

Fusion Compiler™ User Guide

Version T-2022.03-SP4, September 2022

SYNOPSYS®

Copyright and Proprietary Information Notice

© 2022 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <https://www.synopsys.com/company/legal/trademarks-brands.html>.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Contents

New in This Release	28
Related Products, Publications, and Trademarks	28
Conventions	28
Customer Support	29
Statement on Inclusivity and Diversity	30
1. Working With the Fusion Compiler Tool	31
Physical Synthesis Design Flow Overview	32
Formal Verification	33
Place and Route Design Flow Overview	34
Fusion Compiler Concepts	36
Power Intent Concepts	37
UPF Concepts	37
UPF Flows	38
Multiple-Patterning Concepts	39
Mask Constraints	42
User Interfaces	42
Starting the Command-Line Interface	43
Exiting the Fusion Compiler Tool	44
Entering fc_shell Commands	44
Interrupting or Terminating Command Processing	45
Getting Information About Commands	45
Displaying Command Help	46
Using Application Options	46
Using Variables	47
Viewing Man Pages	47
Using Tcl Scripts	48
Adding Changes to a Script With Checkpoints	49
Defining Checkpoints	51
Configuring Checkpoints	53
Defining Checkpoint Behaviors	53
Associating Checkpoints and Checkpoint Behaviors	55

Contents

Querying Checkpoints and Checkpoint Behaviors	59
Viewing Your Checkpoint History	60
Using Setup Files	61
Using the Command Log File	62
Enabling Multicore Processing	62
Configuring Multithreading	63
Configuring Distributed Processing	64
Reporting Multicore Configurations	65
Removing Multicore Configurations	65
Running Tasks in Parallel	66
Running Commands in Parallel on Your Local Host	67
Running Commands in the Background	67
Running Commands in Parallel	68
2. Preparing the Design	70
Defining the Search Path	71
Setting Up Libraries	71
Working With Design Libraries	72
Setting Up Reference Libraries	73
Library Configuration	74
Restricting Library Cell Usage	75
Restricting the Target Libraries Used	76
Specifying Library Subset Restrictions	78
Using Pin Access Checker Utility	79
Analyzing Libraries	81
Introduction to Redundancies, Outliers, Equivalences, and Gaps in Library Cell Families	82
Identifying Redundancies, Outliers, Equivalences, and Gaps in Library Cell Families	87
Comparing Libraries	91
Reading the Design	96
Reading RTL Files in SystemVerilog, Verilog, or VHDL Format	97
Reading Designs Using the -autoread Option	99
Reading Designs Using the VCS Command-line Options	99
Reading Verilog Gate-Level Netlist Files	100
Reading Mixed Gate-Level Netlists and RTL Files	100
Embedding Tcl Commands in RTL Code	101

Contents

Mitigating Design Mismatches	102
Importing the Floorplan Information	102
Reading DEF Files	102
Fixing Site Name Mismatches	104
Validating DEF Files	104
Physical Constraints Extracted From the DEF File	105
Using Automatic Floorplanning	108
Creating Constraints for Auto Floorplanning	109
Setting Up Multivoltage Designs	112
Applying the Multivoltage Power Intent	112
Loading and Applying UPF Information	113
Using Incomplete UPF Information	114
Specifying UPF Constraints for Physical-Only Cells	115
Saving UPF Information	115
Preparing the Power Network	116
Creating Logical Power and Ground Connections	116
Creating Floating Logical Supply Nets	118
Defining Voltage Areas	119
Merging Voltage Area Shapes	120
Resolving Overlapping Voltage Areas	121
Modifying the Stacking Order	123
Defining Guard Bands	123
Defining Gas Stations	125
Querying Voltage Areas	125
Modifying Voltage Areas	126
Controlling Physical-Feedthrough Nets in Voltage Areas	126
Removing Voltage Areas	129
Inserting Multivoltage Cells	130
Inserting Level Shifters	130
Inserting Isolation Cells	130
Associating Power Strategies With Existing Multivoltage Cells	131
Controlling the Placement of Multivoltage Cells	131
Enabling Improved Buffering for Multivoltage Nets	131
Analyzing Multivoltage Information	131
Specifying Timing Constraints and Settings	132
Specifying Logical Design Rule Constraints	133
Specifying Clock Gating Constraints	134
Specifying Physical Constraints for Placement and Legalization	136
Defining Keepout Margins	136
Defining an Outer Keepout Margin	137

Contents

Defining an Inner Keepout Margin	138
Defining Area-Based Placement Blockages	138
Defining a Hard Placement Blockage	139
Defining a Hard Macro Placement Blockage	140
Defining a Soft Placement Blockage	140
Defining a Partial Placement Blockage	141
Define a Blockage of a Predefined Category	141
Defining Blockages That Exclude Registers	142
Defining Blockages That Exclude Relative Placement Groups	143
Defining Blockages That Allow Relative Placement Cells Only	143
Defining Blockages That Allow Buffers Only	143
Querying Placement Blockages	144
Removing Placement Blockages	144
Defining Placement Bounds	144
Defining Move Bounds	145
Defining Group Bounds	147
Querying Placement Bounds	147
Removing Placement Bounds	148
Defining Placement Attraction	148
Specifying Locations for Unmapped Cells	149
Defining Cell Spacing Constraints for Legalization	150
Reporting Cell Spacing Constraints	152
Removing Cell Spacing Constraints	152
Specifying Placement Settings	152
Performing Placement With Inaccurate Constraints at Early Stages	153
Generating Automatic Group Bounds for Clock Gating Cells	153
Controlling the Placement Density	154
Controlling Congestion-Driven Restructuring During Placement	155
Reducing Congestion	156
Considering Wide Cell Density During Placement	156
Considering the Effects of Cell Pins During Placement	157
Considering the Congestion Effects Due to the Nondefault Routing Rules of Clock Nets	157
Considering the Effects of Clock Gating Cells of Sequential Arrays During Placement	158
Considering Legalization Effects During Placement	158
Considering DFT Connections During Placement	158
Considering the Dynamic Power QoR During Placement	158
Performing IR-Drop-Aware Placement	159
Controlling IR-Drop-Aware Placement	160
Spreading Repeater Cells During Placement	162

Contents

Specifying Legalization Settings	162
Minimizing Large Displacements During Legalization	163
Optimizing Pin Access During Legalization	163
Enabling Advanced PG Net Checks	163
Enabling Advanced Legalization Algorithms	163
Setting Up for Variant-Aware Legalization	165
Defining Equivalent Cell Groups	166
Enabling Variant-Aware Legalization	167
Checking if Library Cells Are Legally Placeable	167
Controlling the Optimization of Cells, Nets, Pins, and Ports	168
Resolving Multiple References With the Uniquify Process	169
Preserving Cells and Nets During Optimization	169
Restricting Optimization to Cell Sizing Only	171
Preserving Networks During Optimization	171
Marking the Clock Networks	172
Disabling Design Rule Checking (DRC)	172
Preserving Pin Names During Sizing	174
Preserving Ports of Existing Hierarchies	175
Isolating Input and Output Ports	175
Fixing Multiple-Port Nets	176
Controlling the Addition of New Cells to Modules, Hierarchical Cells, and Voltage Areas	178
Specifying a Cell Name Prefix for Optimization	179
Specifying Settings for Preroute Optimization	179
Specifying Parasitic Estimation Settings for the Preroute Optimization	180
Enabling Global-Route-Layer-Based (GRLB) Preroute Optimization	180
Enabling Route-Driven Estimation (RDE) for Preroute Optimization	180
Specifying Automatic Via Ladder Insertion Settings for Preroute Optimization	181
Specifying Via Ladder Candidates for Library Pins	181
Assigning Nondefault Routing Rules to Critical Nets	183
Enabling Area Recovery in Regions of High Utilization	184
Enabling Advanced Logic Restructuring	184
Setting Up for Power-Related Features	185
Annotating the Switching Activity	185
Using RTL Switching Activity With a Name-Mapping File	186
Scaling the Switching Activity	187
Specifying Switching Probability for Supply Nets	188
Enabling Leakage Power Optimization for the compile_fusion Command	189

Contents

Reporting Leakage Power	190
Enabling Power Optimization for the <code>clock_opt</code> Command	191
Performing Conventional Leakage-Power Optimization	191
Performing Dynamic-Power Optimization	192
Performing Total-Power Optimization	192
Improving Yield By Limiting the Percentage of Low-Threshold-Voltage (LVT) Cells	193
Updating Activity for Improved Power Optimization	194
Enabling the Power Integrity Features	195
Setting Up for Dynamic Power Shaping	197
Setting Up for Voltage-Drop-Aware Placement	198
Manually Enabling Dynamic Power Shaping and Voltage-Drop-Aware Placement	198
Specifying the Routing Resources	199
Specifying the Global Layer Constraints	199
Reporting Global Layer Constraints	200
Removing Global Layer Constraints	201
Specifying Net-Specific Layer Constraints	201
Removing Net-Specific Routing Layer Constraints	202
Specifying Clock-Tree Layer Constraints	202
Setting the Preferred Routing Direction for Layers	205
Handling Design Data Using the Early Data Check Manager	205
Applying Mega-Switch Command Settings	207
Applying Required Settings for Advanced Technology Nodes	207
Applying Required Settings for High Performance Cores	208
Applying Required Settings for Improving Specific QoR Metrics	208
<hr/>	
3. Physical Synthesis	210
Performing Unified Physical Synthesis	210
Using the <code>compile_fusion</code> Command	211
Performing Prechecks	213
Generating Verification Checkpoints During Compilation	215
Controlling Mapping and Optimization	215
Ungrouping or Preserving Hierarchies During Optimization	216
Controlling Boundary Optimization	217
Controlling Datapath Optimization	220
Datapath Extraction	220
Datapath Implementation	222
Optimizing Datapaths	223

Contents

Controlling Mux Mapping	224
Controlling Sequential Mapping	225
Mapping Sequential Elements Exactly as Described in RTL	226
Mapping of Scan Cells	227
Mapping of Synchronous Reset or Preset Registers	227
Specifying Synchronous Input Types for Library Cell Pins	229
Controlling Sequential Output Inversion	229
Preventing Connections to Register QN Pins	230
Mapping of Synchronous Enable Logic to a MUX	231
Identification of Shift Registers	231
Controlling Register Replication	232
Controlling Register Merging	234
Selectively Removing or Preserving Constant and Unloaded Registers	235
Reporting Cross-Probing Information for Optimized Registers	237
Controlling High-Fanout-Net Synthesis	238
Performing Multibit Register Optimization	238
Performing Integrated Multibit-Register Optimization	239
Creating a Multibit Register Bank From Specific Single-Bit Registers	241
Specifying Naming Styles for Multibit Registers	242
Reporting Multibit Registers	244
Identifying Why Multibit Banking Is Not Performed	244
Improving the Banking Ratio	246
Viewing Multibit Components in the GUI	246
Running Concurrent Clock and Data Optimization	247
Reducing Dynamic Voltage Drop	250
Specifying Optimization Targets at the Preroute Stage	250
Transferring to Back End	251
Performing Test Insertion and Scan Synthesis	251
Test Insertion Before Compile	252
Scan Synthesis During Compile	253
Minimizing Glitches on Asynchronous Set and Reset Lines of Scan Registers	254
Specifying Settings for Performance, Power, and Area Improvement	254
Enabling High-Effort Timing Mode	254
Enabling Enhanced Delay Optimization to Improve Total Negative Slack	255
Enabling High-Effort Area Mode	255
Enabling the Embedded Area Optimization Flow	255
Performing Design Analysis	255
Reporting Commands and Examples	256

Contents

Measuring Quality of Results	261
Comparing QoR Data	262
Setting Your Baseline Run	265
Changing the QoR Display Style	266
Sorting and Filtering the Data	267
Exploring the Detailed Comparison Data	272
Reporting Logic Levels in Batch Mode	276
Querying Specific Message IDs	278
4. Clock Gating	279
Introduction to Clock Gating	280
Naming Convention for Tool-Inserted Clock Gates	282
Clock Gating in the Compile Log File	282
Clock-Gate Levels and Stages	284
Clock-Gating Prerequisite Conditions	286
Clock-Gating Enable Condition	287
Clock-Gating Setup Condition	287
Setting Up Clock Gating	287
Setting the Clock-Gating Style	288
Setting Clock-Gating Options	289
Enabling or Disabling Clock Gating on Design Objects	289
Clock-Gating Enable Source Selection	290
Clock Gating Flows	291
Inserting Clock Gates in Multivoltage Designs	291
Inserting Clock Gates in an RTL Design	291
Replacing Clock Gates	292
Controlling Clock-Gate Latencies	293
Integrated Clock-Gate Latency Estimation	293
User-Specified Clock-Gate Latency	295
Controlling the Number of Clock-Gate Levels	298
Clock-Gate Multilevel Expansion	299
Clock-Gate Collapsing	300
Controlling Tool-Inserted Clock-Gate Levels	300
Activity-Based Optimization	301
Merging Clock Gates	302
Setting Clock Gating Transformations	303

Contents

Setting Routing Rules for Clock Gates	304
Clock Gating and Multibit Registers	305
Placement-Aware Clock Gating	305
Fanin-Based Sequential Clock Gating	306
Setting Sequential Clock Gating Options	308
Reporting Sequential Clock Gates	309
Reporting Clock-Gating Results	310
Self-Gating Optimization	312
Self-Gating Flow	314
Library Requirements for Self-Gating	315
Setting Up for Self-Gating	315
Setting Self-Gating Options	315
Controlling the Selection of Self-Gating Objects	316
Reporting and Querying Self-Gating Cells	317
<hr/>	
5. Clock Tree Synthesis	319
Prerequisites for Clock Tree Synthesis	319
Defining the Clock Trees	320
Deriving the Clock Trees	321
Identifying the Clock Roots	321
Identifying the Clock Endpoints	323
Defining Clock Tree Exceptions	325
Defining Sink Pins	325
Defining Insertion Delay Requirements	326
Defining Ignore Pins	327
Ensuring Clock Tree Exceptions are Valid	328
Restricting Optimization on the Clock Network	329
Setting Don't Touch Settings	330
Setting Size-Only Settings	330
Copying Clock Tree Exceptions Across Modes	331
Deriving Clock Tree Exceptions From Ideal Clock Latencies	331
Handling Endpoints With Balancing Conflicts	332
Verifying the Clock Trees	334
Setting Clock Tree Design Rule Constraints	336
Specifying the Clock Tree Synthesis Settings	337
Specifying the Clock Tree References	337
Deriving Clock Tree References for Preexisting Gates	338

Contents

Restricting the Target Libraries Used	339
Setting Skew and Latency Targets	340
Enabling Local Skew Optimization	340
Specifying the Primary Corner for Clock Tree Synthesis	341
Preventing Specific Clocks From Being Synthesized	341
Preserving Preexisting Clock Trees	341
Enabling Clock Tree Power Reduction Techniques	341
Reducing Electromigration	342
Handling Inaccurate Constraints During Clock Tree Synthesis	342
Defining Clock Cell Spacing Rules	343
Creating Skew Groups	344
Defining a Name Prefix for Clock Cells	345
Using the Global Router During Initial Clock Tree Synthesis	345
Specifying Constraints for Clock Nets	345
Reducing Signal Integrity Effects on Clock Nets	346
Specifying Settings for Clock Latency Adjustments	346
Reporting the Clock Tree Settings	347
Implementing Clock Trees and Performing Post-CTS Optimization	347
Performing Standalone Clock Trees Synthesis	348
Synthesizing, Optimizing, and Routing Clock Trees With the <code>clock_opt</code> Command	349
Considering Voltage Drop Information During Clock Tree Synthesis	349
Using Nondefault Routing Rules for Critical Nets During Optimization	350
Performing Concurrent Clock and Data Optimization During the <code>clock_opt</code> Command	350
Controlling Multibit Optimization Performed During the <code>clock_opt</code> Command	351
Performing Power or Area Recovery on the Clock Network	352
Performing IR-Drop-Aware Placement During the <code>clock_opt</code> Command	352
Controlling Concurrent Clock and Data Optimization	353
Limiting the Latency Adjustment Values	354
Excluding Boundary Paths	354
Excluding Specific Path Groups	355
Excluding Specific Scenarios	355
Excluding Specific Sinks	355
Controlling Timing Optimization Effort	356
Controlling Hold Time Optimization Effort	356
Controlling the Adjustment of I/O Clock Latencies	356
Performing Dynamic-Voltage-Drop-Driven Concurrent Clock and Data Optimization During the <code>route_opt</code> Command	357
Specifying Optimization Targets at the Preroute Stage	357

Contents

Specifying Optimization Targets at the Postroute Stage	358
Enabling Buffer Removal at the Postroute Stage	360
Reporting Concurrent Clock and Data Timing	360
Enabling Automatic Zero Balance Point on Macros	362
Skewing Latch and Discrete Clock Gates	362
Splitting Clock Cells	362
Balancing Skew Between Different Clock Trees	363
Defining the Interclock Delay Balancing Constraints	364
Generating Interclock Delay Balancing Constraints Automatically	365
Running Interclock Delay Balancing	365
Performing Global-Route-Based Optimization Using Machine Learning Data .	366
Routing Clock Trees	368
Inserting Via Ladders During Clock Tree Synthesis, Optimization, and Clock Routing	368
Marking Clocks as Propagated After Clock Tree Synthesis	369
Performing Postroute Clock Tree Optimization	369
Performing Voltage Optimization	370
Marking Clock Trees as Synthesized	371
Removing Clock Trees	372
Implementing Multisource Clock Trees	373
Introduction to Multisource Clock Trees Structures	374
Implementing a Regular Multisource Clock Tree	376
Implementing a Regular Multisource Clock Tree Using Integrated Tap Assignment	377
Implementing a Regular Multisource Clock Tree With an H-Tree-Only Global Clock Tree Structure	378
Implementing a Structural Multisource Clock Tree	380
Implementing a Structural Multisource Clock Tree Using Integrated Subtree Synthesis	381
Inserting Clock Drivers	384
Inserting Clock Drivers for Designs With Multiple Levels of Physical Hierarchy	387
Synthesizing the Global Clock Trees	387
Inserting Clock Drivers for Designs With Multiple Levels of Physical Hierarchy	389
Creating Clock Straps	389
Routing to Clock Straps	393
Analyzing the Clock Mesh	398
Performing Automated Tap Insertion and H-Tree Synthesis	400
Specifying Tap Assignment Options and Settings	402
Building the Local Clock Subtree Structures	403

Contents

Analyzing the Clock Tree Results	407
Generating Clock Tree QoR Reports	407
Reporting Clock Tree Power	409
Creating Collections of Clock Network Pins	410
Analyzing Clock Timing	411
Analyzing Clock Trees in the GUI	411
<hr/>	
6. Routing and Postroute Optimization	412
Introduction to Zroute	413
Basic Zroute Flow	415
Prerequisites for Routing	416
Defining Vias	417
Reading Via Definitions from a LEF File	418
Creating a Via Definition	418
Defining Simple Vias	418
Defining Custom Vias	419
Inserting Via Ladders	420
Defining Via Ladder Rules	421
Generating Via Ladder Rules for Electromigration Via Ladders	423
Generating Via Ladder Rules for Performance Via Ladders	425
Via Ladder Rule Files	427
Via Ladder Association File	427
Constraint-Based Via Ladder Insertion	428
Defining Via Ladder Constraints	429
Defining Global Via Ladder Constraints	429
Defining Instance-Specific Via Ladder Constraints	430
Inserting Via Ladders	431
Protecting Via Ladders	433
Verifying Via Ladders	434
Updating Via Ladders	435
Manual Via Ladder Insertion	436
Querying Via Ladders	437
Removing Via Ladders	438
Checking Routability	438
Routing Constraints	444
Defining Routing Blockages	446
Reserving Space for Top-Level Routing	447

Contents

Querying Routing Blockages	448
Removing Routing Blockages	448
Defining Routing Guides	448
Using Routing Guides to Control the Routing Direction	450
Using Routing Guides to Limit Edges in the Nonpreferred Direction	451
Using Routing Guides to Control the Routing Density	451
Using Routing Guides to Prioritize Routing Regions	452
Using Routing Guides to Encourage River Routing	453
Querying Routing Guides	453
Removing Routing Guides	453
Deriving Routing Guides	453
Deriving Pin Access Routing Guides	454
Deriving Metal Cut Routing Guides	455
Controlling Routing Around the Block Boundary	455
Inserting Metal Shapes in the Preferred Direction	456
Inserting Routing Guides Along the Nonpreferred-Direction Edges	460
Inserting Routing Blockages Along the Boundary Edges	462
Removing Perimeter Constraint Objects	464
Routing Nets Within a Specific Region	464
Defining Routing Corridors	465
Assigning Nets to a Routing Corridor	466
Verifying Routing Corridors	467
Modifying Routing Corridors	467
Reporting Routing Corridors	468
Removing Routing Corridors	468
Using Nondefault Routing Rules	468
Defining Nondefault Routing Rules	469
Assigning Nondefault Routing Rules to Nets	480
Controlling Off-Grid Routing	484
Preventing Off-Grid Routing	484
Discouraging Off-Grid Routing for Vias	486
Routing Must-Join Pins	486
Controlling Pin Connections	489
Controlling Pin Tapering	492
Specifying the Tapering Method	492
Controlling the Tapering Width	493
Controlling Via Ladder Connections	494
Setting the Rerouting Mode	494
Routing Application Options	495
Routing Clock Nets	495
Routing Critical Nets	496

Contents

Routing Secondary Power and Ground Pins	497
Verifying the Secondary Power and Ground Pin Attributes	498
Setting the Routing Constraints	498
Routing the Secondary Power and Ground Pins	500
Routing Signal Nets	501
Global Routing	502
Global Routing During Design Planning	505
Timing-Driven Global Routing	505
Crosstalk-Driven Global Routing	507
Incremental Global Routing	507
Track Assignment	507
Detail Routing	509
Routing Signal Nets by Using Automatic Routing	513
Shielding Nets	515
Defining the Shielding Rules	516
Performing Preroute Shielding	517
Soft Shielding Rules During Signal Routing	520
Performing Postroute Shielding	521
Shielding Example	521
Performing Incremental Shielding	522
Reporting Shielding Information	522
Querying Shield Shapes	522
Reporting Shielding Statistics	523
Performing Shielding Checks	524
Performing Postroute Optimization	524
Performing Postroute Logic Optimization	525
Performing Postroute Optimization Using the <code>hyper_route_opt</code> Command	527
Fixing DRC Violations Caused by Pin Access Issues	528
Analyzing and Fixing Signal Electromigration Violations	529
Loading the Signal Electromigration Constraints	531
Analyzing Signal Electromigration	531
Fixing Signal Electromigration Violations	533
Performing ECO Routing	534
Routing Nets in the GUI	536
Modifying Routed Nets	537
Cleaning Up Routed Nets	539
Analyzing the Routing Results	540

Contents

Generating a Congestion Report	540
Generating a Congestion Map	542
Performing Design Rule Checking Using Zroute	545
Performing Signoff Design Rule Checking	548
Performing Design Rule Checking in an External Tool	549
Performing Layout-Versus-Schematic Checking	549
Reporting the Routing Results	552
Reporting the Wire Length	552
Using the DRC Query Commands	553
Saving Route Information	554
Deriving Mask Colors	554
Inserting and Removing Cut Metal Shapes	555
7. Chip Finishing and Design for Manufacturing	557
Inserting Tap Cells	557
Using the create_tap_cells Command	558
Using the create_cell_array Command	562
Inserting Exterior Tap Walls	564
Inserting Interior Tap Walls	565
Inserting Tap Meshes	567
Inserting Dense Tap Arrays	568
Performing Boundary Cell Insertion	569
Specifying the Boundary Cell Insertion Requirements	569
Specifying the Library Cells for Boundary Cell Insertion	570
Specifying Boundary Cell Placement Rules	572
Specifying the Naming Convention for Boundary Cells	573
Creating Routing Guides During Boundary Cell Insertion	573
Creating Placement Blockages During Boundary Cell Insertion	574
Reporting the Boundary Cell Insertion Requirements	574
Removing Boundary Cell Insertion Requirements	575
Inserting Boundary Cells	575
Verifying the Boundary Cell Placement	575
Finding and Fixing Antenna Violations	576
Defining Antenna Rules	577
Calculating the Maximum Antenna Ratio	577
Calculating the Antenna Ratio for a Pin	588
Specifying Antenna Properties	591
Analyzing and Fixing Antenna Violations	593

Contents

Inserting Diodes During Detail Routing	594
Inserting Diodes After Detail Routing	595
Inserting Redundant Vias	596
Inserting Redundant Vias on Clock Nets	597
Inserting Redundant Vias on Signal Nets	598
Viewing the Default Via Mapping Table	599
Defining a Customized Via Mapping Table	600
Postroute Redundant Via Insertion	601
Concurrent Soft-Rule-Based Redundant Via Insertion	602
Near 100 Percent Redundant Via Insertion	604
Preserving Timing During Redundant Via Insertion	605
Reporting Redundant Via Rates	605
Optimizing Wire Length and Via Count	607
Reducing Critical Areas	607
Performing Wire Spreading	608
Performing Wire Widening	609
Inserting Metal-Insulator-Metal Capacitors	610
Inserting Filler Cells	611
Standard Filler Cell Insertion	612
Controlling Standard Filler Cell Insertion	614
Checking for Filler Cell DRC Violations	617
Fixing Remaining Mask Design Rule Violations	618
Abutting Standard Cells With Specific Filler Cells	630
Threshold-Voltage-Based Filler Cell Insertion	632
Controlling Threshold-Voltage-Based Filler Cell Insertion	633
Removing the Threshold-Voltage Filler Cell Information	634
Removing Filler Cells	634
Inserting Metal Fill	634
<hr/>	
8. IC Validator In-Design	635
Preparing to Run IC Validator In-Design Commands	636
Setting Up the IC Validator Environment	636
Enabling IC Validator Multicore Processing	636
Running IC Validator on Specific Hosts	637
Running IC Validator Using a Job Scheduler	638
Running IC Validator Using Hybrid Multicore Processing	639
Defining the Layer Mapping for IC Validator In-Design Commands	639
Performing Signoff Design Rule Checking	640

Contents

Running the signoff_check_drc Command	640
Setting Options for Signoff Design Rule Checking	642
Reading Blocks for Signoff Design Rule Checking	644
Signoff Design Rule Checking	645
Signoff DRC Results Files	647
Viewing the Violations in an ICV Heat Map	648
Configuring an ICV Heat Map	651
Highlighting Violations From the Error Browser Onto an ICV Heat Map	657
Automatically Fixing Signoff DRC Violations	659
Creating an Autofix Configuration File	659
Setting Options for Signoff DRC Fixing	660
Running the signoff_fix_drc Command	662
Automatically Fixing Double-Patterning Odd-Cycle Violations	666
Summary Report for Automatic Design Rule Fixing	666
Checking Signoff Design Rules Interactively in the GUI	668
Displaying Objects for Design Rule Checking	669
DRC Toolbar	670
Setting Options for Interactive Design Rule Checking	672
Improving Instance Voltage Drop by Augmenting the Power Grid	674
Standard Power Grid Augmentation	675
Setting Options for Power Grid Augmentation	677
Timing-Driven Power Grid Augmentation	678
Guided Power Grid Augmentation	680
Removing PG Augmentation Shapes	681
Inserting Metal Fill With IC Validator In-Design	682
Setting Options for Signoff Metal Fill Insertion	683
Performing Metal Fill Insertion	686
Pattern-Based Metal Fill Insertion	686
Track-Based Metal Fill Insertion	691
Using an IC Validator Parameter File	694
Typical Critical Dimension Metal Fill Insertion	699
Timing-Driven Metal Fill Insertion	699
Incremental Metal Fill Insertion	704
Signoff Metal Fill Result Files	706
Querying Metal Fill	707
Viewing Metal Fill in the GUI	708
Reporting the Metal Density	709
Viewing Density Heat Maps in the GUI	712
Removing Metal Fill	714
Removing Metal Fill With the IC Validator Tool	715
Modifying Metal Fill	716

Contents

Performing Real Metal Fill Extraction	716
Automatically Fixing Isolated Vias	717
Setting Options for Fixing Isolated Vias	717
Running the signoff_fix_isolated_via Command	718
Checking for Isolated Vias	718
Checking and Fixing Isolated Vias	719
9. Routing Using Custom Router	721
Using Custom Router in the Fusion Compiler Tool	722
Before Using Custom Router	723
Defining Routing Constraints	724
Defining the Bus Routing Style	725
Creating Differential Groups	727
Inserting Shields on the Nets	728
Defining the Net Priority	730
Defining Minimum Wire Lengths	731
Defining Matching Wire Lengths	733
Managing Constraint Groups	738
Using Custom Routing Application Options	739
Bus Routing Options	741
Corner Type	741
Intra-shield Placement	742
Pin-Trunk Offset	743
Trunk Splitting	743
Tapoffs	744
Track Adherence Options	744
Differential-Pair Options	745
Shielding Options	745
Single-Loop Matching	746
Routing With the Custom Router	747
Shielding the Nets	748
Checking the Routing Results	748
Using a Hybrid Routing Flow	748
Using a DDR Routing Flow	750
10. Physical Datapath With Relative Placement	755

Contents

Introduction to Physical Datapath With Relative Placement	755
Benefits of Relative Placement	756
Relative Placement Flow	757
Creating Relative Placement Groups	757
Adding Objects to a Group	759
Adding Leaf Cells	759
Specifying Orientations for Leaf Cells	760
Adding Hard Macro Cells	760
Adding Relative Placement Groups	760
Creating Hierarchical Relative Placement Groups	761
Using Hierarchical Relative Placement for Straddling	762
Using Hierarchical Relative Placement for Compression	764
Adding Blockages	765
Adding Cells Within a Predefined Relative Placement Area	766
Specifying Options for Relative Placement Groups	766
Anchoring Relative Placement Groups	767
Aligning Leaf Cells Within a Column	770
Aligning by the Left Edges	770
Aligning by the Right Edges	770
Aligning by Pin Location	771
Overriding the Alignment When Adding Objects	773
Controlling the Tiling Within Relative Placement Groups	774
Applying Compression to Groups With Straddling Leaf Cells	775
Specifying the Orientation of Relative Placement Groups	776
Specifying a Keepout Margin	778
Handling Fixed Cells During Relative Placement	778
Allowing Nonrelative Placement Cells	779
Controlling the Optimization of Relative Placement Cells	779
Controlling Movement When Legalizing Relative Placement Groups	780
Handling Tap Columns and Relative Placement Group Overlaps	780
Changing the Structures of Relative Placement Groups	781
Generating Relative Placement Groups for Clock Sinks	782
Performing Placement and Legalization of Relative Placement Groups	783
Relative Placement in a Design Containing Obstructions	783
Legalizing Relative Placement Groups in a Placed Design	784
Creating New Relative Placement Groups in a Placed Design	785
Analyzing Relative Placement Groups	786
Checking Relative Placement Groups Before Placement	786

Contents

Analyzing the Placeability of a Relative Placement Group	787
Reporting Relative Placement Constraint Violations	787
Querying Relative Placement Groups	788
Analyzing Relative Placement in the GUI	788
Saving Relative Placement Information	789
Summary of Relative Placement Commands	789
<hr/>	
11. Hierarchical Implementation	791
Performing Hierarchical Synthesis Using Abstracts	792
Partitioning and Planning the Full Chip Design	792
Hierarchical Synthesis Flow When Floorplans are not Available	793
Hierarchical Synthesis Flow When Floorplans are Available	795
Synthesizing a Subblock	796
Synthesizing the Top-Level	797
Performing Hierarchical Synthesis Using ETMs	798
Running the Hierarchical Synthesis Feasibility Analysis Flow Using ETMs ..	799
Running the Hierarchical Synthesis Flow Using ETMs	801
Example Script of Hierarchical Synthesis Using ETMs	803
Performing Prechecks for Hierarchical Synthesis Flows	803
Providing Block-Level Test Models	803
Specifying Block-Level Power Intent	804
Overview of Abstract Views	805
Creating Abstract Views	807
Creating Abstracts With Power Information	808
Creating Abstracts for Signal Electromigration Analysis	809
Handling Multiple Levels of Physical Hierarchy	809
Reporting Abstract Inclusion Reasons	810
Making Changes to a Block After Creating an Abstract	811
Creating a Frame View	812
Linking to Abstract Views at the Top-Level	812
Linking to Subblocks With Multiple Labels	814
Specifying the Editability of Blocks From the Top-Level	814
Preparing for Top-Level Closure With Abstracts	815
Checking Designs With Abstracts for Top-Level-Closure Issues	816
Handling Design Data Using the Early Data Check Manager	819

Contents

Prerequisites for Handling Early Design Data	820
Early Data Checks, Policies, and Strategies	821
Setting the Policy for Early Data Checks	824
Reporting Early Data Check Records	825
Generating a Report of Early Data Check Records	827
Performing Top-Level Closure With Abstract Views	828
Creating ETMs and ETM Cell Libraries	829
Creating ETMs and ETM Cell Libraries in the Fusion Compiler Tool	830
Creating ETMs in the PrimeTime Tool	830
Creating ETM Cell Libraries in the Library Manager Tool	832
Linking to ETMs at the Top Level	833
Performing Top-Level Closure With ETMs	834
<hr/>	
12. RedHawk and RedHawk-SC Fusion	835
Running Rail Analysis Using RedHawk-SC Fusion	836
An Overview for RedHawk Fusion and RedHawk-SC Fusion	838
RedHawk Fusion and RedHawk-SC Fusion Data Flow	840
RedHawk/RedHawk-SC Fusion Analysis Flow	841
Running RedHawk Fusion Commands in the Background	842
Setting Up the Executables	843
Specifying RedHawk and RedHawk-SC Working Directories	843
Preparing Design and Input Data for Rail Analysis	845
Generating Rail Analysis Script Files	848
Supporting RedHawk-SC Customized Python Script Files	849
Specifying Ideal Voltage Sources as Taps	854
Validating the Taps and Finding Invalid Taps	856
Finding Invalid Taps	857
Finding Substitute Locations for Invalid Taps	858
Removing Taps	859
Missing Via and Unconnected Pin Checking	859
Setting Checking Options	860
Checking Missing Vias and Unconnected Pins	861
Viewing Error Data	862
Fixing Missing Vias	863
Running Rail Analysis with Multiple Rail Scenarios	864
Specifying a Design Scenario for Rail Analysis	865

Contents

Creating and Specifying Rail Scenarios for Rail Analysis	865
Performing Voltage Drop Analysis	868
Viewing Voltage Drop Analysis Results	870
Performing PG Electromigration Analysis	871
Viewing PG Electromigration Analysis Results	872
Displaying the PG Electromigration Map	872
Checking PG Electromigration Violations	873
Performing Minimum Path Resistance Analysis	876
Viewing Minimum Path Resistance Analysis Results	877
Tracing Minimum Path Resistance Using the Mouse Tool	879
Performing Effective Resistance Analysis	882
Performing Distributed RedHawk Fusion Rail Analysis	883
Working With Macro Models	886
Generating Macro Models	886
Creating Block Contexts	888
Performing Signoff Analysis	891
Writing Analysis and Checking Reports	892
Displaying Block-Level Rail Results	895
Generating Instance-Based Analysis Reports	896
Generating Geometry-Based Analysis Reports	897
Displaying Maps in the GUI	898
Displaying ECO Shapes in the GUI	904
Voltage Hotspot Analysis	905
Generating Hotspots	906
Reporting Hotspots	907
Removing Hotspots	909
Voltage Hotspot Analysis Examples	910
Querying Attributes	912
<hr/>	
13. ECO Flow	914
Generic ECO Flow for Timing or Functional Changes	915
Freeze Silicon ECO Flow	916
Signoff ECO Flow	916
Incremental Signoff ECO Flow	918
ECO Fusion Flow	920

Contents

ECO Fusion Power Integrity Flow	921
Manually Instantiating Spare Cells	922
Automatically Adding Spare Cells	924
Adding Programmable Spare Cells	926
Making ECO Changes Using the <code>eco_netlist</code> Command	927
Making ECO Changes Using Netlist Editing Commands	928
Using ECO Scripts for Netlist Editing	928
Resizing Cells	928
Reverting Changes Made During Resizing	929
Adding Buffers on Nets	929
Adding Buffers on Routed Nets	930
Specifying the Net Names, Buffers Types, and Their Locations	930
Adding Buffers in a Specified Configuration	931
Adding Buffers at Specified Locations	931
Adding Buffers at a Specified Interval	932
Adding Buffers at an Interval That is a Ratio of the Net Length	932
Adding Buffers on a Bus in a Specified Pattern	933
Controlling How Buffers are Added	935
Specifying Settings for Multivoltage Designs	936
Specifying Settings for the Freeze Silicon ECO Flow	936
Optimizing the Fanout of a Net	937
Reporting Available Sites for Placing ECO Cells	938
Identifying and Reverting Nonoptimal ECO Changes	938
Placing ECO Cells	938
Controlling Placement When Using the <code>place_eco_cells</code> Command	939
Controlling Legalization When Using the <code>place_eco_cells</code> Command	940
Placing ECO Cells With Minimal Physical Impact (MPI)	943
Placing and Mapping ECO Cells to Spare Cells	943
Specifying Mapping Rules for Programmable Spare Cells	944
Mapping ECO Cells to Specific Spare Cells	944
Mapping ECO Cells to Logically Equivalent Spare Cells	944
Updating Supply Nets for ECO Cells	945
Recording the Changes Made to a Layout	945
Performing Prerequisite Check for Group Repeater Insertion and Placement	946
Adding a Group of Repeaters	947
Defining a group of repeaters	948

Contents

Grouping a list of repeaters	948
Setting Constraints for a Group of Repeaters	949
Reporting the Constraints Assigned to a Group of a Repeaters	949
Removing Constraints for a Group of Repeaters	949
Placing Group Repeaters Before Routing	949
Performing On Route Placement of Repeaters	950
Placing Group Repeaters For Multibit Registers	950
Specifying Locations for Repeater Groups	951
Allowing Repeater Groups Over Macros	951
Specifying Cut Space and Cut Distance for Repeater Groups	952
Specifying Horizontal and Vertical Spacing for Repeater Groups	952
Specifying Library Cells as Repeaters	952
Avoiding Overlapping Repeaters With Existing Tap Cells	952
Avoiding Crosstalk During Group Repeater Insertion	952
Previewing Repeater Groups	953
Unplacing the Repeaters	954
Removing Repeater Groups	954
Querying Group Repeater	954
Performing Auto Grouping Flow	955
Performing Manual Grouping Flow	955
Cell Input Mode	956
Swapping Variant Cell	956
Setting Constraints of Variant Cell	957
Setting Application Option	957
Grouping Variant Cell	958
Running and Placing Group Repeaters	964
Troubleshooting	964
Fixing Multivoltage Violations	971
Fixing Buffers	972
Fixing Diodes	973

About This User Guide

The Synopsys Fusion Compiler tool provides a complete netlist-to-GDSII or netlist-to-OASIS® design solution, which combines proprietary design planning, physical synthesis, clock tree synthesis, and routing for logical and physical design implementation throughout the design flow.

This guide describes the Fusion Compiler implementation and integration flow. For more information about the Fusion Compiler tool, see the following companion volumes:

- *Library Manager User Guide*
- *Fusion Compiler Design Planning User Guide*
- *Fusion Compiler Data Model User Guide*
- *Fusion Compiler Timing Analysis User Guide*
- *Fusion Compiler Graphical User Interface User Guide*
- *Fusion Compiler Multivoltage User Guide*
- *Fusion Compiler Power Analysis User Guide*

This user guide is for design engineers who use the Fusion Compiler tool to implement designs.

To use the Fusion Compiler tool, you need to be skilled in physical design and synthesis and be familiar with the following:

- Physical design principles
- The Linux operating system
- The tool command language (Tcl)

This preface includes the following sections:

- [New in This Release](#)
- [Related Products, Publications, and Trademarks](#)
- [Conventions](#)
- [Customer Support](#)
- [Statement on Inclusivity and Diversity](#)

New in This Release

Information about new features, enhancements, and changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the Fusion Compiler Release Notes on the SolvNetPlus site.

Related Products, Publications, and Trademarks

For additional information about the Fusion Compiler tool, see the documentation on the Synopsys SolvNetPlus support site at the following address:

<https://solvnetplus.synopsys.com>

You might also want to see the documentation for the following related Synopsys products:

- Design Compiler®
- IC Validator
- PrimeTime® Suite
- StarRC™

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates syntax, such as <code>write_file</code> .
<i>Courier italic</i>	Indicates a user-defined value in syntax, such as <code>write_file design_list</code>
Courier bold	Indicates user input—text you type verbatim—in examples, such as <code>prompt> write_file top</code>
Purple	<ul style="list-style-type: none">• Within an example, indicates information of special interest.• Within a command-syntax section, indicates a default, such as <code>include_enclosing = true false</code>
[]	Denotes optional arguments in syntax, such as <code>write_file [-format fmt]</code>

Convention	Description
...	Indicates that arguments can be repeated as many times as needed, such as <i>pin1 pin2 ... pinN</i> .
	Indicates a choice among alternatives, such as low medium high
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Bold	Indicates a graphical user interface (GUI) element that has an action associated with it.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy .
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.

Customer Support

Customer support is available through SolvNetPlus.

Accessing SolvNetPlus

The SolvNetPlus site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNetPlus site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNetPlus site, go to the following address:

<https://solvnetplus.synopsys.com>

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

If you need help using the SolvNetPlus site, click REGISTRATION HELP in the top-right menu bar.

Contacting Customer Support

To contact Customer Support, go to <https://solvnetplus.synopsys.com>.

Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

1

Working With the Fusion Compiler Tool

The Fusion Compiler tool supports the following functionality:

- Extraction and timing analysis
- Physical synthesis and optimization
- Placement and optimization, including relative placement
- Clock tree synthesis
- Routing
- Chip finishing
- Top-level closure for hierarchical designs
- Engineering change orders (ECO)
- Reporting
- ASCII output interfaces

It takes as input a Verilog netlist, timing constraints, logic and physical libraries, and foundry-process data. It generates as output an OASIS- or GDSII-format file of the layout or a Design Exchange Format (DEF) file of placed netlist data ready for a third-party router. The Fusion Compiler tool can also output the design at any time as a binary design database or as ASCII files (Verilog, DEF, and timing constraints).

