget for net C = 2/8 * 4.75 = 1.19

- The `setThresholdBudgetRatio –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 `setThresholdBudgetRatio -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 delay through the net between partition port and pin of the gate on the path. For input ports it is the delay between partition port and an input of the first gate on the path. For output pins, it is the delay between output of the last gate on the path and an output port.

- `RC Adjustment (RC)`

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 `setThresholdBudgetRatio -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 `setThresholdBudgetRatio -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
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 739, 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(fixDel)= 0.000
Clock Skew(clkSkew): 0.000
```

```
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
+ 0.000 - 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   Arrival  Required Slew    Load    Instance
                               Time    Time                              Location

 ---------------------------------------------------------------------------------

             clk1 ^                    0.000    1.090    0.000   0.007

  i_sub1/    CP ^ -> Q ^ df1q  0.182   0.182    1.272    0.063   0.005   (43.12,
 sub_FF                  x1                                             366.80)

 buf1        A ^ -> Z ^  bfx0  0.455   0.637    1.727    1.003   0.040   (43.96,
                                                                         398.16)

 i_sub2/     D ^         df1q  0.004   0.641    1.731    1.003   0.040   (43.12,
 sub_FF                  x1                                             766.64)

 ---------------------------------------------------------------------------------
```

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 {losdclko_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`

- `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`

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 the 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)
```

```
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 */
```

20

RC Extraction

- Overview on page 748
- Before You Begin on page 750
- Native RC Extraction on page 751
- Generating a Capacitance Table on page 755
  - Inputs for Generating a Capacitance Table on page 755
  - Capacitance Table Generation Flow on page 756
  - Output of Generating a Capacitance Table on page 757
  - Generating Capacitance Table With Specified Scaling Factors on page 759
- Reading a Capacitance Table on page 761
- Correlating Native Extraction With Sign-Off Extraction on page 762
  - Comparing the ICT File with the Technology File on page 763
  - Correlating SPEF Files Using the Ostrich Utility on page 763
  - Comparing SPEF Files Using a Perl Script on page 766
  - Defining the Scaling Factor on page 769
- Sign-Off Extraction Using QRC on page 770
  - Inputs for QRC Sign-Off Extraction on page 771

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 751.

- Native Sign-Off RC Extraction

  Provides full chip or incremental sign-off extraction support for the sign-off timing and SI optimization flow. Performing native sign-off extraction provides the following benefits:

  - No need to create the cell library with the `runLibGen` command.
  - No data translation required for the routed and parasitic file.
  - No need to dump a LEF or DEF file, which removes any name mapping issues that result through the LEF or DEF interface.
  - Increased performance and reduced run times.

  Note: Native sign-off extraction requires the following two licenses: SOCE-XL (`SOC_Encounter_XL`) and QRC-XL (`Encounter_QRC_Extraction_XL`). Encounter accesses these licenses when running the `extractRC` command.

  For more information, see "Native Sign-Off RC Extraction" on page 752.

- Standalone Sign-Off RC Extraction using QRC

  Note: Fire & Ice® QX merged with Assura™ RCX to form the new QRC standalone sign-off extraction engine. As a result, Encounter will now call QRC to perform all standalone sign-off extraction operations. Fire & Ice QX will continue to be supported through this release, but will be removed in the next major release of the software.

  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 770.

The following command supports native sign-off incremental extraction:

- `setExtractRCMode -signoff [-fullchip]`

  where:

  The `-signoff` parameter performs native sign-off extraction in either full-chip or incremental extraction mode. Encounter determines which mode to use based on the extent of the logical and physical design changes in the design.

  The `-fullchip` parameter forces the software into full-chip extraction mode. When selected, Encounter performs only full-chip extraction.

The following table summarizes the type of extraction used during the design process.

Extraction Type	When
Native (Default)	Used during optimization both before and after clock tree synthesis.
Native (Detailed)	Used during post-route optimization.
Native Sign-Off RC Extraction	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 are used to multiply 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) extraction—Creates Resistance Capacitance Database (RCDB) then generates an equivalent SPEF file.
- Standalone QRC Extraction—Creates Standard Parasitics Extraction File (SPEF).

Specifying Temporary File Locations

You can specify a temporary file location for native sign-off 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.

  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.

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 generate it by using IceCaps, a built-in functionality of Fire & Ice. 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 755.
3. Read in the capacitance table values. For more information, see "Reading a Capacitance Table" on page 761.
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 762.

Native Sign-Off RC Extraction

There are two modes for native sign-off RC extraction:

- Full chip

  Performs a native sign-off extraction on the complete design.

- Incremental

  Performs a sign-off 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.

The following figure shows the native sign-off RC extraction flow.

Requires a NanoRoute-routed design.
Note: Incremental extraction is not dependent on NanoRoute except for the initialization of the internal extraction database. Running NanoRoute in full-chip or incremental mode automatically initializes the internal extraction database.

1. Load the design into Encounter. Refer to the *Importing and Exporting Designs* chapter in the *Encounter User Guide*.

2. Run native sign-off extraction using the `extractRC` command. The first extraction that Encounter performs is always full-chip extraction. Encounter performs all subsequent extractions in either full-chip or incremental mode, depending on the amount of changes in the design.

3. Perform the logical and physical design changes by calling the ECO (or other) commands. See *Incremental Extraction Support for Design Changes* for more information.

4. Run native sign-off extraction using the `extractRC` command. The shaded area specifies the internal operations of the `extractRC` command. If the logical and physical design changes are extensive, Encounter may choose to perform a full-chip extraction regardless of whether the software is in incremental extraction mode.

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 -signoff
  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 -signoff
  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
  setSIMode -extractionEngine incrementalQRC
  optDesign -postRoute -si
  ```

Generating a Capacitance Table

A capacitance table enables more accurate extraction results. A capacitance table contains capacitance and resistance values. The capacitance values in the capacitance table are calculated using 2-D and 3-D field solvers. The resistance values are defined in the process technology file. Each technology requires one capacitance table. To consider process corner variations, you can generate three capacitance tables, one for each process corner. You can then use the `defineRCCorner` command to specify the extraction values to be used during timing analysis.

Important

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. When capacitance tables are present in the technology file, Encounter will not use the dielectric and thickness values.

Inputs for Generating a Capacitance Table

To generate a capacitance table, you specify the fabrication process information in an ICT file. 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 either a constant sheet resistivity value or value-width pairs as a piecewise linear function. When you generate a capacitance table with width-based sheet resistance, the resistance values are generated for each layer.
- 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.
- The names of the wells.

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 759.

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 three separate capacitance tables using the ICT files. Use the `generateCapTbl` command or the `generateCapTbl` standalone executable (recommended) for each of the ICT file to generate a capacitance table and replace the rough estimate of the parasitic parameters in the technology file with accurate data from a 2-D and a 3-D field solver.

Important

Generating a capacitance table is CPU-intensive and can take a full day to run for newer technologies. Use the `generateCapTbl` standalone executable which can be found in the `bin` directory of your Encounter hierarchy. The standalone executable runs independent of Encounter; it has the same syntax as the `generateCapTbl` command.

Output of Generating a Capacitance Table

Encounter creates an extended capacitance table for extraction. The extended capacitance table is created using information in the LEF file. The 3-D tables are generated using either the Cadence field solver (Coyote) or QuickCap. QuickCap is not a Cadence product. It requires a separate license from Random Logic Corporation.

Extended 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 following sample files show portions of extended capacitance tables for Coyote and QuickCap.

Example 20-1 Extended Capacitance Table

```
PROCESS_VARIATION ...
# THREE_PROCESS_CORNERS
LAYER M1
    MinWidth 0.12000
  # MinSpace 0.12000
  # Height 0.91500
    Thickness 0.20840
    TopWidth 0.17500
    BottomWidth 0.13100
    WidthDev 0.00000
    Resistance 0.1365 0.131 0.1282 0.162 0.1249 0.185 0.1179 0.255 0.1123
        0.395 0.1114 0.435 0.1104 0.495 0.1072 0.735
    ThermalC1 0.00324
    ThermalC2 0.00000
END
```

```
Layer M2
...
LAYER M3
...
LAYER M4
...
LAYER M5
...
LAYER M6
...
VIA CONT
    TopLayer M1
    BottomLayer POLY
    Resistance 12.50000
END
VIA VIA_1
...
VIA VIA_2
...
VIA VIA_3
...
VIA VIA_4
...
VIA VIA_5
...
END_PROCESS_VARIATION
...
EXTENDED_CAP_TABLE ...
# SolverExe: /abc/def/ghi/install/sun4v/tools/bin/coyote
# Solver Type: coyote
        1.01            6            6            1     0.870005
    0.290005          3.9            0            0            5
           3            1            5            0            1
           0            0            0          120          720
         440          300          160          120          140
         140          280          420          560          700
           0         0.12         0.72         0.44          0.3
        0.16         0.12    1.3888889    2.2727273    3.3333333
        6.25    8.3333333         0.14         0.14         0.28
        0.42         0.56          0.7         1000 0.0001126342
           5            0    1.3888889    2.2727273    3.3333333
```

```
        6.25    8.3333333  5.76285e-06  1.55548e-05 2.876825e-05
 6.74432e-05 9.558965e-05            5            0    1.3888889
...
END_EXTENDED_CAP_TABLE
```

Example 20-2 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 20-3 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 20-4 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 tables 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.

- `-res` *`resistanceFactor`*

  Specifies the resistance scaling factor.

Reading a Capacitance Table

To consider process corner variations, you can read three capacitance tables, one for each process corner. You can also read two capacitance tables, one for the best and one for the worst case corner. If you read in a single capacitance table, the software uses that 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.

Carefully observe the command log for any possible error messages. The number of metal layers specified in the PCS 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.

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 770.
2. Generate a capacitance table file using the ICT process file(s). For more information, see "Generating a Capacitance Table" on page 755.
3. Read in the generated capacitance table. For more information, see "Reading a Capacitance Table" on page 761.
4. Generate a SPEF file using the Timing `extractRC` and `rcOut` commands.
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 763.

   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 766.
6. Specify these scaling factors using `setRCFactor` before running native (detailed) extraction. For more information, see "Defining the Scaling Factor" on page 769.
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.

Comparing the ICT File with the Technology File

If the via and sheet resistance values defined in the ICT file are different from those defined in the LEF file, you might not get correlation between native and sign-off extraction. To correct the sheet resistance values, use a script called `correctTechRes.pl`. You can run the script from the UNIX prompt. This script resides in the `$ENCOUNTER/bin` directory.

The script compares the sheet resistance values in the ICT file and technology file derived from the LEF information, and if the values are different, the values in the ICT file are used to update the technology file.

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.

To correlate the SPEF files, complete the following steps:

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.

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.

8. 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.

Note: You may want to set the `minCap` value higher than the default 10 fF in order to correlate the results better for nets that have much higher total capacitance.

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 767.
- 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.

Report File Example

```
#Ref. Cap file:     lambda_aftercrosstalkfix.spef
#Cmp. Cap file:     lambda_detailrc.spef
#-------------------------------------------------------------------#
```

```
#                        Total Capacitance Statistics                 #
#-----------------------------------------------------------------------#
#Total Capacitance range considered:                0.0100 -    5.0000 [pF]
#Total number of nets:                              418399
#Total number of nets with nonzeoro 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_res_scale) "NUMBER"
  ```

- Use the following Tcl command:

  ```
  setRCFactor -defcap scaleFactor -detcap scaleFactor -xcap scaleFactor
  -res 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 is also used with native extraction to generate an RC scaling factor. For more information, see "Correlating Native Extraction With Sign-Off Extraction" on page 762.

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.

21

Calculating Delay

- Overview on page 774
- Data Preparation on page 774
  - Operating Conditions on page 774
  - ECSM Libraries on page 775
  - Using IPDB-based ECSM Libraries on page 775
- Delay Calculation Modes and Related Controls on page 777
- Choosing A Delay Calculation Engine on page 778
- Running Delay Calculation on page 778

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™ delay calculator engine

  Used to provide quick delay calculation for design prototyping, optimization, and general timing analysis.

- SignalStorm® nanometer 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.
- SignalStorm also generates a report file, which provides delay information, the signal slew rate, and interconnect loading information.

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 automatically translates the Liberty (`.lib`) timing library data into ECSM format when you choose to use SignalStorm for delay calculation. When translating the `.lib` timing data, Encounter uses the current configuration of minimum and maximum timing libraries and operating conditions specified. The ECSM data is stored in memory only, not to disk.

Using IPDB-based ECSM Libraries

If you have existing ECSM/IPDB libraries from SPICE characterization, you can still load them into Encounter using backward compatible options.

Note: Using the automatic translation mode and loading ECSM IPDBs are mutually exclusive; they cannot be used together in the same design.

➤ Set the following mode variable in your Encounter initialization script (*current_working_directory/*`.encrc file`):

```
set dbgSignalStormCompatibleMode 1
```

The `setDelayCalMode` command parameters and forms used to load ECSM IPDBs become available. In the automatic translation mode (`dbgSignalStormCompatibleMode 0`), these parameters and forms are disabled.

Loading ECSM Libraries

You can also use the ECSM/IPDB-based mode to load an existing ECSM library.

1. Choose *Design - Design Import* and click on the *Misc.* tab.

   This opens the Misc. page of the Design Import form.

2. Specify the name of the ECSM library in the *ECSM DB* field in the SignalStorm Library panel.

3. Click *Load*.

You can also load ECSM libraries using the `loadECSMDB` command:

```
loadECSMDB ECSM_DB_NAME
```

Or, you can add the following line to the `.conf` file:

```
rda_Input(ui_sigstormlib) "myECSM.ipdb"
```

Selecting ECSM Operating Corners

If you load an existing ECSM library into Encounter using the ECSM/IPDB-based mode, you must set the operating conditions to use for delay calculation.

1. Choose *Timing – Specify Analysis Condition – Specify Delay Calculation Mode*.

   The Delay Calculation Mode form displays.

2. Select *SignalStorm* for the calculation mode.

3. Specify the ECSM process corners to use for maximum and minimum delay calculation in the *Slow Corner Process Name* and *Fast Corner Process Name* fields.

   Note: Click the *Check Available Process Names* button to find out which process corners have been loaded into the library.

4. Click *OK* or *Apply*.

You can also specify operating corners using the `setDelayCalMode` command:

```
setDelayCalMode -engine signalStorm -slowCorner ecsmProceesCornerName
       -fastCorner ecsmProceesCornerName
```

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.

	Fanouts >= 1,000	**1,000 > Fanouts >= 100**	**100 > Fanouts and Elmore > 5 ps**	**5 ps > Elmore**
Algorithm	Uses the default delay parameters.	Uses simplified delay calculation mode.	Uses full RC delay calculation mode.	Uses simplified delay calculation mode.
Cell Delay	Uses lumped C lookup with a default value of 0.5 pF. The value is controlled by the `setDefaultNetLoad` command.	Uses lumped C lookup from total parasitics on the net.	Uses full RC.	Uses lumped C lookup from total parasitics on the net.
Wire Delay	Uses the default value of 1 ns. The value is controlled by the `setDefaultNetDelay` command.	Uses Elmore delay.	Uses full RC.	Uses Elmore delay.
Driver Slew Rates	Uses the default value of 0 ps. The value is controlled by the `setInputTransitionDelay` command.	Uses lumped C lookup from total parasitics on the net.	Uses full RC.	Uses lumped C lookup from total parasitics on the net.
Interconnect Slew Degradation	None	None	Uses full RC. (Slews are degraded across interconnect.)	None
Controls	Fanout threshold is controlled by the `setUseDefaultDelayLimit` command. The default value is 1,000 fanouts.	Fanout threshold is controlled by the `-elmore` option in the `buildTimingGraph` 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 displays.

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.

Running Delay Calculation

1. Choose *Timing – Calculate Delay*.

2. (Optional) Select *Ideal Clock* to force the clock nets on each flip-flop input pin to use 0.

   Note: Use the `setAnalysisMode` command parameters to control how timing and delay calculation are performed before and after clock tree synthesis

3. (Optional) Specify an SDF filename in which to save the delay information in the *SDF Output File* field. By default, Encounter saves the delay information in the default SDF file named `topModule.sdf`.

4. Click *OK* or *Apply* to calculate delay.

Alternatively, you can run delay calculation by issuing the following text command:

```
delayCal
```

For example, the following command calculates both interconnect and cell delays, and saves the results to the SDF file named `TOPCHIP_SP.sdf`:

```
delayCal -sdf TOPCHIP_SP.sdf
```

22

Timing Analysis

- Overview on page 780
- Timing Analysis Features on page 781
- Before You Begin on page 782
- Timing Analysis Results on page 783
- Reading Timing Libraries on page 783
- Reading Timing Constraints on page 784
- Setting Operating Conditions on page 788
- Calculating Clock Latency on page 789
- Defining RC Corners on page 790
- Specifying Timing Analysis Modes on page 792
- Clock Path Pessimism Removal on page 815
- Analyzing Timing Problems on page 821

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 departure 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, such as TLF, `.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

.

setAnalysisMode	Clock Propagation	Clock Latency
`-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 `set_clock_latency` command is overridden by overlapping `set_propagated_clock` constraints.

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 values

  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 788.

- 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 timing analysis. You can set the RC corner as best, typical, or worst. The software uses these values for late and early setup and hold times during timing analysis. For more information, see "Defining RC Corners" on page 790.
- 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 792.

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.

Timing analysis generates the following reports, which you can use to determine whether your design operates at the required clock frequency:

- Timing slack report
- Timing violation report
- Transition timing violation report
- Clock waveform report

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 *MaxTiming 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 performs bestcase-worstcase analysis.

If you read in both Min and Max library groups, the software uses Max library delay values for setup and Min library delay values for hold time calculations. 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
```

Specifying Different Timing Libraries for Setup and Hold Checks

You use the `setTimingLibrary` command to change the library that is used for minimum delay and maximum delay path calculations. The default value of the `setTimingLibrary` command is as follows:

```
setTimingLibrary –max Max –min Min
```

The command uses the delay values from Max library during setup checks and the delay values from the Min library for hold checks. To perform setup analysis at the BC corner and hold analysis at the WC corner, use the following command:

```
setTimingLibrary –min Max –max Min
```

You can accordingly set the operating condition by using the following command:

```
setOpCond -maxLibrary stdcellbest -max bccom -minLibrary stdcellwst -min wccom
```

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
- get_libs
- get_pins
- get_ports
- group_path
- remove_from_collection
- remove_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`

Setting Operating Conditions

Integrated circuits display performance differences depending on the fabrication PVT characteristics. The nominal values for these parameters and the corresponding delay information are contained in the technology library. Each technology library contains one or more operating condition. Each operating condition is identified by name and specifies the PVT characteristic. 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.

- `-forceLibrary`

  Note: This parameter provides backward compatibility with previous versions of Encounter. Cadence does not recommend the use of this parameter.

  Sets the operating condition for each library. Use this parameter to specify different operating conditions for different libraries in a single session. For example, consider the following commands:

  ```
  setOpCond -max max -forceLibrary wcTestLib
  setOpCond -min min -forceLibrary bcTestLib
  setOpCond -max max -forceLibrary wcTestLib1
  setOpCond -min min -forceLibrary bctestLib1
  ```

```
setOpCond -max max -forceLibrary wcTestLib2
setOpCond -min min -forceLibrary bcTestLib2
```

In this case, the software sets the `max` operating condition for the cells defined in the library `wcTestLib`, where `max` is defined in the `wcTestLib`. Similarly, the operating condition of the cells defined in the `bcTestLib2` library is set to `min`, where `min` is defined in the `bcTestLib2` library. In setup analysis, the cells in each library, the `wcTestLib`, `wcTestLib1`, and `wcTestLib2` libraries use the `max` operating condition defined in the corresponding libraries. Similarly, in hold analysis the cells in `bcTestLib`, `bctestLib1`, and `bcTestLib2` use the corresponding `min` operating conditions.

- `-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 max -maxLibrary wcTestLib -min min -minLibrary bcTestLib
  ```

  In this case, the software uses the `max` operating condition defined in `wcTestLib` for setup analysis and the `min` operating condition defined in `bcTestLib` for hold 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 on the design.

The Encounter software sets the clock latency for various combinations of analysis mode settings as follows:

- `setAnalysisMode -skew true -clockPropagation sdcControl` (Default Setting)
  - If the `set_propagated_clock` command is specified in the timing constraints file, the clock latency is computed from the netlist (propagated mode).
  - If no `set_propagated_clock` is specified and `set_clock_latency` command is in the timing constraint file, the clock latency specified in the constraint is used (ideal mode).
  - 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, no 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 setup and hold checks.

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.

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 794

  In single analysis mode, delay values are scaled based on one operating condition.

- Best-Case Worst-Case (BC-WC) Timing Analysis Mode on page 798

  The BC and WC operating conditions or corners are vastly different. Therefore two paths on a chip cannot use two different operating conditions at the same time for setup or hold analysis.

  In BC-WC analysis mode the Encounter software checks the design for two extreme operating conditions. 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 809

  OCV is the small difference in the operating parameter value across the chip. Each timing arc in the design can have a minimum and a maximum delay to account for the on-chip process, voltage and temperature variation.

  In OCV mode, the software calculates the delay for one path based on maximum operating condition while calculating the delay for another path based on minimum operating condition for setup or hold checks.

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 path are the shortest and the longest paths respectively.

The following figure shows setup check with late (shown in solid line) launch clock and early capture clock (shown in dotted line).