The Fusion Compiler tool provides a unified

- Data model, which enables sharing of libraries, design data, constraints, and design intent throughout the entire implementation flow and is architected to support ultra-large designs with a small memory footprint
- Graphical user interface (GUI), which enables seamless visual analysis of timing paths, power intent, schematic, and layout, with full cross-probing between the RTL and netlist
- Design mismatch mitigation infrastructure, which provides user controls to mitigate incomplete or mismatched library and design data

The following topics describe how to use the Fusion Compiler tool:

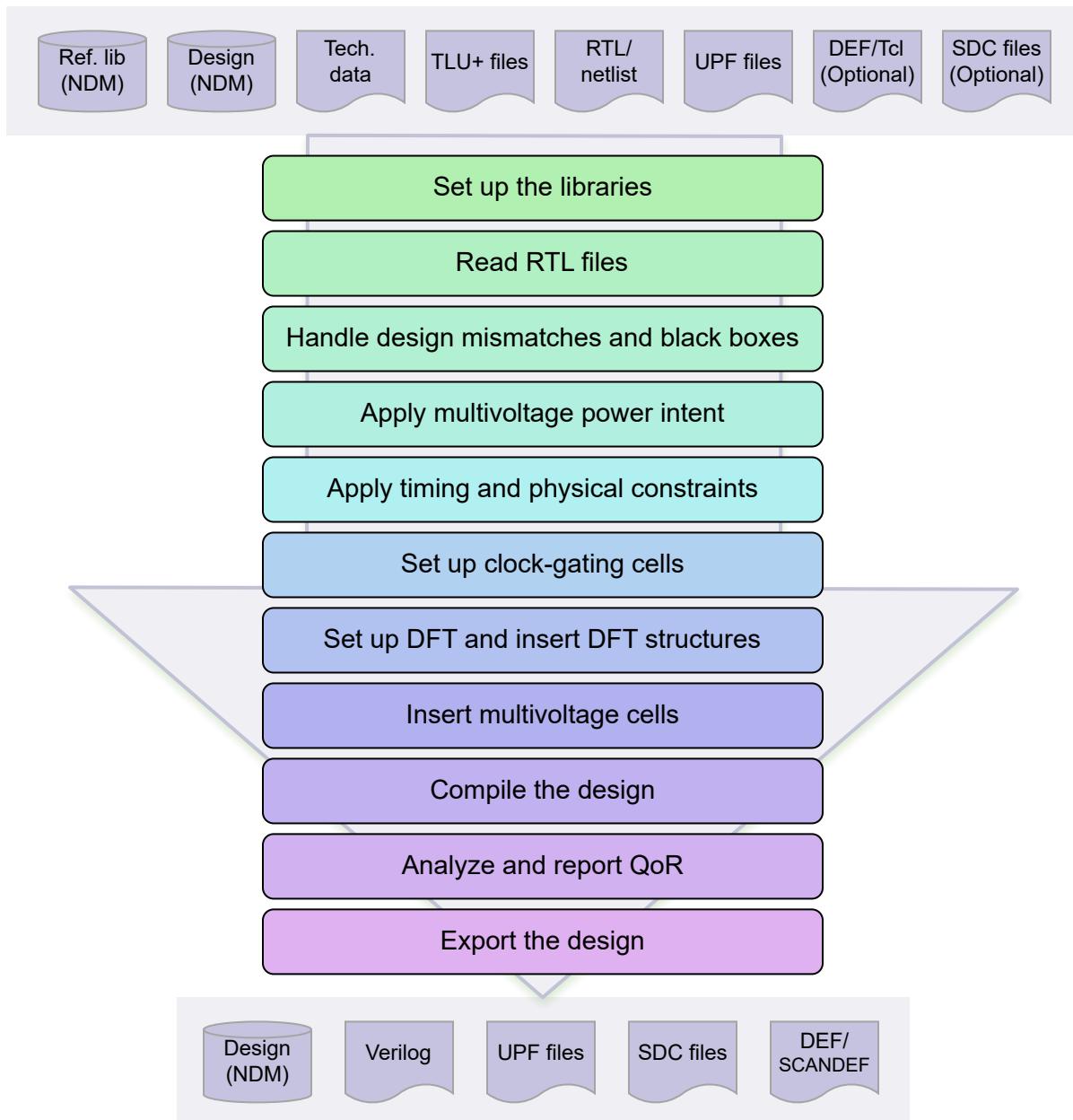
- [Physical Synthesis Design Flow Overview](#)
- [Formal Verification](#)
- [Place and Route Design Flow Overview](#)
- [Fusion Compiler Concepts](#)
- [User Interfaces](#)
- [Entering fc_shell Commands](#)
- [Using Application Options](#)
- [Using Variables](#)
- [Viewing Man Pages](#)
- [Using Tcl Scripts](#)
- [Adding Changes to a Script With Checkpoints](#)
- [Using Setup Files](#)
- [Using the Command Log File](#)
- [Enabling Multicore Processing](#)

For information about working with design data in the Fusion Compiler tool, see the *Fusion Compiler Data Model User Guide*.

Physical Synthesis Design Flow Overview

[Figure 1](#) shows the basic physical synthesis design flow.

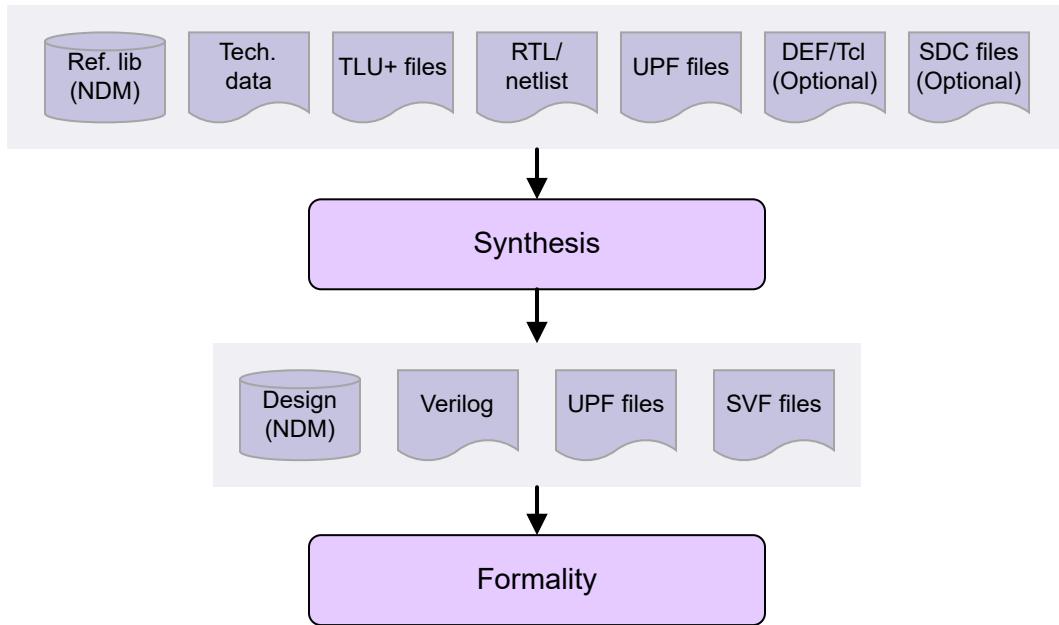
Figure 1 Fusion Compiler Physical Synthesis Flow



Formal Verification

The Formality tool uses formal techniques to prove or disprove the functional equivalence of two designs. It performs RTL-to-RTL, RTL-to-gate, and gate-to-gate verifications. Functional equivalence checking does not take into account static timing verification. The

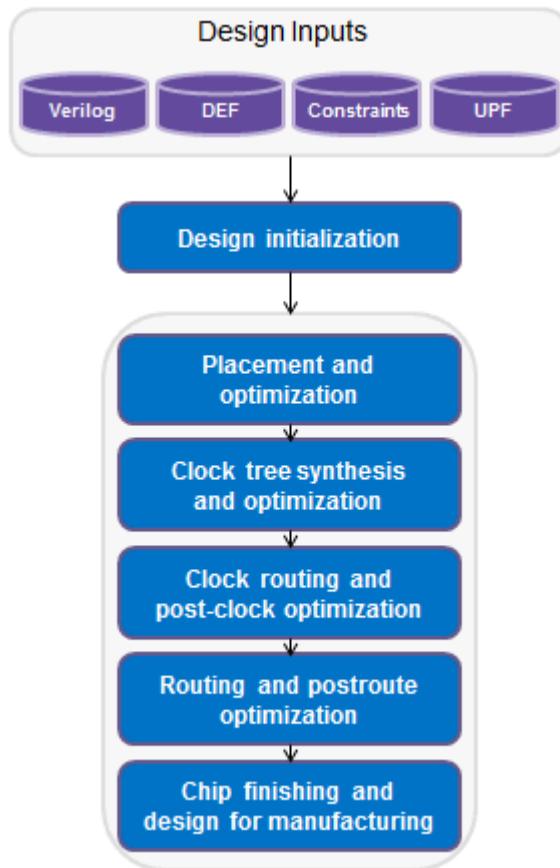
following figure shows an overview of the design flow from the Fusion Compiler tool to the Formality tool:



Place and Route Design Flow Overview

[Figure 2](#) shows the basic place and route design flow using the Fusion Compiler tool.

Figure 2 Fusion Compiler Place and Route Flow



To run the Fusion Compiler place and route flow,

1. Set up the libraries and prepare the design data, as described in [Preparing the Design](#).
2. Perform design planning and power planning.

When you perform design planning and power planning, you create a floorplan to determine the size of the design, create the boundary and core area, create site rows for the placement of standard cells, set up the I/O pads, and create a power plan.

For more information about design planning and power planning, see the *Fusion Compiler Design Planning User Guide*.

3. Perform placement and optimization.

To perform placement and optimization, use the `place_opt` command.

The `place_opt` command addresses and resolves timing closure for your design. This iterative process uses enhanced placement and synthesis technologies to generate

legalized placement for leaf cells and an optimized design. You can supplement this functionality by optimizing for power, recovering area for placement, minimizing congestion, and minimizing timing and design rule violations.

4. Perform clock tree synthesis and optimization.

To perform clock tree synthesis and optimization, use the `clock_opt` command.

Fusion Compiler clock tree synthesis and embedded optimization solve complicated clock tree synthesis problems, such as blockage avoidance and the correlation between preroute and postroute data. Clock tree optimization improves both clock skew and clock insertion delay by performing buffer sizing, buffer relocation, gate sizing, gate relocation, level adjustment, reconfiguration, delay insertion, dummy load insertion, and balancing of interclock delays.

For more information about clock tree synthesis and optimization, see [Clock Tree Synthesis](#).

5. Perform routing and postroute optimization, as described in [Routing and Postroute Optimization](#).

The Fusion Compiler tool uses Zroute to perform global routing, track assignment, detail routing, topological optimization, and engineering change order (ECO) routing. To perform postroute optimization, use the `route_opt` command. For most designs, the default postroute optimization setup produces optimal results. If necessary, you can supplement this functionality by optimizing routing patterns and reducing crosstalk or by customizing the routing and postroute optimization functions for special needs.

6. Perform chip finishing and design for manufacturing tasks, as described in [Chip Finishing and Design for Manufacturing](#).

The Fusion Compiler tool provides chip finishing and design for manufacturing and design for yield capabilities that you can apply throughout the various stages of the design flow to address process design issues encountered during chip manufacturing.

7. Save the design.

Fusion Compiler Concepts

This topic introduces the following concepts used in the Fusion Compiler tool:

- [Power Intent Concepts](#)
- [Multiple-Patterning Concepts](#)

Power Intent Concepts

The Fusion Compiler tool uses the Unified Power Format (UPF) to specify the power intent for multivoltage designs. This topic provides an overview of the UPF concepts and the supported UPF flows.

UPF Concepts

The UPF language establishes a set of commands used to specify the low-power design intent for electronic systems. Using UPF commands, you can specify the supply network, switches, isolation, retention, and other aspects relevant to power management of a chip design. The same set of low-power design specification commands is to be used throughout the design, analysis, verification, and implementation flow. Synopsys tools are designed to follow the official UPF standard.

The UPF language provides a way to specify the power requirements of a design, but without specifying explicitly how those requirements are implemented. The language specifies how to create a power supply network for each design element, the behavior of supply nets with respect to each other, and how the logic functionality is extended to support dynamic power switching to design elements. It does not contain any placement or routing information.

In the UPF language, a *power domain* is a defined group of elements in the logic hierarchy that share a common set of power supply needs. By default, all logic elements in a power domain use the same primary supply and primary ground. Other power supplies can optionally be defined for a power domain as well. A power domain is typically implemented as a contiguous *voltage area* in the physical chip layout, although this is not a requirement of the language.

Each power domain has a *scope* and an *extent*. The *scope* is the level of logic hierarchy where the power domain exists. The *extent* is the set of logic elements that belong to the power domain and share the same power supply needs. In other words, the scope is the hierarchical level where the power domain exists, whereas the extent is what is contained within the power domain.

Each scope or hierarchical level in the design has *supply nets* and *supply ports*. A *supply net* is a conductor that carries a supply voltage or ground throughout a given power domain. A supply net that spans more than one power domain is said to be “reused” in multiple domains. A *supply port* is a power supply connection point between two adjacent levels of the design hierarchy, between parent and child blocks of the hierarchy. A supply net that crosses from one level of the design hierarchy to the next must pass through a supply port.

A *power switch* (or simply *switch*) is a device that turns on and turns off power for a supply net. A switch has an input supply net, an output supply net that can be switched on or off, and at least one input signal to control switching. The switch can optionally have multiple input control signals and one or more output acknowledge signals. A *power state table*

lists the allowed combinations of voltage values and states of the power switches for all power domains in the design.

Where a logic signal leaves one power domain and enters another at a substantially different supply voltage, a *level-shifter* cell must be present to convert the signal from the voltage swing of the first domain to that of the second domain.

Where a logic signal leaves a power domain and enters a different power domain, an *isolation* cell must be present to generate a known logic value during shutdown. If the voltage levels of the two domains are substantially different, the interface cell must perform both level shifting when the domain is powered up and isolation when the domain is powered down. A cell that can perform both functions is called an *enable level shifter*.

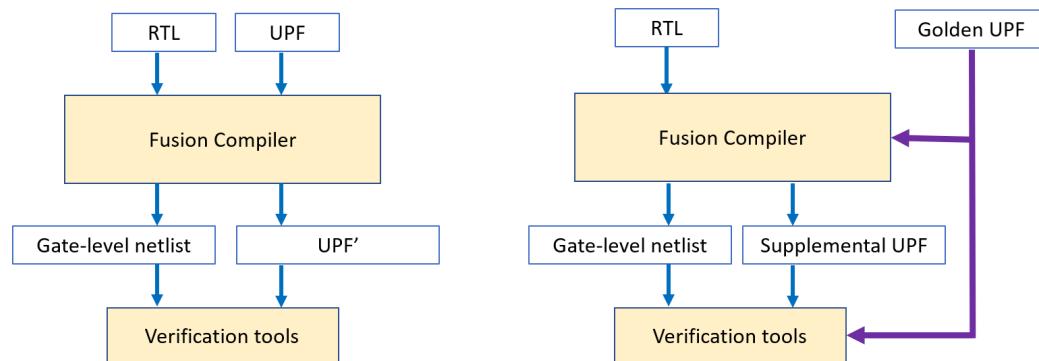
In a power domain that has power switching, any registers that are to retain data during shutdown must be implemented as *retention registers*. A retention register has a separate, always-on supply net, sometimes called the backup supply, which keeps the data stable while the primary supply of the domain is shut down.

UPF Flows

The Fusion Compiler tool supports both the traditional UPF flow and the golden UPF flow. The golden UPF flow is an optional method of maintaining the UPF multivoltage power intent of the design. It uses the original “golden” UPF file throughout the synthesis, physical implementation, and verification steps, along with supplemental UPF files generated by the Fusion Compiler tool.

[Figure 3](#) compares the traditional UPF flow with the golden UPF flow.

[Figure 3](#) UPF-Prime (Traditional) and Golden UPF Flows



The golden UPF flow maintains and uses the same, original “golden” UPF file throughout the flow. The Fusion Compiler tool writes power intent changes into a separate “supplemental” UPF file. Downstream tools and verification tools use a combination of the golden UPF file and the supplemental UPF file, instead of a single UPF’ or UPF” file.

The golden UPF flow offers the following advantages:

- The golden UPF file remains unchanged throughout the flow, which keeps the form, structure, comment lines, and wildcard naming used in the UPF file as originally written.
- You can use tool-specific conditional statements to perform different tasks in different tools. Such statements are lost in the traditional UPF-prime flow.
- Changes to the power intent are easily tracked in the supplemental UPF file.
- You can optionally use the Verilog netlist to store all PG connectivity information, making `connect_supply_net` commands unnecessary in the UPF files. This can significantly simplify and reduce the overall size of the UPF files.

To enable the golden UPF flow, use the following application option setting before you load the UPF:

```
fc_shell> set_app_options -as_user_default \
    -list {mv.upf.enable_golden_upf true}
```

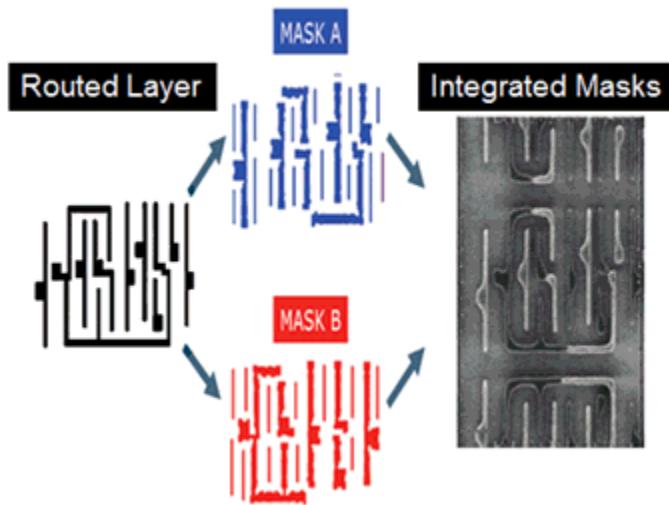
To load supplemental UPF files, use the `-supplemental` option with the `load_upf` command.

For more information about using the golden UPF flow, see [SolvNet article 1412864, "Golden UPF Flow Application Note."](#)

Multiple-Patterning Concepts

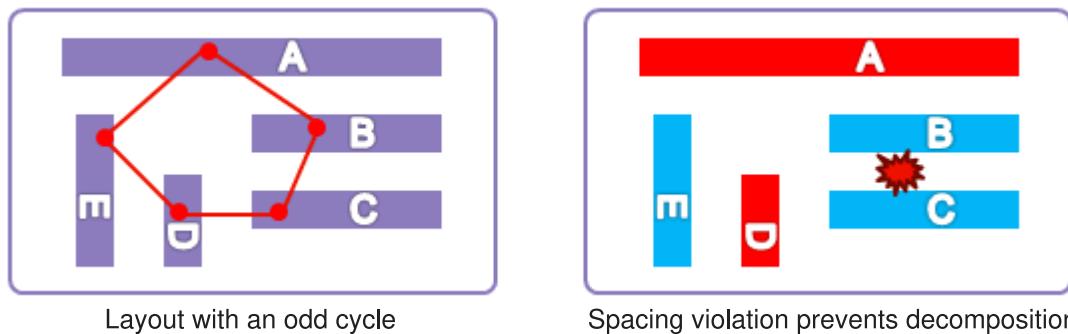
At the 20-nm process node and below, printing the required geometries is extremely difficult with the existing photolithography tools. To address this issue, a new technique, *multiple patterning*, is used to partition the layout mask into two or more separate masks, each of which has an increased manufacturing pitch to enable higher resolution and better printability. [Figure 4](#) shows an example of double-patterning, where the layout mask is partitioned into two separate masks, MASK A and MASK B.

Figure 4 Double-Patterning Example



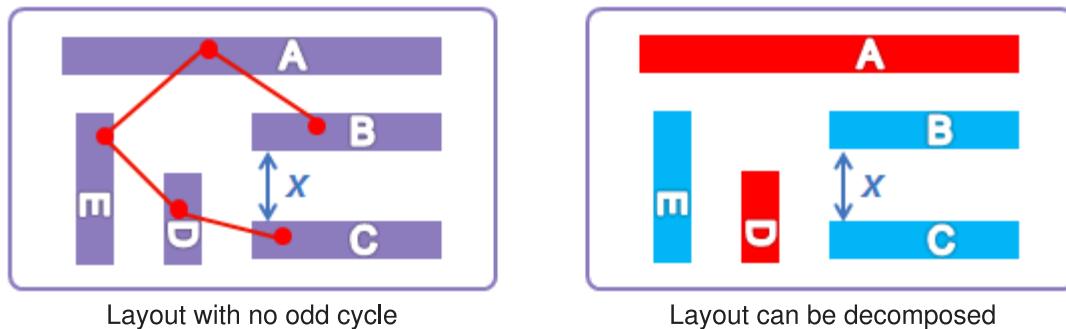
To use multiple patterning, you must be able to decompose the layout into two or more masks, each of which meets the multiple-patterning spacing requirements. A multiple-patterning violation occurs if your layout contains a region with an odd number of neighboring shapes where the distance between each pair of shapes is smaller than the multiple-patterning minimum spacing. This type of violation, which is called an *odd cycle*, is shown in [Figure 5](#).

Figure 5 Odd-Cycle Violation



If the spacing between any pair in the loop is greater than the multiple-patterning minimum spacing, no violation occurs and the layout can be decomposed. For example, in [Figure 6](#), if the spacing, x , between segments B and C is greater than the multiple-patterning minimum spacing, there is no odd cycle and the layout can be decomposed.

Figure 6 No Odd-Cycle Violation



The Fusion Compiler tool ensures that the generated layout is conducive to double patterning by considering the multiple-patterning spacing requirements during placement and routing and preventing odd cycles.

In general, double patterning is performed only on the bottom (lowest) metal layers, which are referred to as *multiple-patterning layers*. The metal shapes on the multiple-patterning layers must meet the multiple-patterning spacing requirements, whether they are routing shapes or metal within the standard cells and macros. The metal shapes on other layers do not need to meet the stricter multiple-patterning spacing requirements.

Multiple-patterning considerations affect all parts of the place and route flow. Depending on your standard cell library, you follow either an uncolored or precolored multiple-patterning flow.

- You use the *uncolored* flow if the cells in your standard cell library have sufficient spacing to the cell boundaries to ensure that multiple-patterning violations do not occur during placement. This type of library is referred to as a *correct-by-construction library*; most multiple-patterning libraries are correct-by-construction libraries.

In the uncolored flow, the tool determines the appropriate mask settings for the pins and net shapes.

- You use the *precolored* flow if the cells in your standard cell library have assigned masks on the metal shapes inside the cells. These assigned masks are often referred to as *colors* and this type of library is referred to as a *precolored library*.

In the precolored flow, the tool must consider these mask assignments to ensure that multiple-patterning violations do not occur during placement. The tool also uses the mask assignments to determine the appropriate mask settings for the pins and net shapes.

The mask assignments are represented as *mask constraints* in the Fusion Compiler tool. You must ensure that the mask constraints are properly set before starting place and route. For information about the mask constraints, see [Mask Constraints](#).

Mask Constraints

Mask constraints indicate the mask requirements for the metal shapes of the physical pins and nets in a block that uses multiple-patterning technology. These mask requirements drive placement and routing to ensure that the resulting layout is multiple-patterning compliant.

Note:

Mask constraints are used only for the precolored flow; they are not necessary in the uncolored flow.

You can set mask constraints on timing-critical nets (net shapes, routing rules, and routing blockages) and vias. For nets, the mask constraint is defined in the `mask_constraint` attribute. For vias, the mask constraints are defined in the `lower_mask`, `upper_mask`, and `cut_mask` attributes. [Table 1](#) shows the supported values for these attributes.

Table 1 Mask Constraint Values

Attribute value	Description
<code>same_mask</code>	This constraint means that the mask color is not yet been determined. Shapes with this attribute must be at least the multiple-patterning minimum spacing distance from any other colored metal shape.
<code>mask_one</code>	This constraint means that the shape has the mask1 color. Shapes with this attribute must be at least the multiple-patterning minimum spacing distance from other mask1-colored metal shapes.
<code>mask_two</code>	This constraint means that the shape has the mask2 color. Shapes with this attribute must be at least the multiple-patterning minimum spacing distance from other mask2-colored metal shapes.
<code>mask_three</code>	This constraint means that the shape has the mask3 color. Shapes with this attribute must be at least the multiple-patterning minimum spacing distance from other mask3-colored metal shapes.
<code>any_mask</code>	This constraint means that the shape is not colored. Shapes with this attribute must be at least the standard minimum spacing distance from other metal shapes; the multiple-patterning minimum spacing rules do not apply to these shapes.

User Interfaces

The Fusion Compiler tool operates in the X Windows environment on Linux. It provides a flexible working environment with both a shell command-line interface and a graphical user interface (GUI). The command-line interface is always available during a Fusion Compiler session. You can start or exit a session in either the shell or the GUI, and you can open or close the GUI at any time during a session.

The tool uses the Tool Command Language (Tcl), which is used in many applications in the EDA industry. Using Tcl, you can extend the `fc_shell` command language by writing reusable procedures and scripts (see the *Using Tcl With Synopsys Tools* manual).

The following topics describe how to start and exit the tool using the command-line interface.

- [Starting the Command-Line Interface](#)
- [Exiting the Fusion Compiler Tool](#)

For information about using the GUI, see the *Fusion Compiler Graphical User Interface User Guide*.

The following topics describe how to use the user interfaces of the Fusion Compiler tool:

- Starting the GUI
- Closing the GUI

Starting the Command-Line Interface

The command-line interface is a text-only environment in which you enter commands at the command-line prompt. It is typically used for scripts, batch mode, and push-button operations.

Before you start the command-line interface, ensure that the path to the bin directory is included in your `$PATH` variable.

To start `fc_shell`,

1. Make sure the path to the bin directory is included in your `PATH` variable.
2. Enter the `fc_shell` command in a Linux shell.

```
% fc_shell
```

You can include other options on the command line when you start the shell. For example, you can use

- `-file script_file_name` to execute a script
- `-x command` to execute a command
- `-output_log_file file_name` to create a log file of your session
- `-help` to display a list of the available options (without starting the shell)

At startup, the tool performs the following tasks:

1. Creates a command log file.
2. Reads and executes the setup files.
3. Executes any script files or commands specified by the `-file` and `-x` options, respectively, on the command line.
4. Displays the program header and `fc_shell>` prompt in the shell.

See Also

- [Using the Command Log File](#)
- [Using Setup Files](#)
- [Using Tcl Scripts](#)

Exiting the Fusion Compiler Tool

You can end the session and exit the Fusion Compiler tool at any time. To exit the tool, use the `exit` or `quit` command.

Note:

When you exit the tool from the command line, the tool exits without saving the open blocks.

Entering fc_shell Commands

You interact with the Fusion Compiler tool by using `fc_shell` commands, which are based on the Tool Command Language (Tcl) and include certain command extensions needed to implement specific Fusion Compiler functionality. The Fusion Compiler command language provides capabilities similar to Linux command shells, including variables, conditional execution of commands, and control flow commands. You can

- Enter individual commands interactively at the `fc_shell>` prompt in `fc_shell`
- Enter individual commands interactively on the console command line in the GUI
- Run one or more Tcl scripts, which are text files that contain `fc_shell` commands (see [Using Tcl Scripts](#))

When entering a command, an option, or a file name, you can minimize your typing by pressing the Tab key when you have typed enough characters to specify a unique name; the Fusion Compiler tool completes the remaining characters. If the characters you typed

could be used for more than one name, the Fusion Compiler tool lists the qualifying names, from which you can select by using the arrow keys and the Enter key.

If you need to reuse a command from the output for a command-line interface, you can copy and paste the portion by selecting it, moving the pointer to the fc_shell command line, and clicking with the middle mouse button.

When you run a command, the Fusion Compiler tool echoes the command output (including processing messages and any warnings or error messages) in fc_shell and, if the GUI is open, in the console log view. By default, the tool does not use page mode, so the output might scroll. To enable page mode, set the `sh_enable_page_mode` variable to true.

Interrupting or Terminating Command Processing

If you enter the wrong options for a command or enter the wrong command, you can interrupt command processing and remain in fc_shell. To interrupt or terminate a command, press Ctrl+C.

Some commands and processes cannot be interrupted. To stop these commands or processes, you must terminate fc_shell at the system level. When you terminate a process or the shell, no data is saved.

When you use Ctrl+C, keep the following points in mind:

- If a script file is being processed and you interrupt one of its commands, the script processing is interrupted and no further script commands are processed.
- If you press Ctrl+C three times before a command responds to your interrupt, fc_shell is interrupted and exits with this message:

Information: Process terminated by interrupt.

Getting Information About Commands

The following online information resources are available while you are using the Fusion Compiler tool:

- Command help, which is information about an Fusion Compiler command
- Man pages

See Also

- [Viewing Man Pages](#)

Displaying Command Help

Command help consists of either a brief description of a Fusion Compiler command or a list of the options and arguments supported by an Fusion Compiler command.

- To display a brief description of a Fusion Compiler command, enter the `help` command followed by the command name. For example, to display a brief description of the `report_timing` command, use the following command:

```
fc_shell> help report_timing
```

- To display the options supported by a Fusion Compiler command, enter the command name with the `-help` option on the command line. For example, to see the options supported by the `report_timing` command, use the following command:

```
fc_shell> report_timing -help
```

Using Application Options

The Fusion Compiler tool uses application options to control the tool behavior. Application options use the following naming convention:

category[*.subcategory*]*.option_name*

where *category* is the name of the engine affected by the application option. Some application option categories have subcategories to further refine the area affected by the application option.

Application options have either a global scope or a block scope.

- Block-scoped application options apply only to the block on which they are set. They are saved in the design library and are persistent across tool sessions.
- Global-scoped application options apply to all blocks, but only within the current session. They are not saved in the design library; you must specify them in each `fc_shell` session. You might want to consider adding the global settings to your `.synopsys_fc.setup` file.

To get a list of available application options, use the `get_app_options` command. By default, this command lists all application options. To restrict the reported application options, provide a pattern string as an argument to the command.

For example, to list all available application options, use the following command:

```
fc_shell> get_app_options
```

To list all available timer application options, use the following command:

```
fc_shell> get_app_options timer.*
```

To generate a report of application options, use the `report_app_options` command.

For detailed information about application options, see Application Options in the *Fusion Compiler Data Model User Guide*.

See Also

- [Using Setup Files](#)

Using Variables

In general, the Fusion Compiler tool modifies default behavior by using application options rather than application variables; however it does support user-defined Tcl variables, as well as a minimal number of application variables, such as the `search_path` variable.

To list the variables and their values, use the `printvar` command. For example, to list all variables defined in the current session, use the following command:

```
fc_shell> printvar *
```

To print the value of the `search_path` variable, use the following command:

```
fc_shell> printvar search_path
```

See Also

- [Defining the Search Path](#)

Viewing Man Pages

To display the man page for a Fusion Compiler command or application option, enter the `man` command followed by the command or application option name. For example, to see the man page for the `report_timing` command, use the following command:

```
fc_shell> man report_timing
```

To display the man page for a Fusion Compiler application option, enter the `man` command followed by the option name. You can also view the following types of summary pages for application options:

- Category summaries

To view a man page that summarizes all of the application options for a specific category, enter the `man` command followed by `category_options`. For example, to see the man page that summarizes all timer application options, use the following command:

```
fc_shell> man timer_options
```

- Subcategory summaries

To view a man page that summarizes all of the application options for a specific subcategory, enter the `man` command followed by `category.subcategory_options`. For example, to see the man page that summarizes all common route application options, use the following command:

```
fc_shell> man route.common_options
```

- Command summaries

To view a man page that summarizes all of the application options for a specific command, enter the `man` command followed by `command_options`. For example, to see the man page that summarizes all application options that affect the `report_timing` command, use the following command:

```
fc_shell> man report_timing_options
```

If you enter the `man` command on the `fc_shell` command line, the man page is displayed in the Fusion Compiler shell and in the console log view if the GUI is open. If you enter this command on the console command line in the GUI, the man page is displayed in the GUI man page viewer.

Using Tcl Scripts

You can use Tcl scripts to accomplish routine, repetitive, or complex tasks. You create a command script file by placing a sequence of Fusion Compiler commands in a text file. Any Fusion Compiler command can be executed within a script file.

In Tcl, a pound sign (#) at the beginning of a line denotes a comment. For example,

```
# This is a comment
```

For more information about writing scripts and script files, see the *Using Tcl With Synopsys Tools* manual.

Use one of the following methods to run a Tcl script:

- Use the `-file` option with the `fc_shell` command when you start the Fusion Compiler tool.
- Use the `source` command from the `fc_shell` command line.
- Choose File > Execute Script in the GUI.

If an error occurs when running a command, the Fusion Compiler tool raises the `TCL_ERROR` condition, which immediately stops the script execution. To tolerate errors and allow the script to continue executing, either

- Check for `TCL_ERROR` error conditions with the `Tcl catch` command on the commands that might generate errors.
- Set the `sh_continue_on_error` variable to `true` in the script file.

See Also

- [Starting the Command-Line Interface](#)
- [Adding Changes to a Script With Checkpoints](#)

Adding Changes to a Script With Checkpoints

When running experiments on a block, you might modify your golden flow scripts in two ways:

- By making flow changes, such as setting application options, modifying constraints, or making other changes to improve the quality of results
- By generating reports (either additional reports or reports at new places in the flow) to help debug an issue

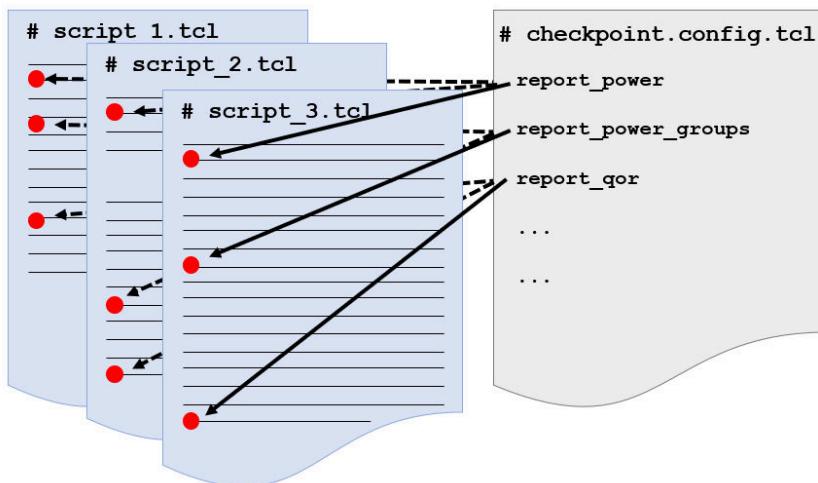
In most flows, you apply changes through a small number of static, modifiable files. Here is a typical example to run the `place_opt` flow, where you source a `pre_place_opt_settings.tcl` file to add flow changes and a `generate_reports.tcl` file to run all your reporting at the end of the script:

```
open_block ./design:init_design
source ./scripts/pre_place_opt_settings.tcl
remove_buffer_trees -all
place_opt -from initial_place -to initial_drc
create_placement -incremental -timing_driven -congestion
place_opt -from initial_drc -to -initial_opto
place_opt -from final_place -to final_opto
source ./scripts/generate_reports.tcl
```

In some cases, complex changes require modifying your golden flow script directly, which might be discouraged or difficult to do in many design environments. These changes can be time consuming to implement, especially if you are implementing them at multiple stages in the design flow, across multiple blocks, or for several different flow experiments.

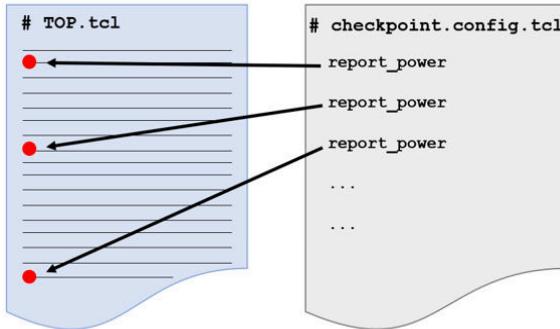
The checkpoint system streamlines this process and allows you to run experiments without modifying your golden flow scripts. When you use the checkpoint system, you define checkpoints for important steps in your golden flow. These checkpoints enwrap the commands associated with the given steps. By default, the checkpoints run only the code they enwrap. That is, simply inserting checkpoints in your script does not change your flow. However, you can associate these checkpoints with flow changes or reports that you want to run before, after, or in place of the code the checkpoints enwrap. By defining these associations in a portable configuration file, you isolate your golden flow scripts from the changes you want to apply for a particular run.

The following figure shows how you can use checkpoints to instantly apply a set of flow changes or reports to multiple designs, without modifying each design's golden flow script. In this case, the configuration file has been copied to each design's run directory.

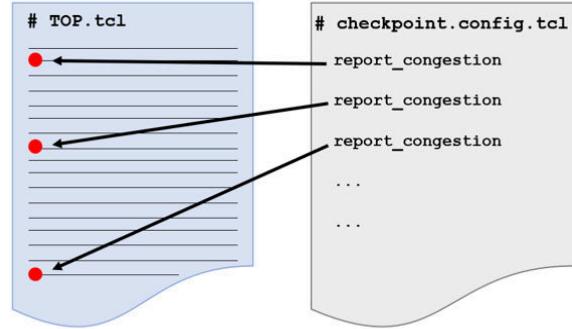


In addition, when multiple users are working on a project, each user can have an individual checkpoint.config.tcl file with his or her preferred settings:

User A's Configuration



User B's Configuration



The checkpoint system also allows you to

- Insert flow changes and reports with more precision than you could using a typical non-checkpointed flow script
- Reference a single source of truth where you can find all changes that have been applied to your default golden flow
- Report a history of the checkpoints that were run, including the flow changes or reports that were run at those checkpoints
- Report runtime and memory information for each checkpoint that was run
- Generate unique and descriptive report names

The general process for using the checkpoint system is as follows:

1. [Insert checkpoints in your script.](#)
2. [Configure your checkpoints](#) by
 - Defining the flow changes or reports you want to associate with the checkpoints
 - Associating the flow changes or reports you defined with your checkpoints
3. Run your script.

Defining Checkpoints

To define a checkpoint, insert the `eval_checkpoint` command in your script. Specify a unique name for the checkpoint and wrap the checkpoint around a block of code.

Note the following:

- If you define a checkpoint with the same name as an existing checkpoint, the tool automatically adds a unique suffix to the name of the newly-defined checkpoint when it encounters the checkpoint in a run. For example, if you duplicate a checkpoint named placement, the tool renames the duplicate to placement2 when it encounters the checkpoint in a run.
- Do not nest checkpoints within other checkpoints. Although the tool does not prevent you from defining nested checkpoints, the tool ignores nested checkpoints during a run, executing only the outermost checkpoints (note that the Tcl code inside nested checkpoints still runs, but runs as though the checkpoints do not exist).

Suppose you have a simple place_opt script:

```
open_block ./design:init_design
source ./scripts/pre_place_opt_settings.tcl
remove_buffer_trees -all
place_opt -from initial_place -to initial_drc
create_placement -incremental -timing_driven -congestion
place_opt -from initial_drc -to -initial_opto
place_opt -from final_place -to final_opto
```

The following example wraps a checkpoint around each major step in your golden flow. The checkpoints are named remove_buffers, place_opt_to_initial_drc, incr_placement, place_opt_to_initial_opto, and place_opt_to_final_opto.

```
open_block ./design.nlib:init_design
source ./scripts/pre_place_opt_settings.tcl

eval_checkpoint remove_buffers {
    remove_buffer_trees -all
}

eval_checkpoint place_opt_to_initial_drc {
    place_opt -from initial_place -to initial_drc
}

eval_checkpoint incr_placement {
    create_placement -incremental -timing_driven -congestion
}

eval_checkpoint place_opt_to_initial_opto {
    place_opt -from initial_drc -to initial_opto
}

eval_checkpoint place_opt_to_final_opto {
    place_opt -from final_place -to final_opto
}
```

By default, the checkpoints run the Tcl code they enwrap. That is, simply inserting checkpoints with the `eval_checkpoint` command does not change your flow. However, you can associate these checkpoints with flow changes or reports that you want to run before, after, or in place of the code the checkpoints enwrap, as described in [Configuring Checkpoints](#).

See Also

- [Configuring Checkpoints](#)
- [Querying Checkpoints and Checkpoint Behaviors](#)

Configuring Checkpoints

For a checkpoint to affect your flow, you must associate it with a flow change or report that you want to run before, after, or in place of the code the checkpoint enwraps.

The process of defining flow changes and reports and associating them with individual checkpoints is described in the following topics:

- [Defining Checkpoint Behaviors](#)
- [Associating Checkpoints and Checkpoint Behaviors](#)

Defining Checkpoint Behaviors

You can define two types of checkpoint behaviors: reports and actions. Define a checkpoint report to generate one or more reports supported by the tool or custom reports you create yourself. Define a checkpoint action to execute any kind of flow change before, after, or in place of the code block enwrapped by a checkpoint. For example, you might set an application option to a specific value before a particular checkpoint.

Define these behaviors in your `checkpoint.config.tcl` file, which is automatically sourced by the checkpoint system.

Defining Checkpoint Reports

To define a checkpoint report, use the `create_checkpoint_report` command in your `checkpoint.config.tcl` file. Specify a unique name for the report and define the Tcl commands to generate the report.

The following example defines a checkpoint report named `timing`, which writes the output of the `report_qor` and `report_timing` commands to disk:

```
create_checkpoint_report timing {
    set name [get_current_checkpoint -name]
    set pos [get_current_checkpoint -position]

    report_qor -nosplit > ./checkpoint/$name.$pos.qor.rpt
```

```

report_qor -summary -nosplit >> ./checkpoint/$name.$pos.qor.rpt

report_timing -nosplit -max_paths 10 \
> ./checkpoint/$name.$pos.path.rpt
}

```

Notice the use of the `get_current_checkpoint` command with the `-name` and `-position` options to give the generated reports a meaningful name. When these reports are generated, their names reflect the checkpoint that triggered them (`get_current_checkpoint -name`), as well as whether they were generated before or after the checkpoint (`get_current_checkpoint -position`). For example, if this checkpoint report is executed after a checkpoint named `checkpoint_A`, the generated reports are saved to the checkpoint directory, and the name of the QoR report is `checkpoint_A.after.qor.rpt`.

The next example defines a checkpoint report named `app_options`, which writes all your non-default application options to disk:

```

create_checkpoint_report app_options {
    set name [get_current_checkpoint -name]
    set pos [get_current_checkpoint -position]

    report_app_options -non_default \
        > ./checkpoint/$name.$pos.app_options.rpt
}

```

Defining Checkpoint Actions

To define a checkpoint action, use the `create_checkpoint_action` command in your `checkpoint.config.tcl` file. Specify a unique name for the action and define the Tcl commands that constitute the action.

The following example defines a checkpoint action named `gropto`, which enables global-route based buffering:

```

create_checkpoint_action gropto {
    set_app_options -name place_opt.initial_drc.global_route_based \
        -value true
}

```

The next example defines a checkpoint action named `placer_high_effort_cong`, which runs high-effort congestion reduction:

```

create_checkpoint_action placer_high_effort_cong {
    set placer_command [get_current_checkpoint -script]
    set cong_option "-congestion_effort high"
    eval $placer_command $cong_option
}

```

Notice the use of the `get_current_checkpoint -script` command and option, which retrieves the command originally enwrapped by the checkpoint and sets its congestion effort to high (`-congestion_effort high`). The `-script` option is typically used to define actions that replace the contents of a command. In this example, the action modifies the `-congestion_effort` option of the command that is enwrapped by the checkpoint associated with this action.

Associating Checkpoints and Checkpoint Behaviors

After defining checkpoints with the `eval_checkpoint` command and checkpoint behaviors with the `create_checkpoint_*` command, you must associate the two so that the tool executes the behaviors when it encounters the checkpoints in a run, as described in the following topics:

- [Associating Checkpoints With Checkpoint Reports](#)
- [Associating Checkpoints With Checkpoint Actions](#)

See Also

- [Defining Checkpoints](#)
- [Defining Checkpoint Behaviors](#)

Associating Checkpoints With Checkpoint Reports

To associate a checkpoint report with one or more checkpoints, use the `associate_checkpoint_report` command in your `checkpoint.config.tcl` file.

- Specify the checkpoint report name with the `-enable` option.
- To enable the report to run before one or more checkpoints, specify the checkpoint names with the `-before` option.
- To enable the report to run after one or more checkpoints, specify the checkpoint names with the `-after` option.

When associating a report with multiple checkpoints, you can list the checkpoint names or specify the asterisk wildcard character.

Assume you have defined three checkpoints in your script named `remove_buffers`, `place_opt_to_initial_opto`, and `place_opt_to_final_opto` using the `eval_checkpoint` command, and two checkpoint reports named `timing` and `app_options` using the `create_checkpoint_report` command.

To enable your `timing` report to run after the `place_opt_to_initial_opto` and `place_opt_to_final_opto` checkpoints, use the following command:

```
associate_checkpoint_report -enable timing \
    -after { place_opt_to_initial_opto place_opt_to_final_opto }
```

To enable your app_options report to run before all the checkpoints in your script, use the following command:

```
associate_checkpoint_report -enable app_options -before *
```

You can also add the command like this, which enables the report to run before the remove_buffers checkpoint and any checkpoints beginning with the phrase place_opt:

```
associate_checkpoint_report -enable app_options \
    -before remove_buffers place_opt*
```

The following is a portion of your checkpointed script showing what happens when the script runs. Note that the highlighted code is not actually in your script, but shows the checkpoint behaviors that are executed when the tool encounters the checkpoints in the run.

- The code that is commented out identifies the original script body
- The blue code identifies the checkpointed Tcl commands in your golden flow
- The green code identifies the app_options report, configured to run before each checkpoint
- The purple code identifies the timing report, configured to run after the place_opt_to_final_opto checkpoint

```
# eval_checkpoint remove_buffers {
#     remove_buffer_trees -all
# }
report_app_options -non_default > ./checkpoint/remove_buffers.before.app_options.rpt
remove_buffer_trees -all
...
# eval_checkpoint place_opt_to_final_opto {
#     place_opt -from final_place -to final_opto
# }
report_app_options -non_default > ./checkpoint/place_opt_to_final_opto.before.app_options.rpt
place_opt -from final_place -to final_opto
report_qor -nosplit > ./checkpoint/place_opt_to_final_opto.after.qor.rpt
report_qor -summary -nosplit >> ./checkpoint/place_opt_to_final_opto.after.qor.rpt
report_timing -nosplit -max_paths 10 > ./checkpoint/place_opt_to_final_opto.after.path.rpt
```

Associating Checkpoints With Checkpoint Actions

To associate a checkpoint action with one or more checkpoints, use the associate_checkpoint_action command in your checkpoint.config.tcl file.

- Specify the checkpoint action name with the `-enable` option.
- To enable the action to run before one or more checkpoints, specify the checkpoint names with the `-before` option.
- To enable the action to run after one or more checkpoints, specify the checkpoint names with the `-after` option.
- To enable the action to run instead of the code block enwrapped by the checkpoint, specify the checkpoint names with the `-replace` option.

When associating an action with multiple checkpoints, you can list the checkpoint names or specify the asterisk wildcard character.

Assume you have defined two checkpoints in your script named `place_opt_to_initial_drc` and `incr_placement` with the `eval_checkpoint` command, and two checkpoint actions named `gropto` and `placer_high_effort_cong` with the `create_checkpoint_action` command.

To enable your `gropto` action to run before the `place_opt_to_initial_drc` checkpoint, use the following command:

```
associate_checkpoint_action -enable gropto \
    -before place_opt_to_initial_drc
```

To enable your `placer_high_effort_cong` action to run instead of the code block enwrapped by the `incr_placement` checkpoint, use the following command:

```
associate_checkpoint_action -enable placer_high_effort_cong \
    -replace incr_placement
```

The following is a portion of your checkpointed script showing what happens when the script runs. Note that the highlighted code is not actually in your script, but shows the checkpoint behaviors that are executed when the tool encounters the checkpoints in a run.

- The code that is commented out identifies the original script body
- The blue code identifies the checkpointed Tcl commands in your golden flow
- The green code identifies the `gropto` action, configured to run before the `place_opt_to_initial_drc` checkpoint
- The purple code identifies the `placer_high_effort_cong` action, configured to run in place of the code block enwrapped by the `incr_placement` checkpoint

```

# eval_checkpoint remove_buffers {
#     remove_buffer_trees -all
# }
remove_buffer_trees -all

# eval_checkpoint place_opt_to_initial_drc {
#     place_opt -from initial_place -to initial_drc
# }
set_app_options -name place_opt.initial_drc.global_route_based -value true
place_opt -from initial_place -to initial_drc

# eval_checkpoint incr_placement {
#     create_placement -incremental -timing_driven -congestion
# }
## Runs 'create_placement -incremental -timing_driven -congestion -congestion_effort high'
set placer_command [get_current_checkpoint -script]
set cong_option "-congestion_effort high"
eval $placer_command $cong_option
create_placement -incremental -timing_driven -congestion

```

Associating a Checkpoint With Multiple Behaviors

When you associate a checkpoint with multiple behaviors, the tool executes those behaviors in the following order:

- Any reports configured to run before the checkpoint
- Any actions configured to run before the checkpoint
- The checkpointer code block, or any actions configured to replace it
- Any actions configured to run after the checkpoint
- Any reports configured to run after the checkpoint

If multiple reports or multiple actions are configured to run at the same position (`-before`, `-after`, or `-replace`) in the same checkpoint, the tool executes them in the order in which they were associated with the checkpoint with the `associate_checkpoint_*` commands. For example, suppose you define a timing report before an area report. If you associate the area report to run after a checkpoint before you associate the timing report to run after the same checkpoint, the tool runs the area report first when it encounters the checkpoint.

See Also

- [Defining Checkpoints](#)
- [Defining Checkpoint Behaviors](#)
- [Querying Checkpoints and Checkpoint Behaviors](#)

Querying Checkpoints and Checkpoint Behaviors

To return a list of all the checkpoints or checkpoint behaviors in the current session, or to return detailed information for any checkpoint or checkpoint behavior, use the `get_checkpoint_data` command.

- To return a list of all the checkpoints that have been executed, specify the `-list_names` option:

```
fc_shell> get_checkpoint_data -list_names
remove_buffers place_opt_to_initial_drc
incr_placement place_opt_to_initial_opto
place_opt_to_final_opto
```

- To return detailed information for a checkpoint, specify the checkpoint with the `-name` option. The tool returns the output as a Tcl dictionary:

```
fc_shell> get_checkpoint_data -name place_opt_to_initial_drc
memory 173.85 start_time 2.34 end_time 2.34 before_runtime 2.34
before_report_runtime 0.00 after_report_runtime 0.00 self_runtime
0.00 before_reports {} after_reports {app_options timing}
before_actions {gropto} after_actions {} replace_actions {}
```

- To return a list of all the checkpoint behaviors defined in your configuration, specify the `-list_reports` or `-list_actions` option:

```
fc_shell> get_checkpoint_data -list_reports
timing app_options
```

```
fc_shell> get_checkpoint_data -list_actions
gropto placer_high_effort_congestion
```

- To return the contents and associations of a checkpoint report or action, specify the report or action with the `-report` or `-action` option:

```
fc_shell> get_checkpoint_data -report app_options
contents {
    set name [get_current_checkpoint -name]
    set pos [get_current_checkpoint -position]

    report_app_options -non_default \
        > ./checkpoint/$name.$pos.app_options.rpt
} before_patterns {*} after_patterns {}
```

See Also

- [Viewing Your Checkpoint History](#)

Viewing Your Checkpoint History

To view your checkpoint history, navigate to the checkpoint directory. This directory contains a `checkpoint_history.rpt` file that captures the history of the checkpoints the tool encounters during each run, along with the runtime and memory usage for each checkpoint. The checkpoint directory also contains any checkpoint reports you have written to it.

In some cases, you might want to clear the full or partial contents of your checkpoint history. A typical example is when a checkpointed run fails and you want to clear some or all of the data associated with that run, depending on whether you plan to rerun a portion of the flow or the entire flow.

- To clear your full checkpoint history, use the `reset_checkpoints` command.

```
fc_shell> reset_checkpoints
```

The `reset_checkpoints` command clears the full contents of the checkpoint directory, including any reports you have written to it. However, before clearing the checkpoint directory, the `reset_checkpoints` command saves a timestamped copy of the directory in your run directory.

- To clear a portion of the data in your `checkpoint_history.rpt` file, use the `-from` option to specify the name of a checkpoint.

This option removes all checkpoints including and following the specified checkpoint from your `checkpoint_history.rpt` file; it does not remove any checkpoint reports you have written to the checkpoint directory.

For example, suppose your `checkpoint_history.rpt` file contains the following data after your run a full checkpointed flow:

Checkpoints	Memory	StartTime	...
remove_buffers	32290.86	2020-03-23_14:34:17	...
place_opt_to_initial_drc	36389.15	2020-03-23_17:21:56	...
incr_placement	37002.09	2020-03-23_19:13:02	...
place_opt_to_initial_opto	45655.59	2020-03-23_23:44:36	...
place_opt_to_final_opto	46322.34	2020-03-23_30:12:55	...

Suppose you want to rerun your flow beginning with incremental placement. To clear the last three checkpoints from your `checkpoint_history.rpt` file before rerunning that portion of your flow, use the following command:

```
fc_shell> reset_checkpoints -from incr_placement
```

After clearing the checkpoints, your checkpoint_history.rpt file lists only the first two checkpoints in your flow:

Checkpoints	Memory	StartTime	...
remove_buffers	32290.86	2020-03-23_14:34:17	...
place_opt_to_initial_drc	36389.15	2020-03-23_17:21:56	...

Similarly, the `get_checkpoint_data -list_names` command returns only the `remove_buffers` and `place_opt_to_initial_drc` checkpoints:

```
fc_shell> get_checkpoint_data -list_names
remove_buffers place_opt_to_initial_drc
```

Using Setup Files

When you start the Fusion Compiler tool, it automatically executes the commands in the `.synopsys_fc.setup` file.

The tool looks for this file both in your home directory and in the project directory (the current working directory in which you start the Fusion Compiler tool). The file is read in the following order:

1. The `.synopsys_fc.setup` file in your home directory
2. The `.synopsys_fc.setup` file in the project directory

The setup files can contain commands that perform basic tasks, such as initializing application options and setting GUI options. You can add commands and Tcl procedures to the setup files in your home and project directories. For example,

- To set application options that define your Fusion Compiler working environment, create setup files in your home directory.
- To set project- or block-specific application options that affect the processing of a block, create a setup file in the design directory.

See Also

- [User Interfaces](#)
- [Using Application Options](#)
- [Using Variables](#)
- [Using Tcl Scripts](#)

Using the Command Log File

The command log file records the commands processed by the Fusion Compiler tool, including setup file commands and application option settings. By default, the Fusion Compiler tool writes the command log to a file named `fc_command.log` in the directory from which you invoked `fc_shell`.

You can change the name of the command log file by setting the `sh_command_log_file` variable in your `.synopsys_fc.setup` file. You should make any changes to this variable before you start the Fusion Compiler tool. If your user-defined or project-specific setup file does not contain this variable, the Fusion Compiler tool automatically creates the `fc_command.log` file.

Each Fusion Compiler session overwrites the command log file. To save a command log file, move it or rename it. You can use the command log file to

- Produce a script for a particular implementation strategy
- Record the physical implementation process
- Document any problems you are having

Enabling Multicore Processing

Several functions in the Fusion Compiler tool support multicore processing, whether through multithreading, distributed processing, or parallel command execution. Multicore processing improves turnaround time by performing tasks in parallel much more quickly than if they were run sequentially on a single core.

Note:

A single machine has one or more CPUs and each CPU has one or more cores. The total number of cores available for processing on a machine is the number of CPUs multiplied by the number of cores in each CPU.

When using multicore processing, you need one Fusion Compiler license for every 16 parallel tasks. For example, to run 17 parallel tasks, you need 2 Fusion Compiler licenses.

In most cases, you configure multicore processing by using the `set_host_options` command. The following topics describe how to use the `set_host_options` command to configure multicore processing:

- [Configuring Multithreading](#)
- [Configuring Distributed Processing](#)

- [Reporting Multicore Configurations](#)
- [Removing Multicore Configurations](#)

The following topic describes how to use distributed processing to run the same task on several blocks in a hierarchical design:

- [Running Tasks in Parallel](#)

The following topic describes how to use parallel command execution for checking and reporting commands:

- [Running Commands in Parallel on Your Local Host](#)

Configuring Multithreading

Multithreading performs tasks in parallel by using multiple cores on the same machine, using a single process memory image. When using multithreading, each parallel task is called a thread. For the best performance during multithreading, you should limit the number of threads to the number of available cores, which is the number of CPUs in the machine times the number of cores per CPU.

The following commands support multithreading configured by the `set_host_options` command:

- `place_opt`
- `clock_opt`
- `check_legality`, when advanced legalization is enabled by setting the `place.legalize.enable_advanced_legalizer` application option to `true`.
- `insert_via_ladders`
- `route_auto`, `route_global`, `route_track`, `route_detail`, and `route_eco`

Note:

When you run routing with a single thread, the result is deterministic; if you start with the same block, you always get the same result. However, if you use multiple threads, the routing results are not deterministic; the final routing is slightly different between runs due to the varying division of tasks between threads. Global routing supports a deterministic mode for multicore routing. To enable this mode, set the `route.global.deterministic` application option to `on`.

- `route_opt`
- `signoff_check_drc`

- signoff_fix_drc
- signoff_create_metal_fill
- signoff_fix_isolated_via
- write_def

By default, all commands use a single thread. To enable multithreading for those commands that support it, set the `-max_cores` option of the `set_host_options` command to a value greater than one and less than or equal to the number of cores available on your machine, which is the number of CPUs in the machine times the number of cores per CPU. The number of cores specified by the `-max_cores` option applies to all commands that support multithreading.

When you enable multithreading, multithreaded commands create and use the specified number of threads, even if the number is more than the number of available cores. You must set an appropriate number of threads, so that the command does not try to use more resources than it has. Overthreading can reduce performance because the extra threads compete for resources. For best performance, do not run more than one thread per available core.

For example, if your machine has two CPUs and each CPU has three cores, specify six as the maximum number of threads:

```
fc_shell> set_host_options -max_cores 6
```

Configuring Distributed Processing

Distributed processing performs tasks in parallel by using multiple machines; each process uses its own process memory image. When using distributed processing, each parallel task is called a process. For the best performance during distributed processing, you should limit the number of processes to the number of available cores, which is the sum of the number CPUs times the number of cores per CPU for each host machine.

The following commands support distributed processing configured by the `set_host_options` command:

- analyze_rail
- create_placement -floorplan
- signoff_check_drc
- signoff_fix_drc
- signoff_create_metal_fill
- signoff_fix_isolated_via

When you configure distributed processing, you can specify one or more of the following settings:

- The job submission command (the `-submit_command` option)
If you do not specify this option, the tool uses the `rsh` command to submit the parallel processes.
- The list of host machines (the `host_names` argument)
- The maximum number of processes (the `-num_processes` option)

By default, the tool assigns a name to each configuration you define with the `set_host_options` command. To specify the configuration name, use the `-name` option. You use the configuration name to select the configuration to use for specific commands and to remove a configuration.

For example, to specify a distributed processing configuration that uses the `qsub` command to submit the parallel processes, use the following command:

```
fc_shell> set_host_options -name dp_config \
    -submit_command [list qsub -P bnormal -cwd]
```

Reporting Multicore Configurations

To report the values set by the `set_host_options` command, use the `report_host_options` command.

Removing Multicore Configurations

To remove the multicore configurations defined by the `set_host_options` command, use the `remove_host_options` command. To remove all multicore configurations, use the `-all` option. To remove a specific multicore configuration, specify the configuration name with the `-name` option.

Running Tasks in Parallel

To efficiently run the same task on several blocks in your design, you can use the `run_block_script` command to enable distributed processing and perform the tasks in parallel. The `run_block_script` command accepts a Tcl script and applies the commands in the script to the blocks you specify.

For example, to compile the subblocks of a design in parallel in a hierarchical flow,

1. Write the script to run on the blocks in the design.

The following example performs the steps necessary to compile each subblock in the design. You can add additional commands to the script as needed to perform other operations.

```
open_lib $block_libfilename
open_block $block_refname.design
commit_upf
set_editability -from_level 1 -value false
compile_fusion -from initial_map -to initial_map
create_abstract -read_only
create_frame
save_block $block_refname -as $block_refname/initial_map
compile_fusion -from logic_opto -to logic_opto
create_abstract -read_only
create_frame
save_block $block_refname -as $block_refname/logical_opt
compile_fusion -from initial_place -to initial_place
create_abstract -read_only
create_frame
save_block $block_refname -as $block_refname/initial_place
save_block
save_lib
close_block -force -purge
close_lib
```

2. Run the `run_block_script` command to process each block in parallel by using the script created in step 1.

The following example specifies the settings for distributed processing with the `set_host_options` command and runs the `block_synthesis.tcl` synthesis script on the subblocks in parallel by using the Load Sharing Facility (LSF). The command writes data and log files to the `my_work_dir` directory.

```
fc_shell> set block_list [get_attribute \
    [get_cells -hierarchical -filter "is_soft_macro==true" \
        ref_full_name]
fc_shell> set block_list [lsort -unique $block_list]
fc_shell> set_host_options -name myhost -submit_command "sh"
fc_shell> run_block_script \
    -host_options myhost \
```

```
-script block_synthesis.tcl \
-blocks $block_list \
-work_dir my_work_dir
```

To control the order in which blocks are processed, use the `-run_order` option with the `top_down` or `bottom-up` argument. When you specify the `-run_order top-down` option, the tool delays processing for a child block until all parent blocks for the child block are processed. When you specify the `-run_order bottom-up` option, the tool begins by processing the child blocks, then processes the parent blocks. By default, blocks are processed in a bottom-up order.

Running Commands in Parallel on Your Local Host

You can improve runtime by running checking and reporting commands in parallel on your local host. You can run the commands either in the background (with the `redirect -bg` command) or in the foreground (with the `parallel_execute` command). These techniques do not require the use of additional licenses beyond the licenses required for the parent run. They also do not require you to use or configure distributed processing.

When you use these techniques, consider the following guidelines:

- To reduce runtime and memory usage, run the `update_timing` command before running the commands in parallel; otherwise, each command that requires updated timing runs the `update_timing` command independently.
- To pass variables from a child process to the parent process, you must write the contents of the variables to a file during the child process, and then read that file in the parent process.

See Also

- [Running Commands in the Background](#)
- [Running Commands in Parallel](#)

Running Commands in the Background

To improve runtime, you can run checking and reporting commands in the background while you run other commands in the foreground. This is useful in interactive sessions when you want to continue your work while the tool generates a report.

To run commands in the background, use the `redirect` command with the `-bg` option. When you use this command, `fc_shell` returns immediately to execute the next command. If you issue an `exit` command in the parent process, the tool waits for all `redirect -bg` commands to complete before exiting.

To list the commands supported by the `redirect -bg` command, use the `list_commands -bg` command. You can run either a single command or source a Tcl script that contains

only supported commands. If the command or script includes a `redirect -bg` command, the `-bg` option is ignored.

You can run at most two jobs in the background. If you specify more than two background jobs, they are queued.

To specify the maximum number of cores to use for the background jobs, use the `-max_cores` option with the `redirect` command. The number of cores available for the parent process (as specified by the `-max_cores` option of the `set_host_options` command) is reduced by the number of cores allocated for background jobs.

The following example redirects a Tcl script to run in the background:

```
fc_shell> set_host_options -max_cores 8
fc_shell> redirect -bg -max_cores 3 -file bg_log.out \
           {source bg_script.tcl}
Information: redirect -bg with max_cores 3 started. The maximum number of
cores available in parent is reduced to 5. (BGE-004)
```

Reporting Background Jobs

To report the background jobs submitted with the `redirect -bg` command, use the `report_background_jobs` command. This command reports both the completed jobs and the jobs currently running in the background, as shown in the following example:

```
fc_shell> report_background_jobs
JOB 'redirect -bg -file {background.log} source bg_script.tcl
-max_cores 4 ' completed
JOB(pid:13010) 'redirect -bg -file {background_1.log}
source bg_script_1.tcl -max_cores 3 ' is running
```

To omit the completed jobs, use the `-reset` option with the `report_background_jobs` command.

Running Commands in Parallel

To improve runtime, you can run checking and reporting commands in parallel and return to the parent process after the longest running command in the parallel execution list completes. This is useful in batch scripts.

To run commands in parallel, use the `parallel_execute` command, as shown in the following example:

```
fc_shell> update_timing
fc_shell> parallel_execute {
           report_cmd1 log_file1
           report_cmd2 log_file2
           report_cmd3 log_file3
           ...
}
```

To list the supported commands, use the `-list_allowed_commands` option with the `parallel_execute` command.

To specify the maximum number of cores to use when running the `parallel_execute` command, use the `-max_cores` option. If you do not use this option, the tool uses the value of the `-max_cores` option from the `set_host_options` command. If you do not specify the maximum number of cores with either command, the tool runs the commands sequentially instead of in parallel.

To run commands in parallel as a background job, use the `redirect -bg` command to run the `parallel_execute` command, as shown in the following example:

```
fc_shell> redirect -bg -max_cores 3 -file bg_log.out {
    parallel_execute {
        report_cmd1 log_file1
        report_cmd2 log_file2
    }
}
```

2

Preparing the Design

The Fusion Compiler tool uses a design library to store your design and its associated library information. This topic describes how to create a design library and how to prepare and save your design.

These steps are explained in the following topics:

- [Defining the Search Path](#)
- [Setting Up Libraries](#)
- [Using Pin Access Checker Utility](#)
- [Analyzing Libraries](#)
- [Reading the Design](#)
- [Mitigating Design Mismatches](#)
- [Importing the Floorplan Information](#)
- [Setting Up Multivoltage Designs](#)
- [Specifying Timing Constraints and Settings](#)
- [Specifying Logical Design Rule Constraints](#)
- [Specifying Clock Gating Constraints](#)
- [Specifying Physical Constraints for Placement and Legalization](#)
- [Specifying Placement Settings](#)
- [Specifying Legalization Settings](#)
- [Controlling the Optimization of Cells, Nets, Pins, and Ports](#)
- [Specifying Settings for Preroute Optimization](#)
- [Setting Up for Power-Related Features](#)
- [Specifying the Routing Resources](#)

- [Handling Design Data Using the Early Data Check Manager](#)
 - [Applying Mega-Switch Command Settings](#)
-

Defining the Search Path

The Fusion Compiler tool uses a search path to look for files that are specified with a relative path or with no path.

To specify the search path, use the `set_app_var` command to set the `search_path` application variable to the list of directories, in order, in which to look for files. When the tool looks for a file, it starts searching in the leftmost directory specified in the `search_path` variable and uses the first matching file it finds.

You can also use the Tcl `lappend` command to add your directories to the default search path, which is the directory from which you invoked the tool. For example,

```
fc_shell> lappend search_path ./mylibdir
```

Setting Up Libraries

A *block* is a container for physical and functional design data. A *design library* is a collection of related blocks, together with technology data that applies to the block collection. A *chip design* consists of one or more blocks, often stored in different design libraries. A design library uses instances of blocks defined in lower-level libraries, called *reference libraries*. A design library can serve as a reference library for another design library.

To learn about setting up libraries, see the following topics:

- [Working With Design Libraries](#)
- [Setting Up Reference Libraries](#)
- [Library Configuration](#)
- [Restricting Library Cell Usage](#)
- [Restricting the Target Libraries Used](#)
- [Specifying Library Subset Restrictions](#)

Working With Design Libraries

You can create, open, query, save, or close a design library, using an absolute path, a relative path, or no path, by using the following commands:

- `create_lib`

This command creates the library in memory and sets it as the current library. When you run this command to create a new design library, you must specify the library name. Slash (/) and colon (:) characters are not allowed in library names. The following command creates the `my_libA` library using a relative path:

```
fc_shell> create_lib ../../my_lib_dir/my_libA
{my_libA}
```

- `open_lib`

This command opens the specified library, makes that library the current library, and opens all its associated reference libraries. Opening a library means loading it into memory and making its blocks accessible. The following example opens the `my_libA` library saved on disk:

```
fc_shell> open_lib my_libA
Information: Loading library file '/usr/lib/StdCells.ndm' (FILE-007)
Information: Loading library file '/usr/lib/RAMs.ndm' (FILE-007)
Information: Loading library file
  '/usr/lib/PhysicalOnly.ndm' (FILE-007)
{my_libA}
```

- `current_lib`

By default, the library most recently opened is the current library. You can explicitly set any open library to be the current library by using the `current_lib` command. For example,

```
fc_shell> current_lib my_libA
{my_libA}
```

- `save_lib`

When you create or change a library, the changes are stored in memory only. To save a library to disk, use this command. For example,

```
fc_shell> save_lib lib_A
Saving library 'lib_A'
1
```

- `close_lib`

When you no longer need access to data in a library, you can close it by using the `close_lib` command. Be sure to save the changes in the library before you close it. For example,

```
fc_shell> close_lib  
Closing library 'lib_A'  
1
```

In addition, you can use the `current_lib`, `get_libs`, and `report_lib` commands to query design libraries.

For more information, see the Design Libraries topic in the *Fusion Compiler Data Model User Guide*.

Setting Up Reference Libraries

You can specify a reference library list for a design library when you create the design library by using the `-ref_libs` option of the `create_lib` command. You can also change the reference library list at any time by using the `set_ref_libs` command.

Use the following commands to specify, rebind, and report reference libraries:

- `create_lib -ref_libs`

You can specify a relationship between a new design library and its lower-level reference libraries by using the `create_lib` command. For example,

```
fc_shell> create_lib lib_B \  
      -ref_libs {../LIBS/lib_c ../STND/stdhvt.ndm} ...  
{lib_B}
```

- `set_ref_libs -ref_libs`

For an existing design library, open the library and then use the `set_ref_libs` command to specify the reference libraries. For example,

```
fc_shell> current_lib  
{lib_B}  
fc_shell> set_ref_libs \  
      -ref_libs {../LIBS/lib_C ../STND/stdhvt.ndm}  
../LIBS/lib_C ../STND/stdhvt.ndm
```

- `report_ref_libs`

To report the reference libraries of a design library, use the `report_ref_libs` command.

For example,

```
fc_shell> create_lib lib_A -ref_libs \
    {../libs/SCLL.ndm ../libs/SCHH.ndm ../BLOCKS/MACROS}
{lib_A}
fc_shell> report_ref_libs...
Name Path Location
-----
*+ SCLL ../libs/SCLL.ndm /remote/project/libs/SCLL.ndm
*+ SCHH ../libs/SCHH.ndm /remote/project/libs/SCHH.ndm
* MACROS ../BLOCKS/MACROS /remote/project/BLOCKS/MACROS
"*" = Library currently open
 "+" = Library has technology information
```

- `set_ref_libs -rebind`

When you make a change that invalidates the reference library list, such as moving a reference library to a new location, you need to rebind the reference libraries. To do so, use the `-rebind` option, which rebinds each reference library path specified by the `search_path` variable to libraries that are currently loaded in memory. For example,

```
fc_shell> current_lib
{lib_A}
fc_shell> set_app_var search_path {. . ./REFLIBS .. /CLIBS}
. . ./LIBS .. /BLOCKS
fc_shell> set_ref_libs -rebind
.. ./REFLIBS/lib_C .. ./REFLIBS/lib_D .. ./CLIBS/stdhvt.ndm}
```

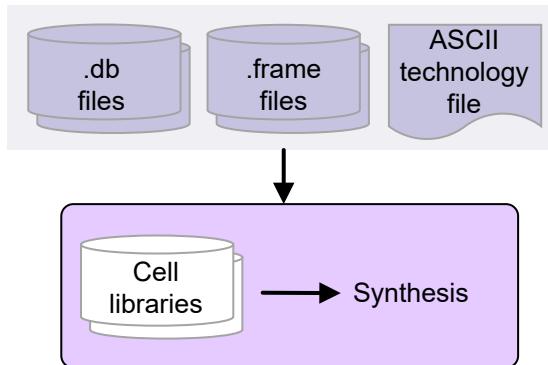
Rebinding a library does not affect the bindings of blocks already existing in the design library. To rebind these blocks using an updated reference library list, use the `-rebind` option with the `link_block` command.

Library Configuration

Library configuration allows you to specify which vendor libraries to use as reference libraries for the current design. You specify the technology file, physical libraries, and logic libraries by using the `search_path` and `link_library` variables, and then you use the `create_lib` or `set_ref_libs` command to assemble the cell libraries.

During library configuration,

- The Fusion Compiler tool automatically calls the Library Manager tool without user intervention to generate cell libraries, as shown in the following figure:



- The tool saves the generated cell libraries to disk and adds them to the reference library list of the design library.
- These cell libraries are the same as when the cell libraries are created during library preparation in the Library Manager tool.

For more information, see the Configuring Cell Libraries topic in the *Fusion Compiler Data Model User Guide*.

Restricting Library Cell Usage

By default, the Fusion Compiler tool can use all of the library cells available in the cell libraries when performing optimization or clock tree synthesis on the block. To restrict the cell usage, use the `set_lib_cell_purpose` command after the block is in memory. The command has a block scope and is persistent across tool sessions. The specified settings are restored when the block is reopened.

To specify how the tool can or cannot use library cells, use the `-include` and `-exclude` options with the `set_lib_cell_purpose` command. Both of these options accept one or more of the following values: `all`, `cts`, `hold`, `optimization`, `none`.

- The `-include` option sets the `included_purposes` attribute on the specified library cells.
- The `-exclude` option sets the `excluded_purposes` attribute on the specified library cells.

Note:

If a library cell has a `dont_use` attribute, it is excluded from all uses, which is the same as if you specified `set_lib_cell_purpose -include none` for that cell.

When the tool performs optimization, which includes setup, hold, and logical DRC fixing, it can use library cells that have an included purpose of `optimization`, `hold`, or both. When the tool performs clock tree synthesis, it can use library cells that have an included purpose of `cts`.

For example, to disallow the use of a set of library cells for all uses, use the following command:

```
fc_shell> set_lib_cell_purpose -include none lib_cells
```

To allow a set of library cells to be used only for clock tree synthesis, use the following commands:

```
fc_shell> set_lib_cell_purpose -include none lib_cells
fc_shell> set_lib_cell_purpose -include cts lib_cells
```

To allow a set of library cells to be used for all uses except clock tree synthesis, use the following command:

```
fc_shell> set_lib_cell_purpose -exclude cts lib_cells
```

To use the asterisk wildcard character (*) to query a collection of cells, you must use the `get_lib_cells` command. However, the `set_lib_cell_purpose` command excludes synthetic modules by default. Therefore, you should use the `set_synlib_dont_use` command to exclude synthetic modules. For example,

```
fc_shell> set_lib_cell_purpose \
      -include none [get_lib_cells */*01_*_AB]
Warning: The 'set_lib_cell_purpose' command cannot be used on synthetic
library cell 'standard:*OPMOD.DW01_ADD_AB.timing'. (NDMUI-511)
0
```

Restricting the Target Libraries Used

By default, the tool can select any library cell from the target library during optimization. In some cases, you might want to restrict the library cells used for clock tree synthesis and optimization. For example, you might want to exclude specific double-height cells from the target library for some design blocks during optimization.

To restrict the library cells used for clock tree synthesis and optimization,

1. Specify the library subset by using the `set_target_library_subset` command.

By default, the library subset restriction applies to

- The top block and all its subblocks

To set it for a specific subblock, use the `-objects` option.

- Both clock and data paths.

To set it for only the clock or data paths, use the `-clock` or `-data` option.

You can further restrict the target library subset setting as follows:

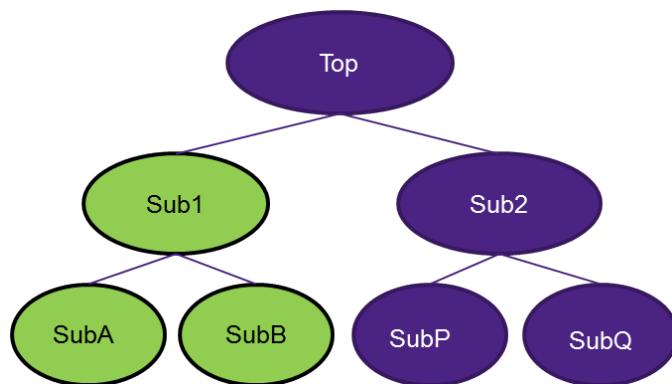
- Specify a list of cells from the libraries that should not be used by using the `-dont_use` option.
- Specify that these libraries cannot be used for any other objects, other than the specified objects, by using the `-only_here` option.

2. Enable the use of the target library subset by setting the `opt.common.enable_target_library_subset_opt` application option to 1.

When you set target library subsets, remember the following points:

- The subset restriction applies to hierarchical cells but not to leaf cells.
- The command enforces the subset restriction on the specified blocks and their subdesigns in the hierarchy, except those subdesigns where a different subset restriction is set.
- A subset specified at a lower level supersedes any subset specified at a higher level.

Figure 7 Logic Hierarchy of Design



For example, assume your design has a logic hierarchy as shown in [Figure 7](#) and you want to implement the following library restrictions during optimization and clock tree synthesis:

- Use only the cells from the library named LVT_lib for the Sub1 block and its subblocks, SubA and SubB.
- Do not use the cells from this library anywhere else in the design.

To do so, use the following settings:

```
fc_shell> set_target_library_subset -objects {top/Sub1} \
    -only_here [get_lib_cells LVT_lib/*] [get_libs LVT_lib]
fc_shell> set_app_options \
    -name opt.common.enable_target_library_subset_opt -value 1
```

In addition to these settings, assume you specify the following setting:

```
fc_shell> set_lib_cell_purpose -include cts \
    {HVT_lib/buf1 HVT_lib/buf2 LVT_lib/buf1 LVT_lib/buf2}
```

Then, when adding buffers on the clock network during clock tree synthesis, the tool uses

- The buf1 and buf2 cells from the LVT_lib library for the block named Sub1 and its subblocks
- The buf1 and buf2 cells from the HVT_lib library for the rest of the design

Reporting Target Library Subsets

To find out which target library subsets have been defined for a top block or hierarchical cell, use the `report_target_library_subset` command.

Reports that are generated by reporting commands, such as `report_cells` and `report_timing`, show the `td` attribute attached to the cells that are specified by the `-dont_use` or `-only_here` option.

Removing Target Library Subsets

To remove a target library subset restriction from a top block or hierarchical cell, use the `remove_target_library_subset` command.

Specifying Library Subset Restrictions

You can restrict the mapping and optimization of sequential cells and instantiated combinational cells to a subset of reference libraries and library cells. To specify one or more library subset restrictions, use the `define_libcell_subset` command followed by the `set_libcell_subset` command.

Note:

The library subset restrictions apply to leaf cells, but not to hierarchical cells.

To set library subset restrictions on leaf cells,

1. Define a subset of library cells by using the `define_libcell_subset` command.

After the library cells are grouped into a subset, they are not available for general mapping during optimization, but for sequential cells and instantiated combinational cells only.

2. Restrict the optimization of sequential and instantiated combinational cells by using the `set_libcell_subset` command.

During optimization, the tool uses the library subset defined in step 1 to optimize sequential cells (both mapped and unmapped) and mapped combinational cells.

In the following example, the `define_libcell_subset` command groups the SDFLOP1 and SDFLOP2 library cells into a subset called `special_flops`, and then the `set_libcell_subset` command restricts the mapping of the LEAF1 leaf cell to the `special_flops` library subset.

```
fc_shell> define_libcell_subset \
    -libcell_list "SDFLOP1 SDFLOP2" -family_name special_flops
fc_shell> set_libcell_subset \
    -object_list "HIER1/LEAF1" -family_name special_flops
```

Reporting Library Subset Restrictions

To report library subsets specified for sequential cells and instantiated combinational cells, use the `report_libcell_subset` command.

Removing Library Subset Restrictions

To remove library subsets, use the `remove_libcell_subset` command.

Using Pin Access Checker Utility

The pin access checker (PAC) utility helps you to identify the pin accessibility issues in the library development stage. This helps you to enhance the cell layout as you cannot modify the cell layout later, after the library is finalized and released to the Place and Route group.

The routing DRCs identify the pin access issues after routing. When the cells are abutted or placed too close, two or multiple adjacent pins compete for common routing resources and causes routing DRCs.

The benefits of PAC utility are listed as follows:

- Produces a place and route database with various permutations of abutted cell placement.
- Generates a place and route database and assesses the ability to place and pin accessibility of the library created based on the reports.

The PAC performs various checks listed as follows:

- Placement of cells
- Number of accessible tracks for each library cell's pins
- Legalization check after placement
- Design rule check after routing
- All the earlier listed checks with PG routes

You can use the `create_pin_check_lib` command to create or open a new library based on the input technology and reference libraries. PAC creates and stores blocks in this library. You must provide minimum three inputs by using the following application options with the `create_pin_check_lib` command:

- Technology file: Use the `-technology` option to provide the technology file information.
- Reference library: Use the `-ref_libs` option to provide the reference library information.
- Routing direction for metal layers: Use the `set_attribute [get_layers Mx] routing_direction horizontal/vertical` command to define routing direction in Tcl script. You must specify the name of the script using the `pin_check.place.preplace_option_file` application option.

To enable the pin access check utility, you can use the `check_libcell_pin_access` command. You can use this command and the `pin_check.*` application option to enable the following pin access checks:

- Legalization
- Router
- Automatic derivation of cell spacing rule

Use the `-mode` option with the `check_libcell_pin_access` command to perform multiple checks by specifying different modes.

```
create_pin_check_lib myTest.nlib -ref_libs ../lib/refLib.ndm
                     -technology ../tf/refTech.tf
```

```
set_app_options -as_user_default -name
pin_check.place.preplace_option_file -value

check_libcell_pin_access -mode analyze_lib_cell
check_libcell_pin_access -mode analyze_lib_pin
check_libcell_pin_access -mode design_under_test
```

If you have been using the earlier Tcl based version, you can translate your old scripts into new Fusion Compiler commands by using the `translate_pin_check_ui` command.

```
translate_pin_check_ui -from old_script.tcl -to new_script.tcl
```

Analyzing Libraries

The Fusion Compiler tool provides the following library analysis capabilities, which can help achieve the performance, power, and area goals of your design:

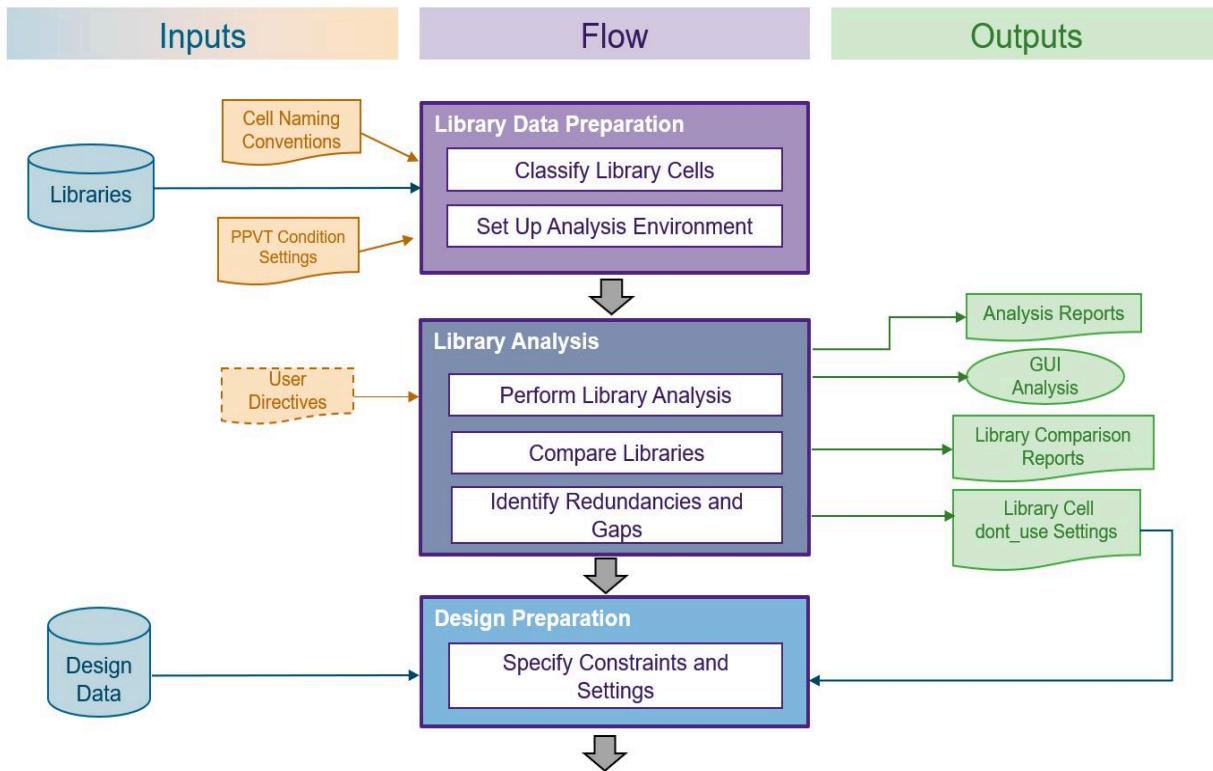
- Identify redundant, outlier, and equivalent cells in library cell families, which you can exclude from optimization during the subsequent steps in the implementation flow.
- Identify gaps in library cell families, which you can provide as feedback to your library vendor with the goal of improving the reference libraries.
- Compare the delay, power, area, and other characteristics of equivalent cells in different libraries or different versions of the same library, which can help you identify the appropriate libraries to use for design implementation.

A family of library cells is a collection of cells that share some common properties. The tool can identify a family of library cells based on

- User-identified library cells properties, such function, threshold voltage, and so on
- User-defined library cells lexical attributes and settings
- User-specified custom library cell families

The following figure shows the high-level flow for performing library analysis.