Figure 22-1 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 22-2 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:

launch clock late path + data clock late path <= capture clock early path + clock period – 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:

launch clock early path + data clock early path >= capture clock late path + hold

Example 22-1 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:

Launch clock late path delay = 0.8 + 0.6 =1.4

Data late path delay = 3.6

Capture clock early path delay = 0.8 + 0.5 = 1.3

Setup = 0.2

Data arrival time = 1.4 + 3.6 = 5

Data required time = 4 + 1.3 - 0.2 = 5.1

Slack = 5.1 - 5 = 0.1

Example 22-2 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:

Launch clock early path delay = 0.6 + 0.4 = 1.0

Data early path delay = 1.0

Capture clock late path delay = 0.6 + 0.3 = 0.9

Hold = 0.1

Data arrival time = 1 + 1 = 2

Data Required Time = 0.1 + 0.9 = 1

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
   ```

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
```

For zero slack value in a setup check, the following condition should be met:

Launch clock_latepath (max) + data_latepath (max) <= Capture clock_earlypath (max) + clock_period – setup

Where, max is the delay at the WC operating condition.

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)	`set_clock_latency –source –late –max` *`value`* Or, `set_clock_latency –source –late` *`value`*

Clock Path (Operating Condition)	Constraint Used
Capture clock early path (max)	`set_clock_latency –source –early –max` *`value`* Or, `set_clock_latency –source –early` *`value`*

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)	`set_clock_latency -max` *`value`*
Capture clock early path (max)	`set_clock_latency -max` *`value`*

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
```

For zero slack value in a hold check, the following condition should be met:

Launch clock_earlypath (min) + data_earlypath (min)>= Capture clock_latepath (min) + hold

Where, min is delay at the BC operating condition.

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)	`set_clock_latency –source –early –min` *`value`* Or, `set_clock_latency –source –early` *`value`*

Clock Path (Operating Condition)	Constraint Used
Capture clock late path (min)	set_clock_latency –source –late –min value Or, set_clock_latency –source –late value

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)	`set_clock_latency -min` *`value`*
Capture clock late path (min)	`set_clock_latency -min` *`value`*

Note: You can also use one library containing two operating conditions in this mode.

Example 22-3 Setup Check in BC-WC Timing Analysis Mode

The following shows the setup check on the path from FF1 to FF2.

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 22-4 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.

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
   ```

5. Set the analysis mode to hold and propagated clock mode.

   ```
   setAnalysisMode –checkType hold
   ```

6. Generate the timing reports for hold.

   ```
   report_timing
   ```

Using setTimingDerate with BC-WC Analysis Mode

When you use the `setTimingDerate` command in BC-WC timing analysis mode, the Encounter software uses scaling factors to consider on-chip variation for each BC and WC corner.

The following figure shows a graph of the clock and data delay.

A timing path around the WC and BC operating conditions or corners might have setup or hold violations if the data or clock delay values vary as a result of on-chip variation.

A timing path has setup violations if the delay value of the capture clock is less as compared to its value at the WC corner. In the figure above, the corner 1 represents the setup violation. The slack value at corner 1 is more pessimistic than the WC corner. To specify the pessimism and emulate minimum delay for capture clock, you can scale the delay by a scale factor of less than 1.

Similarly, corner 3 represents setup violation, and corners 2 and 4 represent hold violations.

The design might have the following setup and hold violations around all four corners due to on-chip variation:

- Corner 1 has setup violations if the delay value of the capture clock is less than its value at the WC corner. To specify the pessimism, you scale the capture clock delay value by a scale factor of less than 1.

Note: Alternatively, you can use the `setTimingDerate` command to scale data path and launch clock delay by a value greater than 1 instead of scaling the capture clock delay by a value less than 1.

- Corner 2 has hold violations if the delay value of the data or launch clock is less than their value at the WC corner. To specify the pessimism, you scale the data and launch clock delay by a scale factor of less than 1.
- Corner 3 has setup violations if the delay value of the data or launch clock is more than their value at the BC corner. To specify the pessimism, you scale the data and launch clock delay by a scale factor of more than 1.
- Corner 4 has hold violations if the delay value of the capture clock is more than its value at the BC corner. To specify the pessimism, you scale the capture clock delay value by a scale factor of more than 1.

The following table summarizes the delay values used for all four corners:

Corner	Violation	Data Path Delay	Launch Clock Delay	Capture Clock Delay
(1) WC	Setup	WC	WC	WC x scaling value<1
(2) WC	Hold	WC x scaling value<1	WC x scaling value<1	WC
(3) BC	Setup	BC x scaling value>1	BC x scaling value>1	BC
(4) BC	Hold	BC	BC	BC x scaling value>1

Note: You can use the `setTimingDerate –data –clock –min –max –early –late` command to scale the delays. The software scales both cell and net delays.

The following table lists the parameters of the `setTimingDerate` command that you can set for scaling the delay values for data, launch and capture clock paths:

Check	Data	Launch Clock	Capture Clock
Setup	-late -max -data	-late -max -clock	-early -max -clock
Hold	-early -min -data	-early -min -clock	-late -min -clock

By default, setup analysis is done at WC corner and hold analysis is done at BC corner. To check setup at BC and hold at WC, change the default values of the `setTimingLibrary` command. For example around the BC corner, you can use the following command:

```
setTimingLibrary –min Min –max Min.
```

Example 22-5 Setup and Hold Checks in Corner 3 and Corner 4

The following figure shows the delays at BC corner.

The setup and hold checks are calculated for the path from FF1 to FF2.

The following values are assumed in this example:

setTimingDerate –max –late 1.2 (corner 3) for setup check
setTimingDerate –min -clock –late 1.1 (corner 4) for hold check
clock source latency = none
Clock mode = propagated clock mode
Wire delay = 0
Clock period = 4

The software computes the slack for setup check as follows:

Launch clock late path delay = 0.9 x 1.2 = 1.08

Data late path delay = 2.3 x 1.2 = 2.76

Capture clock early path delay = 0.5 + 0.3 = 0.8

Setup = 0.2

Data arrival time = 1.08 + 2.76 = 3.84

Data required time = 4 + 0.8 – 0.2 = 4.6

Slack = 4.6 – 3.84 = 0.76

The software computes the slack for hold check as follows:

Launch clock early path delay = 0.5 + 0.4 = 0.9

Data early path delay = 1.0

Capture clock late path delay = 1.1 x 0.8 = 0.88

Hold = 0.1

Data arrival time = 1.0 + 0.9 = 1.9

Data required time = 0.88 + 0.1 = 0.98

Slack = 1.9 – 0.98 = 0.92

The following table shows the parameters that were scaled in this example:

Corner	Violation	Data	Launch Clock	Capture Clock
3	Setup	Delay x 1.2	Delay x 1.2	Delay
4	Hold	Delay	Delay	Delay x 1.1

To perform timing analysis to report setup and hold violations for BC setup and BC hold corner, complete the following steps:

1. Read in the fast library by including the following line in the configuration file:
   `set rda_Input(ui_timelib,min) "${libDir}/fast.libî`
2. Scale the late clock and data paths for setup.
   `setTimingDerate –max –late 1.2`
3. Scale the late paths for hold. This is the capture clock path.
   `setTimingDerate –min –late 1.1`
4. Use the min library for both setup and hold calculations.
   `setTimingLibrary -max Min -min Min`
5. Select the BC operating condition from the fast library.
   `setOpCond best –library fast`
6. Set the analysis mode to setup and propagated clock mode.

```
setAnalysisMode -checkType setup –skew true -clockPropagation sdcControl
```

7. Generate the timing reports for setup.

```
report_timing
```

8. Set the analysis mode to hold and propagated clock mode.

```
setAnalysisMode -checkType hold
```

9. Generate the timing reports for hold.

```
report_timing
```

Example 22-6 Setup and Hold Violations in Corner 1 and Corner 4

The following script sets the scaled on-chip variation of 20 percent near the WC and 10 percent for BC corner for setup and hold paths respectively:

```
set rda_Input(ui_timelib,min) "${libDir}/fast.libî
set rda_Input(ui_timelib,max) "${libDir}/slow.libî
setTimingDerate -max -early 0.8 -late 1.0
setTimingDerate -min -early 1.0 -late 1.1
setTimingLibrary -max Max -min Min
setOpCond –min Best –minLibrary fast –max Worst –maxLibrary slow
setAnalysisMode -checkType setup –skew true -clockPropagation sdcControl
report_timing
setAnalysisMode -checkType hold
report_timing
```

The following table shows the parameters that were scaled in this example:

Violations	Data	Launch Clock	Capture Clock
SETUP	Delay	Delay	Delay x 0.8
HOLD	Delay	Delay	Delay x 1.1

Example 22-7 Setup and Hold Violations in Corner 1 and Corner 2

The following script sets the scaled on-chip variation of 20 percent near the WC corner for setup and hold paths:

```
set rda_Input(ui_timelib,max) "${libDir}/slow.libî
setTimingDerate -early 0.8 -late 1.0
setTimingLibrary -max Max -min Max
setOpCond Worst –library slow
setAnalysisMode  -checkType setup –skew true -clockPropagation sdcControl
report_timing
setAnalysisMode  -checkType hold
report_timing
```

The following table shows the parameters that were scaled in this example:

Violations	**Data**	**Launch Clock**	**Capture Clock**
SETUP	Delay	Delay	Delay x 0.8
HOLD	Delay x 0.8	Delay x 0.8	Delay

Note: The `setTimingDerate` command does not scale delays set using constraints. For example latency values set using `set_clock_latency` or arrival times set using `set_input_delay` are not scaled. You can use `reportTimingDerate` to report current settings.

On-Chip Variation (OCV) Timing Analysis Mode

OCV mode assumes that capture clock, launch clock, and data path can have maximum or minimum delays in setup and hold analysis.

OCV is crucial in the clock insertion part of the flow where you need to consider the minimum delay and maximum delay clock and data signals concurrently using min and max libraries or using derating factors and values from a single library.

To set the timing analysis mode as OCV, use the `-analysisType onChipVariation` parameter of the `setAnalysisMode` command.

Following constraints are supported in OCV mode:

- `set_clock_latency`
- `set_driving_cell`
- `set_drive`
- `set_load`
- `set_resistance`

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.

For zero slack value in a setup check, the following condition should be met:

Launch clock_latepath (max) + data_latepath (max) <= Capture clock_earlypath (min) + clock_period –setup

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)	`set_clock_latency –source –late –max` *`value`* Or, `set_clock_latency –source –late` *`value`*
Capture clock early path (min)	`set_clock_latency –source –early –min` *`value`* Or, `set_clock_latency –source –early` *`value`*

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)	`set_clock_latency -max` *`value`*
Capture clock early path (max)	`set_clock_latency -min` *`value`*

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.

For zero slack value in a hold check, the following condition should be met:

Launch clock_earlypath (min) + data_earlypath (min)>= Capture clock_latepath (max) +hold

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)	`set_clock_latency –source –early –min` *`value`* Or, `set_clock_latency –source –early` *`value`*

Clock Path (Operating Condition)	Constraint Used
Capture clock late path (max)	`set_clock_latency –source –late –max` *`value`* Or, `set_clock_latency –source –late` *`value`*

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)	`set_clock_latency -min` *`value`*
Capture clock late path (min)	`set_clock_latency -min` *`value`*

Example 22-8 Setup Check in OCV Timing Analysis Mode

The following figure shows the setup 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:

Launch clock late path delay (max) = 0.7 + 0.6 = 1.3

Data late path delay (max) = 3.5

Capture clock early path delay (min) = 0.5 + 0.3 = 0.8

Setup = 0.2

Data arrival time = 1.3 + 3.5 = 4.8

Data required time = 4 + 0.8 – 0.2 = 4.6

Slack = 4.6- 4.8 = -0.2

Example 22-9 Hold Check in OCV Timing Analysis Mode

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:

Launch clock early path delay (min) = 0.5 + 0.4 = 0.9

Data early path delay (min) = 1.2

Capture clock late path delay (max) = 0.7 + 0.5 = 1.2

Hold = 0.1

Data arrival time = 0.9 + 1.2 = 2.1

Data required time = 0.1 + 1.2= 1.3

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
      –clockProagation sdcControl
   ```

4. Generate the timing reports for setup.

   ```
   report_timing -late
   ```

5. Generate the timing reports for hold.

   ```
   report_timing -early
   ```

Using setTimingDerate with OCV Analysis Mode

When you use the setTimingDerate command in OCV timing analysis mode, the software uses `-min` with the early paths and `-max` with the late paths. The `setTimingDerate` command affect the following paths in OCV mode:

Violations	Data	Launch Clock	Capture Clock
SETUP	`-max (-late)` `-data`	`-max (-late)` `-clock`	`-min (-early)` `-clock`
HOLD	`-min (-early)` `-data`	`-min (-early)` `-clock`	`-max (-late)` `-clock`

Performing Timing Analysis in OCV Mode With Single Library And Operating Condition

To perform timing analysis in OCV analysis mode, complete the following steps:

1. Read in the minimum and maximum libraries by sourcing the configuration file with the following lines:

   ```
   set rda_Input(ui_timelib,max) "${libDir}/stdcell.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 -library stdcell WCCOM
   ```

3. Set the analysis mode to OCV and propagated clock mode.

   ```
   setAnalysisMode –analysisType onChipVariation –skew true
      –clockPropagation sdcControl
   ```

4. Scale the minimum delays to 80 percent of WCCOM using slow library.

   ```
   setTimingDerate -early 0.8 -late 1.0
   ```

   Note: The minimum delays are used for both setup and hold checks.

5. Generate the timing reports for setup.

   ```
   report_timing -late
   ```

6. Generate the timing reports for hold.

   ```
   report_timing -early
   ```

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.

In the on-chip variation methodology (using the `setAnalysisMode -analysisType BcWc` and the `setTimingDerate` commands or `setAnalysisMode -analysisType ocv`), during setup checks if both the launch clock late path and the capture clock early path share a portion of the clock network, then for the common clock network, a pessimism equal to the difference in maximum and minimum delay values is introduced in the slack values.

When you use the `setAnalysisMode -analysisType BcWc` and the `setTimingDerate` commands, the software calculates maximum delay value by multiplying the delay by the scale value that you set using `setTimingDerate –late`. Similarly, the software calculates the minimum delay value by multiplying the delay by the scale value that you set using `setTimingDerate -early`.

When you use the `setAnalysisMode -analysisType ocv` 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 `setTimingDerate` command.

As shown in Figure 22-3, 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.

Figure 22-3 Example Signal Path

The setup check equation for the example in Figure 22-3 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:

$$t_{1max} + t_{2max} + t_{3max} \leq t_{4min} + t_{pa} - t_{su} + t_{cppr}$$

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 `setTimingDerate` command for setup check delays in Figure 22-3 on page 816 in scaled on-chip variation analysis methodology:

```
setTimingDerate -max –late 1 –early 0.9 –clock
```

With the `setTimingDerate` 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} = 1.0ns * 1.0 - 1.0ns * 0.9 = 0.1ns

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 0.5, and late path has delay of 1ns, a pessimism equal to 0.5 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.

   ```
   setTimingDerate -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
   ```

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	Ce1.938ll	Delay	Arrival Time	Required Time
	clk ^			0.000	0.088
ck_0	A ^ -> Y ^	BUFX2	0.091	0.091	0.178
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

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	Ce1.938ll	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_timing`

  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.

- 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.

23

Debugging Timing Results

- Timing Debug Flow on page 827
- Generating Timing Debug Report on page 828
- Displaying Violation Report on page 828
- Analyzing Timing Results on page 828
- Creating Path Categories on page 832
- Using Categories to Analyze Timing Results on page 838
- Editing Table Columns on page 842
- Cell Coloring on page 844
- Viewing Schematics on page 846

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.

- 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 23-1

Launch and capture latency components are not aligned. Therefore there can be large clock-latency mismatch in this path.

Example 23-2

The cycle adjustment bar in the required time indicates presence of multicycle path.

Example 23-3

Large input delay in an I/O path is represented by the blue bar in the arrival time.

Example 23-4

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

- 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.
- 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 846.
- 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

- 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.

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

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.

Following figure shows categories created by the Create Path Category form.

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                 Total  Passing Failing WNS       TNS
name                     path   Path    path
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.

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:

- `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:

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.

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*. The 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.

24

Optimizing Timing

- Overview on page 848
- Before You Begin on page 848
- Results on page 849
- Performing Optimization Before Clock Tree Synthesis on page 851
- Performing Post-CTS Optimization on page 856
- Performing Postroute Optimization on page 859
- Using Useful Skew on page 864
- Using Virtual Partitioning for Chip-Level Interface Circuit Timing Closure on page 867
- Optimizing Timing in On-Chip Variation Analysis Mode on page 868
- Using Conformal Constraint Designer During Timing Optimization on page 871
- Optimizing Timing Using a Rule File on page 875
- Optimizing Timing When the Constraint File Includes the set_case_analysis Constraint on page 875
- Using the Footprintless Flow on page 875
- Using Cell Footprints on page 876
- Viewing Added Buffers, Instances, and Nets on page 877
- Default Naming Conventions on page 877

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 875.

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.

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.

- Total number of instances added and deleted
- 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).

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   |
+--------------------+---------+---------+---------+---------+- --------+---------+
+----------------+----------------------+---------+
|                |        Real          |  Total  |
|    DRVs        +---------+------------+---------|
|                |    Nr   | Worst Vio  |    Nr   |
+----------------+---------+------------+---------+
|   max_cap      |    3 |   -0.004 |    4 |
|   max_tran     |   0 |   .000 |   0 |
|   max_fanout   |    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*.

Performing Optimization Before Clock Tree Synthesis

- Correcting Violations in Pre-CTS Mode for the First Time on page 852
- Performing Rapid Timing Optimization for Design Prototyping on page 852
- Using Additional Pre-CTS Timing Optimization Parameters on page 852
- Performing Incremental Pre-CTS Optimization on page 854

- Changing Default Settings in Pre-CTS Mode on page 855

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
```

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`
- `macro`
- `flopOrLatch`
- `clkSrc`
- `moduleIOCstr`
- `noModuleIOCstr`
- `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 856.

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
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 856
- Performing Incremental Post-CTS Optimization on page 857
- Changing Default Settings in Post-CTS Mode on page 858

Correcting Violations in Post-CTS Mode

- To optimize timing after the clock tree is built, use the following commands:

  ```
  optDesign -postCTS
  ```

  `optDesign` corrects DRVs, then 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
  ```

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 852.

  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 reclaim area during optimization, use the following commands:

```
setOptMode -reclaimArea true
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 852.

  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:

`setAnalysisMode`	`optDesign` sets `-clockPropagation autoDetectClockTree` and `-clkSrcPath true` 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, 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 859
- Correcting Violations in Postroute Mode on page 861
- Correcting Signal Integrity Violations on page 862

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

Important

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 Fire & Ice® or `starRC` to generate the SPEF, which enables the Encounter extraction engine to match the routes with the SPEF RC graph.

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
  ```

  You can benefit from correcting hold violations before correcting setup violations because hold repairs might cause more setup violations, which you can correct in a later step.

  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.

  The command sets the `setupTargetSlack` value to `0.1`. Hold repair does not degrade slack to less than the `setupTargetSlack` value. You can override this value by specifying `setOptMode -setupTargetSlack` before you run `optDesign`.

- To correct setup and hold violations, use the following commands:

  ```
  optDesign -postRoute
  optDesign -postRoute -hold
  ```

The software repairs a hold violation only if it does not make setup slack worse than the setup target slack on a path.

- 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 true
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 setup optimization based on SDF files, use the following commands:

```
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.

Correcting Signal Integrity Violations

- To correct signal integrity violations when optimizing timing in postroute mode, use the following command:

```
optDesign -postRoute -si
```

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:

`setAnalysisMode`	`optDesign` sets `-clockPropagation autoDetectClockTree` and `-clkSrcPath true`. You can override other parameters.
`setClockDomains`	`optDesign` uses the parameters you specify. The default is all path groups.
`setExtractRCMode`	`optDesign` sets the extraction mode to `-detail`. You cannot change this mode. Ensure that you set the appropriate extraction scale factor.
`setTrialRouteMode`	You can add parameters, but never override the default settings. `optDesign` sets the `-handlePreroute true` parameter.

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 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 Virtual Partitioning for Chip-Level Interface Circuit Timing Closure

The Encounter software provides a top-level interface timing analysis flow to run 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 full timing graph.

To run timing analysis or optimization using the virtual partition flat-top flow, 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 or restore placement data.

   ```
   setPlaceMode -fp -ignoreScan true
   placeDesign
   ```

   or

   ```
   restoreDesign placed.place.gz
   ```

3. Run trial route or restore routing data.

   ```
   setTrialRouteMode -detour false -floorplanMode true
   trialRoute
   ```

   or

   ```
   restoreRoute routed.route.gz
   ```

   Note: Steps 1, 2, and 3 are optional. You can source a routed design instead.

4. Use the `createVirtualPartition` command to identify interface logic for all the partitions.

   ```
   createVirtualPartition -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.

5. Run `timeDesign` or `optDesign`.

```
optDesign -preCTS -incr
```

or

```
timeDesign -preCTS
```

6. Clear the virtual partition marking after you complete timing optimization or analysis.

   `clearVirtualPartition`

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.

For more information on OCV and MMMC, see

- On-Chip Variation (OCV) Timing Analysis Mode on page 809
- Performing Multi-Mode Multi-Corner Timing Analysis and Optimization on page 1013

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   \
            -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]
```

Important

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 871
- Integrated Approach on page 872

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:

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 timng 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.
- 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 876.