Figure 8 Library Analysis Flow



For more information about the different types of library analysis flows, see the following topics:

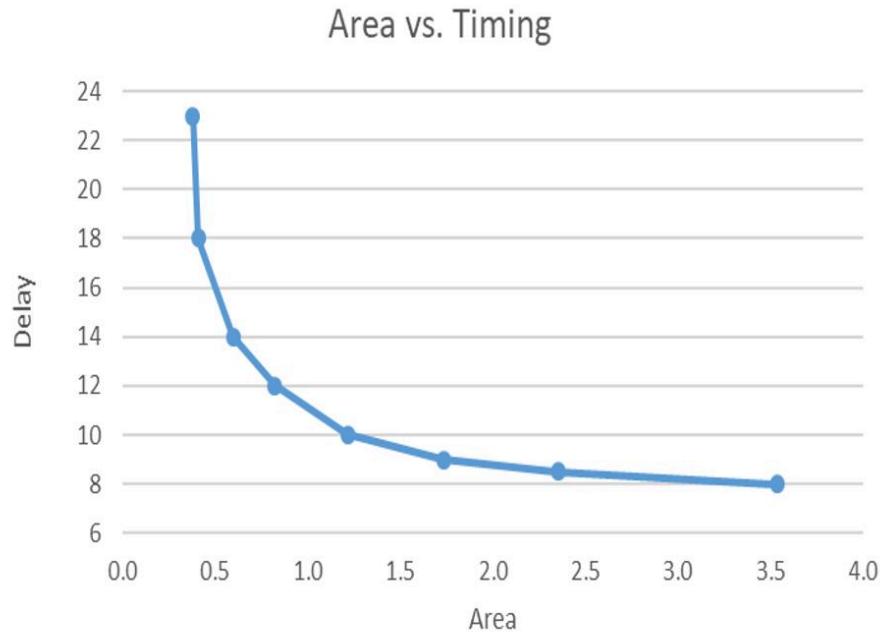
- [Introduction to Redundancies, Outliers, Equivalences, and Gaps in Library Cell Families](#)
- [Identifying Redundancies, Outliers, Equivalences, and Gaps in Library Cell Families](#)
- [Comparing Libraries](#)

Introduction to Redundancies, Outliers, Equivalences, and Gaps in Library Cell Families

When a family of cells with the same function is ordered in terms of increasing drive strength, specific cell property values should be monotonically increasing or decreasing. For example, when a family is ordered in terms of increasing drive strength, the area of cells should be monotonically increasing, power should be increasing, and delay (for the same output load and input slew) should be decreasing.

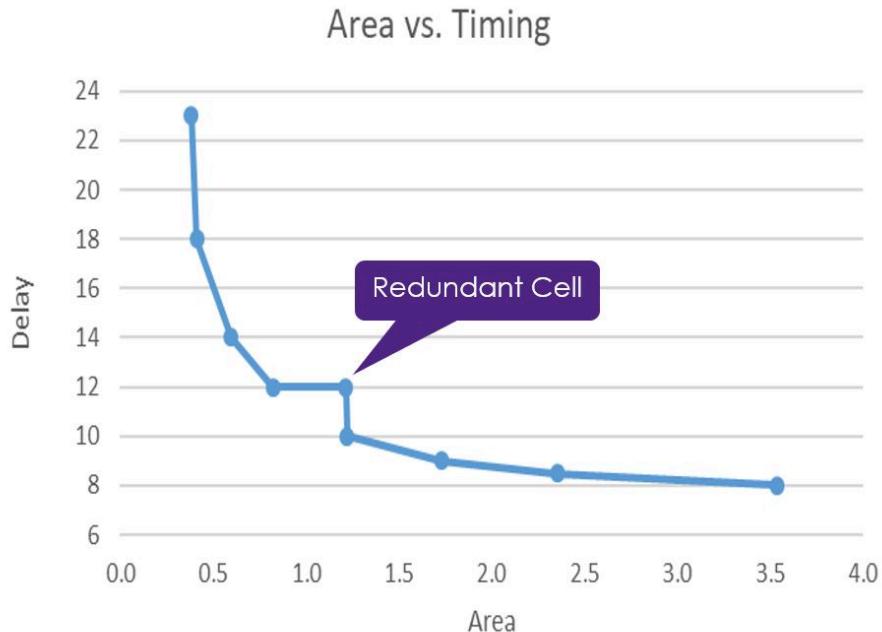
The following figure shows a monotonic distribution of cell area versus timing for a family of cells with the same function.

Figure 9 Monotonic Distribution in Library Cell Family



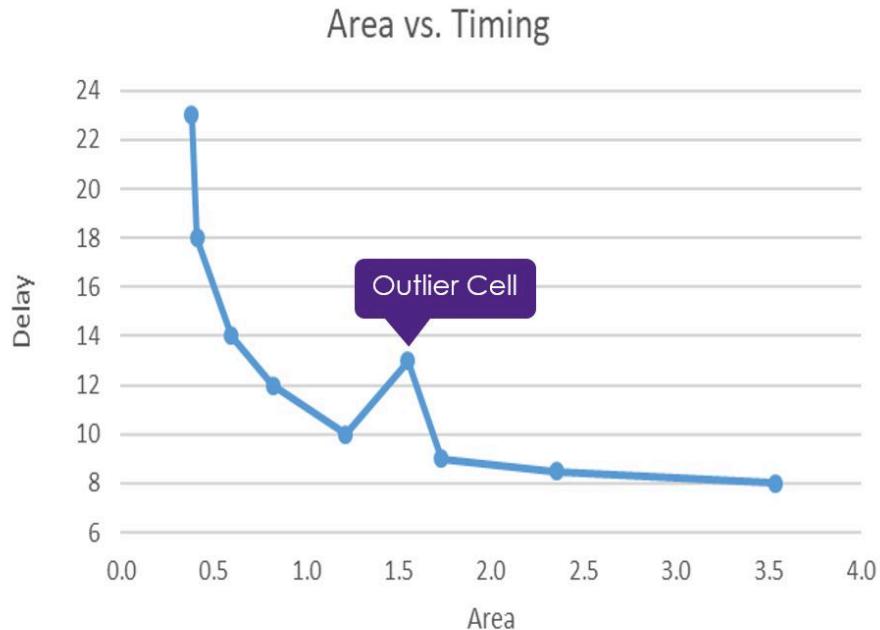
A cell is considered redundant if all its metrics, such as area, delay, power, and so on, are equal or worse than another cell, as shown in the following plot of cell area versus timing for a family of cells with the same function.

Figure 10 Redundant Cell in Library Cell Family



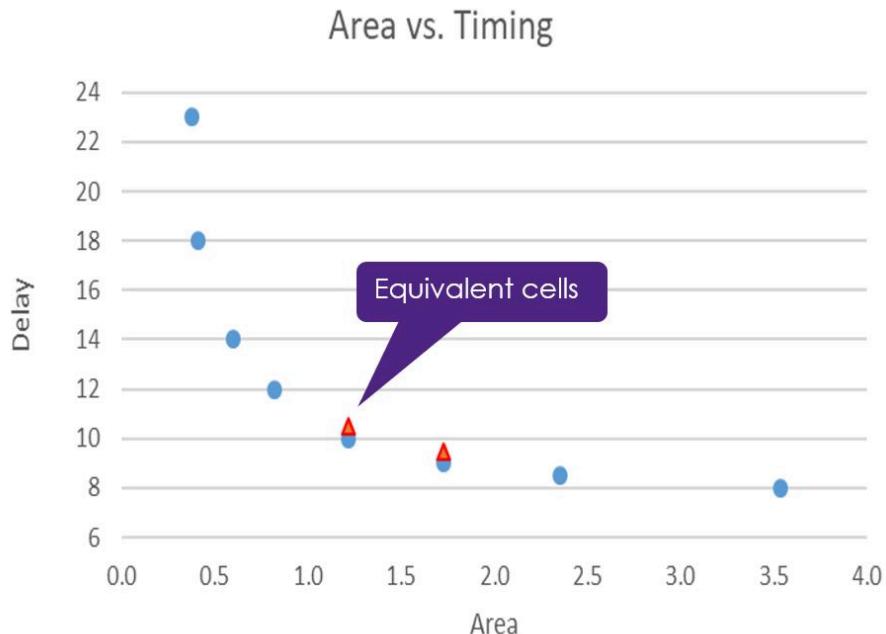
A cell is considered an outlier if its metrics has a very large deviation from the expected values, as shown in the following plot of cell area versus timing for a family of cells with the same function.

Figure 11 *Outlier Cell in Library Cell Family*



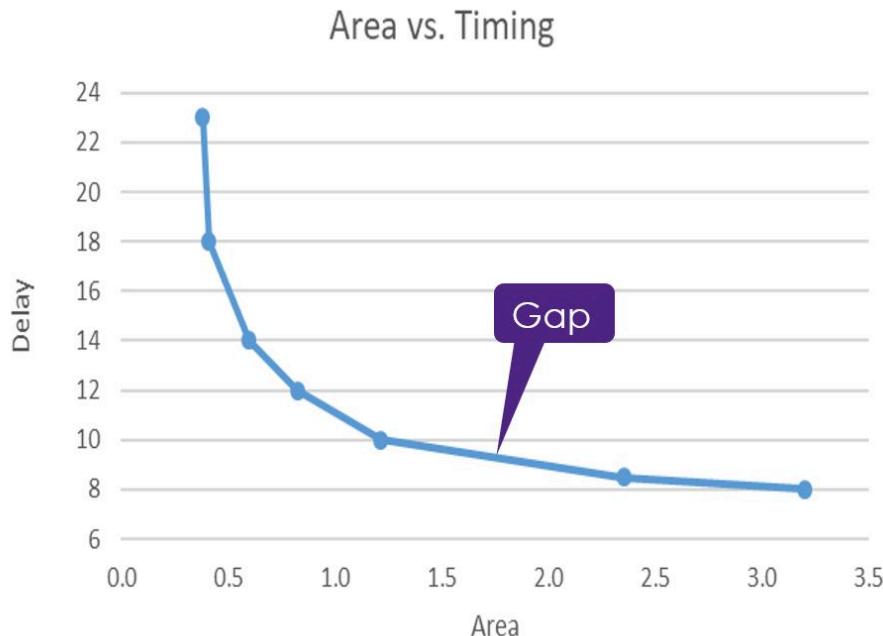
Cells are considered equivalent if all their properties are the same for all process, voltage, and temperature (PVT) values, as shown in the following plot of cell area versus timing for a family of cells with the same function.

Figure 12 *Equivalent Cells in Library Cell Family*



A gap is when there is a large difference between two successive drive strengths, as shown in the following plot of cell area versus timing for a family of cells with the same function.

Figure 13 Gap in Library Cell Family



Identifying Redundancies, Outliers, Equivalences, and Gaps in Library Cell Families

To identify redundancies, outliers, equivalences, and gaps in cell families, perform the following steps:

1. Create a collection of the libraries you want to analyze, which is known as a libset, by using the `create_libset` command, as shown in the following example:

```
fc_shell> create_libset -name myLibset -libs {abcd1 abcd2}
```

When you perform library analysis, you can specify the libset name, instead of specifying a list of libraries. In addition, the tool uses the libset name, instead of using a list of the library names, when it generates reports, plots, and so on during analysis.

You can include a cell library in more than one libset.

2. Set up the library analysis environment by using the `create_environment` command.

The `create_environment` command associates a predefined libset with a process label, process number, voltage, and temperature, which is then used for analyzing the libraries.

You can specify the process label, process number, voltage, and temperature of the analysis environment by using one of the following methods:

- Specify the process label, process number, voltage, and temperature values by using the `-ppvt` option, as shown in the following example:

```
fc_shell> create_environment -name env1 -libset myLibset \
-ppvt {{SS1p, 1.0, 0.6, 125}}
```

- Specify a predefined operating condition from the reference library by using the `-op_conds` option, as shown in the following example:

```
fc_shell> create_environment -name env2 -libset myLibset \
-op_conds SS1p6125
```

- Specify corners from which to infer the process label, process number, voltage, and temperature values by using the `-corners` option, as shown in the following example:

```
fc_shell> create_environment -name env3 -libset myLibset \
-corners nworst
```

- Specify that the tool infers the process label, process number, voltage, and temperature values from the scenario and UPF associated with the current design by using the `-auto_infer_design` option, as shown in the following example:

```
fc_shell> create_environment -name env4 -libset myLibset \
-auto_infer_design
```

3. (Optional) Specify a naming convention for library cells and perform lexical classification by using lexical attributes.

A library cell naming convention can help the tool identify cell families in the subsequent steps of the flow.

For example, assume a cell in your library is named AND2D4WSULVT. This name is the concatenation of the AND2, D4, WS, and ULVT string, which represent cell characteristics that are associated to a lexical attribute, as shown in the following table:

Table 2 Lexical Attributes for the Naming Convention of an Example Library

Column	Lexical Attribute	Allowed Pattern	Values for Example Cell
1	lexical_cell_prefix_name	[a-zA-Z0-9]*	AND2

Column	Lexical Attribute	Allowed Pattern	Values for Example Cell
2	lexical_drive1_name	D[0-9]*	D4
3	lexical_well_substrate_bias_architecture	WS	WS
4	lexical_device_threshold_voltage	ULVT SVT LVT MVT	ULVT

You can specify the naming convention for this library by using the `set_lib_cell_naming_convention` command, as shown in the following example:

```
fc_shell> set_lib_cell_naming_convention -library $lib \
    -column 1 -pattern {[a-zA-Z0-9}*} -attribute
    "lexical_cell_prefix_name"
fc_shell> set_lib_cell_naming_convention -library $lib \
    -column 2 -pattern {D[0-9]*} -attribute "lexical_drive1_name"
fc_shell> set_lib_cell_naming_convention -library $lib \
    -column 3 -pattern {WS} -attribute
    "lexical_well_substrate_bias_architecture"
fc_shell> set_lib_cell_naming_convention -library $lib \
    -column 4 -pattern {ULVT|SVT|LVT|MVT} -attribute
    "lexical_device_threshold_voltage"
```

Note:

For a list of all the predefined lexical attributes, see the man page for the `set_lib_cell_naming_convention` command.

After you specify the naming convention, perform lexical classification by using the `classify_lib_cell_attributes` command, as shown in the following example:

```
fc_shell> classify_lib_cell_attributes -libset myLibset
```

To report the lexical classification results, use the `report_lib_cell_classification_attributes` command.

To report the library cells which are not lexical classified under the given naming convention, use the `report_lexical_unclassified_lib_cells` command. If there are lexically unclassified cells, specify additional naming conventions and rerun lexical classification.

4. (Optional) Exclude specific cells from library analysis by using the `set_use_for_library_analysis` command.

By default, the tool excludes physical-only cells during library analysis. However, you can exclude additional cells, such as library cells with a `dont_use` setting, as shown in the following example:

```
fc_shell> set exclude_cells [filter_collection [get_lib_cells $libs/*]
\ "dont_use == true"]
fc_shell> set_use_for_library_analysis $exclude_cells false
```

To report the cells that are excluded, use the `report_lexical_ignored_lib_cells` command.

5. Identify all library cell families by using the `identify_lib_cell_families` command.

To report the tool-identified library cell families, use the `report_lib_cell_families -all_initial` command.

To manually define library cell families, use the `define_lib_cell_family` command. To report these user-defined cell families, use the `report_lib_cell_families -all_userdef` command.

6. Analyze the libraries by using the `run_library_analysis` command, as shown in the following example:

```
fc_shell> run_library_analysis -name myLibset_run1 -environment env1
```

The name of the environment you specify with the `-environment` option must correspond to an environment you previously created by using the `create_environment` command.

7. Identify redundancies in library cell families by using the `find_library_redundancies` command, as shown in the following example:

```
fc_shell> find_library_redundancies -analysis myLibset_run1
```

The name of the analysis you specify with the `-analysis` option must correspond to the name of an analysis you previously performed by using the `run_library_analysis` command.

In addition to redundant cells, the `find_library_redundancies` command also identifies outlier and equivalent cells in library families.

8. Identify gaps in library cell families by using the `find_library_gaps` command, as shown in the following example:

```
fc_shell> find_library_gaps -analysis myLibset_run1
```

The name of the analysis you specify with the `-analysis` option must correspond to the name of an analysis you previously performed by using the `run_library_analysis` command.

9. Report the library cell gaps, redundancies, outliers, and equivalence identified during library analysis by using the `report_library_analysis` command, as shown in the following example:

```
fc_shell> report_library_analysis -analysis myLibset_run1 \
    -analysis_type redundancy -detail
fc_shell> report_library_analysis -analysis myLibset_run1 \
    -analysis_type outlier -detail
fc_shell> report_library_analysis -analysis myLibset_run1 \
    -analysis_type equivalent -detail
fc_shell> report_library_analysis -analysis myLibset_run1 \
    -analysis_type gap -detail
```

10. Generate plots by using the `gui_plot_lib_cells_attributes` command, as shown in the following example:

```
fc_shell> gui_plot_lib_cells_attributes -libset myLibset \
    -lib_cell_type all -x_attrib_name leakage -y_attrib_name
    avgsl_fall_delay \
    -chart_type scatter
```

Comparing Libraries

To compare libraries, perform the following steps:

1. Create a collection of the libraries you want to analyze, which is known as a libset, by using the `create_libset` command, as shown in the following example:

```
fc_shell> create_libset -name myLibset -libs {abcd1.db abcd2.db}
```

When you perform library analysis, you can specify the libset name, instead of specifying a list of libraries. In addition, the tool uses the libset name, instead of using a list of the library names, when it generates reports, plots, and so on during analysis.

You can include a cell library in more than one libset.

2. Set up the library analysis environment by using the `create_environment` command.

The `create_environment` command associates a predefined libset with a process label, process number, voltage, and temperature, which is then used for analyzing the libraries.

You can specify the process label, process number, voltage, and temperature of the analysis environment by using one of the following methods:

- Specify the process label, process number, voltage, and temperature values by using the `-ppvt` option, as shown in the following example:

```
fc_shell> create_environment -name env1 -libset myLibset \
    -ppvt {{SS1p, 1.0, 0.6, 125}}
```

- Specify a predefined operating condition from the reference library by using the `-op_conds` option, as shown in the following example:

```
fc_shell> create_environment -name env2 -libset myLibset \
    -op_conds SS1p6125
```

- Specify corners from which to infer the process label, process number, voltage, and temperature values by using the `-corners` option, as shown in the following example:

```
fc_shell> create_environment -name env3 -libset myLibset \
    -corners nworst
```

- Specify that the tool infers the process label, process number, voltage, and temperature values from the scenario and UPF associated with the current design by using the `-auto_infer_design` option, as shown in the following example:

```
fc_shell> create_environment -name env4 -libset myLibset \
    -auto_infer_design
```

3. (Optional) Specify a naming convention for library cells and perform lexical classification by using lexical attributes.

A library cell naming convention can help the tool identify cell families in the subsequent steps of the flow.

For example, assume a cell in your library is named AND2D4WSULVT. This name is the concatenation of the AND2, D4, WS, and ULVT strings, which represent cell characteristics that are associated to a lexical attribute as shown in the following table:

Table 3 Lexical Attributes for the Naming Convention of an Example Library

Column	Lexical Attribute	Allowed Pattern	Values for Example Cell
1	lexical_cell_prefix_name	[a-zA-Z0-9]*	AND2
2	lexical_drive1_name	D[0-9]*	D4
3	lexical_well_substrate_bias_arch_itecure	WS	WS

Column	Lexical Attribute	Allowed Pattern	Values for Example Cell
4	lexical_device_threshold_voltage	ULVT SVT LVT MVT	ULVT

You can specify the naming convention for this library by using the `set_lib_cell_naming_convention` command, as shown in the following example:

```
fc_shell> set_lib_cell_naming_convention -library $lib \
    -column 1 -pattern {[a-zA-Z0-9]*} -attribute
    "lexical_cell_prefix_name"
fc_shell> set_lib_cell_naming_convention -library $lib \
    -column 2 -pattern {D[0-9]*} -attribute "lexical_drive1_name"
fc_shell> set_lib_cell_naming_convention -library $lib \
    -column 3 -pattern {WS} -attribute
    "lexical_well_substrate_bias_architecture"
fc_shell> set_lib_cell_naming_convention -library $lib \
    -column 4 -pattern {ULVT|SVT|LVT|MVT} -attribute
    "lexical_device_threshold_voltage"
```

Note:

For a list of all the predefined lexical attributes, see the man page for the `set_lib_cell_naming_convention` command.

After you specify the naming convention, perform lexical classification, by using the `classify_lib_cell_attributes` command as shown in the following example:

```
fc_shell> classify_lib_cell_attributes -libset myLibset
```

To report the lexical classification results, use the `report_lib_cell_classification_attributes` command.

To report the library cells which are not lexical classified under the given naming convention, use the `report_lexical_unclassified_lib_cells` command. If there are lexically unclassified cells, specifies addition naming conventions and rerun lexical classification.

4. (Optional) Exclude specific cells from library analysis by using the `set_use_for_library_analysis` command.

By default, the tool excludes physical-only cells during library analysis. However, you can exclude additional cells, such as library cells with a `dont_use` setting, as shown in the following example:

```
fc_shell> set exclude_cells [filter_collection [get_lib_cells $libs/*]
\ "dont_use == true"]
fc_shell> set_use_for_library_analysis $exclude_cells false
```

To report the cells that are excluded, use the `report_lexical_ignored_lib_cells` command.

5. Identify all library cell families by using the `identify_lib_cell_families` command.

To report the tool-identified library cell families, use the `report_lib_cell_families -all_initial` command.

To manually define library cell families, use the `define_lib_cell_family` command, and to report these user-defined cell families, use the `report_lib_cell_families -all_userdef` command.

6. Analyze the libraries by using the `run_library_analysis` command, as shown in the following example:

```
fc_shell> run_library_analysis -name myLibset_run1 -environment env1
```

The name of the environment you specify with the `-environment` option must correspond to an environment you previously created by using the `create_environment` command.

7. Compare libraries by using the `compare_libraries` command.

When doing so, you must specify the two libraries you want to compare by using the `-ref_lib` and `-target_lib` options. In addition, you must specify the name of the library analysis runs, performed by using the `run_library_analysis` command in the previous step, for each of these libraries. To do so use the `-ref_lib_analysis` and `-target_lib_analysis` options, as shown in the following example:

```
fc_shell> compare_libraries -ref_lib $rvt_lib -ref_lib_analysis sa \
-target_lib $lvt_lib -target_lib_analysis sa -verbose -plot
```

When comparing libraries, you might encounter the following situations related to name matching between the cells of the reference and target libraries:

- The cell names in the reference and target libraries can be an exact match, as in the case when it is two different versions of the same library. If so, additional lexical classification is not required.
- The reference and target libraries can have the same naming convention, except less lexical attributes in one library.

For example, assume a cell named ND2D1SVT in the reference library matches a cell named ND2D1 in the target library. In this case, the target library cell names do not contain a `lexical_device_threshold_voltage` attribute, as shown in the following table.

Table 4 Reference and Target Library Lexical Attribute Differences

Lexical Attribute	Reference Library	Target Library
<code>lexical_cell_prefix_name</code>	ND2	ND2
<code>lexical_drive1_name</code>	D1	D1
<code>lexical_device_threshold_voltage</code>	SVT	-

In this situation, you should ignore the `lexical_device_threshold_voltage` attribute when comparing the two libraries. To do so, use the following application option settings before you run the `compare_libraries` command:

```
fc_shell> set_app_options -name libra.complib.match_full_cell_name
           -value false
fc_shell> set_app_options -name
           libra.complib.match_lexical_device_threshold_voltage -value false
fc_shell> compare_libraries -ref_lib $rvt_lib -ref_lib_analysis sa \
           -target_lib $lvt_lib -target_lib_analysis sa -verbose -plot
```

Note:

For a complete list of application options you can use to control name matching during library comparison, see the man page for the `compare_libraries` command.

- The reference and target libraries can have a similar naming convention, but the lexical attributes values might not identical.

For example, assume a cell named ND2D1SVT in the reference library matches a cell named nand2_svth_x1 in the target library. In this case, the reference and target libraries have the same lexical attributes, which have similar but not the same values, as shown in the following table.

Table 5 Reference and Target Library Lexical Attribute Differences

Lexical Attribute	Reference Library	Target Library
<code>lexical_cell_prefix_name</code>	ND2	nand2
<code>lexical_drive1_name</code>	D1	x1

Lexical Attribute	Reference Library	Target Library
lexical_device_threshold_voltage	SVT	svth

In this situation, you must create a lexical attribute mapping file that contains the following information:

```
lexical_cell_prefix_name      ND2    nand2
lexical_drive1_name          D1     x1
lexical_device_threshold_voltage  SVT    svth
```

Then, when you run the `compare_libraries` command, you must specify this lexical attribute mapping file by using the `-lex_attr_mapping` option, as shown in the following example:

```
fc_shell> set_app_options -name libra.complib.match_full_cell_name
           -value false
fc_shell> compare_libraries -ref_lib $rvt_lib -ref_lib_analysis sa \
           -target_lib $lvt_lib -target_lib_analysis sa \
           -lex_attr_mapping attribute_map_file -verbose -plot
```

- The reference and target library naming convention is completely different.

In this situation, you must create a custom mapping file that contains the cell pairs you want to compare, as shown in the following example:

```
INVD2BWP      INV1D1BWP
INVD2BWP      BUFFD4BWP
ND2D2BWP      ND2D4BWP
DFQD4BWP      SDFQD4BWP
```

Then, when you run the `compare_libraries` command, you must specify this mapping file, by using the `-custom_pairs` option, as shown in the following example:

```
fc_shell> set_app_options -name libra.complib.match_full_cell_name
           -value false
fc_shell> compare_libraries -ref_lib $rvt_lib -ref_lib_analysis sa \
           -target_lib $lvt_lib -target_lib_analysis sa \
           -custom_pairs custom_map_file -verbose -plot
```

Reading the Design

Before you read a design, you must create or open the design library associated with the design, as described in [Setting Up Libraries](#).

The tool can read RTL designs in SystemVerilog, Verilog, or VHDL and gate-level netlists in Verilog format.

Use the following methods to read designs:

- [Reading RTL Files in SystemVerilog, Verilog, or VHDL Format](#)
- [Reading Designs Using the -autoread Option](#)
- [Reading Designs Using the VCS Command-line Options](#)
- [Reading Verilog Gate-Level Netlist Files](#)
- [Reading Mixed Gate-Level Netlists and RTL Files](#)

By default, when the tool reads the RTL or gate-level netlist files, it creates a block in the current design library and increments its open count. The tool determines the top-level module of the block by identifying the module that is not instantiated by any other modules in the specified files and uses the top-level module name as the block name.

Use the following commands to work with blocks:

- `create_block`: Creates a block
- `open_block`: Opens an existing block
- `current_block`: Sets or reports the current block
- `save_block`: Saves a block
- `close_blocks`: Closes a block

See Also

- [Blocks](#)
- [Working With Design Libraries](#)

Reading RTL Files in SystemVerilog, Verilog, or VHDL Format

Use the `analyze` and `elaborate` commands followed by the `set_top_module` command.

The `analyze` command automatically creates HDL template libraries on disk based on the name of the library provided by the `-hdl_library` option. The location of the HDL template libraries is controlled by the `hdlin.hdl_library.default dirname` application option. When the `-hdl_library` option is not specified, the default library name is WORK. To specify a different default library name, use the `hdlin.hdl_library.default name` application option.

Note:

All of the application options that control the RTL reading behavior have the `hdlin` prefix.

The `elaborate` command builds the module specified without linking the rest of the design. Design linking can be performed only after the entire design is in memory, so linking is not performed by the `elaborate` command. This allows multiple `elaborate` commands to be run before performing the single linking of the entire design. The top-level module must be one of the modules that is elaborated.

Linking of the design and setting the top-level module is done using the `set_top_module` command. The top-level module is given as an argument to the `set_top_module` command. The top-level module must be a module that was previously elaborated with the `elaborate` command. The `set_top_module` command sets the specified module to be the top-level design, links the entire design, and creates a single block to be used for the remainder of the synthesis flow.

The following script reads VHDL files using template libraries and creates a block called `top`. You do not need to specify the location of the template libraries on disk, which is automatically created by the `analyze` command based on the `-hdl_library` option.

```
analyze -format vhdl -hdl_library BOT_HDL_LIB bot.vhd
analyze -format vhdl -hdl_library MID_HDL_LIB mid.vhd
analyze -format vhdl top.vhd
elaborate top
set_top_module top
```

If the top-level design is analyzed to an HDL template library other than the default library, you should provide the HDL template library name for the top-level design using the `-hdl_library` option. For example,

```
analyze -format vhdl -hdl_library BOT_HDL_LIB bot.vhd
analyze -format vhdl -hdl_library MID_HDL_LIB mid.vhd
analyze -format vhdl -hdl_library TOP_HDL_LIB top.vhd
elaborate -hdl_library TOP_HDL_LIB top
set_top_module top
```

You can optionally specify the location of the HDL template libraries on disk by using the `define_hdl_library` command. For example,

```
define_hdl_library BOT_HDL_LIB -path ./TEMPLATES/BOT_HDL_LIB
define_hdl_library MID_HDL_LIB -path ./TEMPLATES/MID_HDL_LIB
define_hdl_library WORK -path ./TEMPLATES/WORK
analyze -format vhdl -hdl_library BOT_HDL_LIB bot.vhd
analyze -format vhdl -hdl_library MID_HDL_LIB mid.vhd
analyze -format vhdl top.vhd
elaborate top
set_top_module top
```

Reading Designs Using the **-autoread** Option

To enable the tool to automatically read a list of RTL files, specify the **-autoread** option with the `analyze` command, as shown in the following example. When this option is specified, the RTL language used to analyze the RTL files is automatically determined by the file name extension. To control the file name extensions, use the following application options: `hdlin.autoread.verilog_extensions`, `hdlin.autoread.sverilog_extensions`, `hdlin.autoread.vhdl_extensions`, and `hdlin.autoread.exclude_extensions`.

```
set DESIGN_NAME top
analyze -autoread -top ${DESIGN_NAME} ${RTL_FILE_LIST}
elaborate ${DESIGN_NAME}
elaborate -hdl_library TOP_HDL_LIB top
set_top_module ${DESIGN_NAME}
```

Reading Designs Using the VCS Command-line Options

The `analyze` command with the VCS command-line options provides compatibility with VCS simulation scripts and makes reading large designs easier. When you use the VCS command-line options, the tool automatically resolves references for instantiated designs by searching the referenced designs in the specified libraries and then loading these referenced designs.

To read designs containing many HDL source files and libraries, specify the **-vcs** option with the `analyze` command. You must enclose the VCS command-line options in double quotation marks.

For example,

```
analyze -vcs "-verilog -y mylibdir1 +libext+.v -v myfile1 \
+incdir+myincludedir1 -f mycmdfile2" top.v
elaborate ${DESIGN_NAME}
set_top_module ${DESIGN_NAME}
```

To read SystemVerilog files with a specified file extension and Verilog files in one `analyze` command, use the **-vcs "+systemverilogext+ext"** option. When you do so, the files must not contain any Verilog 2001 styles.

For example, the following command analyzes SystemVerilog files with the `.sv` file extension and Verilog files:

```
analyze -format verilog -vcs "-f F +systemverilogext+.sv
elaborate ${DESIGN_NAME}
set_top_module ${DESIGN_NAME}
```

Reading Verilog Gate-Level Netlist Files

Use the `read_verilog` command to read Verilog netlist files using a specialized Verilog netlist reader. This command cannot be used to read Verilog RTL files. Only the `analyze` and `elaborate` commands should be used to read RTL files. Use the `set_top_module` command after reading the netlist to specify the top-level module and link the design to create the block.

```
read_verilog top.netlist.v
set_top_module top
```

When the file has multiple independent modules, use the `-top` option to identify the top-level module. For example,

```
read_verilog -top top2 netlists.v
set_top_module top2
```

To specify a new block name other than `top_module_name.design`, use the `-design` option. For example,

```
read_verilog -top my_top -design desA ../my_data/my_des.v
set_top_module my_top
```

Reading Mixed Gate-Level Netlists and RTL Files

You can read a combination of Verilog netlists and RTL files. Use the `read_verilog` command to read the Verilog netlists. Use the `analyze` and `elaborate` commands to read the RTL files. After all the netlist files and RTL files have been read into the tool, use the `set_top_module` command to link the designs to create the block for synthesis.

The following example shows how you can read a mix of netlist and RTL files. As shown in this example, when you elaborate the top-level module with parameters, the top-level module name also contains the parameterized module name. You can control the naming style for parameterized modules using the following application options:

`hdlin.naming.template_naming_style`, `hdlin.naming.template_parameter_style`, and `hdlin.naming.template_separator_style`.

```
# Read netlist files
read_verilog NETLIST/bot.v
# Analyze design files
analyze -format sverilog -hdl_library MID RTL/mid.sv
analyze -format sverilog                               RTL/top.svd
# Elaborate top design module
elaborate -parameters "N=8,M=3" top
# Set the top level module to resolve references
set_top_module top_N8_M3
```

Embedding Tcl Commands in RTL Code

To embed Tcl command in RTL code, use the `fc_tcl_script_begin` and `fc_tcl_script_end` directives. The following Tcl commands can be embedded in an RTL design:

- `set_attribute`
- `set_ungroup`
- `set_size_only`
- `set_dont_touch`
- `set_dont_retime`
- `set_implementation`
- `set_optimize_registers`
- `get_modules`
- `get_cells`
- `get_pins`
- `get_nets`
- `get_ports`

The following RTL example contains an embedded Tcl command that applies a `size_only` setting on all the registers with the `iPort1.dout_reg` name prefix:

```
module bot ( myIntf1.datIn iPort1 );
    always @(posedge iPort1.clk or negedge iPort1.rst)
    begin
        if (iPort1.rst == 1'b0)
        begin
            iPort1.dout <= '0;
        end else begin
            iPort1.dout <= iPort1.iData1 + iPort1.iData2;
        end
    end
// synopsys fc_tcl_script_begin
// set_size_only [get_cells iPort1.dout_reg*]
// synopsys fc_tcl_script_end
endmodule
```

Mitigating Design Mismatches

During early design development, frequent design updates can cause incomplete or inconsistent data, such as pin mismatches between a block-level design and the reference to it from a top-level design. To enable the tool to successfully link a block even if it has certain types of design mismatch, set a mismatch configuration for the block.

By default, the tool cannot link any block with inconsistent or mismatching data. When you set mismatch configurations, the tool either ignores or fixes several different types of mismatching data and continues the linking process. Blocks linked with mismatching data can be used only for feasibility analysis.

To learn how to mitigate design mismatches, see the *Fusion Compiler Data Model User Guide*.

Importing the Floorplan Information

A floorplan contains physical constraints such as the core area and shape, port locations, macro locations and orientations, and so on, which are required for performing physical synthesis and optimization.

If you have a floorplan for your block, read in the floorplan as a DEF file, as described in [Reading DEF Files](#).

If you do not have a floorplan for your block, you can perform design planning and generate a floorplan as described in the *Fusion Compiler Design Planning User Guide*. During the early stages of design process, you can use automatically-generated floorplan information to run physical synthesis, as described in [Using Automatic Floorplanning](#).

Reading DEF Files

To read the floorplan information from a DEF file, use the `read_def` command.

```
fc_shell> read_def block.def
```

Note:

When possible, use DEF v5.8 or later, as this version supports more types of physical objects and obstructions than previous versions.

By default, the `read_def` command

- Annotates the floorplan information onto the current block

To annotate the information onto a different block, use the `-design` option to specify the block name.

- Preserves the existing floorplan information

In incremental mode,

- The placement area is imported based on the current core area and site rows in the DEF files
- Physical constraints that can have only one value are overwritten by the value from the latest DEF file; for example, port location and macro location are overwritten.
- Physical constraints that can have accumulated values are recomputed; that is, core area can be recomputed based on the existing value and the site row definitions in the latest DEF file. Placement keepouts from different DEF files are accumulated and the final keepout geometry is computed internally during synthesis.

To remove the existing floorplan information before annotating the floorplan information from the DEF file, use the `-no_incremental` option. In this mode, the placement area is imported based on the site rows in the DEF files.

- Uses rule-based name matching for macros and ports

Rule-based name matching automatically resolves name differences by using the tool's intelligent name matching capability. By default, when rule-based name matching is enabled, the following characters are considered equivalent:

- Hierarchical separators { / _ . }

For example, a cell named `a.b_c/d_e` is automatically matched with the string `a/b_c.d/e` in the DEF file.

- Bus notations { [] __ () }

For example, a cell named `a [4] [5]` is automatically matched with the string `a_4__5_` in the DEF file.

To disable rule-based name matching and require exact name matches between the DEF file and the block, set the `file.def.rule_based_name_matching` application option to `false`.

For more information, see “Rule-Based Name Matching” in the *Fusion Compiler Data Model User Guide*.

- Ignores any objects in the DEF file that do not exist in the block, except for PG objects

To allow new non-PG objects to be created from the DEF file, use the `-add_def_only_objects` option to specify the types of objects to create. Specify one or more of the following keywords:

- `cells`

The tool creates the cells that exist only in the DEF file and connects their power and ground pins as defined in the DEF file; it does not connect the signal, clock, or tie pins even if these connections are defined in the DEF file. The tool also does not create new hierarchical cells; any hierarchy specified in the DEF file must already exist in the block.

- `nets`

The tool creates the signal, clock, and tie nets that exist only in the DEF file and connects them to the ports specified in the DEF PINS section; it does not connect the nets to any other ports or pins in the netlist even if these connections are defined in the DEF file. The tool does not create new hierarchical nets; any hierarchy specified in the DEF file must already exist in the block.

- `ports`

The tool creates the signal, clock, and tie ports that exist only in the DEF file and connects them to the nets specified in the DEF PINS section.

- `all`

The tool creates the non-PG cells, nets, and ports that exist only in the DEF file, as if you had specified `cells`, `nets`, and `ports`.

Fixing Site Name Mismatches

If the site names used in the DEF file do not match the site names defined in the technology file, use the `-convert_sites` option to specify the site name mapping. For example, if the DEF file uses a site named CORE, but the technology file defines only a site named unit, use the following command to convert the site names when reading the DEF file:

```
fc_shell> read_def -convert_sites { {CORE unit} } block.def
```

Validating DEF Files

To analyze the input DEF files before annotating the floorplan information on the block, enable check-only mode by using the `-syntax_only` option. The check-only mode provides diagnostic information about the correctness and integrity of the DEF file. The check-only mode does not annotate any floorplan information onto the block.

```
fc_shell> read_def -syntax_only block.def
```

Physical Constraints Extracted From the DEF File

The `read_def` command extracts physical constraint information from DEF files and annotates it on the block. However, only the following physical constraints are extracted and annotated:

- [Placement Area](#)
- [Port Locations](#)
- [Cell Locations](#)
- [Placement Blockages](#)
- [Site Rows](#)
- [Routing Tracks](#)
- [Placement Bounds](#)
- [Routing Blockages](#)
- [Preroutes](#)

To visually inspect the extracted physical constraints, use the layout view in the GUI. All physical constraints extracted from the DEF file are automatically added to the layout view.

Placement Area

Placement area is computed based on the site array information.

Port Locations

For each port with the location specified in the DEF file, the tool sets the location on the corresponding port in the block.

Note:

If the DEF file does not contain port-location information, the tool inherits the port locations from the locations of the pad cells, as described in .

Example 1 DEF Port Location Information

```
PINS 2 ;
    -Out1 + NET Out1 + DIRECTION OUTPUT + USE SIGNAL +
        LAYER M3 (0 0) (4200 200) + PLACED (80875 0) N;
    -Sel0 + NET Sel0 + DIRECTION INPUT + USE SIGNAL +
        LAYER M4 (0 0) (200 200) + PLACED (135920 42475) N;
END PINS
```

Ports with changed names and multiple layers are supported.

Example 2 DEF Port Locations With Changed Names and Multiple Layers

```
PINS 2 ;
  - sys_addr\[23\].extra2 + NET sys_addr[23] + DIRECTION INPUT +USE
  SIGNAL
    + LAYER METAL4 ( 0 0 ) ( 820 5820 ) + FIXED ( 1587825 2744180 ) N ;
    - sys_addr[23] + NET sys_addr[23] + DIRECTION INPUT + USE SIGNAL +
  LAYER
    METAL3 ( 0 0 ) ( 820 5820 ) + FIXED ( 1587825 2744180 ) N ;
END PINS
```

Example 3 DEF Port Orientation Information

```
PINS 1;
  - OUT + NET OUT + DIRECTION INPUT + USE SIGNAL
    + LAYER m4 ( -120 0 ) ( 120 240 )
    + FIXED ( 4557120 1726080 ) S ;
END PINS
```

Cell Locations

For each cell with a location and the FIXED attribute specified in the DEF file, the tool sets the location on the corresponding cell in the block. [Example 4](#) shows DEF macro location and orientation information, where the letters E and W denote east rotation and west rotation respectively.

Example 4 DEF Cell Location Information

```
COMPONENTS 2 ;
  - macro_cell_abx2 + FIXED ( 4350720 8160 ) E ;
  - macro_cell_cdy1 + FIXED ( 4800 8160 ) W ;
END COMPONENTS
```

Placement Blockages

The `read_def` command imports hard, soft, and partial placement blockages defined in the DEF file.

Note:

DEF versions before version 5.7 did not support partial blockages. In addition, if your floorplanning tool creates a DEF file with DEF version 5.6, you need to manually add the `#SNPS_SOFT_BLOCKAGE` pragma to specify a soft blockage, as shown in [Example 7](#).

Example 5 DEF Hard Placement Blockage Information

```
BLOCKAGES 50 ;
...
  - PLACEMENT RECT ( 970460 7500 ) ( 3247660 129940 )
...
END BLOCKAGES
```

Example 6 DEF Version 5.7 Soft Placement Blockage Information

```
BLOCKAGES 50 ;
...
- PLACEMENT + SOFT RECT ( 970460 7500 ) ( 3247660 129940 ) ;
...
END BLOCKAGES
```

Example 7 DEF Version 5.6 Soft Placement Blockage Information

```
BLOCKAGES 50 ;
...
- PLACEMENT RECT ( 970460 7500 ) ( 3247660 129940 ) ; #SNPS_SOFT_BLOCKAGE
...
END BLOCKAGES
```

Example 8 DEF Partial Placement Blockage Information

```
BLOCKAGES 50 ;
...
- PLACEMENT + PARTIAL 80 RECT ( 970460 7500 ) ( 3247660 129940 ) ;
...
END BLOCKAGES
```

Site Rows

Site row information in the DEF file defines the placement area.

Example 9 DEF Site Row Information

```
ROW ROW_0 core 0 0 N DO 838 BY 1 STEP 560 0;
```

Routing Tracks

The track information in the DEF file defines the routing grid for designs based on standard cells. This information can be used during routing, and track support can enhance congestion evaluation and reporting to make it match more closely with the routing results.

Example 10 DEF Routing Track Information

```
TRACKS X 330 DO 457 STEP 660 LAYER METALL1 ;
TRACKS Y 280 DO 540 STEP 560 LAYER METALL1 ;
```

Placement Bounds

If REGIONS defining bounds exist in the DEF file, the `read_def` command imports those placement bounds. Also, if any cells in the related GROUP are attached to the region,

fuzzy cell matching occurs between these cells and the ones in the block.matched cells are attached to the bounds in the following ways:

- If there are regions in the block with the same name as in the DEF, the cells in the related group are attached to the region by the `add_to_bound` command in incremental mode.
- If the region does not exist in the block, it is created with the same name as in the DEF file by applying the `create_bound` command; matched cells in the related group are also attached.

Example 11 DEF Placement Bound Information

```
REGIONS 1 ;
- c20_group ( 201970 40040 ) ( 237914 75984 ) + TYPE FENCE ;
END REGIONS
GROUPS 1 ;
- c20_group
  cell_abc1
  cell_sm1
  cell_sm2
  + SOFT
  + REGION c20_group ;
END GROUPS
```

Routing Blockages

Routing blockages are extracted similar to the way that placement blockages are extracted.

Preroutes

The tool extracts preroutes that are defined in the DEF file.