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`:

- `setBufFootPrint`
- `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
FE_MDBC	Instance added by multi-driver net buffering	`optDesign`
FE_MDBN	Net added by multi-driver net buffering	`optDesign`
FE_OCP_DRV_C	Instance added by DRV fixing	`optDesign`
FE_OCP_DRV_N	Net added by DRV fixing	`optDesign`
FE_OCP_RBC	Instance added by rebuffering	`optDesign`
FE_OCP_RBN	Net added by rebuffering	`optDesign`
FE_OCPC	Instance added by critical path optimization	`optDesign`
FE_OCPN	Net added by critical path optimization	`optDesign`
FE_OFC	Buffer instance added by rule-based buffer insertion	`insertRepeater`/ `optDesign`
FE_OFN	Buffer net added by rule-based buffer insertion	`insertRepeater`/ `optDesign`
FE_PHC	Instance added by hold time repair	`optDesign`

Prefix	Description	Command
FE_PHN	Net added by hold time repair	optDesign
FE_PSBC	Instance added by buffer insertion in optDesign -postRoute	optDesign
FE_PSBN	Net added by buffer insertion in optDesign -postRoute	optDesign
FE_PSC	Instance added by postroute setup repair	optDesign
FE_PSN	Net added by postroute setup repair	optDesign
FE_RC	Instance created by netlist restructuring	optDesign
FE_RN	Net created by netlist restructuring	optDesign
FE_USC	Instance added during useful skew optimization	optDesign

25

Interactive ECO

- Overview on page 880
- Before You Begin on page 880
- Results on page 880
- Adding Buffers on page 880
- Changing the Cell on page 883
- Deleting Buffers on page 884
- Displaying Buffer Trees on page 886
- Running ECO Placement on page 888
- Naming Conventions for Interactive ECO on page 889
- Comparing Physical Design Data on page 889

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.

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. 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.

4. In the *Terms* field, choose of the following:

- To connect the added buffer to drive all the receivers, specify *All Terms*. 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 Terms*. This provides full flexibility for building an arbitrary buffer connection.

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*.

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.

   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) Evaluate how the changing the selected cells will affect timing, click the *Eval* (for a single selection) or *Eval All* (for all possible selections) buttons.

6. (Optional) To legalize placement of ECO changes, click *Do Refine Placement*.

7. 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) To evaluate how deleting the selected instance will affect timing, click the *Eval* button.

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.

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.

x y	Specifies the coordinates of the instance.
orientation	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.

- ADDPIN statement

  ADDPIN *netName instName pinName*

netName	Specifies the net from which the pin is added.
instName	Specifies the instance name of the added pin.
pinName	Specifies the name of the added pin.

- DELPIN statement

  DELPIN *netName instName pinName*

netName	Specifies the net from which the pin is deleted.
instName	Specifies the instance name of the deleted pin.
pinName	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*

layerName	Specifies the layer name of wire segment added to the net.
width	Specifies the width of the wire segment added to the net.
x1 y1	Specifies the left or bottom coordinate of wire segment added to the net.
x2 y2	Specifies the right or top coordinate of wire segment added to the net.

- `DELWIRE` statement

  `DELWIRE` *layerName width x1 y1 x2 y2*

layerName	Specifies the layer name of wire segment deleted to the net.
width	Specifies the width of the wire segment deleted to the net.
x1 y1	Specifies the right or top coordinate of wire segment deleted from the net.
x2 y2	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*

botLayerName	Specifies the bottom layer name of the via added to the net.
bx1 by1	Specifies the lower-left coordinate of bottom layer of the via added to the net.
bx2 by2	Specifies the top-right coordinate of bottom layer of the via added to the net.
topLayerName	Specifies the top layer name of via added to the net.
tx1 ty1	Specifies the lower-left coordinate of top layer of the via added to the net.
tx2 ty2	Specifies the top-right coordinate of top layer of the via added to the net.

Note: For turnvias, Encounter reports *topLayerName* only.

- 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.

26

Analyzing and Repairing Crosstalk

- Overview on page 896
- Before You Begin on page 898
- Results on page 898
- Performing Crosstalk Prevention on page 899
- Data Preparation on page 899
- Analyzing Crosstalk on page 900
- Repairing Crosstalk Violations on page 901
- Running the Incremental SI Flow on page 901
- Running the Sign-Off SI Flow on page 903

Overview

Crosstalk is the undesired electromagnetic coupling between signal lines that causes functional failures and delay variation. Crosstalk effects 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:

- Victim driver upsizing
- Buffer insertion
- Aggressor downsizing
- Aggressor rip-up
- 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.

Before You Begin

The following design data files are required in order to repair crosstalk violations:

- Netlist
- SDC (timing information)
- Routed Encounter database or DEF file (placement and routing information)

The following library files are required in order to repair crosstalk violations:

- LEF file (physical library))
- `.cdB` file, ECHO model, or user-defined noise (UDN) model (noise library)
- `.lib` or TLF (timing library)
- Encounter extended capacitance table file
- QRC standalone extraction tech file and library (optional)

Note: Before you begin, ensure that your routed design meets all timing requirements.

Results

Analyzing and repairing crosstalk is an iterative process. The Encounter software performs crosstalk analysis and makes repairs, then repeats the process until all crosstalk violations are repaired.

When finished, the software produces the following outputs:

- DEF file that contains clean timing and no crosstalk violations
- Incremental SDF file
- Various incremental and sign-off reports, located in the log file.

Performing Crosstalk Prevention

1. Place and optimize your design.
2. Run timing and signal integrity driven routing with NanoRoute.

   NanoRoute uses signal-integrity-driven routing to change the topology to reduce coupling capacitance. For more information, see "Preventing and Repairing Crosstalk Problems" on page 620.

Data Preparation

In order to analyze and repair crosstalk violations properly, you must prepare you 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 using the following commands:

   ```
   reportDRV
   fixDRCViolation -postRoute
   or
   optDesign
   timeDesign
   ```

c. Perform RC Extraction and Correlation

You can use native detailed extraction, native incremental extraction, 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
```

or

```
loadConfig your_design.conf
defin your_routed.def
```

Analyzing Crosstalk

The Encounter software uses native Signal Integrity (SI) analysis to perform crosstalk analysis. Encounter calculates delay to cross-coupling effects on each net and produces delays in the form of a standard delay format (SDF) file. This delay is stored in Encounter to incorporate delay due to crosstalk effects.

The resulting glitch violations are stored in the `celtic.eco` file. Native SI analysis generates output files and stores them in the `./celtic` working directory. For more information, see the *CeltIC NDC User Guide*.

➤ Use the Run CeltIC Crosstalk Analysis form or the `runCeltIC` command to specify noise analysis information such as:

- ❑ Mode—Native or sign-off
- ❑ Effort—High or low
- ❑ Process technology
- ❑ Switching windows and TWF file
- ❑ Block noise models

Important

Before performing crosstalk analysis, you must generate RC values for crosstalk coupling capacitance. Use the `setExtractRCMode` command to set RC extraction mode and the `extractRC` command to execute the extraction. Applying switching windows automatically calls the `setExtractRCMode -detail -noReduce` command.

Repairing Crosstalk Violations

The following factors determine how crosstalk violations are repaired:

- Whether you are performing incremental crosstalk repair or sign-off repair.
- The size of your design.

There are two basic ways that the Encounter software repairs crosstalk violations:

- Placement-based repair

  For repairing noise delay violations, the software uses buffer insertion and resizing, aggressor downsizing, aggressor ripup, victim net spacing, and protection to fix critical nets in the design.

- Routing-based repair

  The software uses NanoRoute to repair noise-on-delay violations using the following techniques: wire spacing, net ordering, layer selection, minimization of long parallel wires, and shielding.

Running the Incremental SI Flow

Incremental crosstalk analysis and repair performs an incremental noise analysis and repair on your design. The incremental mode only runs on touched nets that need analysis and repair.

Incremental analysis and repair is an iterative process. It continues to analyze and repair crosstalk until there are no glitch violations remaining, and negative slack with coupling delay equals negative slack without coupling delay.

Note: In order to view the noise nets after incremental crosstalk analysis and repair, you must first rerun the `runCeltic` command to update the `celtic.eco` and `celtic.txt` files.

Incremental crosstalk analysis and repair includes the following tasks:

1. Performing noise analysis using native SI analysis.
2. Performing timing analysis using timing window files (TWFs).
3. Repairing noise-on-delay violations.
4. Repairing glitch violations.
5. Performing signal integrity-aware ECO routing using the integrated version of NanoRoute (native).

You can modify the steps in this flow by setting the appropriate parameters in the `setSIMode` command and running the `optDesign -postRoute -si` command.

➤ To perform incremental analysis and repair on your design, use the Fix Crosstalk form, or type the following commands:

```
setSIMode -extractionEngine incrementalQRC
optDesign -postRoute -si
```

Figure 26-1 Incremental SI Flow

Running the Sign-Off SI Flow

Sign-off crosstalk analysis and repair performs a complete sign-off quality analysis and repair of the entire design.

Sign-off analysis and repair is an iterative process. It continues to analyze and repair crosstalk until there are no glitch violations remaining, and negative slack with coupling delay equals negative slack without coupling delay.

Sign-off crosstalk analysis and repair includes the following tasks:

1. Performing noise analysis using native SI analysis..
2. Performing timing analysis using the native SI timing analyzer.
3. Repairing noise-on-delay violations.
4. Repairing glitch violations.
5. Performing signal integrity-aware ECO routing by calling the standalone version of NanoRoute.
6. Extracting the RC parasitic data by calling standalone QRC extraction.

The previous tasks identify the tools you can use for optimal sign-off quality results. You are not limited to using only these tools when performing the sign-off closure flow.

Note: You can modify the steps in this flow using the appropriate parameters in the `setSIMode` command and running the `optDesign -postRoute -si` command.

➤ To perform a complete sign-off analysis and repair on your design, use the Fix Crosstalk form, or type the following commands:

```
setSIMode -extractionEngine standaloneQRC
optDesign -postRoute -si
```

27

Power Analysis

- Overview on page 907
- Before You Begin on page 908
- Results on page 909
- Displaying Macro Current Source Locations on page 910
- Running Power Analysis with Encounter on page 911
- Running Power Analysis in Simple Estimate Mode on page 913
- Running Power Analysis with VoltageStorm on page 915
- Performing Static Power Calculation on page 916
- Performing CPF-Driven Power-Up Analysis on page 923
- Creating a Power Pad Location File on page 924
- Using Auto Fetch on page 926
- Simple Decoupling Capacitor Modeling on page 927
- Viewing the Power Graph on page 927
- Saving Rail Analysis Values on page 928
- Power/Rail Summary Report on page 929
- Power Graph Report on page 929
- Instance Average Power Report on page 931
- Net Average Power Report on page 931
- Instance Power Data File on page 932
- Instance Voltage File on page 932
- Boundary Voltage File on page 933

- Display Setting File on page 934
- Calculating EM Values for Wires and Vias on page 935
- Log of Physical Connectivity to Power Grid on page 935
- Operating Voltage Recognition on page 936
- Removing Power Consumption for Specified Pins or Ports on page 936

Overview

The Encounter™ power analysis software enables you to

- Create reports and waveforms that tell you the average power consumption of the design.
- Create displays that show either the average or the peak IR drop on the power nets, and report the instance voltages due to IR drop.
- Create displays that show areas in which average power consumption could cause EM violations.

Additionally, an interface to the VoltageStorm® PE power integrity verification solution enables you to perform sign-off power grid analysis and display those results in Encounter. To use this interface, you must have VoltageStorm software and a license to run it.

Important

If you have used previous versions of the Encounter power analysis software, results from the current version might look different than you expect. In versions prior to 2.3, the software reported peak IR drop and EM values. In version 2.3 and later, the software always reports average EM values. By default, the software also reports average IR drop values, but you can specify that peak IR drop values be reported instead.

Power Analysis Modes

Power analysis can be run in two types of floorplanning modes and three types of power analysis modes, for a total of six mode combinations.

The two floorplanning modes are *floorplan* and *layout*.

- **Floorplan Mode**

  Use this mode to run power analysis in the early floorplanning stages, after power rings and stripes are created, but before power rails and follow pins are connected. This mode allows early feedback on power consumption, IR drop, and EM violations while the floorplan is being developed.

- **Layout Mode**

  Use this mode to run power analysis on a complete or near complete floorplan. The chip's power structure must be complete, power rails must be in place, and power pins must connect to the power structures. These power pins must exist in the LEF file for all

cell types used in the design. Cells that are not connected to the power structure are not included in the analysis.

The two power analysis modes are *statistical* and *simulation-based*. In addition, you can run a separate rail analysis.

- **Statistical Mode**

  Use this mode for performing power analysis early in the design flow. This mode has dummy clock, pre-CTS, and post-CTS clock options.

  - In the *dummy clock* option, only a specified percent (probability) of all nets toggle at a specified clock rate.
  - In the *pre-CTS* or *post-CTS* option, the design is divided into the clock domains, and each clock domain's clock rate and net toggling probability can be specified.

  For more information, see Power Analysis: Statistical Mode in the *Encounter Menu Reference*.

- **Simulation-Based Mode**

  Use this mode for performing power analysis at later stages in the design flow. This mode requires a Verilog® value change dump (VCD) input file.

  For more information, see Power Analysis: Simulation-Based in the *Encounter Menu Reference*.

- **Rail Analysis: Average Mode**

  Use this mode for performing rail analysis that examines the IR drop or EM violations of existing power structures.

  For more information, see See Rail Analysis: Average Mode in the *Encounter Menu Reference*.

Before You Begin

Before you can perform power analysis, you must prepare your design by completing a series of tasks. For information about these tasks, see "Setting Up Power Analysis" on page 909.

You must also have the following input files:

- Design file
- Power pad location files

- Instance power data file (optional)
- Verilog VCD file (for simulation-based power analysis)
- Cell libraries containing power grid views (for VoltageStorm)

Setting Up Power Analysis

To set up the design for Power Analysis, complete the following steps:

1. Import the design.

   For more information, see "Importing and Exporting Designs" on page 79.

2. (Optional) Issue the `globalNetConnect` command. This command is required when power and ground nets in the `SPECIALNETS` section of the DEF file do not contain the complete connectivity information.

   For more information, see `globalNetConnect` in the *Encounter Text Command Reference* or Global Net Connections in the *Encounter Menu Reference*.

3. Load or design a floorplan.

   For more information, see "Floorplanning the Design" on page 277.

4. Create the main power structures (rings and stripes).

   For more information, see "Power Planning and Routing" on page 339.

5. Connect the power structures to all the I/O pad instances and blocks.

   For more information, see "Power Planning and Routing" on page 339.

   Note: You can also load the power and ground stripes from a back-end tool in DEF or TDF format, using `defIn` or `tdfIn` commands.

6. Run placement.

   For more information, see "Placing the Design" on page 409.

7. Run *Trial Route*.

8. Run *Extract RC*.

Results

The results of power analysis depend on the tools used to perform the analysis.

- Running power analysis in statistical mode produces the following:
  - A power graph
  - A report that contains the average power usage, worst IR drop, and worst EM violation
  - An instance power file
  - An instance voltage file
  - A boundary voltage file (produced when you issue the `savePartition` command for hierarchical designs)

  Note: The Encounter log file contains a set of messages that indicates how many instances had physical connectivity to the power grid. For more information, see "Log of Physical Connectivity to Power Grid" on page 935.

- Running power analysis in simulation-based mode produces the following:
  - A power graph
  - A report that contains the average power usage, worst IR drop, and worst EM violation
  - An instance power file
  - An instance voltage file
  - A boundary voltage file (produced when you issue the `savePartition` command for hierarchical designs)
- Running rail analysis produces the following:
  - A power graph
  - A report that contains the amount of power used for rail analysis, worst IR drop, and worst EM violation
  - An instance voltage file
  - An instance power file

Displaying Macro Current Source Locations

Use the Display Macros I Source Location form to display the macro current source location for blocks. (The I in the title of this form stands for *current*.) The power pins of a block are

both the power source for the block's contents, and the sinks for the power grid external to the macro.

The white circles in the design display area represent the current source locations for the macro.

- For floorplan mode, the current source locations for blocks are distributed along the grid the block is in. For each area of the block (that is, an area bounded by power stripes or the block macro boundary), four current source locations are displayed, one each for the top, bottom, left, and right side of the area. Adjacent areas can share a source location. The following diagram shows an example of how the software can set up current source locations.

- For layout mode, the displayed macro current source locations are at the power pins, and a current source is displayed for each power pin connection to the block.

Running Power Analysis with Encounter

To run power analysis, complete the following steps:

1. Make sure the timing library is complete with power attributes.

   The timing library file(s) should contain the `cell_leakage_power` library attribute for `.lib` or `CELL_SPOWER` for TLF for all cells. This cell attribute represents the static power usage when a gate is not switching. If this attribute is missing, then no static power usage is reported for average power.

   Another cell library attribute is `internal_power` for `.lib`, or `INTERNAL_ENERGY` or `SC_ENERGY` for TLF. This attribute represents the cell's internal power usage due to

switching and instantaneous short circuit current between power and ground when the internal cell switches state. If this attribute is missing, then no internal power usage is reported for average power.

Each power or ground net should use cells that have the same nominal voltage defined in the library.

If timing libraries do not have nominal voltages defined, the default bias voltage, which is 3.3 V, is used. If a cell does not have a power model in the timing library, the power analysis software assumes 3.3 V as its operating voltage.

Note: The operating voltage is calculated using the voltage value of the majority of the cells. Since filler cells do not have timing or power models, the power analysis software excludes filler cells when calculating the operating voltage. Filler cells are not reported as unclocked instances and are not included in the Instance Voltage file.

If the power or ground net under analysis uses instances that have different nominal voltage values, the power analysis software determines which voltage is used by the greatest number of instances and uses that voltage value for all instances.

Note: A warning message is displayed if nominal voltages are not defined or if multiple nominal voltages are defined for a single net, as shown in the following example:

```
**WARN: Net vdd has 200112 instances at 3.3V, 9 instances at 1.62V.
**WARN: Using 3.3V for all instances.
**WARN: Libraries without nominal voltages defined are defaulted to 3.3V.
**WARN: Please ensure that all libraries have nominal voltages defined.
```

2. Create the power and ground rings for the I/O power pads to connect power to the I/O instances, and create power and ground rings for all the blocks.

3. (Optional) Load the power and ground structures from back end design tools in DEF or TDF formats.

   Use the `defIn` or `tdfIn` command to load the power and ground data.

   Note: For floorplan mode, standard cells do not need to be connected to power and ground. When power analysis runs, it will automatically add the power or ground routes for the standard cells. You can view this by clicking *All Colors* on the main Encounter window and selecting *Power Graph* from the Color Preferences form. You must zoom in to see the graph.

4. Create a pad location file for each net to be analyzed, to indicate voltage source reference points.

   Each power and ground net must have at least one pad location. For example, if the design has three power nets (such as VCC, VDD, VSS), then at least three pad locations are needed for running power analysis, one on each voltage source. These locations are usually at the I/O power pads, but you can change the locations to investigate better

power pad locations. For more information, see Edit Pad Location in the *Encounter Menu Reference*.

5. After all the I/O pad instances and all the blocks are connected to power and ground, run *Place*, *Trial Route*, and *Extract RC*.

   Note: Alternatively, if you have a SPEF file, you can use the `spefIn` command to read RC data from the SPEF file.

   For floorplan mode, it is not necessary to connect the I/O pad instances to power.

6. Run power analysis in statistical mode or simulation-based mode.

   For more information, see Power Analysis: Statistical and Power Analysis: Simulation-Based in the "Power Menu" chapter of the *Encounter Menu Reference*.

Tip

If you make any changes to the design after running power analysis, rerun it to create a new IR drop graph, EM violation display, and power report. Depending on your change, you may need to rerun *Place*, *Trial Route*, or *Extract RC* before you rerun the power analysis.

Running Power Analysis in Simple Estimate Mode

To perform a simple estimate of power consumption using multiplier values instead of a power library, complete the following steps.

1. Create a file that contains the following keywords, along with values for each keyword:

   - `clock_rate` *`value`*

     `value` is in MHz, default is 100

   - `toggle_probability` *`value`*

     default is 0.2

   - `Estimate_switching_power {use_extracted_cap |` *`value`*`}`

     `value` is in pf

   - `Estimate_leakage_power`

     This optional keyword does not use a value

   - `Estimate_internal_power` *`slew load_cap`*

slew is in ns and *load_cap* is in pf)

- `Clock_driver` *cell_name power_in_Watts number_of_flops*

  This keyword can be used in pre-CTS mode, where no clock tree is present, to compute clock tree power and distribute the power to flip-flops

- `cell_internal_energy_and_leakage_power_file` *filename*

  This optional keyword specifies a file that contains the cellname, internal energy in picojoules, and leakage power in Watts

- `create_cell_internal_energy_and_leakage_power_file` *filename*

  This optional keyword specifies that the power analysis software create a cell_internal_energy_and_leakage_power_file

2. Issue the `updatePower` command with the `-estimate` parameter, as shown in the following example:

   ```
   updatePower -estimate estimationfile.txt -pad design.vdd.pp VDD
   ```

   Alternatively, issue the `updatePower` command with the `-estimate` parameter and the `-toggleFile` parameter to trace clock domains, as shown in the following example:

   ```
   updatePower -estimate estimationfile2.txt -toggleFile togglefile.tg -pad
   design.vdd.pp VDD
   ```

Estimation Parameter File

The following example shows the contents of a file named `estimationfile.txt`:

```
clock_rate 200
toggle_probability 0.3
Estimate_switching_power use_extracted_cap
Estimate_leakage_power
Estimate_internal_power 0.120 0.1
```

The following example shows the contents of a file named `estimationfile2.txt,` which contains the `Clock_driver` keyword:

```
clock_rate 200
toggle_probability 0.3
Estimate_switching_power use_extracted_cap
Estimate_leakage_power
Estimate_internal_power 0.120 0.1
Clock_driver CLOCK_BUF 0.0001 2
```

Running Power Analysis with VoltageStorm

If you have VoltageStorm software and a license to run it, you can run VoltageStorm using Encounter menus and commands. For detailed information about using VoltageStorm, see the *VoltageStorm Hierarchical PGS Manual*.

VoltageStorm Power Grid Analysis Flow

VoltageStorm enables a high capacity hierarchical sign-off power grid analysis flow based on power grid views. It helps you verify that the power grid network on your chip does not suffer from IR drop, ground bounce, or electromigration problems.

The following recommended sequence uses both Encounter and VoltageStorm software:

1. Use Encounter to perform a flat power grid analysis and create boundary voltages for specified partitions on a design with partitions specified but not committed.
2. Use Encounter power analysis for each partition to create an instance power file.
3. Execute VoltageStorm for each partition using the boundary voltages generated in the previous step and the instance power file.
4. Use VoltageStorm to create a detailed power grid view of the partition for use at the top level.
5. Use Encounter at the top level of the partitioned design to create the instance power file and include partition power using the instance power data file.
6. Execute VoltageStorm at the top level using power grid views of all partitions.

This sequence ensures:

- Accurate block-level power analysis using boundary voltages.
- Accurate block modeling with a detailed power grid view.
- Accurate top level hierarchical analysis using block power grid views.

The detailed power grid view and block-level power consumption combine to provide detailed grid analysis profiles without loading the block's netlist.

Running VoltageStorm

To use VoltageStorm, complete the following steps:

1. Prepare data for power analysis.

   See "Setting Up Power Analysis" on page 909 for more information.

2. Run power analysis in either statistical or simulation-based mode to generate an instance power file.

   See Power Analysis: Statistical and Power Analysis: Simulation-Based in the *Encounter Menu Reference* for more information.

3. Choose *Power – Run VoltageStorm*.

   This opens the VoltageStorm Analysis form. For more information, see VoltageStorm Analysis in the *Encounter Menu Reference*.

4. Specify the following required values on the VoltageStorm Analysis form:

   - Net name.
   - Library List (see Library Generation - Basic in the *Encounter Menu Reference*).
   - Instance power file (this was generated during power analysis).
   - Pad location file (this was generated before running power analysis).

5. Set additional options on the VoltageStorm Analysis form.

6. Click *OK* or *Apply*.

   If VoltageStorm runs without errors, you will see a message stating that VoltageStorm is exiting normally.

7. Choose *Power – Display Rail Analysis Results*.

   This opens the Display Rail Analysis Results form. For more information, see Display Rail Analysis Results in the *Encounter Menu Reference*.

Performing Static Power Calculation

The power analysis software includes a sign-off quality common power engine (CPE) that calculates the instance-based power-consumption data. CPE is based on proven PowerMeter algorithms that calculate and report the power-consumption data from floorplanning to design sign-off (GDSII).

CPE calculates the power consumption for each instance in a digital design as a function of the switching activity for each net and instance in the design. It uses a probabilistic approach to calculate the switching of each net in the design. CPE can handle not only standard cell

designs but also designs containing large amounts of memory, as well as IP blocks. You can use CPE at the block level and also at the chip level for the analysis of the complete design.

CPE generates an instance-based power-consumption file that contains data on leakage power, internal power, and switching power. You can use its textual output to provide Encounter or VoltageStorm PE with the required instance-based power consumption data.

CPE reports global, as well as specific instance power, clock network power, clock domain power and net switching power. It supports power shut off and also reports the power for power domains and specific power nets.

Note: For more details, refer to the static power calculation information in the *VoltageStorm PowerMeter Manual*.

CPE Flow

Inputs

CPE accepts the following inputs for static power-consumption calculations:

- Design Netlist

  CPE accepts a gate-level Verilog design netlist or OpenAccess netlist from Encounter.

  Note: Encounter requires the netlist to be unique[1].

- LEF File

  CPE accepts multiple LEF files from Encounter.

- VCD, TCF, or SAF File

  - CPE uses the value change dump (VCD) file, which contains the toggle count information, as the basis for the power-consumption calculation. This file can be in compressed (zipped) format with a `.gz` extension.
  - CPE uses the toggle count format (TCF) file, which contains data on the switching activity of the nets in the design, as the basis for the power-consumption calculation. The switching activity includes the toggle count information and the probability of the net or pin being in the logic 1 state.
  - CPE uses the switching activity format (SAF) file, which contains the switching activity information and one default clock frequency, as the basis for the power-consumption calculation. The switching activity is in terms of a percentage that is relative to the default clock frequency.

- SDC File

  CPE accepts a Synopsys design constraints (SDC) file that contains clock domain information from Encounter.

- `.lib` Files

  CPE accepts one or more Synopsys timing libraries (`.lib`) from Encounter. You can specify the operating conditions for the libraries using the Encounter `setOpCond` command. For more details, see *Setting Operating Conditions* in the *Timing Analysis* chapter of the *Encounter User Guide*.

- SPEF File

  CPE accepts a SPEF file that contains the interconnect parasitic information. This file can be in compressed (zipped) format with a `.gz` extension. You can read the SPEF file using the Encounter `spefIn` command.

Outputs

CPE can generate the following outputs:

- Instance Power File

---

1. See *Creating a Flat Verilog Netlist from a DEF File* in the *Importing and Exporting Designs* chapter of the *Encounter User Guide*.

CPE generates an instance-based power-consumption data file (textual format) that you can use with Encounter or VoltageStorm PE. It contains the following data:

- ❑ Instance-based power-consumption data that can be sorted by internal, switching, leakage, and total power
- ❑ Instance-based power-density data
- ❑ Total power-consumption data

■ TCF File

CPE can write out a TCF file that contains data on the switching activity of the nets in the design.

■ Power Reports

- ❑ Reports the power for clock networks (leaf flip-flop power is optional)
- ❑ Reports the power for hierarchical instances and cells (wildcards supported)
- ❑ Reports the power down to the user-specified hierarchy in the design.
- ❑ Reports the power for power domains
- ❑ Reports the power for user-specified power nets
- ❑ Reports the net switching power
- ❑ Reports the power for clock domains (includes the data network associated with the clock)

Power Analysis Flow — Early Stage

1. Load the design.

Refer to the *Importing and Exporting Designs* chapter in the *Encounter User Guide*.

2. Define the global net connectivity for power and ground nets.

Refer to *Global Net Connections* in the *Power Planning and Routing* chapter of the *Encounter User Guide*.

3. (Optional) Read the activity file.

`readActivityFile`

4. (Optional) Set the instance power.

`setPower`

5. Read the parasitics.

   You can read the SPEF file using the Encounter `spefIn` command.

6. Set the mode.

   `setDefaultSwitchingActivity`
   `setSwitchingActivity`
   `setPowerAnalysisMode`

7. Report power.

   `reportPower`

   Note: You can run this command multiple times and in consecutive order.

8. Write the TCF file.

   `writeTcf`

Power Analysis Flow — Late Stage

1. Load the design.

   Refer to the *Importing and Exporting Designs* chapter in the *Encounter User Guide*.

2. Define the global net connectivity for power and ground nets.

   Refer to *Global Net Connections* in the *Power Planning and Routing* chapter of the *Encounter User Guide*.

3. (Optional) Read the activity file.

   `readActivityFile`

4. (Optional) Set the instance power.

   `setPower`

5. Place the design.

6. Route the design.

7. Extract the parasitics.

8. Set the mode.

   `setDefaultSwitchingActivity`
   `setSwitchingActivity`
   `setPowerAnalysisMode`

9. Report power.

   `reportPower`

   Note: You can run this command multiple times and in consecutive order.

10. Write the TCF file.

    `writeTcf`

Supported CPE Text Commands

- `getPowerAnalysisMode`
- `propagateActivity`
- `readActivityFile`
- `reportPower`
- `setDefaultSwitchingActivity`
- `setPower`
- `setPowerAnalysisMode`
- `setSwitchingActivity`
- `writeTcf`

Supported Database Commands

CPE supports the following database commands for customized power reporting:

- `dbInstInternalPower`

  `dbInstInternalPower` *`instPtr`*

  Returns the internal power of the specified instance.

- `dbInstLeakagePower`

  `dbInstLeakagePower` *`instPtr`*

  Returns the leakage power of the specified instance.

- `dbInstSwitchingPower`

  `dbInstSwitchingPower` *`instPtr`*

  Returns the switching power of the specified instance.

- `dbInstTotalPower`

  `dbInstTotalPower` *`instPtr`*

  Returns the total power of the specified instance.

- `dbNetPower`

  `dbNetPower` *`netPtr`*

  Returns the power of the specified net.

- `dbNetProbability`

  `dbNetProbability` *`netPtr`*

  Returns the probability of the specified net being in the logic 1 state. The probability is a value between 0 and 1.

- `dbNetSwitchingPower`

  `dbNetSwitchingPower` *`netPtr`*

  Returns the switching power of the specified net.

- `dbNetToggleDensity`

  `dbNetToggleDensity` *`netPtr`*

  Returns the toggle density of the specified net.

- `dbSetNetProbability`

  `dbSetNetProbability` *`netPtr probability`*

  Sets the probability of the specified net being in the logic 1 state. The probability is a value between 0 and 1.

- `dbSetNetToggleDensity`

  `dbSetNetToggleDensity` *`netPtr toggle_density`*

  Sets the toggle density of the specified net. The toggle density specifies the number of transitions per second. It can be any positive number.

- `dbSetPicoWattPerDBU`

  `dbSetPicoWattPerDBU [1 | 1000 | 1000000 | 1000000000 | 1000000000000]`

  Sets the power unit that the db power query commands will use to report the power.

  - When set to `1`, reports the power queried by the db power query commands in picowatts (pw).
  - When set to `1000`, reports the power queried by the db power query commands in nanowatts (nw). This is the default value.

- When set to `1000000`, reports the power queried by the db power query commands in microwatts (uw).
- When set to `1000000000`, reports the power queried by the db power query commands in milliwatts (mw).
- When set to `1000000000000`, reports the power queried by the db power query commands in watts (w).

Performing CPF-Driven Power-Up Analysis

The Encounter-VoltageStorm interface provides support for CPF-driven power-up analysis to enable you to analyze a switched-on net in the power domain. The power-up analysis flow (Figure 21-1) uses the following production-proven technologies: PowerMeter™, Virtuoso® UltraSim™ Simulator, and VoltageStorm® Dynamic Gate (VSDG).

Figure 27-1 Power-Up Analysis Flow

When performing power-up analysis, PowerMeter generates a SPICE deck, netlist, and stimulus files for the switched net that is powering up. After dynamic power calculation is complete, UltraSim (must be run in standalone mode using the `ultrasim` *`switchrail`*`.sp` command) simulates the design to capture the turn-on time of the switched net (*`switched_net`*`.trn` file). To view the turn-on time waveform of the switched net, you can use the Simvision Waveform Viewer on this file. VSDG then performs rush-current analysis using the *`switchednet`*`.ptipeak` current file that is generated by UltraSim.

To perform rush-current analysis, the current file must be specified using the `-currentFile` parameter of the `runVStorm` command (or specified in the VoltageStorm Analysis form under *PowerMeter Current File(s)*).

The following `runVStorm` command parameters support power-up analysis:

Note: You can also use the VoltageStorm Analysis form (*Power – Analysis – Run VoltageStorm – Advanced* tab) to perform these operations.

- `-powerUp {0 | 1}`
- `-powerUpSpiceCorners` *`corners`*
- `-powerUpSpiceModels` *`files`*
- `-powerUpSpiceNetlists` *`files`*
- `-powerUpSpicePWL "`*`time1 voltage1 time2 voltage2...`*`"`

Note: For more details on how to perform power-up analysis, see the *Power Gating* chapter of the *VoltageStorm Cell-Level Rail Analysis User Guide*.

Creating a Power Pad Location File

The location of voltage sources is the starting point for the rail analysis. The power analysis software obtains voltage source locations for a particular net, such as VCC, VDD, and VSS, from that net's power pad location file. You must mark points on the power structures as voltage source reference points, then store that information in the power pad location file. identify the voltage source reference points Power pad location files are used during all stages of power analysis.

To create a new power pad location file, complete the following steps:

1. Choose *Power – Edit Pad Location.*

   This automatically changes the Encounter display to the Physical view and opens the Edit Pad Location form.

2. Specify the power net or ground net name in the *Net* field.

   You can only edit the pad location file for one net at a time.

3. Click *Auto Fetch*.

   This automatically populates the *Pad Location List* with the pad locations based on properties defined in the LEF file. For information about setting up a LEF file for use with *Auto Fetch*, see "Using Auto Fetch" on page 926.

   Note: You may need to reissue the `globalNetConnect` command for *Auto Fetch* to function properly if the power pads were not defined in the data base at the time that the `globalNetConnect` command was first issued.

4. Click *Save*.

   This opens the Save Pad Location File form.

   Note: As an alternative to saving the power pad location data to a file, you can use the `-keepPadLocDisplay` parameter of the `savePadLocation` command to have the power analysis software automatically save the power pad location data that is displayed in the GUI to a reserved file named `paAutoGen.`*`netName`*`.pp`. The power analysis software will then use this file to perform rail analysis. If you do not specify this parameter, the power analysis software clears the power pad locations displayed in the GUI when the file is saved.

5. Specify a location for the pad location file, then click *OK*.

   This saves the pad location file, clears the *Pad Location List*, and closes the Save Pad Location form.

Pad location displays are updated immediately after clicking *Auto Fetch*, *Delete*, or *Add*. The IR drops leading from the pad locations are displayed after running power analysis. You can also use the Display Pad Location form to display the pad locations in the design display area. The pad locations are displayed as yellow circles for each x and y coordinate, marking the voltage source reference points for running power analysis. If a boundary voltage value exists for the location, it is shown as a solid yellow circle. A voltage value equal to the nominal voltage is treated as if there were no boundary voltage value, and is represented by a hollow yellow circle. The IR drop at pad locations is 0.

Note: The yellow circles display only in the Physical view.

Use the *Clear* button on the Edit Pad Location form after you save the pad location file before you use *Auto Fetch* again. To clear the *Pad Location List*, click *Clear*. Once the *Pad Location List* is cleared, you can begin creating a pad location file for a new net.

Using Auto Fetch

To use *Auto Fetch*, the following conditions must be met:

- The LEF macro for power pads must contain the `CLASS PAD POWER` statement.
- Block power and ground pins must have DEF attributes `+USE POWER` or `+USE GROUND`.

In addition, if a pad contains multiple pins associated with different power and ground nets, Specify `CLASS CORE` for the port of the pin that belongs to the net that uses the pad as a DC source. If none of the ports is `CLASS CORE`, then *Auto Fetch* includes that pad as a DC source for multiple nets.

- If the power or ground pad is routed, the power analysis software assumes a DC source on every layer that connects to the routed port.
- If the power or ground pin is not routed, the power analysis software assumes a DC source only on the closest power or ground wire.

The following example shows the portion of a LEF file with a pad that contains pins for VDD and VSS.

```
MACRO VVDD
    CLASS PAD POWER
    PIN VDD
        USE POWER
        PORT CLASS CORE
        ...
    END VDD
    PIN VSS
        USE GROUND
        PORT
        ...
    END VSS
    ...
```

In this example, since `PIN VDD` has `CLASS CORE` specified, *Auto Fetch* only uses `VDD` as a DC source. If `CLASS CORE` was not specified, both `VDD` and `VSS` pins would be used as DC sources.

Note: Use a different `.pp` file for each net.

Simple Decoupling Capacitor Modeling

You can use the power analysis software to model cells with negative current separately for power and ground pins. Specify a negative value for power in the instance power file for decoupling cells in order to see the effect in the rail analysis. Power analysis does not automatically identify the decoupling cells.

The following example shows how to specify an instance power data file with the decoupling capacitor cells:

```
DECAP_1 -0.001
DECAP_2 -0.001
```

Both `DECAP_1` and `DECAP_2` are modelled to similarly *supply* 1 mA current for a 1 V system.

Viewing the Power Graph

The power analysis software creates a power graph that is a representation of the power and ground net structure used by power analysis to construct the resistor network to perform IR drop and EM analysis.

To view the power graphs, complete the following steps:

1. Use the Display Rail Analysis Results form to select the type of results to be displayed.
2. Click the Physical View icon to the left of the design display area.
3. Open the Color Preferences form by clicking the *All Colors* button above the *Colors* Tool area.
4. Click the V (Visibility) and S (Selectability) button for *Power Graph*.
5. (Optional) To get a better view, deselect *Net*, *SNet*, and *Obstruct*.

   This helps clear the design display area.
6. Zoom in to the area of interest.

To see a power graph report after you run a power analysis or rail analysis, complete the following steps:

1. Use the Display Rail Analysis Results form to select the type of results to be displayed.
2. Click the Physical View icon to the left of the design display area.
3. Open the Color Preferences form by clicking the *All Colors* button above the *Colors* Tool area.

4. Click the V (Visibility) and S (Selectability) button for *Power Graph*.
5. (Optional) To get a better view, deselect *Net*, *SNet*, and *Obstruct*.

   This helps clear the design display area.
6. Select one or more power rails within the design display area.

   You can select multiple rails by holding the `Shift` key while clicking each rail.
7. Press the `Q` key.

   The Attribute Editor for object type Power Graph displays. In addition, a report appears in the Encounter console. The format of the report is shown in the following example:

```
probing node 550 on net "VDD":
    location: ( 1196.6300 436.2400 ) um
    layer: M1
    Bounding Box: ( 1170.8400 435.6100 ) ( 1222.4200 436.8700 ) um
    V: 1596.740801 mV
    instance I: 0.000000e+00 mA
    DC source I: 0.000000e+00 mA
    East neighbour:
        node: 37
        location: ( 1226.4200 436.2400 ) um
        layer: M1
        width: 1.2600 um
        V: 1596.897615 mV
        I: -6.566913e-02 mA
        R: 2.387929e+00 ohm
    West neighbour:
        node: 549
        location: ( 1166.7150 436.2400 ) um
        layer: M1
        width: 1.2600 um
        V: 1596.583330 mV
        I: 6.566913e-02 mA
        R: 2.397948e+00 ohm
```

Saving Rail Analysis Values

To save EM, IR Drop, and Instance Voltage Drop values for any net that has been analyzed, use the Save Rail Analysis Results form.

Power/Rail Summary Report

The power analysis software generates a power/rail summary that report average power by the cell category, the worst average IR drop, and the number of voltage nodes (matrix size) in the rail resistor network. This is an indirect performance indicator. An example of the report, for power analysis using a post-CTS clock, is shown below.

```
##############################################
# The Power Analysis Report for VSS net      #
##############################################
power supply: 0 v
average power(default): 1.2353e+02 mw
    average switching power(default): 2.8075e+00 mw
    average internal power(default): 1.2073e+02 mw
    average leakage power(default): 9.9867e-04 mw
    average user specified power(default): 0.0000e+00 mw
average power by clock domain category:
    clock domain(vclk1, 0.2) : 1.2140e+02 mw
       clock tree power : 1.1902e+02 mw
       non clock tree power : 2.3886e+00 mw
    clock domain(vclk2, 0.2) : 1.2438e+00 mw
       clock tree power : 7.5015e-01 mw
       non clock tree power : 4.9361e-01 mw
    unclock domain(0.2) : 8.8550e-01 mw
average power by cell category:
    core:  7.5507e+00 mw
    block: 1.1598e+02 mw
    io:    5.0000e-09 mw
biggest toggled net: DTMF_INST/m_clk
    no. of terminal: 395
    total cap: 2.4680e+03 ff
```

Power Graph Report

The Power Graph Report provides detailed information for each net rectangle on the power graph.

The following example report shows excerpts from a report that is generated when you issue the `updatePower` command with the `-reportRailAnalysis` parameter for net `VDD`:

```
# HEADER:
NET_NAME           : VDD
```

```
LENGTH_UNIT        : um
CURRENT_UNIT       : mA
RESISTANCE_UNIT    : ohm
WIRE_EM_UNIT       : mA/um
VIA_EM_UNIT        : mA/cut
VOLT_UNIT          : mV
NET_BIAS_VOLTAGE   : 3299.999952
MAX_VDROP_CG       : 0.018798
IR_ANALYSIS_MODE   : AVERAGE
# POWER GRAPH:
# nodeId : ( llx lly ) ( urx ury ) nodeId1 nodeId2 nodeId3 ...
#          where nodeId(n) are neigbhouring nodes of nodeId
#          and nodId == 0 represent DC reference nodes
0 : ( 37.1400 -8.7400 ) ( 38.4800 -3.7400 ) 18
1 : ( -22.1800 -20.6400 ) ( -18.1800 -16.6400 ) 2
# WIRE VOLTAGE CURRENT:
# nodeId1  nodeId2  R layerName Width V1 V2 I
1 2 0.274275 M1 4.0000 3.299994 3.299994 1.025500e-03
2 3 0.274275 M1 4.0000 3.299994 3.299995 1.025500e-03
# VIA VOLTAGE CURRENT:
# nodeIdTop nodeIdBottom R viaName NumCuts V1 V2 I
14 1 0.10625 VIAGEN12_7 16 3.299994 3.299994 1.025500e-03
15 3 0.10625 VIAGEN12_7 16 3.299995 3.299995 1.025500e-03
# WIRE EM:
# nodeId : V EM [Emlimit|NONE] [EM_VIOLATION]
1 : 2.5637e-04 3.299994
2 : 2.5637e-04 3.299994
# VIA EM:
# nodeId : V EM [Emlimit|NONE] [EM_VIOLATION]
14 : 6.4094e-05 3.299994
15 : 6.4094e-05 3.299995
# INSTANCE:
# InstName ((x1 y1) (x2 y2)) vddCurrent worstVoltage
#          ( pinName ( pinNodeId pinCurrent pinVoltage ) ...)
com2 ( -7.2800 -4.1600 ) ( -2.0800 6.2400 ) 7.4844e-06 3.299981 ( VDD ( 10
7.4844e-06 3.299981 ) )
```

Instance Average Power Report

The Instance Average Power Report provides detailed power information for a particular cell, as well as for the net.

The following example report shows the power consumption of cell `inst_1` as well as the total power consumption for the net `vdd!`.