Example 12 DEF Preroute Information

```
SPECIALNETS 2 ;
- vdd
  + ROUTED METAL3 10000 + SHAPE STRIPE ( 10000 150000 ) ( 50000 * )
  + USE POWER ;
...
END SPECIALNETS
```

Using Automatic Floorplanning

During early design stages, a design often undergoes many iterations due to changes such as adding features and macros; as a result, the floorplan contains limited or no physical information.

Automatic floorplanning can generate high-quality floorplans and create missing physical information early in the design cycle, which allows you to

- Achieve better correlation between the results of RTL synthesis and design implementation
- Identify and fix design issues early in the design cycle
- Generate a better starting point for place and route, eliminating costly iterations

During automatic floorplanning , the `compile_fusion` command performs the following tasks:

- Creates the die, rows, and tracks
- Shapes and places voltage areas
- Places macros
- Places pins and I/Os

By default, the top-level floorplan is created with a core utilization of 0.7. For information about setting constraints for automatic floorplanning, see [Creating Constraints for Auto Floorplanning](#).

Creating Constraints for Auto Floorplanning

Use the commands and application options described in this topic to create constraints for auto floorplanning.

Commands

Use the following two commands to set and report auto floorplanning constraints. These two commands do not affect explicit floorplanning.

- `set_auto_floorplan_constraints`

The command sets constraints, such as utilization of the floorplan to be created during compile. For example,

```
fc_shell> set_auto_floorplan_constraints -core_utilization 0.8
```

- `report_auto_floorplan_constraints`

The command reports the constraints set by the `set_auto_floorplan_constraints` command.

To specify macro placement, pin placement, and voltage area shaping constraints, use the following commands:

- Macro placement constraints

```
set_macro_constraints, create_macro_array, create_keepout_margin
```

- Pin placement constraints

```
set_block_pin_constraints, set_individual_pin_constraints
```

- Voltage area shaping constraints

```
set_shaping_options
```

Application Options

Use these application options to set up auto floorplanning.

- `compile.auto_floorplan.initialize` (**default**: auto)

- auto: Creates missing floorplan information.

The tool exits with an error when inconsistent information is provided.

- true: Always creates the core and boundary.

- false: Never creates the core and boundary; uses only existing information.

The tool exits with an error when encountering missing or inconsistent information.

By default, the following objects are preserved:

- Existing placement of fixed macros

- Existing placement of pins and pads

- Existing shaping of voltage areas

- `compile.auto_floorplan.place_pins` (**default**: unplaced)

`compile.auto_floorplan.place_ios` (**default**: unplaced)

`compile.auto_floorplan.place_hard_macros` (**default**: unplaced)

`compile.auto_floorplan.shape_voltage_areas` (**default**: unshaped)

- all: Always places and shapes objects even if they are fixed.

Unplaced and unshaped objects will always be placed and shaped.

- unfixed: Places and shapes objects that are not fixed.

Use the fixed information from DEF files or a Tcl floorplan, and use the `set_fixed_objects` command to modify. Unplaced and unshaped objects will always be placed and shaped.

- unplaced, unshaped: Never places or shapes objects when they are already placed.

Unplaced and unshaped objects will always be placed and shaped.

- none: Never places or shapes objects even if they are not already placed.

This table summarizes how the tool handles fixed, placed, and unplaced objects for each setting of these four application options during auto floorplanning.

Application option setting	Fixed objects	Placed objects	Unplaced objects
all	Placed and shaped	Placed and shaped	Placed and shaped
unfixed	Kept	Placed and shaped	Placed and shaped
unplaced, unshaped	Kept	Kept	Placed and shaped
none	Kept	Kept	Kept

- `compile.auto_floorplan.keep_bounds` (**default:** auto)
 - auto: Removes existing bounds if either of the following two application options is set to all.

`compile.auto_floorplan.place_hard_macros,`
`compile.auto_floorplan.shape_voltage_areas`
 - true: Keeps existing bounds.
 - false: Removes existing bounds.
- `compile.auto_floorplan.keep_placement_blockages` (**default:** auto)
 - auto: Removes existing placement blockages if either of the following two application options is set to all.

`compile.auto_floorplan.place_hard_macros,`
`compile.auto_floorplan.shape_voltage_areas`
 - true: Keeps existing placement blockages.
 - false: Removes existing placement blockages.

Automatic Floorplanning Example

The following example specifies constraints for automatic floorplanning:

```
# Application options with default settings
# set_app_options -name compile.auto_floorplan.initialize -value auto
# set_app_options -name compile.auto_floorplan.place_pins -value unplaced
# set_app_options -name compile.auto_floorplan.place_ios -value unplaced
# set_app_options -name compile.auto_floorplan.place_hard_macros \
#   -value unplaced
# set_app_options -name compile.auto_floorplan.shape_voltage_areas \
#   -value unshaped

# Constraint settings for automatic floorplanning
set_auto_floorplan_constraints -core_utilization 0.75 -side_ratio {1 2}
set_macro_constraints -preferred_location {0 0} RAM0
set_block_pin_constraints -self -sides {1 3}
set_individual_pin_constraints -ports [get_ports reset] -side 1
set_shaping_options -guard_band_size 2
```

Setting Up Multivoltage Designs

The following topics describe the tasks you need to perform when setting up multivoltage designs:

- [Applying the Multivoltage Power Intent](#)
- [Preparing the Power Network](#)
- [Defining Voltage Areas](#)
- [Inserting Multivoltage Cells](#)
- [Controlling the Placement of Multivoltage Cells](#)
- [Enabling Improved Buffering for Multivoltage Nets](#)
- [Analyzing Multivoltage Information](#)

Applying the Multivoltage Power Intent

To learn about applying the multivoltage power intent, see

- [Loading and Applying UPF Information](#)
- [Using Incomplete UPF Information](#)

- Specifying UPF Constraints for Physical-Only Cells
- Saving UPF Information

Loading and Applying UPF Information

To load the power intent and apply it to a multivoltage design,

1. Read the UPF file by using the `load_upf` command.

```
fc_shell> load_upf block.upf
```

2. If you are using the golden UPF flow and have a name-mapping file, read the file by using the `read_name_map` command.

```
fc_shell> read_name_map block.nmf
```

3. (Optional) Verify the UPF consistency and identify any PG conflicts among the netlist, floorplan, and power intent.

To verify the UPF consistency and identify PG conflicts, use the `resolve_pg_nets -check_only` command. This command identifies any issues and reports the changes that are made to resolve these issues when you commit the power intent. If you prefer to resolve the issues differently, you can use manual editing commands to resolve the issues before running the `commit_upf` command.

4. Commit the power intent by using the `commit_upf` command.

```
fc_shell> commit_upf
```

The `commit_upf` command performs global checks for UPF consistency; resolves PG conflicts among the netlist, floorplan, and UPF specification; and associates power strategies with existing multivoltage cells. For more information about associating power strategies with existing multivoltage cells, see [Associating Power Strategies With Existing Multivoltage Cells](#).

5. Report the associations made for the multivoltage cells by using the `report_mv_path` command.

If the tool failed to associate any multivoltage cells, the command reports the causes for these failures. You must successfully commit the power intent before you continue with the design flow.

Note:

After successfully running the `commit_upf` command, the tool issues an error message if you try to use additional UPF commands, except for the `set_related_supply_net`, `connect_supply_net`, `set_design_attributes`, `set_port_attributes`, `find_objects`, and `set_scope` commands. To modify the power intent after running the `commit_upf` command, you must remove the

existing UPF specification by using the `reset_upf` command and then reload the power intent.

Using Incomplete UPF Information

The Early Data Check Manager allows you to check designs for power and multivoltage issues early in the design cycle. You can configure different error conditions in different ways. The tool provides comprehensive reports about the data checks. The general flow is as follows:

1. Use the `set_early_data_check_policy` command to define the global violation handling policy for data checks.
2. Proceed with your tool flow. The tool detects and responds to violations throughout the flow.
3. Use the `report_early_data_checks` command to obtain a report about all violations or specific data checks.
4. Use the `get_early_data_check_records` command to get a Tcl collection of violations that can be used in other commands.
5. Use the `write_early_data_check_config` command to save the settings in a file for future use.
6. Use the `remove_early_data_check_records` command to clear the data in preparation for another iteration.

For more information about the Early Data Check Manager, see the *Fusion Compiler Data Model User Guide*.

For a list of power and multivoltage data checks, see the *Fusion Compiler Multivoltage User Guide*.

Specifying UPF Constraints for Physical-Only Cells

The UPF constraints for physical-only cells can cause issues when the UPF file is read into other tools that do not support physical-only cells, such as verification tools. Therefore, surround the UPF constraints for physical-only cells with the syntax shown in the following example:

```
if {[info exists snps_handle_physical_only] && \
    $snps_handle_physical_only} {

    /* Supply net connections for filler cells */
    connect_supply_net VDDS -ports [get_pins FILLER*/VDD]

}
```

By default, the `snps_handle_physical_only` variable is set to `true` in the Fusion Compiler tool. Therefore, when you load and commit the UPF file, the constraints are applied to the physical-only cells.

When you save a UPF file with the `save_upf` command, the tool uses the same syntax for the user-specified and tool-derived UPF constraints for physical-only cells. Therefore, these UPF constraints are ignored by any tool that does not have the `snps_handle_physical_only` variable set to `true`.

To create a UPF file that contains commands only for specific types of cells, use the `-include` or `-exclude` option with the `save_upf` command. These options accept arguments such as `diode_cells`, `pad_cells`, and `physical_only_cells`. See the `save_upf` command man page for the complete list of arguments.

Command filtering for specified cell types applies to the `connect_supply_net`, `set_related_supply_net`, `create_power_domain`, and `set_isolation` commands. You can apply command filtering to both block-level and full-chip UPF files.

Saving UPF Information

During physical implementation, the tool updates the power intent of the design. To generate an updated UPF file, use the `save_upf` command. You can use this UPF file in the subsequent steps of the design flow.

The `save_upf` command separates the UPF commands based on the input UPF file, and adds a comment indicating the input UPF file and the time it was loaded.

Assume you load, commit, and save UPF information as shown in the following example script:

```
load_upf top_1.upf
load_upf top_2.upf
load_upf top_3.upf
commit_upf
save_upf top.upf
```

The resulting top.upf file contains the following information:

```
## Start - load_upf top_1.upf on Wed Oct 26 17:03:18 2016
...
##
## End - load_upf
## Start - load_upf top_2.upf on Wed Oct 26 17:03:49 2016
...
##
## End - load_upf
## Start - load_upf top_3.upf on Wed Oct 26 17:04:22 2016
...
##
## End - load_upf
```

Preparing the Power Network

To learn about preparing the power network for physical implementation, see

- [Creating Logical Power and Ground Connections](#)
- [Creating Floating Logical Supply Nets](#)

Creating Logical Power and Ground Connections

After you read in the design, you must ensure that there are logical connections between the power and ground nets and the power, ground, and tie-off pins on the cells in your design. If your design does not already have these connections, use the `connect_pg_net` command to create them. This command creates the logical power and ground connections for leaf cells, hierarchical cells, and physical-only cells in both single-voltage and multivoltage designs.

Before creating the logical power and ground connections, you must resolve any PG conflicts among the netlist, floorplan, and UPF specification.

- For multivoltage designs, the conflicts are resolved when you commit the power intent, as described in [Loading and Applying UPF Information](#).
- For single-voltage designs, you must run the `resolve_pg_netx` command to resolve the conflicts. Note that the UPF specification for a single-voltage design is the default power domain generated by the `read_verilog` command.

If your design contains unmapped instances, the tool issues an information message to indicate that only mapped instances are connected.

The `connect_pg_net` command operates in two modes:

- Automatic

In automatic mode, the command derives all power and ground nets, power and ground pins, tie-off pins, and connections from the UPF specification. If the supply nets do not exist, the tool creates them.

To create the logical power and ground connections in automatic mode, use the `-automatic` option with the `connect_pg_net` command.

- Manual

In manual mode, the command makes the connections that you specify. If a specified pin or port has an existing connection, the tool removes the existing connection and then creates the specified connection. The tool verifies that the connections match the power intent. If it finds a mismatch, the tool issues a warning but still creates the connection.

To create logical power and ground connections in manual mode, use the `-net` option to specify the power or ground net and specify the pins and ports to be connected to that net as an argument to the command. For example, to connect the VDD power net to all pins named vdd and the VSS ground net to all pins named vss, use the following commands:

```
fc_shell> connect_pg_net -net VDD [get_pins */vdd]
fc_shell> connect_pg_net -net VSS [get_pins */vss]
```

To connect PG nets to power and ground pins only, use the `-pg` option. To connect PG nets to tie-off pins only, use the `-tie` option. By default, if neither option is specified, the tool makes connections to all power, ground, and tie-off pins. These options cannot be used with an object list or with the `-create_nets_only` and `-net` options.

Regardless of the command options used, the tool always creates connections required to build a complete PG netlist structure, such as top-level PG ports and intermediate hierarchical PG pins.

Creating Floating Logical Supply Nets

A floating logical supply net is a power or ground net that is not connected to any logical pin or port in the block. For example, a power feedthrough net created during power planning is a floating supply net.

To create a floating supply net with the `connect_pg_net` command, use the following steps:

1. Specify the supply net type in the UPF file by using the `supply_net_pg_type` attribute, as shown in the following UPF file example:

```
...
create_supply_net VDD
...
...
set_design_attributes -elements VDD \
    -attribute supply_net_pg_type power
...
```

2. Enable floating supply nets by setting the `mv.pg.create_floating_pg` application option to `true` before you create supply nets with the `connect_pg_net` command, as shown in the following script example:

```
...
...
# Apply the UPF
load_upf $upf_input_file_name
commit_upf
...
...
# create logical supply nets
set_app_options -as_user_default \
    -name mv.pg.create_floating_pg -value true
connect_pg_net -automatic
...
check_mv_design
```

The tool creates the logical supply nets in the topmost hierarchy of the current UPF scope for domain-independent supply nets. For domain-dependent supply nets, the tool creates the logical supply nets in the topmost hierarchy of the domain.

The tool does not create a logical supply net in the following situations:

- There is a conflict between the connection and the supply net type specified with the `supply_net_pg_type` attribute.
- The supply net is connected to a PG pin of an instance.
- The `connect_pg_net` command is not specified in the current or parent physical hierarchy.

Defining Voltage Areas

A voltage area is a physical placement area for the cells associated with a power domain. For multivoltage designs, the power domains are defined in the UPF specification. For single-voltage designs, the tool creates a default power domain when you read the Verilog netlist and associates it with a default voltage area, which comprises the core area of the block.

The placer treats a voltage area the same as an exclusive move bound; it must place the cells in a voltage area within a specified region and it must place all other cells outside of the voltage area. Voltage areas can be rectangular or rectilinear. In addition, they can be disjoint, nested, or overlapping. For overlapping voltage areas, the effective shape of each voltage area is determined by the stacking order of the voltage area shapes.

To define a voltage area, use the `create_voltage_area` command. When you define a voltage area, at a minimum, you must specify the power domains associated with the voltage area. To specify the power domains, use the `-power_domains` option. You can specify one or more power domains; however, all specified power domains must have the same primary supply net. If you specify a single power domain, the name of the voltage area is derived from the name of the power domain. If you specify multiple power domains, you must specify a name for the voltage area by using the `-name` option.

A voltage area consists of one or more rectangular or rectilinear shapes, which can be abutted, disjoint, or overlapping. To define the boundaries of these shapes, use the `-region` option with the `create_voltage_area` command (if you are creating a new voltage area) or the `create_voltage_area_shape` command (if you are adding shapes to an existing voltage area). Note that you can specify one or more shapes when using the `create_voltage_area` command, but only a single shape in each `create_voltage_area_shape` command.

- To specify the boundary of a rectangle shape, use the following format to specify its lower-left and upper-right corners:

```
 {{lx ly} {urx ury}}
```

- To specify the boundary of a rectilinear shape, use the following format to specify the coordinates of its vertices:

```
 {{x1 y1} {x2 y2} {x3 y3} {x4 y4} ...}
```

If a voltage area consists of multiple abutting or overlapping shapes, you can merge the shapes into a minimum set of disjoint shapes based on the stacking order of the shapes. For information about how to merge the voltage area shapes, see [Merging Voltage Area Shapes](#).

The tool also uses the stacking order to resolve overlapping shapes from different voltage areas. For information about resolving overlapping voltage areas, see [Resolving Overlapping Voltage Areas](#).

To ensure that no shorts occur at the boundaries of the voltage areas, you can define guard bands for the voltage areas, which act as hard keepout margins surrounding the voltage areas. If you define guard bands for a voltage area shape, the guard bands are included in the effective boundary of the shape; however, they are not included in the effective placement area of the voltage area. For information about defining guard bands, see [Defining Guard Bands](#).

To modify an existing voltage area, use the `set_voltage_area` command, as described in [Modifying Voltage Areas](#).

Multivoltage designs typically have power domains that are shut down and powered up during the operation of the chip while other power domains remain powered up. When dealing with shutdown domains, there can be some situations in which certain cells in the shutdown portion need to continuously stay active, such as for implementing retention registers, isolation cells, retention control paths, and isolation enable paths. These cells are referred to as always-on cells. To define a special placement area for always-on cells (an always-on well) within a voltage area, define an exclusive move bound within the boundary of the voltage area. For information about defining exclusive move bounds, see [Defining Move Bounds](#).

After creating the voltage areas, run the `check_mv_design` command to verify that the design does not have any multivoltage violations.

Merging Voltage Area Shapes

To merge the voltage area shapes into a minimum set of disjoint shapes, use the `-merge_regions` option with the `create_voltage_area`, `create_voltage_area_shape`, or `set_voltage_area` command.

- When you use the `create_voltage_area -merge_regions` command, the merges the shapes specified with the `-region` option.
- When you use the `create_voltage_area_shape -merge_regions` command, the merges the shape specified with the `-region` option and the existing shapes of the specified voltage area.
- When you use the `set_voltage_area -merge_regions` command, the tool merges all existing shapes of the specified voltage area.

The tool merges the voltage area shapes based on their stacking order. By default, the stacking order is the order in which you define the shapes, with the last shape defined on top. The merged shape replaces the top shape of a set of abutting or overlapping shapes; the other shapes in the set are removed and are no longer associated with the voltage area.

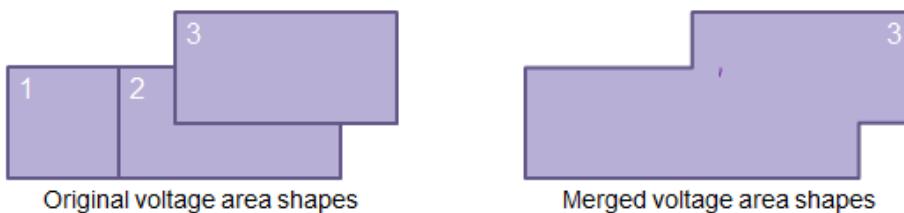
For example, assume that you use the following command to create a voltage area comprising three rectangle shapes, as shown on the left side of [Figure 14](#).

```
fc_shell> create_voltage_area -power_domains {PD1} \
    -region { {{0 0} {10 10}} {{10 0} {30 10}} {{15 5} {35 15}} }
{PD1}
fc_shell> get_voltage_area_shapes -of_objects PD1
{VOLTAGE_AREA_SHAPE_1 VOLTAGE_AREA_SHAPE_2 VOLTAGE_AREA_SHAPE_3}
```

After you use the `-merge_regions` option to merge these shapes, the voltage area consists of a single rectilinear shape, as shown on the right side of [Figure 14](#). The merged voltage area shape is named `VOLTAGE_AREA_SHAPE_3`, which was the last voltage area shape defined when the voltage area was created.

```
fc_shell> set_voltage_area PD1 -merge_regions
Information: Merging abutted and overlapping shapes in voltage_area
'PD1'. (NDMUI-154)
1
fc_shell> get_voltage_area_shapes -of_objects PD1
{VOLTAGE_AREA_SHAPE_3}
```

Figure 14 Merging Voltage Area Shapes



To report the stacking order of the voltage area shapes, use the `report_voltage_areas -verbose` command.

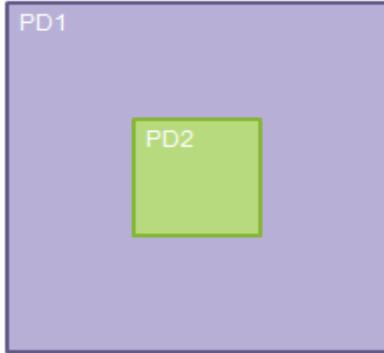
To modify the stacking order of the voltage area shapes, use the `set_voltage_area_shape` command, as described in [Modifying the Stacking Order](#).

Resolving Overlapping Voltage Areas

If voltage area shapes from two or more voltage areas overlap, either completely or partially, the tool uses the stacking order of the shapes to determine the effective shapes of the voltage areas. By default, the stacking order is the order in which you define the shapes, with the last shape defined on top. The tool assigns the overlapped region to the voltage area associated with the top shape. Unlike merging shapes within a voltage area, when the tool resolves overlapping shapes from different voltage areas, it does not remove any shapes; only the interpretation of the shapes changes.

For example, assume you want to create nested voltage areas, as shown in [Figure 15](#).

Figure 15 Nested Voltage Areas



To generate these effective voltage areas, you must specify the outer voltage area first, followed by the inner voltage area, so that the voltage area shape for the inner voltage area is on top:

```
fc_shell> create_voltage_area -power_domains PD1 \
    -region { {0 0} {30 30} }
{PD1}
fc_shell> create_voltage_area -power_domains PD2 \
    -region { {10 10} {20 20} }
{PD2}
fc_shell> get_attribute -objects [get_voltage_areas PD2] \
    -name effective_shapes
{{10.0000 10.0000} {20.0000 20.0000}}
fc_shell> get_attribute -objects [get_voltage_areas PD1] \
    -name effective_shapes
{{0.0000 0.0000} {10.0000 0.0000} {10.0000 20.0000} {20.0000 20.0000}
 {20.0000 10.0000} {10.0000 10.0000} {10.0000 0.0000} {30.0000 0.0000}
 {30.0000 30.0000} {0.0000 30.0000}}
```

If you specify the inner voltage area first, the shape for the outer voltage area is on top and it masks the inner voltage area, so it is ignored by the tool, as shown in the following example:

```
fc_shell> create_voltage_area -power_domains PD2 \
    -region { {10 10} {20 20} }
{PD2}
fc_shell> create_voltage_area -power_domains PD1 \
    -region { {0 0} {30 30} }
{PD1}
fc_shell> get_attribute -objects [get_voltage_areas PD2] \
    -name effective_shapes
fc_shell> get_attribute -objects [get_voltage_areas PD1] \
    -name effective_shapes
{{0.0000 0.0000} {30.0000 30.0000}}
```

To report the stacking order of the voltage area shapes, use the `report_voltage_areas -verbose` command.

To modify the stacking order of the voltage area shapes, use the `set_voltage_area_shape` command, as described in [Modifying the Stacking Order](#).

Modifying the Stacking Order

You can modify the stacking order of the voltage area shapes by using the `set_voltage_area_shape` command.

- To raise a voltage area shape one position, use the `-raise` option.
- To lower a voltage area shape one position, use the `-lower` option.
- To move a voltage area shape to the top, use the `-top` option.
- To move a voltage area shape to the bottom, use the `-bottom` option.
- To move a voltage area shape directly above another shape, use the `-above` option.
- To move a voltage area shape directly below another shape, use the `-below` option.

Defining Guard Bands

Guard bands define hard keepout margins surrounding the voltage areas in which no cells, including level shifters and isolation cells, can be placed. The guard bands guarantee that the cells in different voltage areas are separated so that power planning does not introduce shorts.

By default, voltage areas do not have guard bands. To define guard bands, use the `-guard_band` option with the `create_voltage_area` or `create_voltage_area_shape` command to specify the horizontal and vertical guard band width for each shape specified in the `-region` option. The horizontal guard band width applies to all vertical edges of the voltage area, while the vertical guard band width applies to all horizontal edges of the voltage area.

Note:

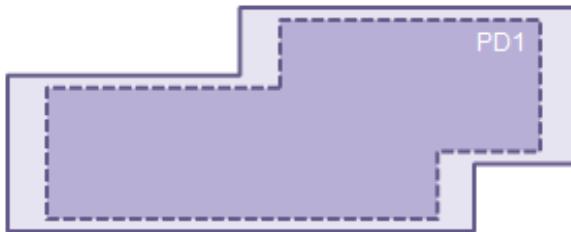
If you also use the `-merge_regions` option, you must specify the guard band widths for each disjoint shape after merging. You would typically use this option only when all the shapes to be merged are abutted or overlapping and therefore merge into a single shape.

The effective boundary of a voltage area shape includes its guard band; however, the effective placeable area of the shape does not.

For example, [Figure 16](#) shows the guard band around the PD1 voltage area that is defined by the following command:

```
fc_shell> create_voltage_area -power_domains PD1 \
    -region {{0 0} {30 0} {30 10} {40 10} {40 30} {20 30} {20 25} {0 25}} \
    -guard_band { {3 1} }
```

Figure 16 Voltage Area Guard Band



To determine the effective guard bands for abutting or overlapping shapes associated with the same voltage area, the tool merges the shapes as described in [Merging Voltage Area Shapes](#) and then applies the guard bands defined for the top shape to the merged shape.

For example, assume that you use the following command to define guard bands for the voltage area shapes shown on the right side of [Figure 14](#):

```
fc_shell> create_voltage_area -power_domains {PD1} \
    -region { {{0 0} {10 10}} {{10 0} {30 10}} {{15 5} {35 15}} } \
    -guard_band { {1 1} {3 3} {3 1} }
{PD1}
```

In this case, the effective guard band is the same as the guard band shown in [Figure 16](#), which is the guard band defined for the merged shape.

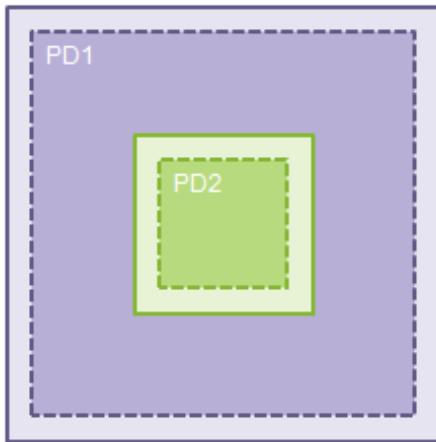
To determine the effective placement areas and guard bands for overlapping shapes associated with different voltage areas, the tool uses the effective boundaries to resolve the shapes as described in [Resolving Overlapping Voltage Areas](#). The top shape retains its placement area and guard band; the effective placement area and guard bands of lower shapes does not include the overlapping region. Note that if abutting shapes have guard bands, they are no longer abutting, but overlapping, due to the effective boundary that includes the guard bands.

For example, assume that you use the following commands to define guard bands for the voltage areas shown in [Figure 15](#):

```
fc_shell> create_voltage_area -power_domains PD1 \
    -region {{0 0} {30 30}} -guard_band { {2 2} }
{PD1}
fc_shell> create_voltage_area -power_domains PD2 \
    -region {{10 10} {20 20}} -guard_band { {2 2} }
{PD2}
```

In this case, the effective placement area of PD1 is reduced by the effective boundary of PD2, which includes its guard band, as shown in [Figure 17](#).

Figure 17 Effective Boundaries of Overlapping Voltage Areas



Defining Gas Stations

Gas stations are small voltage areas that are created for the purpose of minimizing the use of dual-rail buffers on physical feedthrough paths. Only single-rail repeaters are allowed in gas stations, and only optimization steps can add cells to gas stations.

You create gas stations by using the `create_voltage_area` command during the design planning stage. The tool recognizes gas stations automatically and uses them by trading off the cost of a routing detour with the cost of using a dual-rail cell.

Use the `-nwell` and `-pwell` options of the `create_voltage_area` command to specify n-well and p-well supply nets for a voltage area. Defining well supplies for gas station voltage areas provides flexibility for using gas stations for different design styles. If you do not use these options for a voltage area, the well bias values are assumed to be the same as the domain supply values.

The `report_voltage_areas` command lists the n-well and p-well supply nets regardless of whether they are explicitly set.

Querying Voltage Areas

You can query the following information about voltage areas:

- The voltage areas in the current block

To create a collection of voltage areas in the current block, including the default voltage area, use the `get_voltage_areas` command.

- Detailed information about the voltage areas

To display detailed information about voltage areas, including the default voltage area, use the `report_voltage_areas` command.

To include information about the voltage area shapes that comprise each voltage area, use the `-verbose` option with the `report_voltage_areas` command.

- Effective placement area of a voltage area

To display the effective placement area of a voltage area, query the `effective_shapes` attribute of the voltage area.

- Effective guard bands of a voltage area

To display the effective guard bands of a voltage area, query the `effective_guard_band_boundaries` attribute of the voltage area. Note that you can also query this attribute for individual voltage area shapes.

Modifying Voltage Areas

After you have created voltage areas, you can make the following modifications to a voltage area:

- Change the power domains associated with the voltage area

To change the power domains associated with a voltage area, use the `-add_power_domains` and `-remove_power_domains` options with the `set_voltage_area` command.

- Change the voltage area name

To change the name of the voltage area, use the `-name` option with the `set_voltage_area` command.

- Change the voltage area region

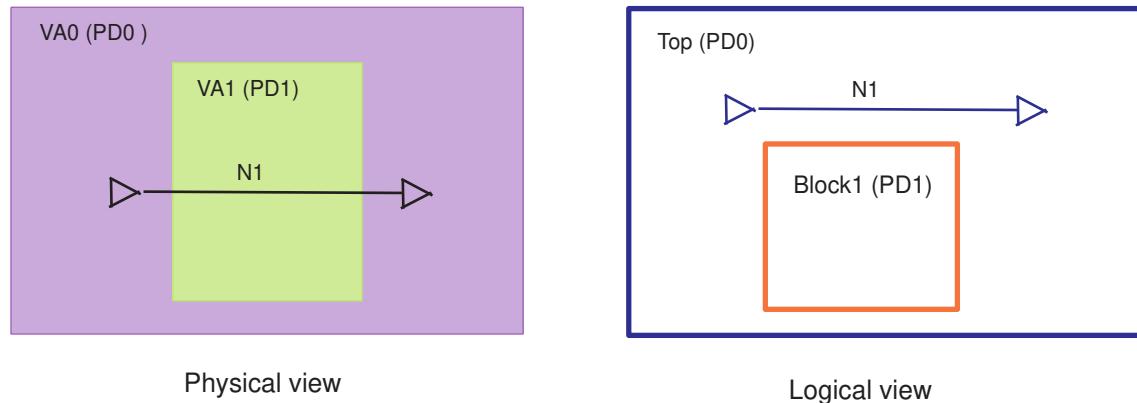
To add shapes to a voltage area, use the `create_voltage_area_shape` command.

To remove shapes from a voltage area, use the `remove_voltage_area_shapes` command.

Controlling Physical-Feedthrough Nets in Voltage Areas

A net is considered to be native to a voltage area if one or more segments of that net are in a logical hierarchy of that voltage area. If a net must physically route over a nonnative voltage area, then it is a physical feedthrough net of that voltage area, as shown in the following figure.

Figure 18 Physical-Feedthrough Nets of Voltage Areas

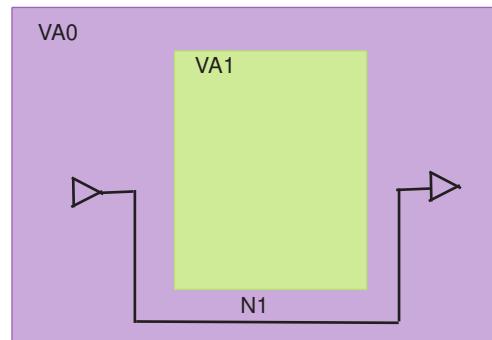


By default, physical-feedthrough nets are allowed in voltage areas. To prevent physical-feedthrough nets in a voltage area, define a voltage area rule using the `create_voltage_area_rule -allow_pass_through false` command, as shown in the following example:

```
fc_shell> create_voltage_area_rule -allow_pass_through false \
-name VA1_rule -voltage_areas VA1
```

With this rule, the N1 net detours around the VA1 voltage area, as shown in Figure 19.

Figure 19 Physical-Feedthrough Nets Disabled for a Voltage Area



By default, optimization does not insert buffers on physical-feedthrough nets of voltage areas. Inserting a buffer on a physical-feedthrough net can cause a mismatch between the

logical and physical view of the buffer. The tool resolves such mismatches by supporting the following types of optimization for such nets:

- Physical-feedthrough buffering

During physical-feedthrough buffering the tool performs the following:

1. Inserts buffers in the logical hierarchy of its drivers or loads and applies the power domain of the nonnative voltage area on the buffer using power-domain-on-instance (PDOI) constraints
2. Places the buffers within the nonnative voltage area

To allow physical-feedthrough buffering, use the `create_voltage_area_rule -allow_physical_feedthrough true` command, as shown in the following example:

```
fc_shell> create_voltage_area_rule \
    -allow_physical_feedthrough true \
    -name VA1_rule -voltage_areas VA1
```

- Logical-feedthrough buffering

During logical-feedthrough the tool performs the following:

1. Adds buffers in the logical hierarchy corresponding to the nonnative voltage area
2. Places the buffers within the nonnative voltage area

To allow logical-feedthrough buffering, use the `create_voltage_area_rule -allow_logical_feedthrough true` command. To specify logical hierarchies in which feedthrough buffers are allowed or not allowed, use the `-include_logical_feedthrough_hierarchy` or `-exclude_logical_feedthrough_hierarchy` option, respectively.

The following example enables logical-feedthrough buffering for the VA1 voltage area and limits the feedthrough buffers to only the U1 and U2 hierarchical cells:

```
fc_shell> create_voltage_area_rule \
    -allow_logical_feedthrough true \
    -include_logical_feedthrough_hierarchy {U1 U2} \
    -name VA1_rule -voltage_areas VA1
```

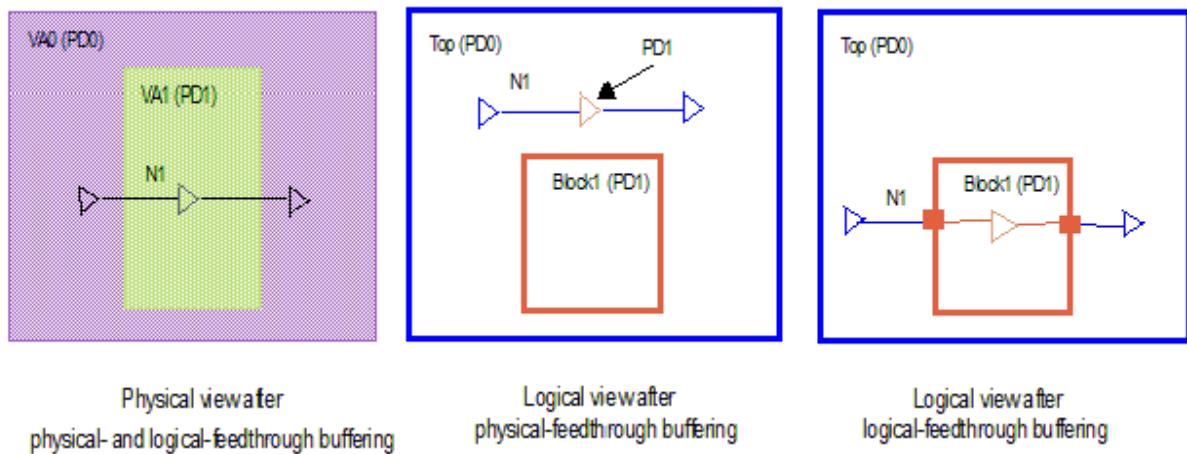
The following example enables logical-feedthrough buffering for the VA2 voltage area and excludes the U3 hierarchical cell from feedthrough buffering:

```
fc_shell> create_voltage_area_rule \
    -allow_logical_feedthrough true \
    -exclude_logical_feedthrough_hierarchy {U3} \
    -name VA2_rule -voltage_areas VA2
```

Note:

To add a logical-feedthrough buffer to a lower-level block, the tool has to add new boundary ports to that block. Therefore, freezing the boundary of a block by using the `set_freeze_ports` command prevents the tool from adding a logical-feedthrough buffer to that block.

Figure 20 Difference in the Logical View After Physical- and Logical-Feedthrough Buffering



To create a default rule that applies to all voltage areas that do not have a specific rule, use the `create_voltage_area_rule` command with the `-default_rule` option, as shown in the following example:

```
fc_shell> create_voltage_area_rule -default_rule \
    -allow_physical_feedthrough true
```

If a voltage area does not have a specific rule, and there is no default rule, feedthrough buffering for that voltage area is controlled by the `opt.common.allow_physical_feedthrough` application option setting.

To report voltage area rules, use the `report_voltage_area_rules` command. To remove voltage area rules, use the `remove_voltage_area_rules` command.

Removing Voltage Areas

To remove voltage areas from a block, use the `remove_voltage_areas` command. To remove all voltage areas from a block, use the `-all` option. To remove specific voltage areas from a block, specify the voltage area names.

Inserting Multivoltage Cells

A multivoltage design requires special multivoltage cells, such as level shifters and isolation cells, at the interface between power domains. Level shifters are required between power domains that operate at different voltage levels, while isolation cells are required between power domains that are in different states (powered-down versus always-on or powered-up).

Inserting Level Shifters

Level-shifter cells function as the interface between power domains that operate at different voltage levels. These cells ensure that the output transition of a driver can cause the receiving cell to switch even though the receiver is operating at a different voltage level.

To insert level shifters in the current block, use the `create_mv_cells` command. The tool inserts the level shifters using the cell mapping and strategy defined in the UPF specification. When the tool inserts a level shifter, it sets a `size_only` attribute on the level shifter and a `dont_touch` attribute on the port-to-level-shifter net.

By default, the command inserts cells only from the generic library. If you use the `-mapped` option with the `create_mv_cells` command, the command inserts cells only from the user-provided logic library and issues an error if a suitable cell is not available.

If the tool does not insert a level-shifter cell, you can use the `analyze_mv_design -level_shifter` command to obtain more information. If you specify a net or pin with the `-through` option, the report lists the power domains and related supplies for the driver and load sides of the net or pin, along with error messages that indicate why a level-shifter cell was not inserted. The report includes errors about the specific insertion point as well as errors that are applicable to the entire path through the net or pin specified with the `-through` option.

Inserting Isolation Cells

Isolation cells are used to selectively shut off the input side of the voltage interface of a power domain; they do not shift the voltage. Isolation cells should be instantiated at the RTL level to prevent formal verification errors. However, you can also insert them using the Fusion Compiler tool.

To insert isolation cells in the current block, use the `create_mv_cells` command. The tool inserts the isolation shifters using the cell mapping and strategy defined in the UPF specification. When the tool inserts an isolation cell, it sets a `size_only` attribute on the level shifter and a `dont_touch` attribute on the port-to-level-shifter net.

By default, the command inserts cells only from the generic library. If you use the `-mapped` option with the `create_mv_cells` command, the command inserts cells only from the user-provided logic library and issues an error message if a suitable cell is not available.

Associating Power Strategies With Existing Multivoltage Cells

The Fusion Compiler tool automatically associates power strategies with existing multivoltage cells when you run the `associate_mv_cells` or `commit_upf` command. If this automatic association is not correct, you can manually modify the associations by using the `set_power_strategy_attribute` command. To determine the power strategies for a power domain, use the `get_power_strategies` command.

Controlling the Placement of Multivoltage Cells

A net that connects a multivoltage cell, such as a level-shifter cell or an isolation cell, to at least one cell in another voltage area is referred to as a multivoltage net. To reduce the length of multivoltage nets, enable advanced multivoltage cell placement by setting the `place.coarse.enable_advanced_mv_cell_placement` application option to `true`.

When you enable this feature, the tool reduces the length of the multivoltage nets by placing the multivoltage cells closer to the voltage area boundaries. To optimize multivoltage nets, the tool uses dual-rail buffers. By reducing the length of multivoltage nets, the tool can reduce the number of dual-rail buffers used during optimization and prevent unoptimized multivoltage nets due to the unavailability of dual-rail buffers.

Enabling Improved Buffering for Multivoltage Nets

Multivoltage nets are nets that logically or physically cross over more than one voltage area. You can enable the use of improved buffering techniques for fixing logical DRC violations on such nets during preroute optimization by setting the `opt.buffering.enable_hierarchy_mapping` application option to `true`.

This application option affects the `compile_fusion` and `clock_opt` commands. Enabling this feature reduces the number of buffers used to fix logical DRC violations. However, it can slightly increase the total negative slack or number of hold violations.

Analyzing Multivoltage Information

To ensure that the design does not have any multivoltage design violations, use the `check_mv_design` command.

You can restrict the types of rules to check with the `check_mv_design` command. For example, the `-isolation` option specifies to check isolation strategy and isolation cell rules, while the `-pg_pin` option specifies to check rules associated with PG pins.

To report multivoltage information for your design, use the commands shown in the following table.

Table 6 Commands for Reporting Multivoltage Information

To do this	Use this command
Report a summary of multivoltage information	<code>report_mv_design</code>
Report information about the multivoltage cells	<code>report_mv_cells</code>
Report information about the multivoltage library cells	<code>report_mv_lib_cells</code>
Report paths with multivoltage constraints and the associated multivoltage cells	<code>report_mv_path</code>
Report the power domains	<code>report_power_domains</code>
Check whether the specified power domains are equivalent	<code>check_equivalent_power_domains</code>
Display a list of equivalent power domains	<code>get_equivalent_power_domains</code>
Report the voltage areas	<code>report_voltage_areas</code>

Specifying Timing Constraints and Settings

Timing constraints describe the timing requirements of a design. An essential part of timing constraints are accurately specified clocks and clock effects, such as latency and uncertainty. When you specify clocks, the tool automatically constrains the paths between the registers driven by these clocks. However, you can change the default behavior of these timing paths by specifying timing exceptions such as paths that should not be analyzed (false paths), paths that require multiple clock cycles (multicycle paths), and so on. In addition, you can constrain the boundary timing paths by specifying input and output delays for input and output ports.

The Fusion Compiler tool uses on-chip variation (OCV) mode to perform timing analysis, which models the effects of variation in operating conditions across the chip. This mode performs a conservative timing analysis that allows both minimum and maximum delays to apply to different paths at the same time. For a setup check, it uses maximum delays for the launch clock path and data path and minimum delays for the capture clock path. For a hold check, it uses minimum delays for the launch clock path and data path and maximum delays for the capture clock path.

A block might operate under several different conditions, such as different temperatures and voltages, and might operate in several different functional modes. For timing analysis, each set of conditions is represented by a *corner* and each functional mode is represented by a *mode*. A *scenario* is a combination of a corner and mode used to perform timing analysis and optimization. Before you start working with a block, you must define the modes, corners, and scenarios that are used for the block, as well as the delay calculation

model and routing layers to use. The routing layer information you specify is used for RC estimation during timing analysis.