```
##################################################
# The Instance Average Power Report for VDD net  #
##################################################
# Power calculation mode: statistic simulation
# Operating bias voltage: 1.62 Volt
# Units for power: Watts
# Format: Instance Total-Power Internal-Power Switching-Power Leakage-Power
#########################################################################
DTMF_INST/i_10048 5.224669e-06 -3.604321e-03 3.609545e-03 5.764000e-10
DTMF_INST/PLLCLK_INST 7.178137e-06 -5.017416e-03 5.024594e-03 1.458000e-10
.#########################################################################
# Sum: 7.353887e-02 3.889689e-04 7.314740e-02 2.499547e-06
# No. of instances: 5942
#########################################################################
```

Net Average Power Report

The Net Average Power Report provides detailed power information for a particular net.

The following example report shows the power consumption of net `CLK`.

```
##############################
#
# The Net Average Power Report
#
# Operating Voltage = 2.5 Volt
#############################################################################
# Net           Load            Toggle          Switching
#               Cap (pF)        Rate (per sec)  Power (Watt)
#############################################################################
CLK             7.709138e+00    4.942857e+07    1.190787e-03
..
#############################################################################
# Total nets = 11
```

```
# Total Capacitance = 23.5889 pF
# Total Switching Power = 0.0037382 Watts
##############################################################################
```

Instance Power Data File

The instance power data file is an ASCII file containing the instance power values for a design. Each line in the file contains a single instance name followed by the instance power value (in watts). This file is created during power analysis, and can also be used as input for subsequent power analysis runs and by the VoltageStorm software.

The following example shows the format of this file:

```
# InstanceName            PowerInWatt
# ---------------------------------
DTMF_INST/ROM_INST        0.005
DTMF_INST/RESULTS_INST    0.01
```

The `InstanceName` column contains the instance name.

The `PowerInWatt` column contains the instance power number, which overwrites or fills in the missing power number in the design that is being analyzed for power consumption.

Specifying the block power consumption in this file eliminates the need to edit a `.lib` file to provide such information. The total current consumption is equally distributed to all power pins.

Instance Voltage File

The instance voltage file is an ASCII file that can be used by any delay calculator to perform instance voltage based calculations. Using this file facilitates IR Drop aware timing.

The following example shows the format of this file for power nets:

```
VERSION        "1.0" ;
CREATION       "Mon Oct 28 11:49:28 2002" ;
CREATOR        "CADENCE" ;
PROGRAM        "First Encounter" ;
DIVIDERCHAR    "/" ;
DESIGN         "dtmf_chip" ;
UNITS          VOLTAGE VOLT 1 ;

INSTANCESUPPLY 5905 VDD 1.8 POWER ;
```

```
- DTMF_INST/i_10048 1.799744 CLKBUFXL ;
- DTMF_INST/RAM_128x16_TEST_INST/i_10074 1.799683 CLKBUFX3 ;
- DTMF_INST/RAM_128x16_TEST_INST/i_14 1.799760 MX2X1 ;
- DTMF_INST/RAM_128x16_TEST_INST/i_13 1.799692 MX2X1 ;
...
```

The following example shows the format of this file for ground nets:

```
VERSION        "1.0" ;
CREATION       "Mon Oct 28 11:50:02 2002" ;
CREATOR        "CADENCE" ;
PROGRAM        "First Encounter" ;
DIVIDERCHAR    "/" ;
DESIGN         "dtmf_chip" ;
UNITS          VOLTAGE VOLT 1 ;

INSTANCESUPPLY 5832 VSS 0 GROUND ;
- DTMF_INST/i_10048 0.000294 CLKBUFXL ;
- DTMF_INST/RAM_128x16_TEST_INST/i_14 0.000158 MX2X1 ;
- DTMF_INST/RAM_128x16_TEST_INST/i_3 0.000327 MX2X1 ;
- DTMF_INST/RAM_128x16_TEST_INST/i_0 0.000333 MX2X1 ;
...
```

Each line in this file contains the following values:

instance_name voltage master_cell_name.

Note: A negative voltage value in this file indicates a supply current. For more information, see "Simple Decoupling Capacitor Modeling" on page 927.

Boundary Voltage File

The boundary voltage file is created by the power analysis software when you issue the `savePartition` command for hierarchical designs.

The format of this file is:

x y layerName [*pad_instance_name* | *voltage*]

where

x y are the coordinates for the DC source location.

layername is the name of the layer that contains the DC source.

Note: Only a generic layer name, such as `M1` or `M2` should be used.

`pad_instance_name` is the name of the power pad.

`voltage` is a real number that indicates the voltage value of the DC source at that location.

Note: You can specify either *`pad_instance_name`* or *`voltage`*, but do not specify both. If a *`voltage`* value is not specified, the ideal voltage value is used.

Display Setting File

The Display Setting file saves the filter-related settings specified on the VoltageStorm Results Display form. By default, this file has a `.dsf` file extension.

The Display Setting file has the following syntax:

```
ir|tc|rc|rj|er|vc|vv|iv {v1 min1 max1} {v2 min2 max2} {v3 min3 max3} {v4 min4 max4}
{v5 min5 max5} {v6 min6 max6} {v7 min7 max7} {v8 min8 max8}
```

Note: The triplets in this file use the same syntax as the `displayVStormResults -filter` command.

The following example shows a `.dsf` file that displays IR drop values from 1.79914 to 1.8:

```
ir {1 1.79989 1.8} {1 1.79978 1.79989} {1 1.79968 1.79978} {1 1.79957 1.79968} {1
0 1.79957} {1 1.79936 1.79946} {1 1.79925 1.79936} {1 1.79914 1.79925}
```

When you save the filter range information, if any required value is not specified, it is saved as `0`. When you load the `.dsf` file, if any required value is not specified, all subsequent values are ignored.

Note: This feature is only available from the Display VoltageStorm Settings form and does not have an equivalent text command option.

Calculating EM Values for Wires and Vias

The following diagram shows an example of a wire and via configuration.

- The EM value for a wire is calculated as the current in the segment divided by the width of the segment. For example, if the current in *M1* is 6 mA and the width is 3 µm, then the calculated EM is 2.0 mA/µm. The segments will display in red if you specify a limit of 1. 647 mA/µm.
- The EM value for a via is calculated as the current in the via connecting two layers divided by the number of cuts. For example, if the current in V12 is 6 mA and there are nine cuts, then the EM is calculated as 0.67 mA/cut. The vias will not display in red if you specify a limit of 1.647 mA/cut.

Log of Physical Connectivity to Power Grid

The Encounter log file contains a set of messages that indicates how many instances had physical connectivity to the power grid. In addition, the log contains messages that indicate why a particular instance is not connected.

- A message that an instance is not geometrically connected to a net indicates that the pin connection to the net may be missing for that instance.
- A message that an instance is not reachable by a DC source on a net indicates that the instance is connected to a power or ground wire, but that wire may not be connected to a portion of the grid that is attached to a DC source.

Operating Voltage Recognition

If timing libraries do not have nominal voltages defined, a default voltage of 3.3 V is used. If multiple voltages are used within the design, the power analysis software determines which voltage is used by the greatest number of instances, then uses that voltage value for all instances.

Removing Power Consumption for Specified Pins or Ports

Sometimes the power analysis software provides a result that is too pessimistic because power consumption is calculated for test nets that contain pins or ports that will not be consuming power in the actual design. To obtain a more accurate result, you can specify to the power analysis software that these pins or ports do not consume power.

To perform power analysis with the removal of power consumption considerations for specified pins or ports, complete the following steps:

1. Set up an `.sdc` file that contains the following statement:

   ```
   set_case_analysis 0 listOfPinOrPortNames
   ```

2. Import the `.sdc` file into the design.
3. Run power analysis in statistical mode. The results will not including power consumption for any pins or ports that you specified in the `set_case_analysis` statement.

28

Verifying Violations

- Overview on page 938
- Verifying Connectivity on page 941
- Verifying Metal Density on page 943
- Verifying Geometry on page 944
- Verifying Process Antennas on page 947
- Verifying Maximum Floating Area Violations on page 950
- Verifying AC Limit on page 951
- Viewing Violations With the Violation Browser on page 952
- Clearing Violations on page 955

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 941
  - `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

- Verifying Geometry on page 944
- `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.

You create violation markers with the Encounter commands, or import markers from another verification tool, such as Assura™ 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 952.

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 945.

- 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.

- 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.

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 distributed processing mode, use the `setMultiCpuUsage` and `setDistributeHost` commands. For more information, see "Running Multiple-CPU Processing" on page 49.
- Route the design.

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.
- 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.

Important

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.

Sample Process Antenna Report

The following example shows 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

`(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 (Irms) at the driver output and compares it to the `ACCURRENTDENSITY` tables in the LEF file that contain the Irms limits for routing layers. It generates an error and attaches a violation marker to a net if the calculated Irms for a net exceeds the `ACCURRENTDENSITY` Irms limit for a routing layer or width used by the net.

Computes the Irms 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 `mA`. The LEF file specifies it in `mA/micron`. To convert the LEF specification, the `verifyACLimit` command reads the value in `mA/micron` and multiplies it by the wire width.

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*)

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.

- 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.

29

Analyzing Yield

- Overview on page 958
- What Effects Does reportYield Consider? on page 958
- Calculating Failure Probabilities on page 959
  - Critical Area Analysis on page 960
  - Defect Data and Cumulative Defect Data Functions on page 961
- Before You Begin on page 961
- Results on page 961
- Interpreting the Yield Map on page 963
  - Displaying the Yield Map on page 963
- Interpreting the Yield Report on page 966
  - Yield Report on page 966
  - Detailed Report on page 970
- Understanding the Yield Technology File on page 972
  - File Format on page 972
  - File Sections and Keyword Statement Descriptions on page 974
  - Yield Technology File Example on page 984
- Formulas and Calculations on page 987
  - Calculating the Probability of Failure for a Metal Layer on page 987
  - Calculating Defect and Cumulative Defect Data on page 987
  - Cost Formulas on page 990

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.

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 979.

- 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 982.

- 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 960, "Open Point Defect" on page 974, and "Short Point Defect" on page 978.

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 987.

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 987.

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 966

- Yield map

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 963.

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 toggle

     To 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*

Tip

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.

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:

  Cell Yield % x *Via Yield %* x *Route Yield % = 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 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:

  Cell Cost + Via Cost + Routing Cost = Total Cost

  For more information, see "Cost Formulas" on page 990.

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 967

- Cell on page 967
- Via on page 967
- Routing on page 968
- Summary on page 970

Header

Displays the report title, design name, dimension, shrink factor, date, and time.

```
Encounter Defect Limited Yield and Cost Report

Title        : test
Design       : asic_entity
Dimension    : 8300.000um x 8300.000um
Shrink Factor: 0.999
Date         : Mon Nov 21 16:31:10 2006
```

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.

```
Layer |          Number of Vias          |             Cost                 | Total
      |    1-cut      2-cut      Array   |     1-cut     2-cut      Array   | Cost
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
    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 METAL1
    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 METAL1
       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 METAL1
 Diameter(nm)   Radius(nm)    Crit area(cm^2)
          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 METAL1
 Diameter(nm)   Radius(nm)    Crit area(cm^2)
          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 *METAL1*.

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 970
- Cells on page 971
- Subareas on page 971

Vias

The via section includes statistics for each via. Following is a via detail section for layer *via12*:

```
Layer        Number  Number          Approx          Via Name
Via12        Cuts      Vias            Cost
              1      5.4287e+04       1.0857e-04       Via2tsl
              1      1.5245e+05       3.0490e-04       Via2
              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, 0.000000
0, 18  [     864,        0] [     913,       48]  0.000002, 0.000000, 0.000007
0, 19  [     913,        0] [     961,       48]  0.000010, 0.000002, 0.000008
0, 20  [     961,        0] [    1009,       48]  0.000010, 0.000002, 0.000010
0, 21  [    1009,        0] [    1057,       48]  0.000011, 0.000002, 0.000010
0, 22  [    1057,        0] [    1105,       48]  0.000010, 0.000002, 0.000011
0, 23  [    1105,        0] [    1153,       48]  0.000010, 0.000002, 0.000000
...
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 a approximate estimate of yield impacts from different yield optimization choices. You can use the template file to experiment with `reportYield`.

Important

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.

- 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 cm^2, is required. This value is written in the file as `cm^2`. A value like 5.0 defects per cm^2 is written as `5.0/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 a separate keyword.

File Sections and Keyword Statement Descriptions

The yield technology file contains the following sections:

- Header on page 974
- Open Point Defect on page 974
- Short Point Defect on page 978
- Via Probability on page 979
- Cell Probability on page 982

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

`<yield_file>`	The first statement in the yield file. Statements outside of (before or after) the `yield_file` statement are ignored.
`<title>`	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.

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/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.

Important

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`^{3}, either `1/cm`^{2}`/nm` or `1/nm`^{3} (`1/cm`^{2}`/nm` means there are *`defectDensity`* defects per `cm`^{2} 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 987.

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`^{3}.
- 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`^{p}.

`<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^2.

`<cumulative_defect_width_and_density>` values are used to define the CDSD(x) function. For more information, see "Cumulative Defect Data Function" on page 988.

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^2.
- 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 977.

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/cm^2 </cumulative_defect_width_and_density>
   <cumulative_defect_width_and_density> 200nm
  0.025/cm^2 </cumulative_defect_width_and_density>
  </open_point_defect>
  ```

- Converting defect bin size data to defect data statements

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^2
Bin 2	Defect width between 150 nm and 200 nm = 0.02 defects/cm^2

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^2 = 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^2 to1/nm^2 with 1e-14), results in the following values:

Bin 1	C_D = 2 * 0.05 x 1e-14 / [$(1/100)^2$ – $(1/150)^2$] = 0.1e-14 * 1.8e4 = 1.8e-11 DSD(125) = C_D * $(1/125)^3$ = 1.8e-11 * $(1/125)^3$ = 9.22e-18/nm^3 at 125nm
Bin 2	C_D = 2 * 0.02 x 1e-14 / [$(1/150)^2$ – $(1/200)^2$] = 0.04e-14 * 5.15e4 = 2.06e-11 DSD(175) = C_D * $(1/175nm)^3$ = 2.06e-11 * $(1/175)^3$ = 3.84e-18/nm^3 at 150nm

For Bin 1, the data means that the number of defects between 125 nm and 126 nm wide is

~9.22e-18/nm^3 * 1nm = 9.22e-18/nm^2 = 9.22e-4/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>
```

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

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 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> 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 via12 M2 </layers>
 <layers> M2 via23 M3 </layers>
<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>
</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 $(1 – 1.2e\text{-}8)^5 * (1 – 1.0e\text{-}7)$ 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>
```

```
 <!-- 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>
```

```
 <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

λ 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 λ 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:

$DSD(x) = C_D * 1/x^p$ for $x >= x_0$

where

C_D is a constant related to the total number of defects per 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 cm^2 per nm of defect size.

> **Note:** Units are converted to defects per nm^2 per nm or defects/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 μ minimum-width wire)

C_i = $-\ln(1-P_i)$ = the cost of feature i (higher costs have worse yield)

C_T = total cost = $\sum C_i$

For N features on the chip:

$$Y = (1-P_1)*(1-P_2)*...*(1-P_N)$$

Taking the ln of both sides gives:

$$\ln(Y)=\ln((1-P_1)*(1-P_2)*...*(1-P_N))=\sum\ln(1-P_i)=-\sum C_i=-C_T$$

so

$$Y = e^{-C_T}$$

and

Yield loss = $1-Y$

For values of P_i <<1, which is true for these types of probability values, it is useful to remember the following approximation:

$$-\ln(1-P_i)\approx P_i$$

30

Creating An Initial Floorplan Using Masterplan

- Overview on page 992
- Masterplan Automatic Floorplanner Flow on page 993
- Data Preparation on page 994
  - Selecting Seeds on page 994
- Importing the Design on page 1000
- Setting Masterplan Global Parameters on page 1001
- Creating an Initial Floorplan on page 1001
  - Creating a Hierarchical Floorplan on page 1002
- Analyzing the Floorplan on page 1003
- Adjusting Macro Placement on page 1005
  - Manual Macro Adjustment on page 1005
  - Masterplan Macro Adjustment on page 1006
  - Marking Refinement Steps on page 1010
  - Restoring Refinement Steps on page 1010
- Placing Cells and Generating Floorplan Guides on page 1011
- Saving the Floorplan on page 1011

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.

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.

Note: Timing aware Masterplan is a Beta feature in the 7.1 release. Contact your Cadence sales representative to arrange for license and support.

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 30-1 on page 993 shows the typical task flow for using Masterplan automatic floorplanner to create an initial floorplan:

Figure 30-1 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.

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.

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.

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 than10x.

  - 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,

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 30-2 on page 998, you should select the modules at this level as seeds because it will produce better grouping for macro placement.

Figure 30-2

In Figure 30-3 on page 999, you should select this macro as a seed because it is larger than 1/4 of the core area.

Figure 30-3 Large Macro

4. Create a seed section.

For example, Figure 30-4 on page 999 shows the block diagram for the design.

Figure 30-4 Design Block Diagram

Based on this diagram:

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 30-2 on page 998).
2. Examine the module `cache`, and select the macro `bigmacro` because it is very large (Figure 30-3 on page 999).
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
sifu1               #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 995)
- Whether to ignore hard macros within a soft seed when adjusting the seed's aspect ratio.
- Hard macro spacing
- Target utilization for floorplan guides

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.

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

  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 a Hierarchical Floorplan

Given a hierarchical top-level floorplan that contains fences or power domains that represent predefined partitions (Figure 30-5 on page 1002), Masterplan places the hard macros, and generates guides for the submodules within the fence boundary and also on the top level (Figure 30-6 on page 1003).

Figure 30-5 Hierarchical Top-Level Floorplan

Figure 30-6 Masterplan Result

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.

Analyzing the Floorplan

After generating a floorplan, analyze the results in order to assess whether improvements are needed:

- 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.

- 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?

Guidelines For a Good Floorplan

In general, a good floorplan should meet the following criteria:

- All macros are placed in the core area.
- Macros do not overlap, or have very little overlap between them.
- Macros match the module locations, or they are physically located with similar macros to minimize connectivity.
- Macro alignment and orientation is correct, and pins are on the edges that you expect.
- Macro grouping is correct.
- There is enough spacing between macros to avoid congestion.

  Note: You might want to restrict logic cells from being placed between macros to avoid routing congestion, but still allow nets to be buffered using this area.
- Guides have the correct relative locations.

Note: Designer knowledge should always take precedence over any recommended guidelines.

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.

Important

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.

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.

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*.
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` *`llx lly 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 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.

Placing Cells and Generating Floorplan Guides

➤ To perform detailed cell placement and generate floorplan guides for soft seeds after hard macro refinement, type the following command:

```
planDesign -refineFPlan
```

Whenever you specify the `-refineFPlan` parameter, Masterplan checks the `MP_status` global variable setting to see if the `planDesign` command has been run previously in the same session.

If it has (`MP_status` is 1), Masterplan performs cell placement and generates floorplan guides for the existing floorplan. If `planDesign` has not been previously run in this session (`MP_status` is 0), the tool performs steps 1 through 4 listed under "Creating an Initial Floorplan" on page 1001, using automatic seed selection to create the seed list. Then Masterplan places the cells and generates the floorplan guides.

Note: Hard macros are marked `FIXED` at the beginning of placing cells and generating floorplan guides.

Saving the Floorplan

➤ To save the floorplan, type the following command:

```
saveFplan fileName.fp
```

31

Performing Multi-Mode Multi-Corner Timing Analysis and Optimization

- Overview on page 1015
  - Multi-Mode Multi-Corner Licensing on page 1015
  - Multi-Mode Multi-Corner Flow Support on page 1015
  - Differences Between 5.2 Multi-Mode Analysis and 6.1 Multi-Mode Analysis on page 1015
- Configuring the Setup for Multi-Mode Multi-Corner Analysis on page 1016
  - Creating Library Sets on page 1017
    - Flow Library Requirements on page 1018
    - Editing A Library Set on page 1018
  - Creating Virtual Operating Conditions on page 1019
    - Editing A Virtual Operating Condition on page 1019
  - Creating RC Corner Objects on page 1020
    - Editing An RC Corner Object on page 1020
    - Creating Pre-Scaled Capacitance Tables on page 1021
  - Creating Delay Calculation Corner Objects on page 1021
    - Editing A Delay Corner Object on page 1022
  - Adding A Power Domain Definition To A Delay Calculation Corner on page 1023
    - Editing A Power Domain Definition on page 1024
  - Creating Constraint Mode Objects on page 1024
    - Editing A Constraint Mode Object on page 1025

- Entering Constraints Interactively on page 1026
- Constraint Support in Multi-Mode and Multi-Mode Multi-Corner Analysis on page 1027
- Creating Analysis Views on page 1028
  - Editing An Analysis View Object on page 1028
- Setting Active Analysis Views on page 1029
  - Guidelines For Setting Active Analysis Views on page 1029
  - Changing the Default Active Analysis View on page 1030
- Checking the Multi-Mode Multi-Corner Configuration on page 1030
  - Changing How the MMMC Browser Displays Configuration Information on page 1030
- Saving Multi-Mode Multi-Corner Configurations on page 1031

- Controlling Multi-Mode Multi-Corner Analysis Through the Flow on page 1031
- Performing Timing Analysis on page 1034
- Generating Timing Reports on page 1034

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.

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. The software does not support multi-mode multi-corner analysis of hierarchical flows, as well as certain flat sub-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 1031.

Differences Between 5.2 Multi-Mode Analysis and 6.1 Multi-Mode Analysis

In both the 5.2 and 6.1 versions of the Encounter software, multi-mode analysis requires that all setup and hold checking is done with regard to a single delay calculation corner.

In 5.2 multi-mode analysis, delays are stored per delay calculation corner, resulting in the need for all modes to share the same delay. This version of multi-mode analysis:

- Has restrictions on the types of SDC constraints in the modes to avoid collisions
- Creates potentially more pessimistic delay calculation
- Requires less memory and CPU due to a single delay being used

In 6.1 multi-mode analysis, separate delays are stored per analysis view. This version of multi-mode analysis:

- Has fewer restrictions on the types of SDC constraints that can be in the modes
- Provides more accurate delay calculation
- Requires more memory and CPU

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 31-1 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. There can be sets of libraries to define different process corners, operating voltages for MSMV power domains, and `.lib` and ECSM library groups used in SignalStorm® interpolation flows.

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.

➤ To create a library set, use the following command:

```
create library set
```

Important

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.

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`:

Flow Library Requirements

You should configure timing library entries in the configuration file as you normally would for a non-multi-mode multi-corner flow. 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.

Editing A Library Set

➤ To change the timing and cdB library files for an existing library set, use the following command:

```
update_library_set
```

Important

You can use the `update_library_set` command before the multi-mode multi-corner view definition file is loaded into the design, or after. However, after the software processes the file, any changes to an existing object when the software is in multi-mode multi-corner mode results in the timing, delay calculation, and RC data being reset.

You can make changes to a library set using the Edit Library Set form, but only before the multi-mode multi-corner view definition file is loaded into the design. After the software processes the file, this form is no longer available.

➤ 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 MMMC*, and double click on the name of the library set you want to edit.

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 exist 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`:

```
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.

Important

You only can edit a virtual operating condition before the multi-mode multi-corner view definition file is loaded into the design. After the software processes the file, this form is no longer available.

➤ 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 MMMC*, 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 Fire & Ice® QX gate-level 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 scaling attributes when running the software in multi-mode multi-corner analysis mode. Scaling factors set using the `setRCFactor` command are ignored in this mode. However, you can still use the `setRCFactor` command to set scaling factors when running Single or Best-Case Worst-Case (BcWc) timing analysis. Use the `setExtractRCMode` command to set the extraction mode for all RC corner objects in the design.

RC corner objects are referenced when creating delay calculation corner objects.

➤ To create an RC corner, use the following command:

`create_rc_corner`

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
```

Important

You can use the `update_rc_corner` command before the multi-mode multi-corner definition file is loaded into the design, or after. However, after the software processes the file, any change to an existing object when the software is in multi-mode multi-corner mode results in the timing, delay calculation, and RC data being reset.

You can make changes to an RC corner object using the Edit RC Corner form, but only before the multi-mode multi-corner view definition file is loaded into the design. After the software processes the file, this form is no longer available.

➤ 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.

Creating Pre-Scaled Capacitance Tables

If you need unique RC scaling per RC corner, you can use the `generateCapTbl` command to prescale an existing capacitance table. For example:

```
generateCapTbl
    -ict myTech_nc.CapTbl
    -cap 1.10
    -res 1.10
    -output myTech_wc.CapTbl

create_rc_corner -name rc-worst -cap_table myTech_wc.CapTbl
```

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 Best-Case and Worst-Case differences.
- ❑ 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`

Important

You can use the `update_delay_corner` command before the multi-mode multi-corner definition file is loaded into the design, or after. However, after the software processes the file, any change to an existing object when the software is in multi-mode multi-corner mode results in the timing, delay calculation, and RC data being reset.

You can make changes to a delay calculation corner object using the Edit Delay Corner form, but only before the multi-mode multi-corner view definition file is loaded into the design. After the software processes the file, this form is no longer available.

➤ 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.

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.

Instances contained within an MSMV power domain often require a different library or operating condition than that used for the default power domain. Power domain definitions within a delay calculation corner object can be used to assign specific libraries and PVT settings per 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:

`update_delay_corner`

Important

You can use the `update_delay_corner` command before the multi-mode multi-corner definition file is loaded into the design, or after. However, after the software processes the file, any change to an existing object when the software is in multi-mode multi-corner mode results in the timing, delay calculation, and RC data being reset.

You can make changes to a power domain definition using the Edit Power Domain form, but only before the multi-mode multi-corner view definition file is loaded into the design. After the software processes the file, this form is no longer available.

➤ Choose *Design – Import Design*, click the *Advanced* tab, select *MMMC*, click the *MMMC Browse* button, and double click on the name of the power domain definition you want to edit.

or:

➤ Choose *Timing – Configure MMMC*, and double click on the name of the power domain definition you want to edit.

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`

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`

Important

You can use the `update_constraint_mode` command before the multi-mode multi-corner definition file is loaded into the design, or after. However, after the software processes the file, any change to an existing object when the software is in multi-mode multi-corner mode results in the timing, delay calculation, and RC data being reset.

You can make changes to a constraint mode object using the Edit Constraint Mode form, but only before the multi-mode multi-corner view definition file is loaded into the design. After the software processes the file, this form is no longer available.

➤ 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 MMMC*, and double click on the name of the constraint mode object you want to edit.

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.

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_input_delay`
- `set_output_delay`
- `set_false_path`
- `set_multicycle_path`
- `set_max_delay`
- `set_clock_uncertainty`
- `set_clock_latency`
- `set_min_delay`
- `set_propagated_clock`
- `set_case_analysis`
- `set_disable_timing`
- `set_annotated_delay`
- `set_annotated_check`
- `set_annotated_transition`
- `set_clock_transition`
- `set_drive`
- `set_driving_cell`
- `set_fanout_load`
- `set_input_transition`
- `set_max_transition`
- `set_resistance`
- `set_load`

The following constraints cannot be isolated between views for either multi-mode, or multi-mode multi-corner analysis:

- `set_max_time_borrow`
- `set_clock_gating_check`
- `group_path`
- `set_min_pulse_width`

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`

Important

You can use the `update_analysis_view` command before the multi-mode multi-corner definition file is loaded into the design, or after. However, after the software processes the file, any change to an existing object when the software is in multi-mode multi-corner mode results in the timing, delay calculation, and RC data being reset.

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. You should limit the number of active views for optimization to 10 views or less.

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

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 31-1 Application Behavior in Multi-Mode Multi-Corner Analysis Mode

Flow Step	Commands	Application Behavior
Placement	`setPlaceMode -timingDriven true` `placeDesign`	Uses the slack values from the default active setup analysis view.
	`placeDesign -inPlaceOpt`	Runs `optDesign -preCTS` on all active analysis views.
Pre-CTS Timing Optimization	`optDesign -preCTS`	Optimizes all active analysis views concurrently.
	`timeDesign -preCTS`	Analyzes all active analysis views concurrently.
Clock Tree Synthesis	`setCTSMode -multiCorner true`	By default, CTS is in multi-corner mode when the `set_analysis_view` command is specified.
	`specifyClockTree`	Reports the RC information for each active analysis view, including the estimated capacitance, resistance, and via resistance for each view.
	`ckSynthesis` `ckECO`	Attempts to balance the skew for all active setup and hold analysis views. For other constraints, such as transition, capacitance, and delay, CTS attempts to meet the requirements for the default analysis view.
	`reportClockTree -view` *`view`*	Reports the trigger slew for each active analysis view 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.

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.
	`timeDesign -postRoute`	Analyzes all active analysis views concurrently.
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 only in the default analysis view.
Timing Signoff	`timeDesign -signoff`	Calls `runQRC` iteratively to generate and annotate SPEF files for RC corners.
	`timeDesign -signoff -si`	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 the results from each active analysis view, as well as an aggregated summary.

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.

- To generate a User Case Analysis table for each analysis view, use the following command:

  `report_case_analysis`

- To generate a report containing pin and delay arc timing context information for each analysis view, use the following command:

  `report_cell_instance_timing`

- To generate a report containing information on disabled timing and timing check arcs for each analysis view, use the following command:

  `report_inactive_arcs`

- To generate a report containing net information for each analysis view, use the following command:

  `report_net`

- To generate a report containing path exception information for each analysis view, use the following command:

  `report_path_exceptions`

- To perform a variety of consistency and completeness checks on the timing constraints for each analysis view, use the following command:

  `check_timing`

32

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 and using the `icecaps` command, see the *Fire & Ice QX Gate-Level Extraction* 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, Fire & Ice QX, IceCaps, and RCgen determine 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_1 ... W_n
  - rho_spacings S_1 ... S_m
  - rho_values R_{11} .... R_{1n}
    ....
    R_{m1} ... R_{mn}

  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_{top}

  `wire_top_enlargement_r` E_{top}

  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 32-1 on page 1043.

- `wire_bottom_enlargement_c` E_{bottom}

`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 32-1 on page 1043.

Figure 32-1 Trapezoidal Wire Shape Resulting from Manufacturing Processes

- `wire_edge_enlargement | wire_edge_enlargement_[r|c]`

  `wee_widths` W_1 ... W_n

  `wee_spacings` S_1 ... S_m

  `wee_adjustments` E_{11} ... E_{1n}

  .

  .

  .

  E_{m1} ... 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 ... 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 ... 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}...E_{1n} ... E_{m1}...E_{mn}

  Specifies the enlargement, either positive or negative, of the wire edge.

See "Wire-Width Values" on page 1045 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 32-2 on page 1044 illustrates these parameters.

Figure 32-2 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:

drawn_width + ((*top* + *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, Fire & Ice QX uses the wire widths in the table, which always include biasing and wire-edge enlargement. If this table is not available, Fire & Ice QX calculates the resistance as follows:

rho * *L* / (*drawn_width* + (*top* + *bottom*) / 2 + (*top* + *bottom*) / 2)

where *rho* is the sheet resistivity.

Fire & Ice QX uses wire-width values in the following order:

1. Drawn width
2. Biased width
3. Edge-enlarged width
4. Resistivity as a function of width

Figure 32-3 on page 1046 illustrates the defining of the conductor layer.

Figure 32-3 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
```

```
                    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 32-4 on page 1048 illustrates the planar dielectric syntax.

Figure 32-4 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 1047 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 32-5 on page 1049 illustrates the defining of the passivation layer.

Figure 32-5 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
```

```
}
```

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
}

##########################################################################
```

```
# 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
```

```
    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
```

```
# 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.
```

```
# 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
}
```

```
##########################################################################
# 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
}

##########################################################################
```

```
# 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” {
```

```
# 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 “PlanarIntraMetal5” {
# Special planar dielectric which intersects “UniformConformal3”
    conformal                 FALSE
    height                    6.47
    thickness                 0.40
    dielectric_constant       3.00
```

```
}
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
```

```
    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 “VA1” {
    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
```

```
}
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
}
```

33

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 1064
- Pre-Mask ECO Changes from an ECO File on page 1066
- Pre-Mask ECO Changes from a New Verilog File on page 1070
- Pre-Mask ECO Changes from a New DEF File on page 1073
- Post-Mask ECO Changes from a New Verilog Netlist on page 1077
- Post-Mask Gate Array Style ECO from a New Verilog Netlist on page 1082

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 an ECO File on page 1066

  Allows you to use an ECO file to make changes to the netlist.

- Pre-Mask ECO Changes from a New Verilog File on page 1070

  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 1073

  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.

- Post-Mask ECO Changes from a New Verilog Netlist on page 1077

  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.

- Post-Mask Gate Array Style ECO from a New Verilog Netlist on page 1082

  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 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.

- 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.

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).

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

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.
- Instances existing only in the `oldchip.def` file are ignored, so they are not added to the current netlist.
- 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.

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.

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,

- 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.

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).

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

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 #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.
   - 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) 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.

4. Specify the spare cell list.

   ```
   specifySpareGate -inst SPARE*
   ```

5. 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.

6. 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`

- 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.

7. (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.

8. (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.

9. 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.

10. (Optional) Add notches.

```
fillNotch
```

Note: Skip this step if you specify freeze metal layers, because this command modifies all layers.

11. (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.

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 `metal1` 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 GAFILL -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`).

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.

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.

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 1107.

The directives are presented in alphabetical order.

- ADDHIERINST on page 1089
- ADDINST on page 1090
- ADDNET on page 1092
- ATTACHTERM on page 1093

- CHANGECELL on page 1095
- CHANGEINST on page 1096
- DELETEBUFFER on page 1098
- DELETEINST on page 1100
- DELETENET on page 1101
- DETACHTERM on page 1102
- INSERTBUFFER on page 1103
- SWITCHTERM on page 1106
- Example ECO File on page 1107

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

`instName`	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.
`moduleName`	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
     [-noNewPorts]
     instName
     cellName
     [numTerms]

[INSTTERM termName netName [hierPortName] … ]
```

Adds an instance of a cell to the netlist.

Important

You cannot include `INSTTERM` directives on the same line as the `ADDINST` directive. You must specify the first `INSTTERM` directive on the line after the `ADDINST` directive. You must specify each additional `INSTTERM` directive on its own line.

Parameters

`cellName`	Specifies the cell for which the `ADDINST` directive creates an instance. If *`cellName`* does not exist in the netlist, the software displays an error message and the directive stops.
`hierPortName`	Specifies the hierarchical port used to connect *`termName`* to *`netName`*. If you specify the *`hierPortName`* parameter, the hierarchical port name must exist in the module that contains *`instName`*. The *`hierPortName`* port must connect to *`netName`*. This parameter lets you use a specific port to maintain the same netlist topology, which simplifies equivalence checking later in the design flow. *Default:* If you do not specify this parameter, the `INSTTERM` directive uses existing ports or creates new hierarchical ports as necessary to connect *`termName`* to *`netName`*.
`instName`	Specifies the name of the instance created.

`INSTTERM`	Connects *`termName`* of *`instName`* to *`netName`*. You can specify one or more `INSTTERM` directives. If the terminal already connects to a current net, the directive first detaches the terminal from that net. If you do not specify the `INSTTERM` directive, the `ADDINST` directive creates new hierarchical ports as necessary to connect *`termName`* to *`netName`*. If *`termName`* or *`netName`* do not exist, the software displays an error message and the `ADDINST` directive stops. If you specify one or more `INSTTERM` directives, you must specify the *`numTerms`* parameter. *Default:* If you do not specify the *`numTerms`* parameter, the `INSTTERM` directives are ignored and the software displays a syntax error message.
`netName`	Specifies the net to connect to *`termName`*.
`-noNewPorts`	Prohibits the `INSTTERM` directives from creating hierarchical ports. If *`termName`* cannot connect to *`netName`* through existing ports, the software displays an error message and the directive stops. *Default:* If you do not specify this parameter, the `INSTTERM` directive creates hierarchical ports as needed.
`numTerms`	Indicates the number of `INSTTERM` directives specified. *Default:* If you do not specify this parameter, the software ignores any `INSTTERM` directives.
`termName`	Specifies the name of the terminal that is connected to *`netName`* by the `INSTTERM` directive.

Example

- The following directives create a new instance, `i1/i2/i3`, of `myBuffer`:

```
ADDINST i1/i2/i3 myBuffer 2
INSTTERM myIn i1/net1
INSTTERM myOut i1/net6 Cntr1Out
```

  A new hierarchical port is created on `i1/i2` if necessary to connect terminal `myIn` of `i1/i2/i3` to `i1/net1`. The terminal `myOut` only uses the existing port `Cntr1Out`.

  The `Cntr1Out` port must exist on `i1/i2` (the module containing `i1/i2/i3`) so `myOut` can connect to net `i1/net6`. `Cntr1Out` must connect to `i1/net6`; if it does not, the directive displays an error message and the directive stops.

ADDNET

`ADDNET` *`netName`*

Adds a net to the netlist.

Parameters

`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.

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.

ATTACHTERM

```
ATTACHTERM
     [-noNewPorts]
     instName
     termName
     netName
     [hierPortName]
     [-Hinst]
```

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

hierPortName	Specifies the hierarchical port used to connect *termName* to *netName*. If you specify the *hierPortName* parameter, the hierarchical port name must exist in the module that contains *instName*. The *hierPortName* port must connect to *netName*. This parameter lets you use a specific port to maintain the same netlist topology, which simplifies equivalence checking later in the design flow. *Default:* If you do not specify this parameter, the ATTACHTERM directive uses existing ports or creates new hierarchical ports as necessary to connect *termName* to *netName*.
-Hinst	Specifies the name of the hierarchical module that contains *instName*.
instName	Specifies the instance containing *termName*, which the ATTACHTERM directive attaches to *netName*. If *instName* does not exist, the software displays an error message and the directive stops.
netName	Specifies the name of the net used to attach to *termName*. If *netName* does not exist, the software displays an error message and the directive stops.
-noNewPorts	Prohibits the ATTACHTERM directive from creating hierarchical ports when it attaches a terminal. If *termName* cannot connect to *netName* through existing ports, the software displays an error message and the directive stops. *Default:* If you do not specify this parameter, the ATTACHTERM directive creates hierarchical ports as needed.

termName	Specifies the terminal of *instName* that is connected to *netName*. If *termName* does not exist, the software 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.

CHANGECELL

```
CHANGECELL
     instName
     newCellName
     [oldCellName]
```

Changes the cell type of an instance to a new cell type. You can only change cells that have the same footprint and functionality. If you provide the existing cell type (*oldCellName*), this directive verifies that the specified instance (*instName*) is a current instance of the existing cell type before changing the cell type to a new type (*newCellName*).

If the current instance is already placed, the modified instance keeps the existing placement. Keeping the old placement might cause overlap violations between the modified instance and neighboring instances if the new cell is larger than the existing cell.

Parameters

instName	Specifies the name of the instance whose cell type you want to change.
newCellName	Specifies the new cell type. The *newCellName* must exist in the library; otherwise, the software displays an error message and the directive stops.
oldCellName	Specifies the existing cell type. If the current cell is not *oldCellName*, the software displays an error message and the directive stops.

Example

- The following directive changes the instance `i1/i2/i3` of type `and2_1x` to cell type `and2_2x`:

  ```
  CHANGECELL i1/i2/i3 and2_2x and2_1x
  ```

  If `i1/i2/i3` does not exist, the software displays an error message and the directive stops. If `and2_2x` does not exist in the library, or if all terminals of `and2_2x` do not exactly match the terminals of the current cell, or if the current cell is not `and2_1x`, the software displays an error message and the directive stops.

CHANGEINST

```
CHANGEINST
     instName
     newCellName
     oldCellName
     nrNewCellFTerm

TERM newTermName oldTermName
```

Changes the cell type (master) for an instance. Unlike the `CHANGECELL` directive, you can exchange the cell type for one with a different footprint but identical functionality. If you provide the existing cell type (*`oldCellName`*), the `CHANGEINST` directive verifies that the specified instance (*`instName`*) is a current instance of the existing cell type before changing the cell type to a new type (*`newCellName`*).

If the current instance is already placed, the modified instance keeps the existing placement. Keeping the old placement might cause overlap violations between the modified instance and neighboring instances if the new cell is larger than the existing cell.

Parameters

`instName`	Specifies the name of the instance you want to change.
`newCellName`	Specifies the new cell type. The *`newCellName`* must exist in the library and all terminals of *`newCellName`* must match the terminals of the current cell type exactly; otherwise, the software displays an error message and the directive stops.
`nrNewCellFTerm`	Specifies the number of terminals belonging to the new cell type.
`oldCellName`	Specifies the existing cell type. If the current cell is not *`oldCellName`*, the software displays an error message and the directive stops.
`TERM`	Specifies the old and new terminal names for the instance. You must specify a `TERM` directive for every terminal name that must change.
	`newTermName` Specifies the name of the terminal on the changed instance.
	`oldTermName` Specifies the name of the terminal on the existing instance.

Example

- The following directives exchange a three-terminal flip-flop master (FF3) for a four-terminal master (FF4) for instance i1/i2.

  ```
  CHANGEINST i1/i2 FF4 FF3 4
  TERM D D
  TERM Q Q
  TERM CK CLK
  ```

  Suppose that FF3 has input D connected to net n1, clock terminal CK connected to net clock, and output Q connected to n2. FF4 has input D connected to n1, clock terminal CLK connected to net clock, output Q connected to n2. Flip-flop FF4 also has output QN. When the master for i2/i2 is changed to FF3, QN is not connected to a net.

DELETEBUFFER

```
DELETEBUFFER
     instName
     keepNetName
     [deleteNetName]
```

Deletes a buffer instance after merging the nets on both sides of the buffer into one net.