For detailed information about specifying

- Clock and clock effects, see the “Defining Clocks” topic in the *Fusion Compiler Timing Analysis User Guide*.
- Exceptions for timing paths and constraints for boundary paths, see the “Constraining Timing Paths” topic in the *Fusion Compiler Timing Analysis User Guide*.
- Modes, corners, and scenarios, see the “Defining Modes, Corners, and Scenarios” topic in the *Fusion Compiler Timing Analysis User Guide*.
- Operating conditions and on-chip variation (OCV) related settings, see the “Specifying Operating Conditions” topic in the *Fusion Compiler Timing Analysis User Guide*.
- Parasitic information for RC estimation and extraction, see the “Performing Parasitic Extraction” topic in the *Fusion Compiler Timing Analysis User Guide*.
- Routing layers, see [Specifying the Routing Resources](#).

Specifying Logical Design Rule Constraints

Minimum capacitance, maximum capacitance, and maximum transition are logical design rule constraints that your design must meet to function as intended. They are technology-specific restrictions that are specified in the logic libraries. However, you can specify more restrictive design rule constraints by using the constraint commands given in [Table 7](#).

During optimization, the Fusion Compiler tries to meet the design rule constraints, even if it means violating optimization constraints such as timing, power, and area goals; these design rule constraints have a higher priority. After optimization, you can use the reporting commands given in [Table 7](#) to identify design rule constraint violations in a block.

Table 7 Design Rule Commands

To do this	Use this command
Specify the minimum allowed capacitance for input ports, library cell pins, leaf cell pins, clocks, or blocks	<code>set_min_capacitance</code>
Specify the maximum allowed capacitance for input ports, library cell pins, leaf cell pins, clocks, or blocks	<code>set_max_capacitance</code>
Specify the maximum allowed signal transition time for input ports, library cell pins, leaf cell pins, clocks, or blocks	<code>set_max_transition</code>
Remove a user-specified minimum capacitance constraint	<code>remove_min_capacitance</code>

Table 7 Design Rule Commands (Continued)

To do this	Use this command
Remove a user-specified maximum capacitance constraint	<code>remove_max_capacitance</code>
Remove a user-specified maximum transition constraint	<code>remove_max_transition</code>
Report minimum capacitance constraint violations	<code>report_constraints -min_capacitance</code>
Report maximum capacitance constraint violations	<code>report_constraints -max_capacitance</code>
Report maximum transition constraint violations	<code>report_constraints -max_transition</code>
Specify the connection requirements for the network that is connected to the specified ports or library cell pins	For ports: <code>set_connection_class</code> For pins: <code>set_attribute \ -name connection_class</code>

Note:

The maximum fanout design rule constraint is not honored by the Fusion Compiler tool. However, you can specify a maximum fanout for the data paths in a block by using the `opt.common.max_fanout` application option. This is a soft optimization constraint. During optimization the tool tries to ensure that data path cells do not drive more than the specified maximum fanout.

Specifying Clock Gating Constraints

By default, the tool identifies preexisting clock-gating cells during the execution of the `analyze` and `elaborate` commands. During compile, the tool inserts one more level of clock-gating cells at the leaf level and implements clock-gating logic using the default

clock-gating settings if you do not specify any clock-gating constraints. You cannot disable clock gating.

Before you set up clock-gating constraints, load your design and libraries in memory and then use the following commands to set up the constraints:

- The `set_clock_gate_style` command

To select the type of integrated clock-gating cell, specify the following options:

Option	Description
<code>-test_point</code>	Specifies the test control position with respect to integrated clock-gating cell. The valid values are <code>none</code> , <code>before</code> , and <code>after</code> .
<code>-observation_output</code>	Specifies the observation output pin.
<code>-target</code>	Specifies the target sequential elements to be clock-gated. The valid values are <code>pos_edge_flip_flop</code> and <code>neg_edge_flip_flop</code> .
<code>-objects</code>	Specifies the objects to which the clock gate applies.

- The `set_clock_gating_options` command

To define the netlist structure of the inserted clock-gating cells, specify the following options:

Option	Description
<code>minimum_bitwidth</code>	Specifies the minimum number of bits to gate together. The default is three.
<code>max_fanout</code>	Specifies the maximum number of cells gated by the same clock-gating cell. The default is infinite.

- The `set_clock_gating_objects` command

To control clock gating for specified objects and override the default clock-gating behavior that is set during compile, specify the following options. An object can be a hierarchical cell, register, power domain, or module.

Option	Description
<code>-include</code>	Specifies a list of objects to gate as per the clock-gating style and clock-gating options.
<code>-exclude</code>	Specifies a list of objects to exclude from clock gating.

Option	Description
<code>-clear</code>	Specifies to remove all clock-gating inclusion or exclusion for the specified list of objects.
<code>-reset</code>	Specifies to remove all criteria previously set by the <code>set_clock_gating_objects</code> command.

- The default

During clock-gating optimization, the tool uses the following setup defaults:

Setup	Default
Maximum fanout	Infinite
Minimum bit-width	Three
Clock-gating cell	Latch-based cells for both positive-edge and negative-edge triggered flip-flops. If only one type of cell is available, the tool inserts inverters on both sides of the clock-gating cell.

For more information, see [Setting Up Clock Gating](#).

Specifying Physical Constraints for Placement and Legalization

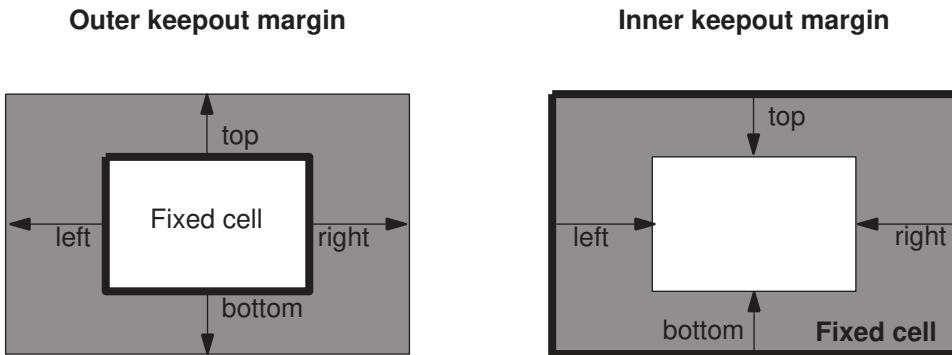
During placement and legalization, the floorplan information dictates where cells are placed. The following topics describe how to specify additional physical constraints that affect placement and legalization:

- [Defining Keepout Margins](#)
- [Defining Area-Based Placement Blockages](#)
- [Defining Placement Bounds](#)
- [Defining Placement Attraction](#)
- [Specifying Locations for Unmapped Cells](#)
- [Defining Cell Spacing Constraints for Legalization](#)

Defining Keepout Margins

A keepout margin is a region (the shaded portions in [Figure 21](#)) around the boundary of fixed cells in a block in which no other cells are placed.

Figure 21 Placement Keepout Margins



An outer keepout margin is a region outside the cell boundary, while an inner keepout margin is a region inside the cell boundary. The width of the keepout margin on each side of the fixed cell can be the same or different, depending on how you define the keepout margin. In addition, keepout margins can be defined as hard or soft. Keeping the placement of cells out of such regions avoids congestion and net detouring and produces better QoR.

To define a keepout margin, use the `create_keepout_margin` command. By default, the command creates a hard keepout margin. To create a soft keepout margin, use the `-type soft` option.

Defining an Outer Keepout Margin

You can define an outer keepout margin on a hard macro, a hierarchical cell, or a leaf cell. When you define an outer keepout margin, you can either specify the keepout distance explicitly or, for hard macros, you can have the tool derive the keepout distance based on the macro pin count.

To explicitly specify an outer keepout margin, use the `-outer` option to specify the margin distance for each side. You specify the left, bottom, right, and top margins using the following format: `{lx by rx ty}`. A value of 0 results in no keepout margin for that side.

For example, to create a hard outer keepout margin with a margin of 10 on each side for a macro named `my_macro`, use the following command:

```
fc_shell> create_keepout_margin -outer {10 10 10 10} my_macro
```

To have the tool derive the outer keepout distance for a hard macro based on its pin count, use the `-tracks_per_macro_pin` option to specify the track-to-pin ratio. When you use this option, the tool calculates the keepout margin from the track width, the number of macro pins, and the specified track-to-pin ratio, which is typically set to a value near 0.5. A larger value results in larger keepout margins. The derived keepout margin is always hard; the `-type` setting is ignored. The derived margins are subject the minimum and maximum

values specified by the `-min_padding_per_macro` and `-max_padding_per_macro` options.

For example, to have the tool derive the outer keepout margin for a macro named `my_macro` by using a track-to-pin ratio of 0.6 with a minimum keepout distance of 0.1 and a maximum keepout distance of 0.2, use the following command:

```
fc_shell> create_keepout_margin -tracks_per_macro_pin 0.6 \
    -min_padding_per_macro 0.1 -max_padding_per_macro 0.2 my_macro
```

Defining an Inner Keepout Margin

You can define an inner keepout margin on a hierarchical cell, but not on a hard macro or a leaf cell. When you define an inner keepout margin, you must specify the keepout distance explicitly.

To explicitly specify an inner keepout margin, use the `-inner` option to specify the margin distance for each side. You specify the left, bottom, right, and top margins using the following format: `{lx by rx ty}`. A value of 0 results in no keepout margin for that side.

For example, to create a hard inner keepout margin with a margin of 10 on each side for a hierarchical cell named `my_hcell`, use the following command:

```
fc_shell> create_keepout_margin -inner {10 10 10 10} my_hcell
```

Defining Area-Based Placement Blockages

An area-based placement blockage is a rectangular region in which cells cannot be placed or in which the types or number of cells is limited. The Fusion Compiler tool supports the following types of area-based placement blockages:

- Hard

A hard blockage prevents the placement of standard cells and hard macros within the specified area during coarse placement, optimization, and legalization.

- Hard macro

A hard macro blockage prevents the placement of hard macros within the specified area during coarse placement, optimization, and legalization.

- Soft

A soft blockage prevents the placement of standard cells and hard macros within the specified area during coarse placement, but allows optimization and legalization to place cells within the specified area.

- Partial

A partial blockage limits the cell density in the specified area during coarse placement, but has no effect during optimization and legalization.

To define placement blockages, use the `create_placement_blockage` command. At a minimum, you must specify the coordinates of the placement blockage. To create a rectangular placement blockage, use the `-boundary` option to specify the lower-left and upper-right coordinates of the rectangle. To create a rectilinear placement blockage, use the `-boundary` option to specify the coordinates of the polygon.

By default, the `create_placement_blockage` command creates a hard placement blockage. To create another type of placement blockage, use the `-type` option to specify the blockage type. A single `create_placement_blockage` command can create just one type of placement blockage.

You can optionally assign a name to a placement blockage by using the `-name` option. You can then reference that blockage by name to query or remove the placement blockage.

Defining a Hard Placement Blockage

To define a hard placement blockage, specify the boundary and optionally a name for the placement blockage.

For example, to create a hard placement blockage in the area enclosed by a rectangle with corners at (10, 20) and (100, 200), use the following command:

```
create_placement_blockage -boundary {10 20 100 200}
```

Note:

Hard placement blockages are honored during placement, legalization, optimization, and clock tree synthesis.

Hard placement blockages can also be defined in the DEF as shown in [Example 13](#).

Example 13 Placement Blockages in DEF

```
BLOCKAGES 2 ;
- PLACEMENT
```

```

RECT ( 0 327600 ) ( 652740 327660 ) ;
- PLACEMENT
RECT ( 0 327600 ) ( 652740 327660 ) ;
END BLOCKAGES
1

```

Defining a Hard Macro Placement Blockage

To define a hard macro blockage, specify the boundary, type (`-type hard_macro` option), and optionally the name for the placement blockage.

For example, to define a hard macro blockage enclosed by a rectangle with corners at (120, 75) and (230, 200), use the following command:

```
create_placement_blockage -boundary {120 75 230 200} \
    -type hard_macro
```

Note:

Hard macro placement blockages are honored during placement, legalization, and optimization. This is the only type of placement blockage that is honored by hard macro placement.

Defining a Soft Placement Blockage

To define a soft blockage, specify the boundary, type (`-type soft` option), and optionally the name for the placement blockage.

For example, to define a soft blockage enclosed by a rectangle with corners at (120, 75) and (230, 200), use the following command:

```
create_placement_blockage -boundary {120 75 230 200} \
    -type soft
```

A soft blockage prevents the initial placement from placing cells within the specified area, but allows legalization, optimization, and clock tree synthesis to do so. However, after placement and optimization, during subsequent incremental placement, the tool can move the cells added during legalization, optimization, and clock tree synthesis out of the soft blockage area. To prevent this, use the following application option setting:

```
set_app_options -name place.coarse.enable_enhanced_soft_blockages \
    -value true
```

Note:

Soft placement blockages are honored during placement, but not during legalization, optimization, or clock tree synthesis.

Defining a Partial Placement Blockage

To define a partial blockage, specify the boundary, type (`-type partial` option), blockage percentage (`-blocked_percentage` option), and optionally the name for the placement blockage.

For example, to define a partial blockage with a maximum allowed cell density of 60 percent (a blocked percentage of 40), enclosed by the rectangle with corners at (10, 20) and (100, 200), use the following command:

```
create_placement_blockage -boundary {10 20 100 200} \
    -type partial -blocked_percentage 40
```

To allow unlimited usage of a partial blockage area, specify a blockage percentage of zero (`-blocked_percentage 0` option).

Note:

Partial placement blockages are honored during placement, but not during legalization, optimization, or clock tree synthesis.

Define a Blockage of a Predefined Category

A category blockage is a special type of partial blockage that controls the placement of a predefined category of cells within the partial blockage.

To create a category blockage,

1. Create the cell category by defining a user attribute for the category by using the `define_user_attribute` command and applying it to the affected cell references or instances by using the `set_attribute` command.

To prevent the cell from being placed in the category blockage, set the attribute to `true`. To allow the cell in the category blockage, set the attribute to `false`.

When defining and applying attributes, observe the following rules:

- A blockage can only be controlled by a single attribute.
- Multiple blockages can be controlled by the same attribute.
- A cell can have multiple attributes, which could impact its placement in multiple category blockages.

- The same attribute can be applied to a reference and an instance.
 - If a cell instance and its reference have different attributes, the attribute on the cell instance takes precedence.
2. Create the category blockage by using the `create_placement_blockage` command with the `-type category`, `-blocked_percentage`, and `-category` options. The argument to the `-category` option is the name of the user attribute.

The following example defines a blockage that prevents a certain category of cells from being placed within it:

```
define_user_attribute -type boolean \
    -classes lib_cell dr_cells
set_attribute [get_lib_cells mylib/dr*] dr_cells true
create_placement_blockage -boundary {{4300 2400} {4500 2700}} \
    -type category -blocked_percentage 45 -category dr_cell
```

The following example defines a blockage that allows only a certain category of cells to be placed within it:

```
define_user_attribute -type boolean \
    -classes lib_cell non_iso
set_attribute [get_lib_cells mylib/*] non_iso true
set_attribute [get_lib_cells mylib/iso*] non_iso false
create_placement_blockage -boundary {{4300 2400}{4500 2700}} \
    -type category -blocked_percentage 30 -category non_iso
```

Note:

Category blockages are honored during placement, but not during legalization, optimization, or clock tree synthesis.

Defining Blockages That Exclude Registers

To define partial blockages that exclude registers, specify the boundary, type (`-type register` option), blockage percentage (`-blocked_percentage` option), and optionally the name for the placement blockage.

For example, to define a partial blockage that excludes registers, but allows a cell density of 50 percent for other cells, use the following commands:

```
create_placement_blockage -boundary {{10 20} {100 200}} \
    -type register -blocked_percentage 50
```

Note:

Register blockages are honored during placement, but not during legalization, optimization, or clock tree synthesis.

Defining Blockages That Exclude Relative Placement Groups

To define partial blockages that exclude relative placement groups, specify the boundary, type (-type rp_group option), blockage percentage (-blocked_percentage option), and optionally the name for the placement blockage.

For example, to define a partial blockage that excludes relative placement groups, but allows a cell density of 100 percent for all other cells, use the following commands:

```
create_placement_blockage -boundary {{10 20} {100 200}} \
    -type rp_group -blocked_percentage 0
```

Note:

Relative placement group blockages are honored during placement, but not during legalization, optimization, or clock tree synthesis.

Defining Blockages That Allow Relative Placement Cells Only

To define partial blockages that allow relative placement cells only, specify the boundary, type (-type allow_rp_only option), blockage percentage (-blocked_percentage option), and optionally the name for the placement blockage.

For example, to defines a partial blockage that allows relative placement cells only, with a maximum allowed cell density of 80 percent (a blocked percentage of 20), and enclosed by the rectangle with corners at (10, 20) and (100,200), use the following command:

```
create_placement_blockage -name rp_only -type allow_rp_only \
    -boundary {10 20 100 200} -blocked_percentage 20
```

These blockages allows the tool to place relative placement groups, which are usually large structures, without disturbing the placement of other cells. After the tool places and legalizes the relative placement groups, you can remove these blockages and allow the tool to place other cells in them.

Note:

Relative-placement-cells-only blockages are honored during placement, but not during legalization, optimization, or clock tree synthesis.

Defining Blockages That Allow Buffers Only

To define a partial blockage that allows buffers only, specify the boundary, type (-type allow_buffer_only option), blockage percentage (-blocked_percentage option), and optionally the name for the placement blockage.

For example, to define a partial blockage that allows only the placement of buffers and inverters, with a cell density of 30 percent, use the following commands:

```
create_placement_blockage -boundary {{10 20} {100 200}} \
    -type allow_buffer_only -blocked_percentage 70
```

Note:

Buffer-only blockages are honored during placement, but not during legalization, optimization, or clock tree synthesis.

Querying Placement Blockages

To return a collection of placement blockages in the current block that match certain criteria, use the `get_placement_blockages` command.

Removing Placement Blockages

To remove placement blockages from a block, use the `remove_placement_blockages` command. To remove all placement blockages from a block, use the `-all` option. To remove specific placement blockages from a block, specify the placement blockage names.

Defining Placement Bounds

A placement bound is a constraint that controls the placement of groups of cells. It allows you to group cells to minimize wire length and place the cells at the most appropriate locations. For example, you might want to define a bound for clock-gating cells or extremely timing-critical groups of cells that you want to guarantee will not be disrupted for placement by other logic.

[Table 8](#) lists the types of placement bounds supported by the Fusion Compiler tool.

Table 8 Types of Placement Bounds

Bound type	Description
Soft move bound	The tool tries to place the cells in the move bound within a specified region, during coarse placement. However, there is no guarantee that the cells are placed inside the bounds. Soft move bounds are honored during coarse placement, but not legalization.
Hard move bound	The tool must place the cells in the move bound within a specified region. Hard move bounds are honored during both coarse placement, and legalization. However, they are not honored during clock tree synthesis.

Table 8 Types of Placement Bounds (Continued)

Bound type	Description
Exclusive move bound	The tool must place the specified cells within the specified region and place all other cells outside of the exclusive move bounds, during both coarse placement and legalization. The placement of newly added buffers in exclusive move bounds is controlled by the <code>opt.buffering.exclusive_hard_bound_buffering_mode</code> application option. By default, during buffering, the tool can add new cells to an exclusive hard move bound if it improves the QoR. If you set this application option to 0, the tool honors the exclusive hard move bound.
Soft group bound	The tool tries to place the cells in the group bound within a floating region; however, there is no guarantee that the cells are placed inside the bounds.
Hard group bound	The tool must place the cells in the group bound within a floating region, whose actual coordinates are determined by the tool.
Dimensionless group bound	The tool determines the shape and location of the group bound based on the effort used to bring cells closer inside the group bound.

To define a placement bound, use the `create_bound` command. When you define a bound, you must use the `-name` option to specify its name.

In general, you also specify the cells and ports to be included in the bound. If a hierarchical cell is included, all cells in the subdesign belong to the bound. However, you can create an empty bound and specify the contents of the bound later by using the `add_to_bound` command. You can remove objects from a bound by using the `remove_from_bound` command.

You must also specify the options required for the specific type of bound you want to create. The following topics describe how to create the various types of move bounds:

- [Defining Move Bounds](#)
- [Defining Group Bounds](#)
- [Querying Placement Bounds](#)
- [Removing Placement Bounds](#)

Defining Move Bounds

A move bound is a fixed region within which to place a set of cells. It comprises one or more rectangular or rectilinear shapes, which can be abutted, disjoint, or overlapping. To define the boundaries of these shapes, use the `-boundary` option with the `create_bound` command (if you are creating a new move bound) or the `create_bound_shape` command

(if you are adding shapes to an existing move bound). Note that you can specify one or more shapes when using the `create_bound` command, but only a single shape in each `create_bound_shape` command.

- To specify the boundary of a rectangle shape, use the following format to specify its lower-left and upper-right corners:

```
{ {lx ly} {urx ury} }
```

- To specify the boundary of a rectilinear shape, use the following format to specify the coordinates of its vertices:

```
{ {x1 y1} {x2 y2} {x3 y3} {x4 y4} ... }
```

Move bounds can be hard, soft, or exclusive.

- To define a soft move bound, use the following syntax:

```
create_bound -name name [-type soft] [-effort effort_level] \
             -boundary {coordinates} [bound_objects]
```

The default effort is `ultra`; you can also specify `low`, `medium`, or `high`.

For example, to define a rectangular soft move bound for the `INST_1` cell instance with its lower-left corner at (100, 100) and its upper-right corner at (200, 200), use the following command:

```
fc_shell> create_bound -name b1 -boundary {100 100 200 200} INST_1
```

- To define a hard move bound, use the following syntax:

```
create_bound -name name -type hard -boundary {coordinates} \
             [bound_objects]
```

For example, to define a rectangular hard move bound for the `INST_1` cell instance with its lower-left corner at (100, 100) and its upper-right corner at (200, 200), use the following command:

```
fc_shell> create_bound -name b2 -type hard \
             -boundary {100 100 200 200} INST_1
```

- To define an exclusive move bound, use the following syntax:

```
create_bound -name name -exclusive -boundary {coordinates} \
             [bound_objects]
```

For example, to define a rectangular exclusive move bound for the `INST_1` cell instance with its lower-left corner at (100, 100) and its upper-right corner at (200, 200), use the following command:

```
fc_shell> create_bound -name b3 -exclusive \
             -boundary {100 100 200 200} INST_1
```

To add shapes to an existing move bound, use the `create_bound_shape` command. To remove shapes from an existing move bound, use the `remove_bound_shapes` command.

Defining Group Bounds

A group bound is a floating region within which to place a set of cells. Group bounds can be hard, soft, or dimensionless.

- To define a soft group bound, use the following syntax:

```
create_bound -name name [-type soft] [-effort effort_level] \
[-dimensions {width height} bound_objects]
```

The default effort is `ultra`; you can also specify `low`, `medium`, or `high`.

For example, to define a soft group bound for the `INST_1` and `INST_2` cell instances with a width of 100 and a height of 100, use the following command:

```
fc_shell> create_bound -name b4 -dimensions {100 100} \
{INST_1 INST_2}
```

- To define a hard group bound, use the following syntax:

```
create_bound -name name -type hard -dimensions {width height} \
[bound_objects]
```

For example, to define a hard group bound for the `INST_1` and `INST_2` cell instances with a width of 100 and a height of 100, use the following command:

```
fc_shell> create_bound -name b5 -type hard -dimensions {100 100} \
{INST_1 INST_2}
```

- To define a dimensionless group bound, use the following syntax:

```
create_bound -name name [-effort effort_level] \
[bound_objects]
```

The default effort is `medium`; you can also specify `low`, `high`, or `ultra`.

For example, to define a dimensionless group bound for the `INST_1` and `INST_2` cell instances in which the tool uses a high level of effort to place the cells closer within the group bound, use the following command:

```
fc_shell> create_bound -name b6 -effort high {INST_1 INST_2}
```

Querying Placement Bounds

To report the placement bounds in a block, use the `report_bounds` command.

To return a collection of placement bounds in the current block that match certain criteria, use the `get_bounds` command.

To return a collection of bound shapes associated with one or more move bounds, use the `get_bound_shapes` command.

Removing Placement Bounds

To remove placement bounds from a block, use the `remove_bounds` command. To remove all placement bounds from a block, use the `-all` option. To remove specific placement bounds from a block, specify the placement bound names.

Defining Placement Attractions

A placement attraction is a constraint that you can use to specify that a large group of cells be placed together in the same vicinity of the placement area. It is a soft constraint that the tool considers during initial placement. However, it has less effect during subsequent incremental placement stages. To specify a more restrictive placement constraint for a smaller group of cells, create a move bound or group bound by using the `create_bound` command.

To define a placement attraction, use the `create_placement_attraction` command. For example, to specify that the cells from the subblocks named add1 and mult1 be placed in the same vicinity, use the following command:

```
fc_shell> set add1_cells [get_flat_cells add1/*]
fc_shell> set mult1_cells [get_flat_cells mult1/*]
fc_shell> create_placement_attraction -name add1_mult1 \
    "$add1_cells $mult1_cells"
```

You can use the `-region` option to specify

- A region in which to place the cells

To do so, specify the lower-left and upper-right coordinates of the region, as shown in the following example:

```
fc_shell> set U1_cells [get_flat_cells U1/*]
fc_shell> create_placement_attraction -name U1 \
    -region {{0 0} {2000 1500}} $U1_cells
```

- A straight line along which to place the cells, such as an edge of a macro cell

To do so, specify the coordinates of the two ends of the line, as shown in the following example:

```
fc_shell> set U2_cells [get_flat_cells U2/*]
fc_shell> create_placement_attraction -name U2 \
    -region {{1000 200} {1000 800}} $U2_cells
```

- A location around which to place the cells, such as a port location

To do so, specify the coordinates of the location, as shown in the following example:

```
fc_shell> set U3_cells [get_flat_cells U3/*]
fc_shell> create_placement_attraction -name U3 \
    -region {{0 700}} $U3_cells
```

The following table shows additional commands available for changing, reporting, and removing placement attractions. For more information, see the command man pages.

Table 9 Commands Related to Placement Attraction

To do this	Use this command
Define a placement attraction	create_placement_attraction
Add cells to an existing placement attraction	add_to_placement_attraction
Remove cells from an existing placement attraction	remove_from_placement_attraction
Report placement attractions	report_placement_attractions
Find existing placement attractions	get_placement_attractions
Remove existing placement attractions	remove_placement_attractions

See Also

- [Defining Move Bounds](#)

Specifying Locations for Unmapped Cells

You can specify fixed locations for unmapped leaf cells, so the tool does not move these cells during compile. To do so, specify the location coordinates for leaf cells by using the `set_cell_location` command.

To prevent the tool from moving a cell during compile, specify a fixed location for the cell by using the `-fixed` option with the `set_cell_location` command to set the `physical_status` attribute to `fixed` on the cell. To change the location of a cell marked with fixed placement status, specify the `-ignore_fixed` option with the command. The

specified coordinates indicate the lower-left corner of the cell boundary. After you run the `set_cell_location` command, the unmapped cell gets the specified location in memory, but the location is not reflected in the layout view before compile. To view the placement of the cell, run the `compile_fusion` command.

This example sets the lower-left corner of the `out1_reg` cell to (20 10) and sets the fixed physical placement status to prevent the tool from moving the cell during compile.

```
fc_shell> set_cell_location -coordinates { 20 10 } out1_reg -fixed
```

Defining Cell Spacing Constraints for Legalization

Cell spacing constraints control the spacing between a standard cell and another standard cell or a boundary (the chip boundary, a hard macro, a hard macro keepout margin, a hard placement blockage, or a voltage area guard band). You assign library cells to groups (the boundaries are in a predefined group named `SNPS_BOUNDARY`), and then define the required spacing between cells in these groups.

By default, there are no spacing constraints between standard cells during legalization. To enhance yield, you can define the valid spacing between standard cells or between a standard cell and a boundary.

Note:

The support for spacing constraints between standard cells and power or ground nets depends on how the net is represented in the block. If the power or ground net is defined as a complete blockage, the legalizer and the `check_legality` command ignore spacing rule violations between standard cells and the power or ground net. If the power or ground net is defined as a partial blockage, the legalizer and the `check_legality` command check for spacing rule violations between standard cells and the power or ground net.

Cell spacing constraints are implemented by attaching labels, which are similar to attributes, to the left and right sides of library cells, assuming that the cell is in its north orientation, and specifying the invalid spacings between these labels.

To define cell spacing constraints,

1. Add labels to the library cells that have spacing constraints by using the `set_placement_spacing_label` command.

You must specify the following information for each label:

- The label name (the `-name` option)
- The library cells to which to apply the label (the `-lib_cells` option)
- The sides of the library cells to which to apply the label (the `-side` option, which can take a value of `right`, `left`, or `both`)

The label definitions are additive; you can specify the same label to be used on the right side of some cells and the left side of other cells. For example, to assign a label named X to the right side of the cellA library cell and the left side of the cellB and cellC library cells, use the following commands:

```
fc_shell> set_placement_spacing_label -name x \
    -lib_cells {cellA} -side right
fc_shell> set_placement_spacing_label -name x \
    -lib_cells {cellB cellC} -side left
```

You can assign multiple labels to a side of a library cell. For example, to assign labels named Y and Z to the right side of the cellB library cell, use the following commands:

```
fc_shell> set_placement_spacing_label -name y \
    -lib_cells {cellB} -side right
fc_shell> set_placement_spacing_label -name z \
    -lib_cells {cellB} -side right
```

2. Define the spacing requirements between the labels by using the `set_placement_spacing_rule` command.

You must specify the following information for each rule:

- The labels being constrained (the `-labels` option)

You must specify exactly two labels in each `set_placement_spacing_rule` command. You can specify any of the labels defined by the `set_placement_spacing_label` command or the predefined SNPS_BOUNDARY label, which includes the chip boundary, hard macro boundaries, a hard macro keepout margins, hard placement blockages, and voltage area guard bands. The two labels can be the same or different.

- The range of invalid spacings, in number of unit tiles

For example, to specify that there must be at least one unit tile between labels X and Y (they cannot abut), use the following command:

```
fc_shell> set_placement_spacing_rule -labels {X Y} {0 0}
```

To specify that there must be at least one unit tile between X labels and any boundary, use the following command:

```
fc_shell> set_placement_spacing_rule \
    -labels {X SNPS_BOUNDARY} {0 0}
```

To specify that two X labels cannot have a spacing of two unit tiles, use the following command:

```
fc_shell> set_placement_spacing_rule -labels {X X} {2 2}
```

To specify that labels X and Z must have a spacing of less than two unit tiles or more than four unit tiles, use the following command:

```
fc_shell> set_placement_spacing_rule -labels {X Z} {2 4}
```

Caution:

The cell spacing constraints are not saved with the block; they apply only to the current session and must be redefined in each session.

Reporting Cell Spacing Constraints

To report the cell spacing rules, use the `report_placement_spacing_rules` command. This command reports the cell spacing labels defined in the current session, the library cells (and their sides) to which they apply, and the rules defined for them.

Removing Cell Spacing Constraints

To remove spacing rules, use the `remove_placement_spacing_rules` command. You must specify which rules to remove.

- To remove all spacing rules, use the `-all` option.
- To remove a specific label and all rules associated with that label, use the `-label` option to specify the label.
- To remove a rule between labels, use the `-rule` option to specify two labels associated with the rule.

Specifying Placement Settings

The tool performs coarse placement at various points during the preroute stage of the design flow. The following topics describe settings for controlling coarse placement:

- [Performing Placement With Inaccurate Constraints at Early Stages](#)
- [Generating Automatic Group Bounds for Clock Gating Cells](#)
- [Controlling the Placement Density](#)
- [Controlling Congestion-Driven Restructuring During Placement](#)
- [Reducing Congestion](#)

- Considering Wide Cell Density During Placement
- Considering the Effects of Cell Pins During Placement
- Considering the Congestion Effects Due to the Nondefault Routing Rules of Clock Nets
- Considering the Effects of Clock Gating Cells of Sequential Arrays During Placement
- Considering Legalization Effects During Placement
- Considering DFT Connections During Placement
- Considering the Dynamic Power QoR During Placement
- Performing IR-Drop-Aware Placement
- Spreading Repeater Cells During Placement

Performing Placement With Inaccurate Constraints at Early Stages

During the early stages of a design cycle, placement constraints can be inaccurate causing the tool to exit during coarse placement. To continue with placement when a region is overutilized due to it being too small or being covered by blockages, set the `place.coarse.handle_early_data` application option to `true`. When you do so, the tool issues warning messages as shown in the following example output:

```
Warning: Utilization of move_bound_0 is 109%. (PLACE-089)
Warning: Placement continues with over utilized regions in the design.
         (PLACE-083)
Warning: Overutilized regions for move_bound_0 are modified by removal of
         partial and full blockages. (PLACE-084)
```

When you enable this feature, the tool prints a warning message if the utilization of a region is more than 90 percent. To change this threshold, use the `place.coarse.utilization_warning_threshold` application option.

Generating Automatic Group Bounds for Clock Gating Cells

The tool can automatically generate group bounds for integrated clock gating cells and the sequential cells they drive. To do so, use the following application option setting:

```
fc_shell> set_app_options -name place.coarse.icg_auto_bound \
    -value true
```

When you enable this feature, the tool creates the automatic group bounds at the beginning of placement and removes them at the end of placement. The tool does not include cells that already belong to another group bound in the automatic group bounds.

To limit the maximum number of fanouts that can be included in an automatic bound, use the `icg_auto_bound_fanout_limit` application option setting, as shown in the following example:

```
fc_shell> set_app_options \
    -name place.coarse.icg_auto_bound_fanout_limit -value 30
```

The default fanout limit is 40.

Controlling the Placement Density

You can control the placement density of a block as described in the following table.

Table 10 Application Options for Controlling Placement Density

To do this	Use this application option
Specify a maximum density that controls how densely the tool can place cells during non-congestion-driven placement	<code>place.coarse.auto_density_control</code> (Default is 0, and the tool follows a preset schedule during the flow if <code>place.coarse.auto_density_control</code> is not set to <code>false</code> . Disabling <code>place.coarse.auto_density_control</code> or manually setting <code>place.coarse.max_density</code> to 0 causes uniform spreading of the cells in the core area)
Specify a maximum utilization that controls how densely the tool can place cells in less congested areas that surround highly congested areas, so that the cells in the congested areas can be spread out to reduce the congestion	<code>place.coarse.congestion_driven_max_util</code> (Default is 0.93)

When specifying the maximum density or maximum utilization, choose a value between 1 and the overall utilization of the block. For example, if the utilization of a block is 40 percent, you can choose values between 1 and 0.4, as shown in the following example:

```
fc_shell> set_app_options -name place.coarse.max_density -value 0.6
fc_shell> set_app_options \
    -name place.coarse.congestion_driven_max_util -value 0.8
```

If you do not specify a value for the maximum density or maximum utilization, the tool automatically derives a value for the maximum density and maximum utilization based on the stage of the design flow, considering the `place.coarse.auto_density_control` application option is not set to `false`.

By default, the `place.coarse_auto_density_control` application option is enabled and set to `enhanced`. This improves the total power and wire length for the design.

Note:

It is not necessary to disable the `place.coarse.auto_density_control` application option to override the `max_density` and `congestion_driven_max_util` values. User settings always take precedence over the tool derived values.

Throughout placement, the tool prints PLACE-027 information messages indicating the values it uses for these settings.

Information: Automatic density control has selected the following settings: `max_density 0.60, congestion_driven_max_util 0.77.` (PLACE-027)

The tool applies the maximum density and congestion-driven maximum utilization settings independently to each placeable area such as a voltage area or exclusive move bound, rather than taking an average over the entire block.

Controlling Congestion-Driven Restructuring During Placement

By default, the tool restructures nets to reduce congestion during

- The `create_placement` command, except when you use the `-timing_driven` option
- The `initial_place` stage of the `compile_fusion` command

You can control the congestion-driven restructuring as follows:

- Specify a restructuring strategy by using the `place.coarse.cong_restruct_strategy` application option.

To perform

- Multiple iterations of net restructuring interleaved with incremental placement, set this application option to `original`
- Multiple iterations of net restructuring embedded within wire-length-driven coarse placement, set this application option to `embed`, which is the default

- Specify the number of iterations by using the `place.coarse.cong_restruct_iterations` application option.

If you do not specify the number of iterations, the tool performs

- Three iterations, if the `place.coarse.cong_restruct_strategy` application option is set to `original`
- One iteration, if the `place.coarse.cong_restruct_strategy` application option is set to `embed`

- Specify an effort level by setting the `place.coarse.cong_restruct_effort` application option to `low`, `medium` (default), `high`, or `ultra`.
- Prevent the tool from increasing the path depth by more than three levels of logic by setting the `place.coarse.cong_restruct_depth_aware` application option to `true`.

Reducing Congestion

To reduce the congestion of your block, use the following settings:

- Consider the congestion of each layer separately and improve the accuracy of congestion reduction in coarse placement by setting the `place.coarse.congestion_layer_aware` application option to `true`.

By default, the tool combines the congestion information of all layers during placement. By considering the congestion of each layer separately, the tool can identify and fix areas with high pin densities that have insufficient resources on the lower layers for making pin connections.

- Expand the virtual area of cells during placement based on the local routing resource needs by setting the `place.coarse.increased_cell_expansion` application option to `true`.

By increasing the virtual area of cells in areas where there is a shortage of routing resources, the tool can minimize the cell density and congestion.

By default, the tool expands the cells in the horizontal direction, which helps reduce congestion in the vertical routing layers. For designs with congestion mainly in the horizontal routing layers, set the `place.coarse.congestion_expansion_direction` application option to `both`. The default is `horizontal`.

By default, the tool uses the global route congestion map to identify highly congested areas that need cell expansion. However, the tool does not consider the congestion due to soft routing rules. To consider the congestion effects of soft routing rules, set the `route.global.export_soft_congestion_maps` application option to `true`.

- Consider the legalization requirements of the cells during coarse placement by setting the `place.coarse.legalizer_driven_placement` application option to `true`.

Considering Wide Cell Density During Placement

If a cell cannot straddle the vertical power straps and the width of the cell is more than half the pitch of the power straps, only one such cell can be placed between the power straps. Therefore, during placement, the tool must minimize the density of such wide cells to ensure that they can be placed and legalized without large displacements.

For advanced technology nodes with wide cells, enable wide-cell modeling during placement by setting the `place.coarse.wide_cell_use_model` application option to `true`.

Considering the Effects of Cell Pins During Placement

When you specify a technology node setting of `7`, `7+`, `5`, `s5`, or `s4` by using the `set_technology -node` command, you can specify that the tool uses a technology-specific pin-cost model during placement by setting the `place.coarse.pin_cost_aware` application option to `true`. The default is `false`. When you enable this feature, during placement, the tool tries to improve pin accessibility by using a technology-specific model to predict the routing resources required to access each pin.

For all other technology nodes, you can control the maximum local pin density during placement by setting the `place.coarse.pin_density_aware` application option to `true`. The default is `false`.

Table 11 Settings for Controlling the Effects of Pins During Placement

<code>set_technology -node</code>	<code>place.coarse.pin_cost_aware</code>	<code>place.coarse.pin_density_aware</code>	The tool does this
<code>7, 7+, 5, s5, or s4</code>	<code>true</code>	<code>true or false</code>	Performs technology-specific pin-cost-aware placement
<code>7, 7+, 5, s5, or s4</code>	<code>false</code>	<code>true</code>	Performs pin-density-aware placement
<code>7, 7+, 5, s5, or s4</code>	<code>false</code>	<code>false</code>	Does not consider the effects of cell pins during placement
Any other setting	<code>true or false</code>	<code>true</code>	Performs pin-density-aware placement
Any other setting	<code>true or false</code>	<code>false</code>	Does not consider the effects of cell pins during placement

Considering the Congestion Effects Due to the Nondefault Routing Rules of Clock Nets

To improve timing and reduce crosstalk, nondefault routing rules are used extensively for routing clock nets. However, these nondefault routing rules require additional space, which can increase the congestion after clock routing.

To consider the effects of the nondefault routing rules of the clock nets during placement, set the `place.coarse.ndr_area_aware` application option to `true`.

Considering the Effects of Clock Gating Cells of Sequential Arrays During Placement

A sequential array, which is commonly used in storage devices, is a group of registers that are arranged as a two-dimensional array. The clock pins of the registers along a row of a sequential array are usually driven by a single clock-gating cell.

To consider the clock-gating cells of a sequential array during placement, set the `place.coarse.seq_array_icg_aware` application option to `true`. Doing so can reduce the congestion caused by the clock nets.

Considering Legalization Effects During Placement

To minimize many cells from being displaced during legalization, the tool should be aware of legalization limits and restrictions during coarse placement. For example, the tool should be aware that it cannot place a wide cell in a specific location due to the position of the power straps at that location.

To consider legalization effects during placement, set the `place.coarse.enhanced_legalizer_driven_placement` application option to `true`. Use this feature for blocks that have many cells with small to medium displacements during legalization and use the RMS displacement values reported during legalization to see if it reduces the displacement.

Considering DFT Connections During Placement

To improve the DFT logic placement by enhancing the tool's awareness of DFT connectivity, while still optimizing for functional placement, set the `place.coarse.enable_dft_modeling` application option to `auto`. The default is `false`.

Considering the Dynamic Power QoR During Placement

To consider the dynamic power QoR during placement, perform the following steps:

1. Annotate switching activity on the design, as described in [Annotating the Switching Activity](#).
2. Ensure that at least one scenario is enabled for dynamic-power optimization by using the `set_scenario_status -dynamic_power true` command.

3. Enable power-driven placement for the `create_placement`, `refine_placement`, `compile_fusion`, or `clock_opt` command by using one of the following methods:
 - Enable low-power-placement by setting the `place.coarse.low_power_placement` application option to `true`.
During low-power placement, the tool tries to minimize the length of high-switching nets to improve the power QoR. However, the tool does not consider the effect on the timing QoR.
 - Enable dynamic-power-driven placement by setting the `place.coarse.enhanced_low_power_effort` application option to `low`, `medium`, or `high`. The default is `none`.
During dynamic-power-driven placement, the tool tries to improve both the timing and power of timing-critical nets and the power of the other nets. This improves the power QoR without affecting the timing QoR.

Performing IR-Drop-Aware Placement

During placement, the tool can use the voltage (IR) drop values of cells to identify areas of high power density and spread the cells with high voltage drop values, which reduces the power density of such areas.

To perform IR-drop-aware placement, use the following steps:

1. Place and legalize the block.
2. Set up for RedHawk Fusion and perform static or dynamic voltage drop analysis by using the `analyze_rail -voltage_drop` command as shown in the following example:

```
fc_shell> source redhawk_setup.tcl
fc_shell> analyze_rail -voltage_drop static -nets {VDD VSS}
```

For more information, see [Performing Voltage Drop Analysis](#).

3. Enable IR-drop-aware placement by setting the `place.coarse.ir_drop_aware` application option to `true`.
4. (Optional) Specify additional settings for IR-drop-aware placement, as described in [Controlling IR-Drop-Aware Placement](#).
5. Rerun placement.

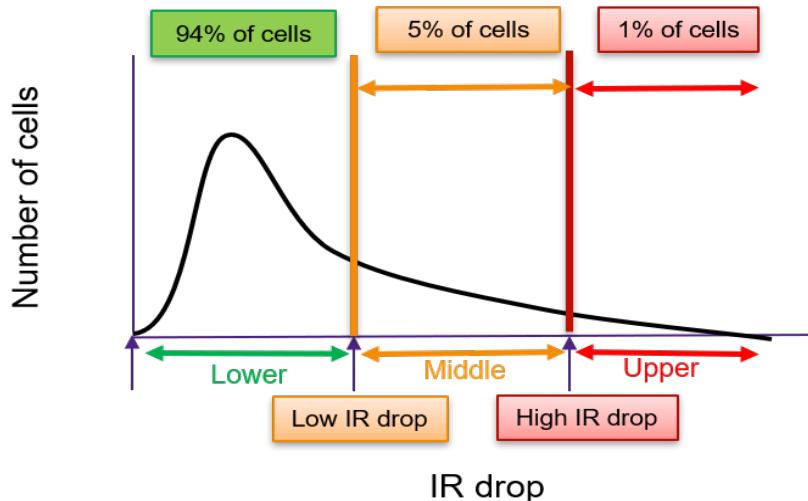
Note:

To perform IR-drop-aware placement, you must have Digital-AF and SNPS_INDESIGN_RH_RAIL license keys.

Controlling IR-Drop-Aware Placement

When you enable IR-drop-aware placement by setting the `place.coarse.ir_drop_aware` application option to `true`, the tool creates three categories of cells based on the total number of cells.

Figure 22 Default Cell Categories for IR-Drop-Aware Placement



By default, the tool selects cells for the three categories by using the criteria shown in [Figure 22](#).

- The upper category consists of the top one percent of the total number of cells with the highest voltage drop values. During placement, the tool spreads these cells the most.
- The middle category consists of the next five percent of the cells. During placement, the tool spreads these cells less than those in the upper category.
- The lower category consists of the rest of the cells. The tool does not spread these cells.

You can change the percentage of cells in the upper and middle categories by using the `place.coarse.ir_drop_default_target_high_percentage` and `place.coarse.ir_drop_default_target_low_percentage` application options.

The following example puts the top 2 percent of cells in the upper category and the next 6 percent of cells in the middle category:

```
fc_shell> set_app_options \
    -name place.coarse.ir_drop_default_target_high_percentage \
    -value 2.0
fc_shell> set_app_options \
```

```
-name place.coarse.ir_drop_default_target_low_percentage \
-value 6.0
```

Instead of using a percentage of the total number of cells to define the cell categories, you can use the cell voltage drop as a percentage of supply voltage by setting the `place.coarse.ir_drop_default_target_high_population` and `place.coarse.ir_drop_default_target_low_population` application options to false. They are true by default.

The following example puts cells with a voltage drop larger than 8 percent of the supply voltage into the upper category and cells with a voltage drop between 4 to 8 percent of the supply voltage into the middle category:

```
fc_shell> set_app_options \
    -name place.coarse.ir_drop_default_target_high_population \
    -value false
fc_shell> set_app_options \
    -name place.coarse.ir_drop_default_target_high_percentage \
    -value 8.0
fc_shell> set_app_options \
    -name place.coarse.ir_drop_default_target_low_population \
    -value false
fc_shell> set_app_options \
    -name place.coarse.ir_drop_default_target_low_percentage \
    -value 4.0
```

You can specify a percentage of the total number of cells to define the upper category and cell voltage drop as a percentage of supply voltage to define the middle category, or vice versa.

The following example puts the top 2 percent of the total cells in the upper category, and the next cells that have a voltage drop larger than 4 percent of supply voltage in the middle category:

```
fc_shell> set_app_options \
    -name place.coarse.ir_drop_default_target_high_population \
    -value true
fc_shell> set_app_options \
    -name place.coarse.ir_drop_default_target_high_percentage \
    -value 2.0
fc_shell> set_app_options \
    -name place.coarse.ir_drop_default_target_low_population \
    -value false
fc_shell> set_app_options \
    -name place.coarse.ir_drop_default_target_low_percentage \
    -value 4.0
```

You can set constraints for specific voltage areas by using the `set_placement_ir_drop_target` command. The following example puts the top 1.5

percent of the cells in the VA1 voltage area into the upper category and the next 4.5 percent of the cells into the middle category:

```
fc_shell> set_placement_ir_drop_target VA1 high 1.5
fc_shell> set_placement_ir_drop_target VA1 low 4.5
```

The following example puts the cells with a voltage drop larger than 10 percent of the supply voltage in the VA2 voltage area into the upper category and the cells with a voltage drop between 5 and 10 percent of the supply voltage into the middle category:

```
fc_shell> set_placement_ir_drop_target VA2 high 10 -irdrop
fc_shell> set_placement_ir_drop_target VA2 low 5 -irdrop
```

To get, report, and reset the voltage-area-based constraints, use the `get_placement_ir_drop_target`, `report_placement_ir_drop_target`, and `reset_placement_ir_drop_target` commands.

Spreading Repeater Cells During Placement

If a block has many chains of repeater cells (buffers, inverters, or pipelined registers) along the edges or corners of macro cells or blockages, the tool might clump the cells along the edges or corners, increasing congestion. You can reduce this congestion by setting the `place.coarse.spread_repeater_paths` application option to `true` before you run the `create_placement`, `compile_fusion`, or `clock_opt` command. Then, the tool reduces congestion along the edges and corners by spreading the repeater cells in an orthogonal direction.

Specifying Legalization Settings

The tool performs legalization at various points during the design flow. The following topics describe settings for controlling legalization:

- [Minimizing Large Displacements During Legalization](#)
- [Optimizing Pin Access During Legalization](#)
- [Enabling Advanced PG Net Checks](#)
- [Enabling Advanced Legalization Algorithms](#)
- [Setting Up for Variant-Aware Legalization](#)
- [Checking if Library Cells Are Legally Placeable](#)

Minimizing Large Displacements During Legalization

To minimize large displacements of cells during legalization, enable

- Orientation optimization by setting the `place.legalize.optimize_orientations` application option to `true`.

When you do so, the tool considers flipping the orientations of cells to reduce the displacements during legalization.

- Stream placement by setting the `place.legalize.stream_place` application option to `true`.

When you do so, the tool moves many cells a small distance, to avoid moving a single cell a large distance during legalization. You can further control stream placement by using the `place.legalize.stream_effort` and `place.legalize.stream_effort_limit` application options.

Optimizing Pin Access During Legalization

For advanced technology nodes, to improve the routability of areas with high pin densities by redistributing the cells, enable pin optimization during legalization by setting the `place.legalize.optimize_pin_access_using_cell_spacing` application option to `true`.

Enabling Advanced PG Net Checks

For advanced technology nodes, such as 12 nanometer, 7 nanometer, or smaller technology nodes, you can enable advanced physical DRC checks between cells and prerouted PG nets by setting the `place.legalize.enable_prerouted_net_check` application option to `true`.

When you enable this feature, you can further control the accessibility checks performed for standard cell pins by using the `place.legalize.advanced_layer_access_check` and `place.legalize.advanced_libpin_access_check` application options. However, the default behavior of this feature is suitable for most designs.

Enabling Advanced Legalization Algorithms

To enable advanced legalization algorithms for 2D rule checking and cell interaction, which can reduce legalization runtime, set the `place.legalize.enable_advanced_legalizer` application option to `true`.

To specify additional advanced legalization rules that are not automatically detected, set the `place.legalize.enable_advanced_legalizer_rules` application option.

When you enable the advanced legalization algorithms, by default, the tool aligns vertical pin shapes during pin access optimization to improve routability. However, you can specify additional strategies for pin access optimization by setting the `place.legalize.optimize_pin_access_strategies` application option as shown in the following table.

Table 12 Settings for the place.legalize.optimize_pin_access_strategies Application Option

To do this	Use this setting
Align horizontal pin shapes	<code>horizontal_align</code>
Move cells away from areas of high pin density	<code>avoid_high_pin_density</code>
Align horizontal pin shapes and move cells away from areas of high pin density	<code>horizontal_align avoid_high_pin_density</code>

During advanced legalization and legality checking, the tool can simultaneously check for DRC violations between multiple cells and PG net shapes. To enable this feature, set the `place.legalize.enable_multi_cell_pnet_checks` application option to `true`.

To enable multi cell checks at the last run of `route_opt` command or after the `route_auto` command, set the following application options to `true`. This helps to improve the runtime for QoR considerations. To enable the following application options, set the `place.legalize.enable_multi_cell_pnet_checks` application option to `true`:

- `place.legalize.enable_multi_cell_access_check`: considers impact on a cell's pin access due to presence of abutting neighbor cells and any prerouted net PG net shapes in the vicinity. Default is `false`.
- `place.legalize.enable_multi_cell_track_capacity_check`: analyzes the track capacity of the cell's pins and abutting neighbor cells. Track capacity estimates the sufficiency of routing tracks in the area to access all the specified pins. To enable this application option, set the `place.legalize.enable_multi_cell_access_check` application option to `true`. Default is `false`.

To enable multithreaded advanced legalization, perform the following steps:

1. Enable multithreaded advanced legalization algorithms by setting the `place.legalize.enable_threaded_advanced_legalizer` application option to `true`.
2. Configure for multithreading as described in [Configuring Multithreading](#).

To perform multithreaded advanced legality checking,

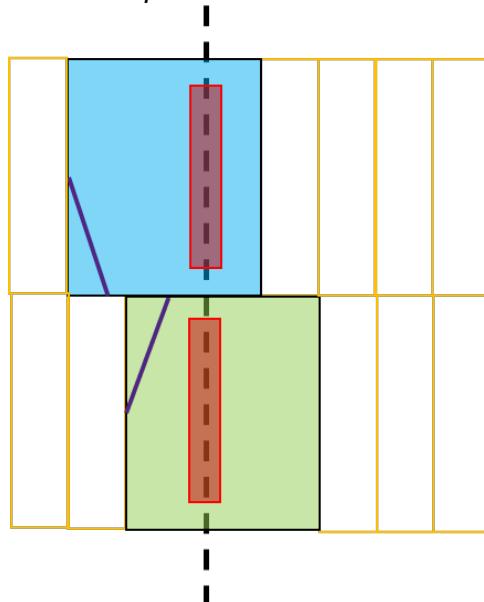
1. Enable advanced legalization algorithms by setting the `place.legalize.enable_advanced_legalizer` application option to `true`.
2. Configure for multithreading as described in [Configuring Multithreading](#).
3. Perform legality checking using the `check_legality` command.

Setting Up for Variant-Aware Legalization

Some libraries provide sets of functionally equivalent cells that can be legalized in different locations. This allows the tool to legalize a cell by replacing it with a variant, instead of moving the cell.

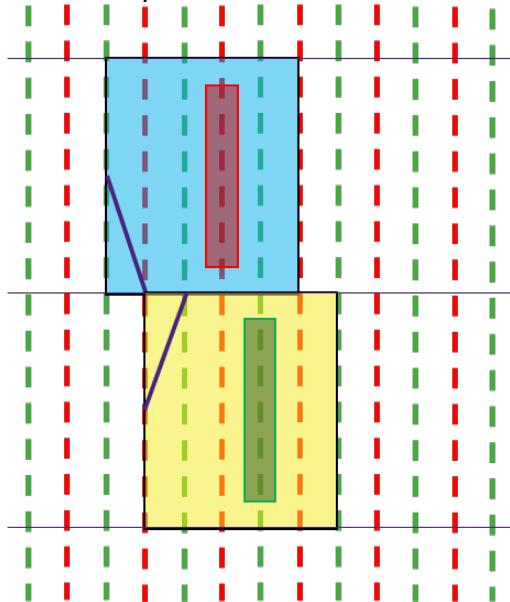
The following figure shows functionally equivalent cells with varying pin locations. With variant-aware legalization, if a cell pin does not align with a track, the tool can try using its variant to align the cell pin to the track.

Figure 23 Equivalent Cells With Different Pin Locations



The following figure shows functionally equivalent cells with different pin colors, which need to be aligned to a track of the same color. With variant-aware legalization, if a cell pin does not align with the corresponding colored track, the tool can try using its variant with a different colored pin and try aligning it to the appropriately colored track.

Figure 24 Equivalent Cells With Pins of Different Colors



The following topics describe the tasks you need to perform to set up the reference libraries and design for variant-aware legalization:

- [Defining Equivalent Cell Groups](#)
- [Enabling Variant-Aware Legalization](#)

Defining Equivalent Cell Groups

The Library Compiler tool supports creating variant-ready libraries from the Liberty source files. The `physical_variant_cells` attribute in the Liberty source defines the variants of a cell. When the `physical_variant_cells` attribute exists in the logic libraries, the compiled library is variant ready.

If your logic library is not variant ready, you can define the equivalent cell variants during the library preparation stage in the Library Manager tool. To do so, create equivalent cell groups by using the `create_cell_groups` command. The following example adds equivalent cell groups to an existing reference library in the Library Manager tool:

```
lm_shell> create_workspace -flow edit LIB.ndm
lm_shell> create_cell_group -name A [get_lib_cells MY_LIB/A_*]
lm_shell> create_cell_group -name B [get_lib_cells MY_LIB/B_*]
lm_shell> check_workspace
lm_shell> commit_workspace
```

You can report the equivalent cell groups in a library by using the `report_cell_groups` command in either the Library Manager tool or the Fusion Compiler tool. This command reports equivalent cell groups that are defined in Liberty source files of variant-ready logic

libraries as well as those defined by using the `create_cell_groups` command in the Library Manager tool.

Enabling Variant-Aware Legalization

To enable variant-aware legalization, set the `place.legalize.enable_variant_aware` application option to `true`.

When variant-aware legalization is enabled, the tool tries to fix legalization DRC violations by swapping the violating cell with a variant, rather than moving it. If the cell has no variants, it is moved to a legal location.

For pin-track and pin-color alignment during variant-aware legalization, set the following application options to `true`:

- `place.legalize.enable_prerouted_net_check`
- `place.legalize.enable_pin_color_alignment_check`

In addition, you must specify the layers to be aligned by using the `place.legalize.pin_color_alignment_layers` application option.

Checking if Library Cells Are Legally Placeable

After you specify all your placement and legalization constraints and settings, you can check if specific library cells can be legally placed in the block by using the `analyze_lib_cell_placement -lib_cells` command.

- To limit the analysis to a specific region of the core area, use the `-region` option. By default, the command searches the entire core area.
- To limit the analysis to a specific number of sites, use the `-trials` option. By default, the command analyzes 1000 random sites to see if the specified library cells can be placed. If you specify a value of 0, the command analyzes all free sites, which can increase runtime.
- To ignore physical design constraints or advanced design rules during placement analysis, use the `-no_pdc` or `-no_adv` option.
- To report only the relative placement groups that do not meet a specific threshold, use the `-threshold` option. The tool reports a cell only if the percentage of sites the cell can be placed, relative to the total number of sites analyzed, is less than the specified threshold.
- To limit the report to a maximum number of cells, use the `-max_cells` option. The default is 100, starting with the library cell with the worst pass rate.

If your block contains library cells that have a very low pass rate, they will cause large displacements and increased runtime during legalization. In such cases, review the power plan, cell layout, and the legalization settings to improve the pass rate.

The following example analyzes all the library cells that have instances in the current block:

```
fc_shell> analyze_lib_cell_placement -lib_cells [add_to_collection \
-unique "" [get_attribute [get_cells -physical_context] ref_phys_block]]
```

Analyzing 215 lib cell(s).

Warning: Routing direction of metal layer PO is neither "horizontal" nor "vertical". PDC checks will not be performed on this layer. (PDC-003)

PDC app_options settings =====

place.legalize.enable_prerouted_net_check:	1
place.legalize.num_tracks_for_access_check:	1
place.legalize.use_eol_spacing_for_access_check:	0
place.legalize.allow_touch_track_for_access_check:	1
place.legalize.reduce_conservatism_in_eol_check:	0
place.legalize.preroute_shape_merge_distance:	0.0

Layer M1: cached 0 shapes out of 869 total shapes.
Layer M2: cached 773 shapes out of 773 total shapes.
Cached 41757 vias out of 99466 total vias.
0.0%...2.0%...4.0%...6.0%...8.0%...10.0%.....

Lib Cell	Pass Rate
saed32_lvt_lsup:LSUPX1_LVT.frame	0.4450
saed32_lvt_lsup:LSUPX8_LVT.frame	0.4780
saed32_lvt_lsup:LSUPX2_LVT.frame	0.4810
saed32_lvt_lsup:LSUPX4_LVT.frame	0.5040
.....	
.....	
.....	
saed32_rvt_std:HADDX1_RVT.frame	0.8600
saed32_lvt_std:OA21X1_LVT.frame	0.8610

Controlling the Optimization of Cells, Nets, Pins, and Ports

The following topics describe how you can control the optimization of cells, nets, pins, and ports:

- [Resolving Multiple References With the Uniquify Process](#)
- [Preserving Cells and Nets During Optimization](#)
- [Restricting Optimization to Cell Sizing Only](#)
- [Preserving Networks During Optimization](#)

- [Marking the Clock Networks](#)
 - [Disabling Design Rule Checking \(DRC\)](#)
 - [Preserving Pin Names During Sizing](#)
 - [Preserving Ports of Existing Hierarchies](#)
 - [Isolating Input and Output Ports](#)
 - [Fixing Multiple-Port Nets](#)
 - [Controlling the Addition of New Cells to Modules, Hierarchical Cells, and Voltage Areas](#)
 - [Specifying a Cell Name Prefix for Optimization](#)
-

Resolving Multiple References With the Uniquify Process

The `uniquify` command resolves multiple references, except those with the `dont_touch` attribute, throughout the hierarchy in the current design.

The `uniquify` process copies and renames any multiply referenced design so that each instance references a unique design. The process removes the original design from memory after it creates the new and unique designs. The original design and any collections that contain it or its objects are no longer accessible. After this process finishes, the tool optimizes each design copy based on the unique context of its cell instance.

To control the names of the reference copies, use the `design.uniquify_naming_style` application option. The default is `%s_%d`, where `%s` denotes the name of the existing reference and `%d` denotes the smallest integer value that forms a unique design name.

This example creates unique reference copies of the specified cells A1, A2, and A3.

```
fc_shell> uniquify {A1 A2 A3}
3
```

This example creates unique reference copies for the filtered cells A and B.

```
fc_shell> uniquify [get_cells {A B}]
2
```

Preserving Cells and Nets During Optimization

To preserve design objects during optimization, use the `set_dont_touch` command. The command places the `dont_touch` attribute on cells, nets, references, and subdesigns in the current design to prevent these objects from being modified or replaced during optimization.

This table shows how the tool preserves the design objects during optimization.

Design objects	Tool behavior
Cells	Excludes the cells from optimization.
Nets	Preserves the connectivity of all pins to the nets.
Modules, designs, library cells	Prevents optimization of all instances that reference them.
Hierarchical cells or modules	Excludes the cells and all child cells from optimization.

When you use the `set_dont_touch` command, keep the following in mind:

- The command prevents the ungrouping of hierarchy.
- A child cell must inherit the same `dont_touch` attribute from its parent module. For example, you cannot remove the `dont_touch` attribute from a child cell when its parent module is marked with the `dont_touch` attribute.
- Use the command on fully mapped logic only, including cells, modules, and designs, as well as nets with fully mapped surrounding logic.
- The `dont_touch` attribute has higher precedence over the `boundary_optimization` and the `size_only` attributes.

To report design objects marked with the `dont_touch` attribute, use the `report_dont_touch` command.

This command sets the `dont_touch` attribute on cell A.

```
fc_shell> set_dont_touch [get_cells A] true
```

To remove the `dont_touch` attribute, use the `remove_attributes` command or the `set_dont_touch` command set to `false`, as shown in the following two commands:

```
fc_shell> remove_attributes [get_cells A] dont_touch
fc_shell> set_dont_touch [get_cells A] false
```

The following table summarizes the commands that you can use to set, remove, or report the `dont_touch` attribute on design objects, including modules, cells, nets, and library cells. Yes denotes the command operation of the `dont_touch` attribute is supported, while No denotes the command operation is not supported.

Table 13 Command Support for the *dont_touch* Attribute on Design Objects

Command	Module	Cell (Instance)	Net	Library cell
set_dont_touch	Yes	Yes	Yes	Yes
set_attribute	Yes	Yes	Yes	Yes
define_user_attribute	No	No	No	No
get_attribute	Yes	Yes	Yes	Yes
remove_attributes	Yes	Yes	Yes	Yes
report_dont_touch	Yes	Yes	Yes	No
set_dont_touch <i>unmapped_object</i>	No	No	Yes	NA

Restricting Optimization to Cell Sizing Only

To allow only sizing on cells or instances during optimization, use the `set_size_only` command. The command places the `size_only` attribute on specified cells or instances to prevent them from being modified or replaced except cell sizing.

To query the `size_only` attribute, use the `report_attributes`, `get_attribute`, or `report_cells` command. To remove the attribute, use the `remove_attributes` command or the `set_size_only` command set to `false`.

This example sets the `size_only` attribute on a specific cell.

```
fc_shell> set_size_only [get_cells cell_name] true
```

This example queries the `size_only` attribute on the `cell_name` cell.

```
fc_shell> get_attribute [get_cells cell_name] size_only
```

This example reports all the `size_only` information on design objects in the current design.

```
fc_shell> report_size_only -all
```

Preserving Networks During Optimization

You can preserve networks, such as clock trees, during optimization by using the `set_dont_touch_network` command. The command places the `dont_touch` attribute

on the clocks, pins, or ports to prevent the cells and nets in the transitive fanout of the network from being modified or replaced.

By default, the command preserves both clock paths and clock as data paths and propagates the `dont_touch` attribute throughout the hierarchy of the network. To preserve clock paths only, specify the `-clock_only` option. To remove the `dont_touch` attribute from the network, specify the `-clear` option.

This example sets the `dont_touch` attribute on the CLK clock network.

```
fc_shell> set_dont_touch_network [get_clocks CLK]
```

This example sets the `dont_touch` attribute on the CLK clock paths only.

```
fc_shell> set_dont_touch_network [get_clocks CLK] -clock_only
```

This example removes the `dont_touch` attribute from the CLK clock network.

```
fc_shell> set_dont_touch_network [get_clocks CLK] -clear
```

Marking the Clock Networks

If a block does not contain clock trees, you should perform optimization using ideal clocks.

To mark all clock networks as ideal, use the following command:

```
foreach_in_collection mode [all_modes] {
    current_mode $mode
    set_ideal_network [all_fanout -flat -clock_tree]
}
```

To model the clock tree effects for placement, you should also define the uncertainty, latency, and transition constraints for each clock by using the `set_clock_uncertainty`, `set_clock_latency`, and `set_clock_transition` commands.

Before performing clock tree synthesis, you must use the `remove_ideal_network` command to remove the ideal setting on the fanout of the clock trees.

Disabling Design Rule Checking (DRC)

You can disable design rule checking (DRC) on clock, constant, scan enable, and scan clock nets by using the `set_auto_disable_drc_nets` command.

To control design rule checking on specific nets in the current design, use the options with the `set_auto_disable_drc_nets` command as shown in [Table 14](#).

Table 14 Options for the `set_auto_disable_drc_nets` Command

Use this option	To do this
<code>-none</code>	Enable DRC for all nets.
<code>-all</code>	Disable DRC on all clock, constant, scan enable, and scan clock nets.
<code>-constant true false</code>	Disable (set to <code>true</code>) or enable (set to <code>false</code>) DRC on constant nets.
<code>-on_clock_network true false</code>	Disable (set to <code>true</code>) or enable (set to <code>false</code>) DRC on clock networks.
<code>-scan true false</code>	Disable (set to <code>true</code>) or enable (set to <code>false</code>) DRC on scan enable and scan clock nets.

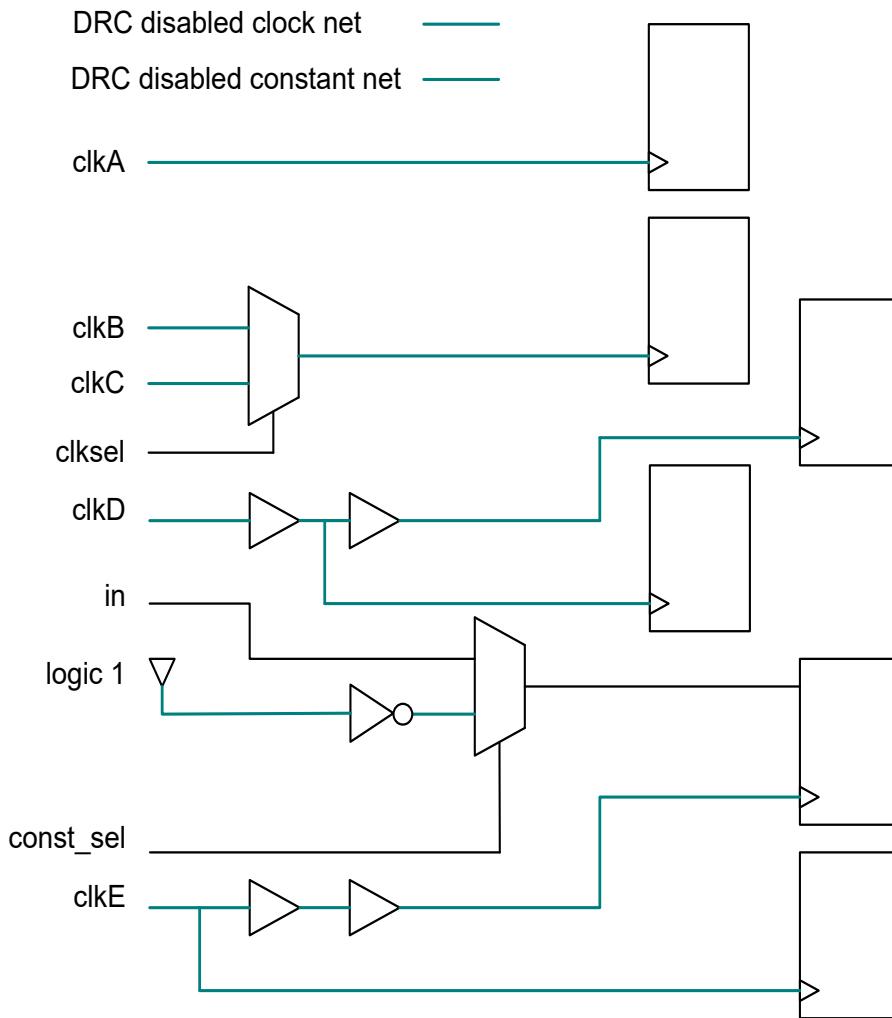
For example, to enable design rule checking for all clock, constant, scan enable, and scan clock nets, specify the `-none` option:

```
fc_shell> set_auto_disable_drc_nets -none
```

To disable design rule checking on nets connecting to clocks and constants, set the `-on_clock_network` and `-constant` options to `true`, as shown in the following example and figure:

```
fc_shell> set_auto_disable_drc_nets \  
      -on_clock_network true -constant true
```

Figure 25 DRC Disabled Clock and Constant Nets Highlighted



Preserving Pin Names During Sizing

By default, optimization can change the pin names of leaf cells during sizing if the functionality of the resulting circuit is equivalent. The Fusion Compiler tool automatically updates its version of the internal constraints to reflect the new pin name if the pin is part of an exception constraint. If this occurs, it means that the original constraints used to constrain the block in the Fusion Compiler tool cannot be used for signoff purposes; instead, you must write out the constraints from the Fusion Compiler tool and include these constraints with the resulting block.

This behavior provides the optimization engine with the most flexibility to select cells and improve the cost functions. You can restrict this sizing capability so that the constraints remain unchanged through optimization by setting the `opt.common.preserve_pin_names`

application option. This application option defaults to the setting of `never`, and accepts values of either `never` or `always`.

To restrict sizing to pin-name equivalent cells, use the following command:

```
fc_shell> set_app_options \
    -name opt.common.preserve_pin_names -value always
```

To restore the default behavior, use the following command:

```
fc_shell> set_app_options \
    -name opt.common.preserve_pin_names -value never
```

Preserving Ports of Existing Hierarchies

During optimization and clock tree synthesis, the tool can create new ports on existing hierarchical blocks. To prevent the tool from doing so, use the `set_freeze_ports` command and set the `freeze_clock_port` attribute to `true` on the corresponding cell instance.

You can prevent the tool from adding clock ports, data ports, or both by using the `-clock`, `-data`, or `-all` option. For example, to prevent the tool from creating additional clock ports on the MBX22 cell instance, use the following command:

```
fc_shell> set_freeze_ports -clock [get_cells MBX22] true
```

To report the freeze-port settings, use the `report_freeze_ports` command.

Isolating Input and Output Ports

You can isolate input and output ports to improve the accuracy of timing models. To insert isolation logic at specified input or output ports, use the `set_isolate_ports` command.

Isolation logic can be either a buffer or a pair of inverters; by default, the command places buffers to isolate

- An input port from its fanout networks
- An output port from its driver

When you specify the `-force` option with the `set_isolate_ports` command, the tool ensures that the library cells specified by the `-driver` option are not sized.

Note that the tool does not place isolation logic on the following ports:

- Bidirectional ports
- Ports defined as clock sources or power pins
- Ports connected to nets that are marked with the `dont_touch` attribute

Example

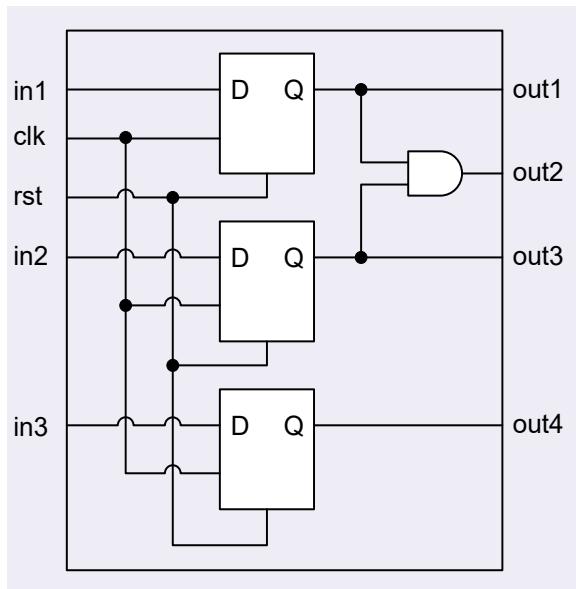
The following commands insert isolation logic to

- Input port in3 with a buffer
- Output port out1 with a buffer
- Output port out3 with the LIBCELL_BUFF library cell
- Output port out4 with a pair of inverters

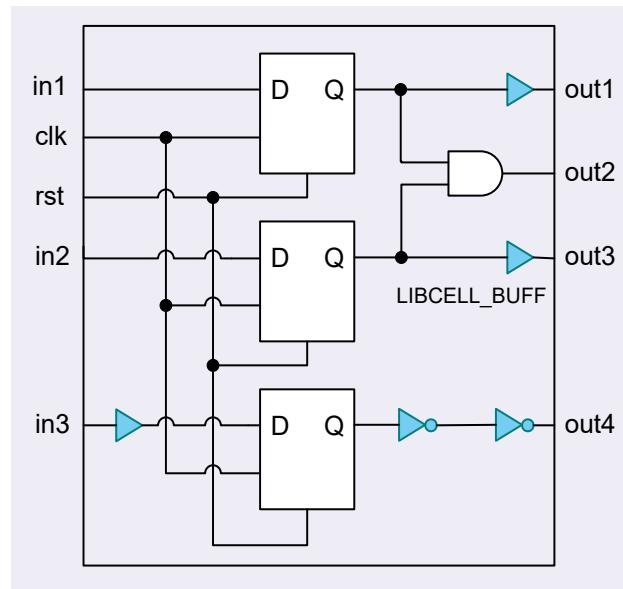
```
fc_shell> set_isolate_ports {in3 out1}
fc_shell> set_isolate_ports -driver LIBCELL_BUFF out3 -force
fc_shell> set_isolate_ports -type inverter out4
```

The following figures show the design without and with the isolation logic inserted by the `set_isolate_ports` commands, including buffers and an inverter pair:

Without port isolation



After running the port isolation commands



Fixing Multiple-Port Nets

Multiple-port nets include

- Feedthrough nets, where an input port feeds into an output port
- Nets connected to multiple output ports, logically equivalent outputs



Multiple-port nets, which are represented with the `assign` statement in the gate-level netlist, might cause design rule violations further in the design flow. By default, the tool does not fix multiple-port nets during optimization.

You can enable multiple-port-net fixing for

- The `logic_opto` and `initial_drc` stages, which are earlier stages of the `compile_fusion` command, by using the `set_fix_multiple_port_nets` command
This command sets the `fix_multiple_port_nets` attribute on the specified objects.
- The `initial_opto` and `final_opto` stages, which are later stages of the `compile_fusion` command, by setting the `opt.port.eliminate_verilog_assign` application option to `true`
This application option is `false` by default.

If you want to fix multiple-port nets, use the `set_fix_multiple_port_nets` command and fix these issues during the early stages of the `compile_fusion` command, when the tool can perform more optimization techniques.

If you do not explicitly specify a multiple-port-net setting using the `set_fix_multiple_port_nets` command, but you set the `opt.port.eliminate_verilog_assign` application option to `true`, the tool issues the following message and uses the `set_fix_multiple_port_nets -all -buffer_constants` command setting to fix multiple-port nets during the `logic_opto` and `initial_drc` stages of the `compile_fusion` command:

```
Information: The opt.port.eliminate_verilog_assign application option has
been
set to true. The tool will run set_fix_multiple_port_nets -all
-buffer_constant by
internally enabling MPN fixing due to the absence of explicit
set_fix_multiple_port_nets
constraint. (MPN-0004)
```

However, if you explicitly specify a multiple-port-net setting using the `set_fix_multiple_port_nets` command, the tool honors it. For example, the following settings enable multiple-port-net fixing for feedthrough nets during the `logic_opto` and

`initial_drc` stages and all nets during the `initial_opto` and `final_opto` stages of the `compile_fusion` command:

```
fc_shell> set_fix_multiple_port_nets -feedthroughs
fc_shell> set_app_options \
    -name opt.port.eliminate_verilog_assign -value true
```

The following settings disable multiple-port-net fixing for all nets during the `logic_opto` and `initial_drc` stages and enables it for all nets during the `initial_opto` and `final_opto` stages of the `compile_fusion` command:

```
fc_shell> set_fix_multiple_port_nets -default
fc_shell> set_app_options \
    -name opt.port.eliminate_verilog_assign -value true
```

During multiple-port-net fixing, the tool performs the following tasks in sequence:

1. Rewires the connections to ensure that each net is connected to one hierarchical port, if possible.
 The tool first performs rewiring to avoid unnecessary buffering. The tool also rewrites across the hierarchy because boundary optimization is enabled by default.
2. Inserts buffers or inverters to the nets that are not fixed by rewiring.

Controlling the Addition of New Cells to Modules, Hierarchical Cells, and Voltage Areas

To control the addition of new cells to

- Modules or hierarchical cells, use the `set_allow_new_cells` command
- Voltage areas, use the `-allow_new_cells` option with the `create_voltage_area_rule` command

You can use these settings to prevent new cells from being added to the top-level hierarchies and voltage areas of abutted designs. These settings are saved in the design library and honored by the tool throughout the implementation flow.

The following example prevents new cells from being added to the top-level module:

```
fc_shell> set_allow_new_cells \
    [get_attribute [current_block] top_module] false
```

The following example prevents new cells from being added to the M1 module:

```
fc_shell> set_allow_new_cells [get_modules M1] false
```

The following example prevents new cells from being added to the U22 hierarchical cell:

```
fc_shell> set_allow_new_cells [get_cells U22] false
```

The following example prevents new cells from being added to the VA1 voltage area:

```
fc_shell> create_voltage_area_rule -name VA1_rule \
    -allow_new_cells false -voltage_areas VA1
```

Specifying a Cell Name Prefix for Optimization

You can specify a name prefix for the cells added on the data nets during optimization by using the `opt.common.user_instance_name_prefix` application option.

The following example specifies a name prefix of

- CF_ for the cells added on the data nets during the `compile_fusion` command
- CO_ for the cells added on the data nets during the `clock_opt` command
- RO_ for the cells added on the data nets during the `route_opt` command

```
fc_shell> set_app_options \
    -name opt.common.user_instance_name_prefix -value "CF_"
fc_shell> compile_fusion

fc_shell> set_app_options \
    -name opt.common.user_instance_name_prefix -value "CO_"
fc_shell> clock_opt
fc_shell> set_app_options \
    -name opt.common.user_instance_name_prefix -value "PO_"
fc_shell> route_opt
```

To specify a name prefix for the cells added on the clock network during clock tree synthesis, use the `cts.common.user_instance_name_prefix` application option, as described in .

Specifying Settings for Preroute Optimization

Optimizing the design for performance, power, and area (PPA) is one of the primary goals of the tool. The following topics describe how to specify settings for performance, power, and area optimization at the preroute stage:

- [Specifying Parasitic Estimation Settings for the Preroute Optimization](#)
- [Specifying Automatic Via Ladder Insertion Settings for Preroute Optimization](#)
- [Assigning Nondefault Routing Rules to Critical Nets](#)

- Enabling Area Recovery in Regions of High Utilization
- Enabling Advanced Logic Restructuring

Specifying Parasitic Estimation Settings for the Preroute Optimization

The following topics describe how you can control parasitic estimation during the preroute stage of the design flow:

- Enabling Global-Route-Layer-Based (GRLB) Preroute Optimization
- Enabling Route-Driven Estimation (RDE) for Preroute Optimization

Enabling Global-Route-Layer-Based (GRLB) Preroute Optimization

To improve correlation with the postroute stage of the design flow, you can enable global-route-layer-based RC estimation during the `compile_fusion` and `clock_opt` commands. With this feature, the tool uses global routes for all nets and identifies the layers with the most appropriate per-unit resistance values. The nets are then constrained to minimum and maximum layers for preroute RC estimation.

To enable this feature, use the `opt.common.use_route_aware_estimation` application option. If you set it to

- `auto`, the feature is enabled only when there is a variation in the per-unit resistance value of the different routing layers
- `true`, the feature is enabled irrespective of the variation in the per-unit resistance value of the different routing layers

If you enable this feature, use the `remove_route_aware_estimation` command to remove all global route based estimation before you perform routing.

Enabling Route-Driven Estimation (RDE) for Preroute Optimization

To improve correlation with the postroute stage of the design flow, the tool can perform global routing, perform extraction based on the global routes, and use this parasitic information when performing optimization.

If the tool detects that the technology used by the design is an advanced technology node that is less than 16 nm, the tool enables this feature by default. To enable it for any technology, set the `opt.common.enable_rde` application option to `true`.

When enabled, the tool performs route-driven parasitic estimation during the `final_opt` stage of the `compile_fusion` and `clock_opt` commands. The parasitic information is

stored in the design library and used during subsequent optimization steps. If you enable this feature, you should enable it for all subsequent preroute optimization steps in the design flow.

During route-driven estimation, the tool honors the capacitance and resistance scaling factors specified with the `-early_cap_scale`, `-late_cap_scale`, `-early_res_scale`, and `-late_res_scale` options of the `set_extraction_options` command.

When route-driven estimation is enabled, the tool ignores the setting of the `opt.common.use_route_aware_estimation` application option, which enables global-route-layer-based (GRLB) RC estimation.

Specifying Automatic Via Ladder Insertion Settings for Preroute Optimization

A via ladder is a stacked via that starts from the pin layer and extends into the upper routing layers. Using via ladders during optimization improves the performance and electromigration robustness of a design. The tool can automatically insert via ladders for cell pins on timing-critical paths during the `compile_fusion` and `clock_opt` commands.

To perform via ladder insertion during preroute optimization,

1. Ensure that the via ladder rules are defined as described in [Defining Via Ladder Rules](#).
2. Specify the via ladders that can be used for specific library pins by using the `set_via_ladder_candidate` command, as described in [Specifying Via Ladder Candidates for Library Pins](#).
3. (Optional) Enable high-performance and electromigration via ladder insertion for critical paths by setting the `opt.common.enable_via_ladder_insertion` application option to `true`.
4. (Optional) Enable the insertion of global-route-based via ladders on pins with via ladder constraints by setting the `route.global.insert_gr_via_ladders` application option to `true`.
5. Perform optimization using the `compile_fusion` or `clock_opt` command.

Specifying Via Ladder Candidates for Library Pins

To specify the valid via ladders that can be used for a given library pin during preroute optimization, use the `set_via_ladder_candidate` command. You can specify only one via ladder each time you use the `set_via_ladder_candidate` command. To specify multiple via ladder candidates for a library pin, use the command multiple times, as shown in the following example:

```
fc_shell> set_via_ladder_candidate [get_lib_pins lib1/AND2/A1] \
-ladder_name "VP4"
```

```
fc_shell> set_via_ladder_candidate [get_lib_pins lib1/AND2/A1] \
    -ladder_name "VP2"
```

The order in which you specify multiple via ladders for the same library pin indicates the priority to use when selecting a via ladder to insert for that pin. Specify the via ladder candidates starting with the highest priority. In the previous example, the tool gives the VP4 via ladder priority over the VP2 via ladder.

Note:

If both the `pattern_must_join` attribute and a via ladder candidate apply to a pin, the via ladder has a higher priority than the pattern-based must-join connection when both methods have the same estimated delay.

To specify that a library pin requires an electromigration via ladder, set the `is_em_via_ladder_required` pin attribute to `true`, as shown in the following example:

```
fc_shell> set_attribute [get_lib_pins lib1/INV4/A] \
    is_em_via_ladder_required true
```

If you set the `is_em_via_ladder_required` pin attribute to `true` for a specific library pin, you must specify an electromigration via ladder as a candidate with the `set_via_ladder_candidate` command.

A via ladder can be identified as an electromigration via ladder by using one of the following methods:

- By using the `forElectromigration=1` construct when defining the via rule in the technology file, as shown in the following example:

```
ViaRule "EMVP1" {
    ...
    ...
    ...
    forElectromigration=1
}
```

- By setting the `for_electro_migration` attribute to `true`, as shown in the following example:

```
fc_shell> set_attribute [get_via_rules EMVP1] \
    for_electro_migration true
```

See Also

- [Defining Via Ladder Rules](#)

Assigning Nondefault Routing Rules to Critical Nets

To improve timing QoR, you can enable the tool to automatically assign nondefault routing rules to nets with critical timing.

When you enable this feature, the tool

- Chooses candidate nets based on slack and length
- Uses timing-driven optimization to assign nondefault routing rules to the nets
- Optimizes buffering for these nets

To enable the feature, follow these steps.

1. Define nondefault routing rules by using the `create_routing_rule` command.
For details, see [Defining Nondefault Routing Rules](#).
2. Report routing rules by using the `report_routing_rules` command.
3. Specify a prioritized list of nondefault routing rules for automatic assignment by using the `opt.common.optimize_ndr_user_rule_names` application option.
4. Enable the automatic assignment of nondefault routing rules during optimization by setting the `compile.flow.optimize_ndr` application option.