Parameters

deleteNetName	Specifies the name of the net connected to the terminal on *instName* opposite to the terminal that connects to *keepNetName*. If *deleteNetName* is not connected to the terminal on *instName* opposite to the terminal that connects to *netName*, the software displays an error message and the directive stops. For example, if *keepNetName* connects to the input of *instName*, *deleteNetName* specifies the name of the net connecting to the output. The software uses the *deleteNetName* 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.
instName	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 *keepNetName*.
keepNetName	Specifies the name of the existing net into which the nets from both of the instance's terminals are merged. The *keepNetName* 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
  ```

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` *`instName`*

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.

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.

DELETENET

`DELETENET` *`netName`*

Deletes a net after deleting all the instance terminal connections to the net. If routing is connected to the net, the routing is deleted.

Parameters

`netName`	Specifies the name of the deleted net. If *`netName`* does not exist, the software displays an error message and the directive stops.

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.

DETACHTERM

```
DETACHTERM
     instName
     termName
     [netName]
     [-Hinst]
```

Disconnects a terminal from a net.

Parameters

`-Hinst`	Specifies the name of the hierarchical module that contains *`instName`*.
`instName`	Specifies the instance that contains *`termName`*.
`netName`	Specifies the net that is already connected to *`termName`*. If *`termName`* does not connect to *`netName`*, or if *`netName`* 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 *`instName`*, 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.

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

`netName`	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.
`nrBuffer`	Specifies the number of buffers attached to the net.
`nrNetTerm`	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.
	`instName` — Specifies the name of the buffer instance to insert.
	`cellName` — Specifies the cell master for the buffer instance.

	nrInstTerm	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 *termName* and *netName* parameters are required. you must specify the INSTTERM directive for each buffer you want to add.	
	termName	Specifies the name of the terminal on the buffer.
	netName	Specifies the net to connect with the terminal.
	portName	Specifies the physical port corresponding to the terminal.
NETTERM	Specifies the net connection between a driver terminal and the added buffer.	
	instName	Specifies the instance containing the terminal.
	netName	Specifies the net connected to the terminal.
	termName	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
    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`.

SWITCHTERM

```
SWITCHTERM
     instName
     termName1
     termName2
```

Exchanges the terminal-to-net connection of *termName1* with that of *termName2* on the specified instance.

Parameters

instName	Specifies the name of the instance containing the terminals *termName1* and *termName2*.
termName1	Specifies the name of the terminal whose net connection is exchanged with that of *termName2*.
termName2	Specifies the name of the terminal whose net connection is exchanged with that of *termName1*.

Example

- The following directive exchanges the net connections of terminal `in1` on instance `i1/i2/i3` with those of terminal `in2`:

  ```
  SWITCHTERM i1/i2/i3 in1 in2
  ```

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.

Important

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

#
```

```
# DELETEINST: Delete block level instance
#
DELETEINST DTMF_INST/m_clk__L6_I6

#
# CHANGECELL: Change block level buffer
#
CHANGECELL DTMF_INST/m_clk__L6_I4 BUFX1

#
# 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

#
# DELETEBUFFER: Delete an existing buffer
#
DELETEBUFFER DTMF_INST/i_10048 resetI

#
# SWITCHTERM: Switch existing terminals
#
SWITCHTERM DTMF_INST/RESULTS_CONV_INST/i_1642 A B
#
# CHANGEINST: Change the master for an instance
#
CHANGEINST DTMF_INST/ARB_INST/i_1 NOR4BX1 NOR2X1 5
TERM Y Y
TERM AN A
TERM B B
#
# INSERTBUFFER: Inserts a buffer on a net
#
INSERTBUFFER DTMF_INST/DMA_INST/n_81 3 3
INST b1 BUFX1 2
    INSTTERM A DTMF_INST/DMA_INST/n_81
    INSTTERM Y n1
INST b2 BUFX1 2
    INSTTERM A n1
    INSTTERM Y n2
INST b3 BUFX1 2
    INSTTERM A n1
    INSTTERM Y n3
NETTERM DTMF_INST/DMA_INST/i_12 Y DTMF_INST/DMA_INST/n_18
NETTERM DTMF_INST/DMA_INST/i_44 B n2
NETTERM DTMF_INST/DMA_INST/i_4294 A1 n3
```

Clock Mesh Specification File

- Overview on page 1111
- Routing Type Definitions on page 1112
- Cutout Definitions on page 1112
- Clock Mesh Definitions on page 1113
- Clock Mesh Specification File Example on page 1129

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 1112
- Cutout Definitions on page 1112
- Clock Mesh Definitions on page 1113

You can specify the following units in a clock mesh specification file:

- Distance unit: µm
- 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` *`typeName`*	
	Specifies the routing type for which you are defining routing attributes. This attribute denotes the beginning of the `RouteTypeDef` section.
`Layer` *`layerName`*	Specifies the metal layer to be used. The name you specify must be a layer name defined in the LEF file.
`Width` *`distance`*	Specifies the width of the metal layer to be used.
`Spacing` *`distance`*	Specifies the spacing to all other wires in the design.
	Default: 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)
- Mesh Stage 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` *`timeValue`*	Specifies the clock period (in picoseconds) for the mesh. *Default:* `0` (frequency also will default to `0`)
`SupplyVoltage` *`voltageValue`*	Specifies the power supply voltage for the mesh. The software uses this value when analyzing clock mesh power. *Default:* `1V`
`MaxPower` *`timeValue`*	Specifies the maximum power value (in milliwatts) for the mesh. **Note:** Currently, the clock mesh tool does not use this value when implementing the mesh. *Default:* `0`
`RootTrans` *`timeValue`*	Specifies the transition time (in picoseconds) at the root pin. *Default:* `0`
`MinDelay` *`timeValue`*	Specifies the minimum phase delay (in picoseconds). **Note:** Currently, the clock mesh tool does not use this value when implementing the mesh. *Default:* `0`
`MaxDelay` *`timeValue`*	Specifies the maximum phase delay (in picoseconds). **Note:** Currently, the clock mesh tool does not use this value when implementing the mesh. *Default:* Largest possible number, based on the machine
`MaxSkew` *`timeValue`*	Specifies the maximum skew between sink pins (in picoseconds). **Note:** Currently, the clock mesh tool does not use this value when implementing the mesh. *Default:* `0`

`MaxBufferTrans` *`timeValue`*

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 and physical attributes of the global 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.

`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
```

`MeshDrivePoint {Center | Root | x,y}`

Specifies the position of the first-stage pre-drivers.

Default: `Center`

`Center`	Positions the pre-drivers near the center of the clock mesh structure.
`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.

`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`.

`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.

`TrunkDriveDist {StrictAttach | Uniform | LoadWeighted | LoadWeightedMatch}`

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.

Mesh Stage Section

This section defines stage-specific parameters for the global mesh, including driver cells, route types, and trunk and branch information.

The following table describes the entries for the mesh stage section:

`Stage`	Denotes the beginning of a particular stage definition. Each stage definition must begin with a `Stage` statement, and close with an `End` statement. **Note:** You must define an even number of stage definitions if you use an inverter for the drive cell (`DriveCell`).
`NumDriver` *`number`*	Specifies the number of drivers that should be inserted at the current stage.

`X` *`number`*

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.

Note: This statement only can be used when you are defining an HTree + Mesh structure.

Default: Uses a single driver at the end of each node.

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.

`DriveCell` *`cellName`*

Specifies the type of driver to be used for the stage being defined.

`RouteTypePair` *`type1 type2`*

Specifies the routing types to be used for the stage. Routing types are defined using the `RouteTypeDef` statement.

`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 of 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`*

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`*

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.

`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.

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]`	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:

`LocalTree`

Denotes the beginning of the local tree definition. The local tree definition must begin with a `LocalTree` statement, and close with an `End` statement.

`Enabled [true | false]`

Enables the implementation of the local tree.

Default: `false`

`RootPos {ClusterCenter | OnMesh | NearMeshInCluster}`

Specifies how to place the buffers.

Default: `ClusterCenter`

`ClusterCenter`

Places buffers at the cluster center of gravity.

`OnMesh` Places buffers on or along the nearest mesh trunk or branch. Buffer placement can extend beyond the bounding box of the cluster.

`NearMeshInCluster`

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...`

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.

`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
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.
    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

    #MeshDrive Point 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

Mesh Stage definition
    #Definition for Stage 1
    Stage
        NumDriver 1
        X 4
        DriveCell CLKBUFX20
        RouteTypePair RT1 RT2
    End

    #Definition for Stage 2
    Stage
        NumDriver 4
        X 2
        DriveCell CLKBUFX20
        RouteTypePair RT1 RT2
    End

    #Definition for 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)
    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
        PrefferedExtraSpace 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
    End

#Marks the end of the clock mesh defintion
End
```

CPF Script Example

```
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
#########################################

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 \
        -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

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
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 conatins 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 "\
        $libdir/technology45_lvll2h_1p1.lib \
"
set lib_ao_bc_extra "\
        $libdir/technology45_lvll2h_1p3.lib \
"
set lib_core_wc_extra "\
```

```
        $libdir/technology45_lvlh2l_0p8.lib \
        $libdir/technology45_headers_0p8.lib \
        $libdir/technology45_sprg_ao_0p8.lib \
"
set lib_core_bc_extra "\
        $libdir/technology45_lvlh2l_1p0.lib \
        $libdir/technology45_headers_1p0.lib \
        $libdir/technology45_srpg_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 \
        $libdir/rom_512x16_fast.lib \
"
#########################
#### Define librray sets
#########################
define_library_set -name ao_wc_0p8 -libraries "$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"

#############################
#### 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
##########################################

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 \
        -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
#######################################################################
##### 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}

update_state_retention_rules -names SRPG_CORE \
        -cell SRPG2Y \
        -library_set tdsp_wc_0v792

end_design
```

Index

Numerics

64-bit mode, starting 39

A

AC current violations, checking 951
AC limit
verifying 951
actual value, in Violation Browser 953
ADDHIERINST 1089
Adding logical hierarchy without creating additional hierarchy 344
ADDINST 1090
ADDNET 1092
Amoeba view
using during placement 455
antenna cells
ensuring connectivity after inserting with NanoRoute router 644
highlighting after inserting with NanoRoute router 645
See also process antenna
antenna violations
as distinguised from process antenna violations by verifyConnectivity and verifyGeometry 945
antennas
types of connectivity and geometry violations reported 945
antennas, trimming 556
area I/O
See also flip chip
placement, and I/O assignment files 74
array vias
in yield report, definition of 967
in yield technology file 982
assign nets, removing from Verilog netlist 96
Astro or Apollo, using with NanoRoute router 648
ATTACHTERM 1093
attributes
NanoRoute router 596
persistency of 596
VoltageStorm cell library
cell_leakage_power 911
internal_power 911
auto fetch pad locations 925
Auto Fetch, using 926
automatic mode of CTS 478
Automatic Power Planning (APP), described 358

B

back-end designer, and approach to basic floorplanning 293
bindkeys, for wire editing 546
Blackbox Timing Analysis, described 177
blackboxes
deleting 201
flows 199
pins 208
R0 transformation 203
specifying, methods of 198
block pins
orientation of, changing, to resolve route congestion 582
block rings
creating 352
specifying coordinates with 353
block-level partitions, creating 267
blocks
changing module status from fence 266
displaying macro current source location 910
boundary voltage file, and power analysis 933
bounding box, defining 304
buffer trees, inspecting routing topology of in interactive ECO 886
buffers
added during timing optimization, viewing 877
attaching to I/O pins in interactive ECO 882
hole punch 259
inserting 260

problems with, in timing analysis 822
bump arrays 122
bumps
assigning 128
I/O assignment files and 75
routing 128
viewing flight lines 128, 140, 157
bus names, and partition pin guides 213
bus routing
constraint for 679
description of 660

C

CAA. *See* critical area analysis
capacitance calculations of the mixed signal router 663
capacitance comparison, graph and report 767
capacitance table
basic 757
creating 756
extended 757
generating 755
reading 761
CAR, editing wires with 564
cell blockage visibility, controlling 563
cell delay specification, and CTS 485
cell padding
and clock buffers 446
and highly localized clocks 446
and placement space 446
cell probability, section in yield technology file 982
cell timing model, and CTS 489
cell_leakage_power, VoltageStorm cell library attribute 911
cells
placing 450
CHANGECELL 1095
CHANGEINST 1096
channel estimation, reporting
example 265
channelless designs, and feedthroughs 246
clock buffers, and cell padding 446
clock fanout buffers, limiting 507
clock grouping, and CTS 486
clock mesh 858
clock nets
routing guide file in CTS for NanoRoute 499
routing with NanoRoute router 618, 619, 626
setting attributes for with NanoRoute router 618
clock tree specification file
automatic gated CTS section 503
clock grouping data section 503
clock tree topology section 503
contents 494
creating 494 to 522
example 494
global tracing directive section 501
macro model data section 500
and maximum skew value 486
NanoRoute attribute section 499
order of items in 494
clock tree synthesis. *See* CTS
clones, orientation of, in partitions 204
clustering, netlist 460
CML file, example for flip chip 165
command-line help and man pages 58
Common Timing Engine. *See* CTE
configuration files
loading 91
Configuration, OpenAccess 40
congestion
checking with NanoRoute router 607
resolving with density screens 582
resolving by reorienting block pins 582
congestion analysis table, generated by NanoRoute router 607
congestion distribution report
in Trial Route 575
usage and routing overflow percentage values in 575
congestion distribution table
acceptable values in 572
valued described 577
congestion map
used by NanoRoute router 588, 608
congestion markers, in Trial Route, described 572
connectivity
loops, detecting 941
types of violations reported 941
verifying 941
console, Encounter, described 46
constraint file
creating for mixed signal router 681

editing for mixed signal router 689
format for mixed signal router 661
loading for mixed signal router 686
sample of mixed signal router 691
constraint files, with timing budgets for partitions 717
constraints
bus 679
differential pair 668
keywords used by mixed signal router 664
matched nets 666
nets 664
shielding 670
conventions, syntax and typographic 34
coordinates, using to specify block rings or core rings 353
core dimensions
margin, described 296
size, described 296
core rings
creating 352
specifying coordinates with 353
core size, determining 301
core-to-I/O distance, specifying in partitioning 206
correlation between native and sign-off extraction, assuring 763
costs
definition of yield costs 966
formulas for yield costs 990
critical area analysis
definition of 959
cross-probing, in Violation Browser 953
crosstalk
correcting violations with NanoRoute router 620
effect on incremental delay 896
overview 896
preventing violations with NanoRoute router 620
crosstalk analysis, specifying noise analysis mode 900
CTE
using for timing-driven routing with NanoRoute router 615
CTS
automatic mode 478
and gated clocks 479
and nets 478
and cell delay specification 485
cell timing model requirement 489
clock buffers, and cell padding 446
clock designs with tight areas 489
clock grouping and automatic mode 486
clock tree specification file
automatic gated CTS section 503
clock grouping data section 503
clock tree topology section 503
contents 494
creating 494 to 522
example 494
global tracing directive section 501
macro model data section 500
NanoRoute attribute section 499
order of items in 494
files required for 476
gating cells, defined 513
hierarchical CTS analysis 487
highly localized clock, and cell padding 446
and INSERTION_DELAY statement 485
log file headings 522
macro model delay specification 485
manual mode 477
modes 477
and module placement utilization 489
padding cells, moving 513
and pin balancing for macro models 489
and pin instance delay specification 485
and port delay specification 485
preparatory procedure 476
and recommended placement utilization 447
tracing clock trees 479
cumulative defect data function
described 961, 988
custom macros
utilizing pre-defined feedthroughs 253
cut areas, and rectilinear partitions 205
cut layers, using NanoRoute router to repair PAE violations on 645
cuts
creating for power domains 380
cutting shielding wires 556

D

data
  checking before importing a design 86
  exporting 100
  importing 100
database
  saving, with NanoRoute router 589
database units (DBU), adjusting the value of for NanoRoute router 590
DBU. *See* database units
debugging
  NanoRoute router 650
Debugging Timing Results 826
decoupling capacitance
  adding 459
  defining cell candidates for 459
  removing 459
decoupling capacitor modeling 927
DEF files
  comparing with database 889
  comparison file format 889
DEF syntax, unsupported 61
defect data function
  described 961, 987
defect-limited yield
  definition of 958
DELETEBUFFER 1098
DELETEINST 1100
DELETENET 1101
demand, defined in Trial Route 576
density screens, using to resolve route congestion 582
density, calculating 301
design
  checking 83
  checking data before importing 86
  display area, described 294
  importing 87
  saving files 97
DETACHTERM 1102
detailed routing
  description of 589
  running in NanoRoute router 604
  violations, troubleshooting 604, 650
die size
  described 296
dielectric layer
  syntax 1047
differential pair routing
  constraint for 668
  description of 656
  shielding 659, 675
differential signal routing 158, 365
diodes
  See also antenna cells, process antennas
diodes, highlighting after inserting with NanoRoute router 645
display area, for floorplanning 294
Display Setting file 934
  VoltageStorm 934
distributing jobs 290
DLY. *See* defect-limited yield
documents, related, list of 35
DRC violations
  recognized by NanoRoute router 626

E

ECO
  naming conventions, table of 889
  netlist, updating 888
  placement 888
    general principles 888
ECO File
  ADDHIERINST 1089
  ADDINST 1090
  ADDNET 1092
  ATTACHTERM 1093
  CHANGECELL 1095
  CHANGEINST 1096
  DELETEBUFFER 1098
  DELETEINST 1100
  DELETENET 1101
  DETACHTERM 1102
  Example 1107
  INSERTBUFFER 1103
  SWITCHTERM 1106
ECO flows
  alternative pre-mask 1070
  description 1064
  GACORE cells 1082
  post-mask from new netlist 1077
  post-mask gate-array 1082
  pre-mask 1066
  pre-mask from new DEF 1073
Edit Route form
  populating 547
editing wires. *See* wire editing, wires

effective resistance, description of 663
effective utilization (EU value) 300
electrostatic discharge cells 122
EM values, calculating for wires and vias 935
Encounter console, described 46
Encounter software, starting 43
encrypting libraries 81
end-cap cells
  placing 439
  removing 440
environment, run-time, setting 39
Estimation Parameter file 914
EU value 300
exporting
  GDSII Stream or OASIS files 105
  TDF files 103
extraction
  and capacitance comparison graph and report 767
  capacitance table
    basic 757
    extended 757
    generating 755
    reading 761
  overview 748
  preparatory procedure 750

F

fabrication process information 1037
fanouts, clock buffer, limiting 507
fat via, definition of 634
feature license matrix, for Encounter products 42
feedback buffer 246
feedthrough insertion, and partitioning 246
feedthroughs 244 to 261
  buffers vs. routing 244
  and channelless designs 246
  channel-type 260
  inserting 244, 245
  pre-defined 253
  replicating across ECO netlists 249
  routing 259
fences
  changing module status to block 266
  constraints 297
  module constraint types 297
  restrictions 297
field solvers, 2-D and 3-D 756
file selection, general description 93
files
  boundary voltage 933
  Display Setting, VoltageStorm 934
  Estimation Parameter 914
  ICT
    comparing with LEF file 763
    sheet and via resistance values in 763
  instance power data 932
  instance voltage 932
  pad location 924
  power pad location
    fetching automatically 925
    saving 925
  required for CTS 476
  routing blockage, generating 197
  TDF, importing and exporting 103
  timing constraints 717
  updating during a session 119
  updating incrementally 119
filler cells
  adding to MSV designs 457
  implant layers 456
  placing 456
  placing gate-array style cells in ECO 457
  removing 457
  replacing during process antenna repair with NanoRoute router 645
Flight 295
flight lines
  colors used by wire editor 551
  definition 295
  for bumps 128, 140, 157
flip chip
  defined 122
  flows
    bump flow 130
  methodology 122
  route feasibility report example 166
  routing bumps to I/O driver cells 140
  testing package routing feasibility 137
  useful tasks 123
floorplan
  data, types 100, 102
  display, described 294
  loading 343
  saving 343
floorplan files, containing routing blockage

data 197
floorplan objects
and partition feedthroughs 259
floorplanning
modes for power analysis
floorplan 907
layout 907
overview 292
relative 340 to 343
row spacing 302
sequence 293
specifying spare cells during 440
steps 293
footprint
using in timing optimization 876
footprintless flow, using for timing optimization 875
formula
used for yield costs 990
used to calculate the probability of failure 987
used to derive defect data and cumulative defect data functions from each other 989
front-end designer, and approach to basic floorplanning 293

G

GACORE cells
ECO flows 1082
gated clocks, and CTS automatic mode 479
gating cells, defined for CTS 513
gcells
definition of 588
gcells, obstructed and overflowed, summarized in Trial Route congestion distribution report 576
GDSII
and FOREIGN statements 106
exporting files to 105
map file format 111
merging files in 107
renaming LEF vias in 106
specifying floating fill in 116
specifying layers for nets in 116
specifying voltage levels in 117
support for GZIP format 105
suppressing text labels in 113
geometry
spacing violation checks during verification of 944
verifying 944
global net connections 353
global routing
description of 588
running in NanoRoute router 604
Global Timing Debug 826
guide constraints 297
guides, module constraint types 297

H

hard blocks
placing 434
hard blocks, changing status of partition from 266
help, command-line 58
hierarchical clock tree analysis, and CTS 487 to 488
hierarchical designs
timing budgeting, methods for 714
hierarchy icon, picture of 681
hole punch buffers, inserting 260
horizontal and vertical tracks, summarized in Trial Route congestion distribution report 576

I

I/O assignment files
multiple I/O rows in 74
ICT file
case sensitivity 1038
commands 1038
contents 1037
example 1051
format 1038
ICT files
comparing with LEF file 763
sheet and via resistance values in 763
implant layers, for filler cells 456
implementation
block-level 192
top-level 194
importing
TDF files 103
tile cell design data 103

incremental delay
crosstalk effect on 896
insert 244
INSERTBUFFER 1103
INSERTION_DELAY statement, and macro models in CTS 485
inserts 245
instance average power report 931
instance groups
adding an instance to 303
deleting 303
saving to netlist 303
specifying 300
instance power data file 932
instance voltage files 932
example for power and ground nets 932
interactive ECO
naming conventions, table of 889
overview 880
interconnect capacitance modeling
creating ICT file 1037 to 1061
internal_power, VoltageStorm cell library attribute 911
I/O assignment files
and area I/O placement 74
bumps and 75
creating 65
described 64
examples of I/O pads in 76
module pins in 78
rule-based, creating 75
I/O pads
core dimension size 296
and I/O assignment files 76
I/O pins, adding buffers to, in interactive ECO 882
island, and routing feedthroughs 260

J

JTAG cells
placing 461

K

keyboard shortcuts, for wire editing 546
keywords
mixed signal router constraint 664

L

layers
changing during wire editing 559
routing, checking metal density of 943
LEF files
checking for availability before importing 86
checking for compatibility with NanoRoute router 590
comparing with ICT file 763
example for flip chip 165
LEF statements
for metal density 943
for process antenna calculations 947
for verifying geometry 944
LEF syntax, unsupported 61
libraries
checking for availability before importing 86
timing 81
license matrix, for Encounter products 42
licenses
checking out 45
log file, congestion analysis table in 607
Logical hierarchy manipulation 344

M

macro current source
location 910
location, displaying for blocks 910
macro model delay, and CTS 485
macro models, pin banancing for 489
MACRO, creating cells in GDSII Stream or OASIS format from 105
macros
placing 434
man pages, command-line 58
manual mode of CTS 477
map file format, GDSII or OASIS, described 111
master partitions, defined 203
Masterplan Automatic Floorplanner
adjusting macro placement
adjusting a specific macro pack 1006
adjusting within a specified area 1007

interactive commands, using 1005
constraint file
creating a seed section 996
seed section, definition of 996
criteria for a good floorplan 1005
data preparation 994
examing modules for seed selecion 997
floorplan analysis
checking macro placement 1004
checking the seed report 1004
visual check 1003
generating an initial floorplan 1001
generating floorplan guides 1011
guidelines for hierarchical floorplanning 1003
ideal designs for use with 992
input files for 994
macro pack, definition of 1006
marking refinement steps 1010
overview 992
performing detailed cell placement 1011
preplacing cells with I/O connections 994
restoring design to previous state during refinement 1010
seed selection
automatic selection 995
definition 994
how seeds are used 994
optimal seed number 995
user defined selection 995
setting global parameters for 1001
task flow 993
use model 1001
matched net routing
constraint for 666
description of 655
maximum fanouts, limiting for clock buffers 507
maximum floating area, verifying 950
maximum skew value, and clock tree specification file 486
maximum wire width, repairing violations of 558
merging GDSII or OASIS files 107
metal density
LEF statements for 943
verifying 943
metal fill
adding over macros 708
connected, definition of 699
cross-vias not used in 699
definition of 696
floating 707
LEF statements for 705
macro density considerations of 709
power strapping recommendations for 709
preparatory procedure 696
recommendations for 705
setting parameters for 704
staggering 698
tied-off 707
tied-off, definition of 699
timing aware 703
trimming 711
problems caused by vias 699
visibility of 697
Milkyway technology file 648
minimum cut violations, fixing 355
minimum spacing violations, fixing 355
mixed signal router
capacitance calculations of 663
constraint file
creating 681
editing 689
format of 661
keywords used in 664
loading 686
sample of 691
constraints
bus 679
diffpair 668
matched nets 666
nets 664
shielding 670
resistance calculations of 663
specialized routing performed by 654
MMMC
specifying for timing optimization 868
mode
standalone mode, NanoRoute router 595
modes
auto query 547
CTS, automatic 478
CTS, manual 477
power analysis 907
power analysis, floorplan 907
power analysis, layout 907

timing-aware metal fill 703
module constraint types
described 297
fence 297
guide 297
and physical design size 298
region 298
soft guide 298
module fences, restrictions 297
module guides 297
module pins, and I/O assignment files 78
module placement utilization, and CTS 489
module status, changing from fence to block 266
-modulePlan
limitations of 434, 450
-modulePadding and 448
multi-threading and 451
modules
placement utilization and CTS 489
MSV (multiple supply voltage)
power analysis 391
MSV designs
adding end-cap cells to 439
adding filler cells to 457
adding well-tap cells to 438
multi-cut vias, using 634
multiple I/O rows, specifying in I/O assignment file 74
multiple supply voltage
flat flow 372
multiple-CPU processing
placement 451
multi-threading
controlling licenses for 43, 52
placement 451
running 52
verifyGeometry 52

N

naming conventions
for ECO 889
for timing optimization 877
NanoRoute router 644
adjusting the database units (DBU) value for 590
Astro or Apollo, using with the router 648
attributes
characteristics of 596
compared with options 596
listed. See `setAttribute` in the *Encounter Text Command Reference*.
clock nets
routing 619
setting attributes for 618
commands to run within Encounter environment 594
congestion analysis table, using 607
congestion map 588, 608
congestion, checking 607
connectivity, power and ground pins 644
crosstalk
correcting violations 620
preventing violations 620
CTE, using 615
database, saving 589
debugging 650
detailed routing 604
evaluating violations in 626
generating tracks for 592
global routing 604
hard spacing, forcing 642
LEF files, checking for problems in 590
nets, shielding 638
nondefault rule routing 642
nondefault spacing in 642
open net messages, meaning of 628
options
compared with attributes 596
crosstalk prevention 622
listed. See `setNanoRouteMode` in the *Encounter Text Command Reference*.
process antenna repair 646
via optimization 635
output files from 592
overlapping cells, checking for 591
overview 588
pin access, improving 590
pins, checking for under power routes 591
postroute optimization in 606
preparatory procedure 590
prerouted nets, skipping during routing 625, 639

routing
accelerating 599
clock nets 618, 626
detailed 589
distributed 599
ECO 624
global 588
shielded 637
signal-integrity driven 622
standalone mode 595
statistics report 593
strategy 603
wide wires 642
routing layers, specifying in DEF 592
shielded routing 637, 638
signal integrity, preventing problems with 620
soft spacing 642
soft spacing routing 642
table of violations 628
tapering wires in 642
timing engine, specifying 615
timing graph, using with standalone NanoRoute router 616
timing-driven routing 632
troubleshooting design problems 650
vias
checking for rotated 591
defining TOPOFSTACKONLY 591
fat, definition of 634
mapping Astro scheme format of 648
minimizing the number of 636
optimizing 634
optimizing in selected nets 635
using multi-cut 634
violations
on upper layers 630
horizontal 626
marker placement for 626
process antenna, repairing 644
table for evaluating 628
types of 626
vertical 626
via 626
native extraction
default mode 751
defined 751
detailed mode 751
flow, illustrated 751
See also extraction, sign-off extraction
net attributes
characteristics of in NanoRoute router 596
net average power report 931
net group names, and partition pin guides 213
net names, and partition pin guides 213
netlist clustering 460
netlists
ECO, replicating feedthrough insertions 249
ECO, updating 888
and feedthroughs 244
preparing 86
saving instance groups to 303
Verilog, concatenating for a partitioned design 286
Verilog, removing Assign nets from 96
nets
and CTS automatic mode 479
deleting violated nets 633
names, and partition pin guides 213
shielding, with NanoRoute router 638
nets routing constraint 664
noise analysis mode, specifying for crosstalk analysis 900
nominal voltage 912
nondefault rule routing, NanoRoute router 642
nonrectangular partitions 205

O

OASIS
and FOREIGN statements 106
exporting files to 105
map file format 111
merging files in 107
renaming LEF vias in 106
specifying floating fill in 116
specifying layers for nets in 116
specifying voltage levels in 117
support for GZIP format 105
suppressing text labels in 113
obstructed gcell information, summarized in Trial Route congestion distribution report 576
OCV 809
on-chip variation 809, 868
online documentation (CDSDoc),

accessing 57
open net messages, generated by NanoRoute router 628
open point defect, section in yield technology file 974
OpenAccess
restoring a design 95
saving a design 95
OpenAccess, configuration 40
operating voltage 912
operating voltage, default 936
optimizing timing. *See* timing optimization
optimizing vias 634
options
NanoRoute router
characteristics and persistency of 596
compared with attributes 596
crosstalk prevention 622
process antenna 646
via optimization 635
orientation, of blocks, changing 582
orientation key 340
Ostrich
parasitics correlation utility 763
OverCon #Gcell, definition of 607
overflow percentage values, in Trial Route congestion distribution report 575
overflowed gcells, summarized in Trial Route congestion distribution report 576
overlapped objects
navigating in Violation Browser 953
overlapping cells, checking for with NanoRoute router 591
overlapping objects
selecting 552

P

package routing feasibility
testing, using APD 137
pad location file 912
creating 924
padding
placing 441
padding cells
moving in CTS 513
parallel job runs 290
partition feedthroughs, defining 259
partition pin guides
and bus names 213
and net group names 213
partition pinS
guide objects 197
partition pins
guide names, changing 213
guides, adding 213
ranges 242
snapping 242
partitioning
block-level 267
channel estimates, reporting example 265
concatenating netlists of 286
core-to-I/O distance, specifying 206
feedthrough insertion 246
loading 287
overview 186
restoring 287
running 266
saving 287
partitions
adding cut areas to rectilinear 206
changing status from hard block 266
master, defined 203
multiple instantiations 203
nonrectangular 205
pin assignments for 210
pin blockage 217
pins 208
rectilinear 240
creating 205
repeated 203
specifying 196
and timing budgets for 717
passivation layer
example 1049
path groups 857
peripheral I/O, and flip chip 122
physical connectivity, and power grid, report on 935
physical design size, and module constraint types 298
physical layout, verifying 944
physical libraries
creating 60
pin access, improving for NanoRoute router 590
pin alignment, illustrated 241
pin assignments

rectilinear 240
and rectilinear edges 206
pin balancing, and macro models 489
pin blockage 217
pin guides
location 213
pin instance delay specification, and CTS 485
pins
blackbox
assigning 208
partition
assigning 208
preventing creation of 197
when created in absence of partition pin guide objects 197
underneath power routes, checking for with NanoRoute router 591
visibility of 551
placement
blockages
assigning attributes to 436
creating 435
hard, description of 436
partial
behavior of 436
description of 436
soft
description of 436
types of 436
checking 453
clock buffers, reserving space for 447
files required for 435
multi-threading and 451
overview of 434
padding
adding 446
preparing for 434
preroute treatment during 436
saving data from 460
tuning 450
placing
end-cap cells 439
filler cells 456
gate-array style filler cells 457
hard blocks 434
hierarchical spare cells 443
JTAG cells 461
padding 441
scan cells 463
spare cells 440
spare modules 441
standard cells 450
tie-off cells 460
well-tap cells 437
planar dielectric
syntax 1047
port delay specification, and CTS 485
post-CTS optimization 856
postroute optimization 859
postroute optimization, in NanoRoute router 606
power analysis
and boundary voltage file 933
decoupling capacitor modeling 927
flow using Encounter and VoltageStorm 915
for MSV designs 391
instance average power report 931
instance power data file 932
and instance voltage file 932
modes
simulation-based 908
statistical 908
net average power report 931
overview 907
pad location file 912, 924
power graph report 929
power graph, viewing 927
power/rail summary report 929
preparatory procedure 908
rail analysis 908
results 909
running 911
setup procedure 909
simple estimate mode 913
and SPEF files 913
and VoltageStorm 915
power domain as a partition 198
power domains
adding power stripes to 356
adding stripes to 381
creating cuts 380
creating rings around 381
timing libraries 388
power graph report 929
power grids, and physical connectivity data report 935
power pad location file
fetching automatically 925
saving 925
power plan template

block rings 360
creating 359
design 360
IP blocks 359
power analysis data 362
stripes 360
power planning, overview 350
power routing, overview 351
power structures, creating, preparatory procedure 350
power/rail summary report 929
example 929
pre-CTS optimization 851
pre-defined feedthroughs 253
preferences
initialization files for 55
session, setting 55
prerouted nets, skipping routing on 625, 639
PrimeTime
format, and timing constraint file creation 717
timing graph, using for timing with NanoRoute router 617
process antenna violations
verifying 947
process antennas
ensuring antenna cell connectivity 644
methods NanoRoute router uses to repair violations from 644
NanoRoute router options for repairing violations from 646
repairing by changing routing layers 645
repairing by diode insertion 645
violations, repairing on nets with multiple pins 644
product licenses
order of checking out 45

R

R0 transformation 203
rail analysis 908
RC extraction. *See* extraction, native extraction, sign-off extraction
RC scaling factors
generating 762
methods for specifying 769
reclaiming area 857
rectilinear boundaries, identification of 240
rectilinear pin assignments 240
rectilinear power domains 380
rectilinear shape, defining 304
rectilinear shapes, and pin assignments 240
redundant vias, using 634
regions, module constraint types 298
related documents, list of 35
relative floorplanning 340
orientation
Encounter 340
overview 340
saving commands for 343
relative floorplans
loading 343
reshaping 341, 342, ?? to 342
restoring 343
repeated partitions. *See* partitions
reports
capacitance comparison 767
instance average power 931
NanoRoute router 593
net average power 931
physical connectivity data and power grids 935
power graph 929
power/rail summary 929
routing statistics 593
used by NanoRoute router 593
verification 938
wire length 593
reportYield
yield effects considered by 958
resistance
calculations of mixed signal router 663
restoring
OpenAccess designs 95
restoring designs
from pre-3.2 versions 98
ring pins
connecting 355
rings
around power domains 381
rotated vias, checking for with NanoRoute router 591
route congestion, resolving
by inserting density screens 582
by reorienting block pins 582
routers
choosing NanoRoute or WRoute 588

routes
removing while editing 548
reshaping 562
routing
See also NanoRoute router
choosing NanoRoute router or WRoute router 588
clock nets with NanoRoute router 618, 626
differential 158, 365
feedthroughs 259
routing blockage file, generating 197
routing bumps to I/O driver cells 140
routing feedthroughs, inserting 260
routing guide file, from CTS 499
routing layers, checking metal density of 943
routing resources, and channelless designs 246
routing strategy, NanoRoute router 603
row configurations, types supported 302
row spacing, standard 302
rule-based I/O assignment files, creating 75
run parallel 290
run-time environment, setting 39

S

Sbox. *See* switch box
scaling factors, RC
specifying, methods for 769
scan cells
placing 463
scan chains
reordering 463
flows 465
native approach 464, 465 to 469
scanDEF approach 464, 469 to 474
scan files
loading 474
saving 474
scan functionality, principles of 463
scan nets, reordering 463
scheme format, mapping LEF vias and layers to 648
search and repair
phases of, in NanoRoute router 605
seed, definition of 994
selecting wire shapes 552
session preferences, setting 55
set_dont_touch constraint, implications of for timing analysis 786
sheet resistance values, and ICT files 763
shielded net routing
coaxial 672
coaxial, description of 658
constraint for 670
description of 656
differential pairs in 659
shared shields in 658
shielded routing
with NanoRoute router 637, 638
shielding wires
cutting 556
short point defect, section in yield technology file 978
SI. *See* signal integrity
signal integrity
crosstalk effect on incremental delay 896
incremental mode 901
NanoRoute router, using to prevent problems 620
preventing problems using NanoRoute router 620
setting NanoRoute router options for 622
signal routers
choosing NanoRoute or WRoute 588
See also extraction, native extraction
sign-off extraction
described 770
requirements 771
simple estimate mode, in power analysis 913
simulation-based mode, in power analysis 908
slot. *See* routing feedthrough
soft guides, module constraint types 298
spare cells
distribution of 441
placing 440
placing when hierarchy is an issue 443
specifying during floorplanning session 440
spare modules
placing 441
special route
loading 352
saving 352

specification file, for CTS. *See* clock tree specification file
SPEF files
comparing files from runQX and extractDetailRC commands 766
and power analysis 913
SRoute
overview 351
Stamp models
defined 82
preparing 82
standalone NanoRoute router
using a timing graph with 616
standard cell density, calculating 301
standard cells
placing 128, 450
statistical mode, in power analysis 908
stripes
adding to power domains 381
stripes, adding to power domains 356
superthreading
definition of 599
features of 599
supply, defined, in Trial Route 576
switch box, definition of 589
SWITCHTERM 1106
syntax conventions 34

T

tapering wires, NanoRoute router 642
target utilization 298
target value, in Violation Browser 953
TDF files, importing and exporting 103
technology file 60
template, for power plans 359
threshold, as used by mixed signal router 669
tie-off cells
adding 460
removing 460
tile cell
importing design data 103
Timing
Debug Results 826
timing analysis
overview 780
preparatory procedure 782
and buffer problems 822
timing budget
analyzing 726
calculating 731
constraints 742
customizing 734
deriving 717
deriving preliminary budget 719
justifying 737
verifying 736
warning report 745
timing budgeting
and hierarchical designs 714
methods for, in hierarchical designs 714
timing budgets
constraint files for partitions, generating 717
for partitions 717
values for, analyzing 736
timing closure design flow, preparing data for 84
timing constraints
file, and set_dont_touch constraint 786
file 717
importing 82
preparing 82
Timing Debug
Global 826
timing engine, specifying for NanoRoute router 615
timing graph
commands to generate for NanoRoute router 616
generating with CTE 616
using with PrimeTime and NanoRoute router 617
using with standalone NanoRoute router 616
timing libraries
for power domains 388
preparing 81
timing models, for cells in CTS 489
timing optimization
added buffers, viewing 877
after routing 861
clock domains 853, 854, 856, 857
correcting signal integrity violations during 862
hold violation repair 861
incremental 854, 857, 858
incremental,path groups 854
incremental,useful skew 854

inputting a SPEF file 863
location of timing reports generated by 851
low-effort mode 852
naming conventions 877
optimizations performed during 848
path groups 853, 856
post-CTS 856
postroute 859
pre-CTS 851
reclaiming area 854, 857
requirements for 848
results of 849
and set_case_analysis timing constraint 875
setup and hold violation repair 861
specifying MMMC environment for 868
useful skew 852, 857
using footprintless flow in 875
using on-chip variation (OCV) analysis for 868
using virtual partitioning during 867
viewing added buffers 877
timing reports, commands for generating 821
timing-driven routing, NanoRoute router 632
tolerance, as used by mixed signal router 667
total density, calculating 301
tracks
generating for NanoRoute router 592
tracks, horizontal and vertical, summarized in Trial Route congestion distribution report 576
Trial Route
analyzing values in congestion distribution table 573
congestion distribution report 575
obstructed and overflowed gcells in 576
congestion distribution table
acceptable values in 572
values described 577
congestion markers, overflow values and 573
data for, deleted during partitioning 266
demand, defined 576
design analysis requirements 572
loading 572
overview 568
preparatory procedure 568
route congestion, resolving 581
saving 572
supply, defined 576
two layer designs 572
troubleshooting
NanoRoute router violations (table) 628
problems with NanoRoute router 650
TU (target utilization) value 298
typographic conventions 34

U

upper layer violations, NanoRoute router 630
useful skew 858
controlling optimization 865
described 864
post-CTS 857, 858, 865
pre-CTS 852, 854, 864
user-defined ring 352

V

verifying
AC limit 951
connectivity 941
geometry 944
maximum floating area 950
metal density 943
physical layout 944
process antenna violations 947
Verilog
checking for availability before importing 86
Verilog netlists
concatenating for a partitioned design 286
creating for entire design 286
creating from a DEF file 87
removing Assign nets from 96
unique for use in CTS, scan, and IPO features 86
vertical and horizontal tracks, summarized in Trial Route congestion distribution report 576
via fill, adding 710
via probability, section in yield technology file 979

via ratio, increasing multiple-cut to single-cut with NanoRoute router 634
via resistance values, and ICT files 763
via rules, as supported by verifyGeometry 946
via syntax 1050
vias
adding 560
changing 561
concurrent insertion of multiple-cut and routing 634
deleting 547
EM values, calculating for 935
example 1050
fat, definition of 634
giving unique names after streamOut or oasisOut 106
inserting multiple-cut with NanoRoute router 634
mapping Astro scheme format by NanoRoute router 648
minimizing the number of 636
moving selected 551
optimizing in selected nets with NanoRoute router 635
optimizing with NanoRoute router 634
postroute optimization of 634
rotated, checking for with NanoRoute router 591
TOPOFSTACKONLY, defining for NanoRoute 591
using multi-cut (redundant) 634
ViewICT
errors and warnings 1038
viewing violations 952
Violation Browser
actual value 953
features of 953
navigating overlapped objects in 953
target value 953
violation markers
Assura or Calibre 952
clearing from design 955
highlighting 952
imported from NanoRoute router to Encounter 626
incremental updates to 952
NanoRoute router placement of 626
overwriting 952
and verifying connectivity 941
viewing 952
violation report
database impact of 941
violations
clearing from design 955
deleting violated nets 633
evaluating in NanoRoute router 626
fixing minimum cut 355
fixing minimum spacing 355
in NanoRoute router, evaluation table 628
repair strategies for 633
types recognized by NanoRoute router 626
viewing 952
virtual partitioning,using for timing analysis or optimization 867
voltage
nominal 912
operating 912
voltage level shifters
and MSV 383
voltage source reference points 912
VoltageStorm 935
Display Setting file 934
power analysis with Encounter 915
running 915
running with Encounter menus and commands 915

W

well-tap cells
adding to MSV designs 438
controlling the distance between 438
placing 437
removing 439
wide wires, routing with NanoRoute router 642
wire editing
bindkeys 546
controlling visibility of cell blockages during 563
using CAR for 564
See also wires
wire groups, adding with wire editor 555
wire length report 593
wire tapering, with NanoRoute router 642
wire width
changing 557
repairing violations of 558

wires
adding 551
changing layers of 559
changing width of 557
correcting maximum width violations of 557
cutting 556
deleting 549
duplicating 558
editing with CAR 564
EM values for, calculating 935
extending automatically 554
merging 560
moving
selected 551
with arrow keys 550
with mouse 549
reshaping routes of 562
selecting 549
splitting 560
stretching 559
trimming 556
WRoute router, when to use 588

X

XML
comment syntax 973
reference for 973
statement syntax 972
46
-xmode 46

Y

yield analysis
effects considered 958
prerequisites for 961
results of 961
support for 45-degree shapes 958
yield costs
definition of 966
yield map
changing bin size of 964
changing colors in 965
changing object types in 965
displaying 963
yield report
probabilities in 966
sections in 966
yield technology file
cell probability section in 982
contents of 972
conventions used in 973
example of 984
header section in 974
open point defect section in 974
short point defect section in 978
using template for 972
via probability section in 979