The following script example creates two nondefault routing rules, `ndr_1w2s` and `ndr_2w3s`, and automatically assigns these nondefault routing rules to timing-critical nets during optimization:

```
create_routing_rule "ndr_1w2s" -multiplier_spacing 2
create_routing_rule "ndr_2w3s" -multiplier_spacing 3 -multiplier_width 2
# Enables feature, defines priority
set opt.common.optimize_ndr_user_rule_names { ndr_1w2s ndr_2w3s }
set compile.flow.optimize_ndr
compile_fusion
```

Controlling Nondefault Routing Rule Assignment

To specify a critical range for nondefault routing rule for optimization, set the `compile.flow.optimize_ndr_critical_range` application option to a value between 0 and 1. The default is 0.05. The specified value y , as shown in the following example, is used to calculate the slack range $[y^*WNS, WNS]$ where WNS is the worst negative slack. Nets in the timing paths that have a slack value within the range are considered for nondefault routing rule assignment.

```
fc_shell> set_app_options \
    -name compile.flow.optimize_ndr_critical_range -value y
```

To set the maximum number of nets considered for automatic nondefault routing rules assignment, set the `compile.flow.optimize_ndr_max_nets` application option to an integer. The default is 2000. For example,

```
fc_shell> set_app_options \
    -name compile.flow.optimize_ndr_max_nets -value 5000
```

When both the `compile.flow.optimize_ndr_critical_range` and `compile.flow.optimize_ndr_max_nets` application options are specified, the stricter constraint has priority.

Enabling Area Recovery in Regions of High Utilization

For designs that cannot be legalized due to areas of high utilization, you can specify that the tool perform area recovery in regions where the utilization is high by setting the `opt.common.small_region_area_recovery` application option to `true`.

When you enable this feature, the tool performs the area recovery during the `compile_fusion` and `clock_opt` commands. However, doing so can slightly degrade the timing QoR.

Enabling Advanced Logic Restructuring

To improve the area, timing, and power QoR, you can enable advanced logic restructuring. To enable this feature for the `final_opto` stage of the `compile_fusion` and `clock_opt` commands, use the `opt.common.advanced_logic_restructuring_mode` application option, as shown in the following table.

Table 15 Settings for the `opt.common.advanced_logic_restructuring_mode` Application Option

To do this	Use this setting
Perform no restructuring	none (default)
Perform restructuring to improve the area QoR	area
Perform restructuring to improve the timing QoR	timing
Perform restructuring to improve the power QoR	power
Perform restructuring to improve the area and timing QoR	area_timing
Perform restructuring to improve the timing and power QoR	timing_power
Perform restructuring to improve the area and power QoR	area_power
Perform restructuring to improve the area, timing, and power QoR	area_timing_power

When restructuring, the tool does not

- Restructure across logical hierarchy
- Consider cells with `dont_touch`, `size_only`, or `fixed` attributes
- Accept the results if it increases the wire length

However, if your design does not have congestion or routing issues, you can specify that the tool accepts the restructuring results even if it increases the wire length by setting the `opt.common.advanced_logic_restructuring_wirelength_costing` application option to `medium` or `none`. The default is `high`.

Setting Up for Power-Related Features

The following topics describe the tasks you need to perform for setting up a design for power-related features:

- [Annotating the Switching Activity](#)
- [Enabling Leakage Power Optimization for the `compile_fusion` Command](#)
- [Enabling Power Optimization for the `clock_opt` Command](#)
- [Improving Yield By Limiting the Percentage of Low-Threshold-Voltage \(LVT\) Cells](#)
- [Updating Activity for Improved Power Optimization](#)
- [Enabling the Power Integrity Features](#)

Annotating the Switching Activity

When performing low-power placement and dynamic-power optimization, you obtain better power savings if you annotate switching activity on the design. You can annotate the switching activity in the following ways:

- Read a switching activity file (SAIF) by using the `read_saif` command.

The switching activity is scenario specific. So, when you use this command, ensure that the current scenario is enabled for dynamic power optimization.

- Set the switching activity by using the `set_switching_activity` command.

When you use this command, you can set the switching activity for a specific mode, corner, or scenario by using the `-mode`, `-corner`, or `-scenario` options. When doing so, ensure that the scenarios you specify or the scenarios corresponding to the modes and corners you specify are enabled for dynamic power optimization.

If you do not specify the switching activity, the tool applies the default toggle rate to the primary inputs and black box outputs and then propagates it throughout the design.

To report the switching activity of a block, use the `report_activity` command. To remove switching activity of specific nets, pins, ports, or the entire block, use the `reset_switching_activity` command.

Using RTL Switching Activity With a Name-Mapping File

To use an RTL SAIF file in the Fusion Compiler tool, use the following steps:

1. After reading and elaborating the design, use the `saif_map -start` command, which creates a name-mapping database during synthesis that the tool then uses for power analysis and optimization.
2. (Optional) Manually change the name mapping as described in [Controlling the Name Mapping](#).
3. Read in the same RTL SAIF file by using the `read_saif` command. Reading an RTL SAIF file does not affect the name-mapping database.
4. Perform optimization and other design changes.
5. Generate a binary name-mapping file that maps RTL object names to gate-level objects by using the `saif_map -write_map` command, which is needed if you want to continue the flow using ASCII files in a later Fusion Compiler session. In the subsequent session, use the `saif_map -read_map` command to read the saved file and begin tracking further changes to object names.
6. (Optional) Generate an ASCII name-mapping file for use with the PrimePower tool by using the `saif_map -write_map -type primepower -essential` command.

Controlling the Name Mapping

In the SAIF-map flow, the name-mapping database tracks the changes to pin, port, cell, and net names that occur during optimization. It contains the original name for each object, so that the original SAIF can be applied even after optimization.

You can manually control the name mapping in the database by using the `saif_map` command as shown in the following table.

Table 16 Options for Manually Controlling the Name Mapping

To do this	Use this option of the <code>saif_map</code> command
Specify a name mapping for an object and overwrite the name mapping that might exist in the database	<code>-set_name</code>

Table 16 Options for Manually Controlling the Name Mapping (Continued)

To do this	Use this option of the <code>saif_map</code> command
Add a name mapping to the existing mapping in the database	<code>-add_name</code>
Apply the name mapping specified with the <code>-set_name</code> or <code>-add_name</code> option to the logically inverted object	<code>-inverted</code> , with <code>-set_name</code> or <code>-add_name</code>
Change an object name using a name rule, when reading in the SAIF file	<code>-change_name</code>
Report all manual name-mapping settings specified	<code>-report</code>
Remove the name mapping settings for specific objects	<code>-remove_name</code>
Reset the name-mapping settings by clearing all manually created name mappings and name rules	<code>-reset</code>
Retrieve name-mapping settings from the database	<code>-get_saif_names</code> , <code>-get_object_names</code>

Scaling the Switching Activity

If the clock frequency used in the SAIF file is different from the clock frequency used in the Fusion Compiler tool, you can scale the switching activity by performing the following steps:

1. Read in the SAIF file by using the `read_saif` command.
2. Enable scaling by setting the `power.use_generated_clock_scaling_factor` application option to `on`.
3. Scale the switching activity by using the `set_power_clock_scaling` command.

When you use this command, you must specify the following:

- The clock objects associated with the switching activity you want to scale
- The clock period used in the SAIF file by using the `-period` option or the ratio between the clock period used in the SAIF file and the clock period used in the Fusion Compiler tool by using the `-ratio` option

In addition, you can specify the scenario for which to apply the scaling by using the `-scenario` option.

The following example reads in a SAIF file, enables scaling, scales the switching activity associated with clocks named CLK1 and CLK2 by a ratio of five, and scales the switching activity associated with clock named CLK3 by a ratio of two:

```
fc_shell> read_saif top.saif
fc_shell> set_app_options -list \
    {power.use_generated_clock_scaling_factor true}
fc_shell> set_power_clock_scaling -ratio 5 {CLK1 CLK2}
fc_shell> set_power_clock_scaling -ratio 2 {CLK3}
```

If you run the `set_power_clock_scaling` command again for the same clock, the tool scales the already scaled switching activity.

When you use the `set_power_clock_scaling` command, the tool scales only the switching activity applied with the `read_saif` command. The tool does not scale the following:

- Switching activity applied with the `set_switching_activity` command
- Switching activity within block abstracts

The scaled switching activity is persistent in the design. You can write it out by using the `write_saif` command and use it in the subsequent steps of the design flow.

Specifying Switching Probability for Supply Nets

The leakage power of a block is scaled based on the switching probability of its supply nets, which represents the fraction of time a supply net is in the `on` state. You can specify the switching probability for one or more supply nets by using the `set_supply_net_probability -static_probability` command. By default, the switching probability is applied to the current scenario and the corresponding mode and corner. To specify modes, corners, or scenarios in which to apply the setting, use the `-modes`, `-corners`, or `-scenarios` option. To get the switching probability of a supply net, use the `get_supply_net_probability` command, and to reset the value, use the `reset_supply_net_probability` command.

By default, the tool propagates supply net activity through power switches and determines the static probability of the switched supply net based on the UPF power switch constraint. For example, consider the following UPF power switch constraint:

```
create_power_switch my_switch \
    -output_supply_port {vout VDDS} \
    -input_supply_port {vin VDD} \
    -control_port {ms_sel ctrl1} \
    -control_port {ms_ctrl ctrl2} \
    -on_state {on vin {ms_ctrl && !ms_sel}}
```

The tool derives the static probability of the supply net named VDDS, which is connected to the output of the power switch, based on the probability of the power switch being *on*. This is derived based on the following:

- The Boolean function specified with the `-on_state` option, which is `ms_ctrl && ! ms_sel`, and the switching activity (static probability) of the nets connected to the corresponding control ports, which are nets named `ctrl1` and `ctrl2`.
- The switching probability of the supply net connected to the input supply port specified with the `-on_state` option, which is the supply net named `VDD`.

The following application options control whether dynamic and leakage power are scaled based on the supply switching activity:

- The `power.scale_dynamic_power_at_power_off` option controls whether dynamic power is scaled. The default is `false` (no scaling).
- The `power.scale_leakage_power_at_power_off` option controls whether leakage power is scaled. The default is `true` (scaling is performed).

Enabling Leakage Power Optimization for the `compile_fusion` Command

By default, the tool performs leakage power optimization during compile. To produce optimal leakage power optimization, use libraries with multithreshold voltage group cells.

To set up leakage optimization,

1. Enable at least one scenario for leakage power by using the `set_scenario_status -leakage_power true` command.

For example,

```
fc_shell> set_scenario_status -leakage_power true SC1
```

If the block has multiple scenarios enabled for leakage power, the tool uses the worst-case leakage value of all the specified scenarios.

2. Set the `threshold_voltage_group` attribute on the reference library cells.

For example,

```
fc_shell> set_attribute [get_lib_cells -quiet lib1_name/*] \
    threshold_voltage_group HVT
```

3. Set the `threshold_voltage_group` attribute to one of the three built-in threshold voltage group types: high, normal, or low threshold voltage.

If there are more than three threshold-voltage groups, you should group them into the three voltage group types as follows:

```
fc_shell> set_threshold_voltage_group_type \
-type high_vt {HVT HVTX}
fc_shell> set_threshold_voltage_group_type \
-type low_vt {LVT LVTX}
fc_shell> set_threshold_voltage_group_type \
-type normal_vt {RVT RVTX}
```

Reporting Leakage Power

To check your leakage power, use the `report_power` command. By default, state-dependent leakage power is used to calculate leakage power. To change the leakage power calculation mode, set the `power.leakage_mode` application option. For example, to set the application option to `average` do the following:

```
fc_shell> set_app_options -name power.leakage_mode -value average
```

To report threshold voltage usage, use the `report_threshold_voltage_groups` command. Classification is based on the `threshold_voltage_group` attribute from the reference library cells or the `default_threshold_voltage_group` attribute on the library. You can also set the attributes on the objects by using the `set_attribute` command.

You can generate several types of reports with the `report_threshold_voltage_groups` command. For example, the summary report (using the `-summary` option) lists the number, area, and percentage of cells in each threshold voltage group. The default report (without using any options) shows the cell count and area based on the following groups: repeater, combinational, register, sequential, clock network, and physical-only cells. For the repeater, register, and clock network cell groups, you can obtain additional information by using the `-detailed` option.

The `-hierarchy` option of the `report_threshold_voltage_groups` command generates a report based on the design hierarchy. The report shows the cell count, area, and area ratio for a top-level block and its subblocks. The `-hierarchy` option is mutually exclusive with the other report types, but can be used with the `-datapath_cells_only` and `-standard_cells_only` options.

You can restrict the hierarchy-based report to specific threshold voltage groups by using the `-hier_vt_groups` option. You can also specify to report only the top-level cells that contain more than a specified number of constituent cells by using the `-hier_threshold` option. These options must be used with the `-hierarchy` option.

If you specify a cell list, a separate report is created for each top-level cell in the list.

Enabling Power Optimization for the `clock_opt` Command

The Fusion Compiler tool can optimize both dynamic and static (leakage) power.

- Dynamic power

This is the energy dissipated due to the voltage or logic transitions in the design objects, such as cells, pins, and nets. The dynamic power consumption is directly proportional to the number and frequency of transitions in the design.

- Static (leakage) power

This is the energy dissipated even when there are no transitions in the circuit. This is also known as leakage power and depends on the device characteristics. The main contributor to leakage power is the sub-threshold-voltage leakage in the device. At lower technology nodes, leakage power consumption contributes significantly to the total power consumption of the circuit.

The following topics describe how to enable the different types of power optimization the tool performs:

- [Performing Conventional Leakage-Power Optimization](#)
- [Performing Dynamic-Power Optimization](#)
- [Performing Total-Power Optimization](#)

Performing Conventional Leakage-Power Optimization

To set up for conventional leakage-power optimization during the preroute optimization stage, perform the following:

1. Ensure that at least one scenario is enabled for leakage-power optimization by using the `set_scenario_status -leakage_power true` command.
2. Enable conventional leakage-power optimization by setting the `opt.power.mode` application option to `leakage`.
3. (Optional) Change the effort level for power optimization by setting the `opt.power.effort` application option to `medium` or `high`. The default is `low`.

When you use these settings, the tool performs conventional leakage-power optimization during the `clock_opt` command. However, it does not perform any dynamic-power optimization.

Performing Dynamic-Power Optimization

To set up for dynamic-power optimization during the preroute optimization stage, perform the following:

1. Annotate switching activity on the design, as described in [Annotating the Switching Activity](#).
2. Ensure that at least one scenario is enabled for dynamic-power optimization by using the `set_scenario_status -dynamic_power true` command.
3. Enable dynamic-power optimization by setting the `opt.power.mode` application option to `dynamic`.
4. (Optional) Change the effort level for power optimization by setting the `opt.power.effort` application option to `medium` or `high`. The default is `low`.

When you use these settings, the tool performs dynamic-power optimization during the `clock_opt` command. However, it does not perform any leakage-power optimization.

Performing Total-Power Optimization

Total power optimization considers the combined leakage- and dynamic-power cost during optimization. To setup for total-power optimization,

1. Setup for leakage-power optimization by performing the following steps:
 - a. Ensure that at least one scenario is enabled for leakage-power optimization by using the `set_scenario_status -leakage_power true` command.
2. Setup for dynamic-power optimization by performing the following steps:
 - a. Annotate switching activity on the design, as described in [Annotating the Switching Activity](#).
 - b. Ensure that at least one scenario is enabled for dynamic-power optimization by using the `set_scenario_status -dynamic_power true` command.
3. Enable total-power optimization at the preroute stage by setting the `opt.power.mode` application option to `total`.
4. (Optional) Change the effort level for power optimization by setting the `opt.power.effort` application option to `medium` or `high`. The default is `low`.

When you use this setting, the tool performs total-power optimization during the `clock_opt` command.

Improving Yield By Limiting the Percentage of Low-Threshold-Voltage (LVT) Cells

The Fusion Compiler tool can perform percentage-LVT-based optimization. By limiting the percentage of LVT cells in a block, you can improve yield of your design. You can also use this feature during the early design phases to get an idea of the quality of the RTL.

To set up for percentage-LVT-based optimization,

- Specify the LVT library cell group as shown in the following example:

```
fc_shell> remove_attributes -quiet \
[get_lib_cells -quiet */*] threshold_voltage_group
fc_shell> set_attribute [get_lib_cells -quiet *lvt*/*] \
threshold_voltage_group LVT
fc_shell> set_threshold_voltage_group_type \
-type low_vt LVT
```

Ensure that the LVT library cells you specify

- Have a valid purpose defined by using the `set_lib_cell_purpose` command and are included in the appropriate target-library subsets defined by using the `set_target_library_subset` command
 - Do not have a `dont_touch` or `dont_use` attribute setting
- Specify the percentage limit for LVT cells by using the `set_multi_vth_constraint -low_vt_percentage` command.

To specify the limit as a percentage of the

- Cell count, use the `-cost cell_count` option
- Cell area, use the `-cost area` option

To reset this constraint, use the `reset_multi_vth_constraint` command, and to report it, use the `report_multi_vth_constraint` command.

After you set up percentage-LVT-based optimization, when you run the `compile_fusion`, `clock_opt`, or `route_opt` command, the tool uses the specified LVT cell percentage constraint for the data path cells in the block. However, to minimize QoR disturbance, the tool limits the percentage-LVT-based optimization performed during the `route_opt` command. Therefore, it is important to enable this feature earlier in the design flow.

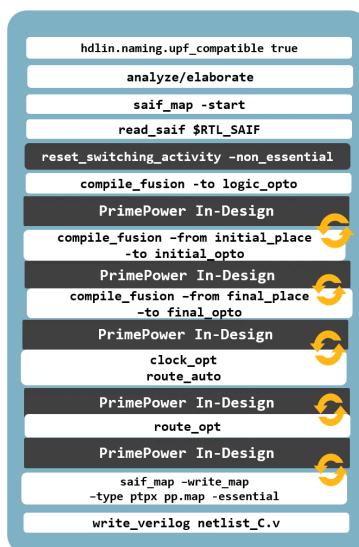
Note:

LVT cells have a smaller cell delay, but higher leakage power dissipation. Therefore, limiting the number of LVT cells reduces leakage-power dissipation. However, to further reduce leakage power, enable leakage power optimization for the different stages of the design flow.

Updating Activity for Improved Power Optimization

The PrimePower In-Design flow in the Fusion Compiler tool uses input waveform information from the RTL FSDB file and performs time-based analysis to update the activity for enhanced power estimation accuracy. [Figure 26](#) shows the PrimePower in-design flow in the Fusion Compiler tool, indicating points in the flow where you can run the PrimePower tool. For the best correlation with standalone PrimePower time-based analysis results, run the PrimePower tool after the `route_opt` command.

Figure 26 The PrimePower In-Design Flow



Use the `set_indesign_primepower_options` command to set PrimePower analysis options for the In-Design flow. You must specify the RTL activity file with the `-fsdb` option and the path to the PrimePower executable with the `-pwr_shell` option. Other options allow you to specify details about the FSDB file, settings for distributed or concurrent processing, settings for scenarios and libraries, and output files.

Use the `update_indesign_activity` command to perform the analysis, which includes the following tasks:

- Invokes the PrimePower tool and reads the design data which includes the netlist, libraries, PVT settings, constraints, parasitics, essential name mapping data, UPF, and so on. Library details come from the design database stored in the Fusion Compiler tool memory.
- Reads the RTL FSDB file into the PrimePower tool and performs time-based analysis to generate a gate-level SAIF.
- Reads the gate-level SAIF back into the Fusion Compiler tool.

Set the power scenario as the current scenario before starting analysis with the `update_indesign_activity` command. Activity is refreshed for the current scenario.

If you specify multiple scenarios using the `-scenarios` option of the `set_indesign_primepower_options` command, the refreshed gate-level SAIF from the PrimePower tool is read back into the Fusion Compiler tool for all of the specified scenarios.

Specifying Distributed Analysis

For faster runtime, use distributed analysis, which is disabled by default in the PrimePower tool.

To enable distributed analysis in the PrimePower tool, specify the following options with the `set_indesign_primepower_options` command:

- `-num_processes`: Use this option to specify the number of hosts to launch. The value must be greater than 1.
- `-submit_command` or `-host_names`: Use either of these options to specify the configuration of host machines.
- `-max_cores`: Use this option to specify the maximum number of cores. The default is 4.

The tool prints the following message if valid options are specified:

```
Information: PrimePower is being called in distributed mode
```

Enabling the Power Integrity Features

The Fusion Compiler tool provides the following techniques for ensuring power integrity of the design:

- Dynamic power shaping (DPS), which reduces transient power by utilizing concurrent clock and data optimization (useful skew) techniques.
- Voltage-drop-aware placement, which use the voltage (IR) drop values of cells to identify areas of high power density and spread the cells with high voltage drop values to reduce the power density of such areas.

To set up for the recommended power integrity flow, perform the following steps:

1. Enable the power integrity flow by setting the `opt.common.power_integrity` application option to `true`.

When you do so, the Fusion Compiler uses the following techniques during the different stages of the design flow:

- a. Dynamic power shaping (DPS) during the `final_opto` stage of the `compile_fusion` command
 - b. Voltage-drop-aware placement during the `clock_opt` command
2. (Optional) Specify that the tool should not reduce the voltage drop at the expense of the timing QoR by reducing the power integrity effort level by setting the `opt.common.power_integrity_effort` application option to `low`.

The default is `high`, and by default the tool tries to reduce the voltage drop to less than eight percent of the supply voltage at the expense of the timing QoR.
 3. (Optional) Specify a maximum voltage drop threshold by using the `opt.common.ir_drop_threshold` application option.

The default is eight percent of the supply voltage. If you change the default effort level by setting the `opt.common.power.integrity_effort` application option to `low`, the tool ignores the threshold specified with the `opt.common.ir_drop_threshold` application option.
 4. Specify settings for dynamic power shaping as described in [Setting Up for Dynamic Power Shaping](#).
 5. Specify settings for voltage-drop-aware placement as described in [Setting Up for Voltage-Drop-Aware Placement](#).
 6. (Optional) Enable IR-driven sizing, which uses the RedHawk dynamic voltage drop analysis results to identify cells involved in voltage drop violations, and then tries to replace those cells with cells having smaller leakage current. To enable this feature, set the `clock_opt.flow.enable_irdrivenopt` application option to `true`, in addition to setting the `opt.common.power_integrity` application option to `true`.

Instead of using the recommended power integrity flow setting, you can manually enable dynamic power shaping or voltage-drop-aware placement for the `compile_fusion` or `clock_opt` commands as described in [Manually Enabling Dynamic Power Shaping and Voltage-Drop-Aware Placement](#)

Setting Up for Dynamic Power Shaping

To set up for dynamic power shaping, perform the following steps:

1. Ensure UPF settings are specified for multiple-power nets.
2. Ensure both setup and hold scenarios are created and are active.
3. (Optional) Control the power analysis and optimization performed during dynamic power shaping as follows:
 - Specify the scenarios to focus on by setting the `ccd.dps.focus_power_scenario` application option.

By default, the tool focuses on all active scenarios. When the design has many active scenarios, specifying one or two scenarios, as shown in this example, can reduce the runtime.

```
fc_shell> set_app_options -name \
    -name ccd.dps.focus_power_scenario -value {{S1 S2}}
```

- Control the type of internal power analysis performed by setting the `ccd.dps.use_case_type` application option.

By default, this application option is set to `autovector`. With this setting, the tool runs internal power analysis with a 20 percent probability of toggling a flip-flop and all flip-flops are effectively clocked every cycle. However, the clock gating is not modeled.

If you set this application option to `activity`, the tool chooses probability of toggling a flip-flop and the clock-gating model based on the activity.

- Define the overall dynamic power optimization goal by setting the `ccd.dps.optimize_power_target_modes` application option.
 - To reduce the global peak current, where the optimization goal is to reduce the total peak current, set this application option to `global_peak`.
 - To reduce the local peak current in windows stepped across the design, where the optimization goal is to reduce voltage drop violations, set this application option to `stepped_psgs`, which is the default.
- Specify the tradeoff between power and setup timing during dynamic power shaping by setting the `ccd.dps.optimize_setup_tradeoff_level` application option to low, medium (default), or high.
- Specify the tradeoff between power and hold timing during dynamic power shaping by setting the `ccd.dps.optimize_hold_tradeoff_level` application option to low, medium (default), or high.

You can further control the power analysis and optimization performed during dynamic power shaping by using application options such as `ccd.dps.use_cases`, `ccd.dps.auto_targets`, and `ccd.dps.auto_target_constraint_parameters`, which are more complex to use. For more information on these application options, see the corresponding man pages.

Setting Up for Voltage-Drop-Aware Placement

To set up for voltage-drop-aware placement, perform the following steps:

1. Specify the voltage-drop-analysis type by using the `rail.analysis_type` application option.

The valid values are `static`, `dynamic` (default), `dynamic_vcd`, and `dynamic_vectorless`.

2. Specify settings required to perform RedHawk voltage drop analysis, as described in [Performing Voltage Drop Analysis](#).

Manually Enabling Dynamic Power Shaping and Voltage-Drop-Aware Placement

To manually enable

- Dynamic power shaping for the `compile_fusion` command, use the following settings:

```
fc_shell> set_app_option -name opt.common.power_integrity \
    -value false
fc_shell> set_app_option -name compile.flow.enable_dps \
    -value true
```

- Voltage-drop-aware placement for the `compile_fusion` command, use the following settings:

```
fc_shell> set_app_option -name opt.common.power_integrity \
    -value false
fc_shell> set_app_option -name compile.flow.enable_irap \
    -value true
```

- Voltage-drop-aware placement for the `clock_opt` command, use the following settings:

```
fc_shell> set_app_option -name opt.common.power_integrity \
    -value false
fc_shell> set_app_option -name clock_opt.flow.enable_irap \
    -value true
```

Specifying the Routing Resources

You can specify the minimum and maximum routing layers both for the block (global layer constraints), for specific nets (net-specific layer constraints), and for unsynthesized clock nets (clock-tree layer constraints). If you specify both global layer constraints and net-specific layer constraints, the net-specific constraints override the global constraints. In addition to constraining the routing layers, you can also specify a preferred routing direction for each layer. The routing layer constraints are used by RC estimation, congestion analysis, and routing. Because these constraints affect RC estimation and congestion analysis as well as routing, you should set these constraints before performing placement on the block.

The following topics describe how to specify routing layer constraints, preferred routing direction, and via ladders :

- [Specifying the Global Layer Constraints](#)
- [Specifying Net-Specific Layer Constraints](#)
- [Specifying Clock-Tree Layer Constraints](#)
- [Setting the Preferred Routing Direction for Layers](#)

Specifying the Global Layer Constraints

To specify the global layer constraints, use the `set_ignored_layers` command. By default, the global layer constraints are used for RC estimation, congestion analysis, and as soft constraints for routing. Use the following options to set the constraints and change the default behavior.

- To specify the minimum and maximum routing layers, use the `-min_routing_layer` and `-max_routing_layer` options. Specify the routing layers by using the layer names from the technology file.

For example, to use layers M2 through M7 for routing, RC estimation, and congestion analysis, use the following command:

```
fc_shell> set_ignored_layers \
           -min_routing_layer M2 -max_routing_layer M7
```

- To allow the use of layers beyond the minimum or maximum routing layer only for pin connections, set the `route.common.global_min_layer_mode` and `route.common.global_max_layer_mode` application options to `allow_pin_connection`.

- To change the constraints to hard constraints, set the `route.common.global_min_layer_mode` and `route.common.global_max_layer_mode` application options to `hard`.
- To specify additional layers to be ignored for RC estimation and congestion analysis, use the `-rc_congestion_ignored_layers` option.

The specified layers must be between the minimum and maximum routing layers. For example, to use layers M2 through M7 for routing and layers M3 through M7 for RC estimation and congestion analysis, use the following command:

```
fc_shell> set_ignored_layers \
    -min_routing_layer M2 -max_routing_layer M7 \
    -rc_congestion_ignored_layers {M2}
```

- To change an existing layer constraint setting, simply reset that option.

When you reset an option, it overrides the existing value of only that option; the other option settings remain unchanged.

For example, assume that you used the previous command to set the minimum routing layer to M2 and the maximum routing layer to M7. To change the maximum routing layer from M7 to M8, but keep the other settings, use the following command:

```
fc_shell> set_ignored_layers -max_routing_layer M8
```

Reporting Global Layer Constraints

To report the ignored layers, use the `report_ignored_layers` command. For example,

```
fc_shell> report_ignored_layers
*****
Report : Ignored Layers
Design : my_design
Version: J-2014.12
Date   : Wed Oct 22 15:58:23 2014
*****
Layer Attribute          Value
-----
Min Routing Layer        M2
Max Routing Layer        M7
RC Estimation Ignored Layers    PO M1 M2 M8 M9 MRDL
1
```

Removing Global Layer Constraints

To remove the global layer constraints, use the `remove_ignored_layers` command. You must specify one or more of the following options:

- `-min_routing_layer`

This option removes the minimum routing layer setting. When you remove the minimum routing layer setting, it also removes the ignored layers for RC estimation and congestion analysis that are below the minimum routing layer.

- `-max_routing_layer`

This option removes the maximum routing layer setting. When you remove the maximum routing layer setting, it also removes the ignored layers for RC estimation and congestion analysis that are above the maximum routing layer.

- `-all`

This option removes the ignored layers for RC estimation and congestion analysis that are between the minimum and maximum routing layers.

- `-rc_congestion_ignored_layers layer_list`

This option removes the specified ignored layers for RC estimation and congestion analysis. The specified layers must be between the minimum and maximum routing layers.

Specifying Net-Specific Layer Constraints

To specify net-specific layer constraints, use the `set_routing_rule` command. To specify the minimum and maximum routing layers, use the `-min_routing_layer` and `-max_routing_layer` options. Specify the routing layers by using the layer names from the technology file.

For example, to use layers M2 through M7 when routing the n1 net, use the following command:

```
fc_shell> set_routing_rule [get_nets n1] \
    -min_routing_layer M2 -max_routing_layer M7
```

By default, net-specific minimum layer constraints are soft constraints, while net-specific maximum layer constraints are hard constraints. You can change the default behavior as follows:

- To change the default constraint strength for all net-specific layer constraints, set the `route.common.net_min_layer_mode` and `route.common.net_max_layer_mode` application options.

Set these application options to `soft` to specify a soft constraint, `allow_pin_connection` to allow the use of lower layers only for pin connections, or `hard` to specify a hard constraint.

- To change the constraint strength for a single net-specific layer constraint, use the `-min_layer_mode` and `-max_layer_mode` options when you define the constraint with the `set_routing_rule` command.

Set these options to `soft` to specify a soft constraint, `allow_pin_connection` to allow the use of lower layers only for pin connections, or `hard` to specify a hard constraint.

- To set the cost of violating soft constraints for all net-specific layer constraints, set the `route.common.net_min_layer_mode_soft_cost` and `route.common.net_max_layer_mode_soft_cost` application options.

Set these application options to `low`, `medium` (the default), or `high`. The cost setting controls the effort expended by the router to avoid violations. The `high` setting can reduce violations at a cost of increased runtime. The `low` setting can reduce runtime at a cost of increased violations.

- To set the cost of violating soft constraints for a single net-specific layer constraint, use the `-min_layer_mode_soft_cost` and `-max_layer_mode_soft_cost` options when you define the constraint with the `set_routing_rule` command.

Set these options to `low`, `medium` (the default), or `high`. The cost setting controls the effort expended by the router to avoid violations. The `high` setting can reduce violations at a cost of increased runtime. The `low` setting can reduce runtime at a cost of increased violations.

Removing Net-Specific Routing Layer Constraints

To remove net-specific routing layer constraints, use the `set_routing_rule -clear` command. Note that when you use this command, it also removes any nondefault routing rules assigned to the specified nets.

Specifying Clock-Tree Layer Constraints

To specify clock-tree layer constraints, use the `set_clock_routing_rules` command. To specify the minimum and maximum routing layers, use the `-min_routing_layer` and

`-max_routing_layer` options. Specify the routing layers by using the layer names from the technology file.

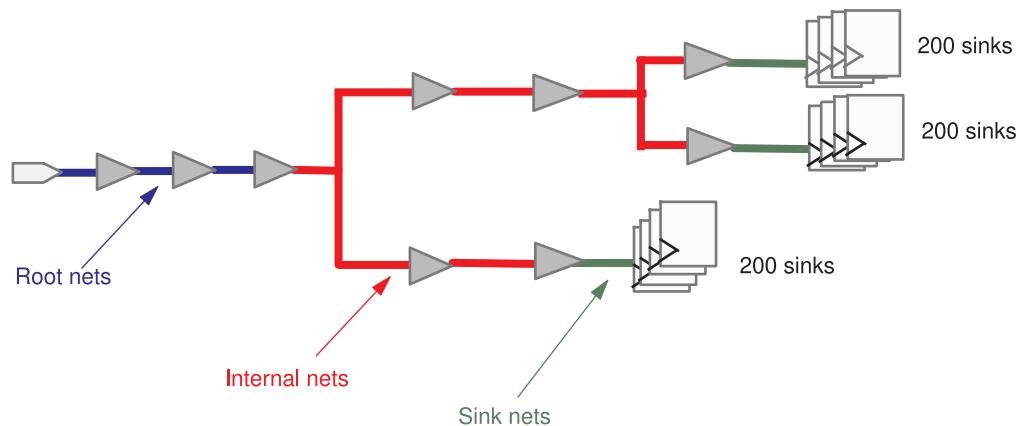
By default, the `set_clock_routing_rules` command assigns the specified layer constraints to all clock trees. [Table 17](#) shows the options used to restrict the layer constraints.

Table 17 Restricting Clock-Tree Layer Constraints

To assign a layer constraint to	Use this option
Specific clock trees	<code>-clocks clocks</code>
Nets connected to the clock root ¹	<code>-net_type root</code>
Nets connected to one or more clock sinks ¹	<code>-net_type sink</code>
Internal nets in the clock tree (all nets except the root and sink nets)	<code>-net_type internal</code>
Specific clock nets	<code>-nets nets</code>

[Figure 27](#) shows the root, internal, and sink nets of a clock tree after clock tree synthesis. By default, the root-net layer constraints are applied to all the single-fanout clock nets starting from the clock root up to the point where the clock tree branches out to a fanout of more than one. Internal-net layer constraints are applied to the nets from this point until the sink nets.

Figure 27 Root, Internal, and Sink Clock Net Types



1. You can use this option with the `-clocks` option to further restrict the assignment. This option is not valid with the `-nets` option.

For example, to use layers M4 through M7 when routing the root nets of the CLK1 clock, use the following command:

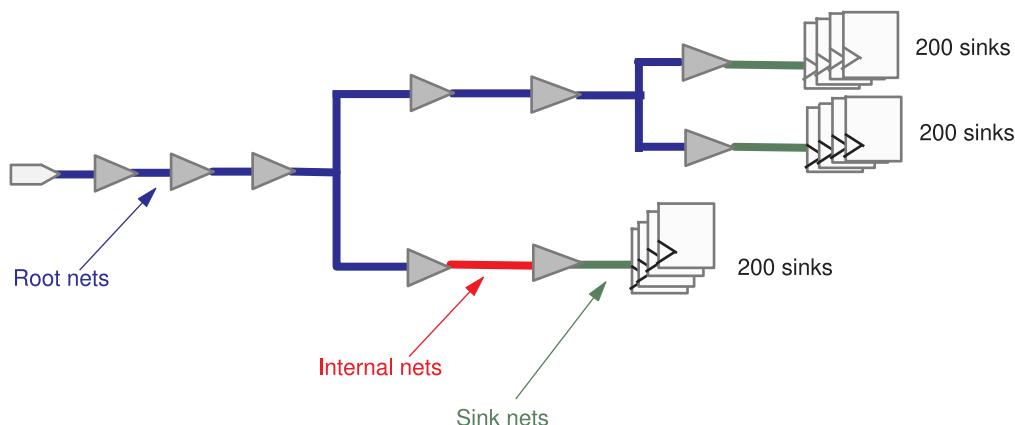
```
fc_shell> set_clock_routing_rules -clocks CLK1 -net_type root \
    -min_routing_layer M4 -max_routing_layer M7
```

To specify a transitive fanout limit to use when identifying root nets, use the `set_clock_tree_options -root_ndr_fanout_limit` command. For example, to specify that any clock net with a transitive fanout of more than 300 sinks be considered as a root net, use the following command:

```
fc_shell> set_clock_tree_options -root_ndr_fanout_limit 300
```

[Figure 28](#) shows the root, internal, and sink nets of the same clock tree when a transitive fanout limit of 300 is specified for identifying the clock root nets.

Figure 28 Using a Fanout Limit for Selecting Root Nets



When calculating the transitive fanout of clock nets for the purpose of identifying root nets, the tool includes only the valid clock sinks; It does not include the ignore pins. If a net identified as a root net is less than 10 microns, the tool uses internal-net layer constraints for that net.

Note:

Specifying a smaller value with the `set_clock_tree_options -root_ndr_fanout_limit` command increases the number of clock nets that are assigned the root-net layer constraints, which can increase routing congestion.

To remove the transitive fanout limit specified with the `set_clock_tree_options -root_ndr_fanout_limit` command, use the `remove_clock_tree_options -root_ndr_fanout_limit` command.

Setting the Preferred Routing Direction for Layers

The Fusion Compiler tool requires that the preferred routing direction is specified for the routing layers defined in the technology file. Typically this information is defined in the cell library. To set or change the preferred routing direction for a layer, use the following syntax to set its `routing_direction` attribute:

```
set_attribute -objects layers
  -name routing_direction
  -value vertical | horizontal
```

Specify the routing layers by using the layer names from the technology file.

The layer direction set with this attribute applies only to the current block.

For example, to set the preferred routing direction to vertical for the M5 and M7 layers, use the following command:

```
fc_shell> set_attribute -objects [get_layers {M5 M7}] \
  -name routing_direction -value vertical
```

Note:

Settings made with the `create_routing_guide` `-switch_preferred_direction` command, which changes the preferred direction within the area that is covered by the routing guide, override the `routing_direction` attribute settings.

To report the user-defined preferred routing direction for one or more routing layers, use the `get_attribute` command. To remove the user-defined preferred routing direction for one or more routing layers, use the `remove_attributes` command.

See Also

- [Preparing Routing Layers](#)

Handling Design Data Using the Early Data Check Manager

During the early iterations of a design cycles, the design data might be incomplete or incorrect. This can prevent you from implementing the design for exploration purposes and so on. The Fusion Compiler Early Data Check Manager allows you to specify how the tool should handle the data checks it performs during the following implementation tasks:

- Design planning
- Multivoltage design implementation
- Optimization

- Placement
- Legalization
- Hierarchical implementation of designs with physical hierarchy

The following table shows the Early Data Check Manager commands available for controlling how the tool handles the data checks and for generating related information.

Table 18 Early Data Check Manager Commands

To do this	Use this command
Specify the policy for handling one or more data checks	<code>set_early_data_check_policy</code>
Obtain a report about data check violations	<code>report_early_data_checks</code>
Get a Tcl collection of violations that can be used in other commands	<code>get_early_data_check_records</code>
Save the settings in a file for future use	<code>write_early_data_check_config</code>
Clear the data check settings in preparation for another iteration	<code>remove_early_data_check_records</code>

You should specify the policies for handling the data checks before you begin implementation.

The following example specifies that the tool should be strict when checking for the SCANDEF information. Therefore, the tool does not proceed with the `create_placement` command because the SCANDEF information is missing.

```
fc_shell> set_early_data_check_policy -policy strict \
           -check place.coarse.missing_scan_def
fc_shell> create_placement
Information: Policy for early data check 'place.coarse.missing_scan_def'
is 'error'. (EDC-001)
Error: No valid scan def found. (PLACE-042)
Information: Ending 'create_placement' (FLW-8001)
fc_shell> report_early_data_checks -check place.coarse.missing_scan_def
Check                         Policy   Strategy   Fail Count
-----
place.coarse.missing_scan_def   error          1
-----
```

The following example specifies that the tool should be lenient with regards to all data checks related to coarse placement. Therefore, the tool proceeds with the `create_placement` command without the SCANDEF information.

```
fc_shell> set_early_data_check_policy -policy lenient \
           -check place.coarse.*  

fc_shell> create_placement  

Information: Policy for early data check 'place.coarse.missing_scan_def'  

           is 'tolerate'. (EDC-001)  

Continuing without valid scan def.  

Start transferring placement data.  

Creating placement from scratch.  

...
...
```

For more information about the Early Data Check Manager, see the *Fusion Compiler Data Model User Guide*.

Applying Mega-Switch Command Settings

You can use mega-switch commands to apply setting for different stages of the tool flow that are applicable for specific situations with the use of a single command.

- [Applying Required Settings for Advanced Technology Nodes](#)
- [Applying Required Settings for High Performance Cores](#)
- [Applying Required Settings for Improving Specific QoR Metrics](#)

Applying Required Settings for Advanced Technology Nodes

To apply placement, legalization, routing, and extraction setting specific to 12 or 7 nanometer technology nodes, use the `set_technology` command with the `-node 12` or `-node 7` option.

When using this command, perform the following steps before placement, optimization, and routing:

1. Load and link the design.
2. Apply the settings required for the technology nodes by using the `set_technology -node` command, as shown in the following example:
`fc_shell> set_technology -node 7`
3. (Optional) Report the technology-node settings by using the `set_technology -report_only` command.
4. Apply your design-specific application option or command settings to override the generic setting of the `set_technology` command.

When you save the design library with the `save_lib` command, the technology-node setting is saved.

Note:

After you specify a technology node with the `set_technology` command, you cannot change it.

For more information about which ASIC technologies are supported by this command, contact Synopsys Support.

Applying Required Settings for High Performance Cores

You can apply tool settings required for achieving the best QoR for high performance cores by using the `set_hpc_options` command. The applied settings affect placement, legalization, optimization, clock tree synthesis, routing, extraction, and timing analysis.

To

- List the supported high performance core types and implementation stages, use the `-list` option
- Specify the type of high performance core, use the `-core` option
- Specify the stage of the implementation flow, use the `-stage` option

If your design uses a 12 or 7 nanometer technology node, apply the node-specific tool settings by using the `set_technology` command before you run the `set_hpc_options` command, as shown in the following example script:

```
set_technology -node 7
set_hpc_options -core A72 -stage clock_opt_cts
...
clock_opt -from build_clock -to route_clock
...
set_hpc_options -core A72 -stage clock_opt_opto
...
clock_opt -from final_opto
...
set_hpc_options -core A72 -stage route_auto
...
route_auto
...
set_hpc_options -core A72 -stage route_opt
...
route_opt
...
```

Applying Required Settings for Improving Specific QoR Metrics

To apply tool settings required for improving specific QoR metrics, use the `set_qor_strategy -stage synthesis` command.

To specify the QoR metrics to improve, use the `-metric` option. To optimize the block for the best

- Timing, use the `timing` setting
- Leakage power and timing, use the `leakage_power` setting
- Total power and timing, use the `total_power` setting.

To further improve total power, at the expense of runtime, you can use the `-mode extreme_power` option, with the `-metric total_power` option.

To specify a target mode for the design flow, use the `-mode` option. The supported values for this option are:

- `balanced`, which is the default mode that optimizes the design for the target metrics specified with the `-metric` option.
- `early_design`, which is suitable for the early stages of the design flow, when the emphasis is on short turnaround time for prototyping purposes.

With this mode, you can reduce runtime at the expense of QoR.

- `extreme_power`, which is recommended for further improvement of total power at the expense of runtime.

This mode should only be specified along with the `-metric leakage_power` or `-metric total_power` option. If you specify this mode with the `-metric timing` option, the tool ignores the specified mode setting and uses the `balanced` mode.

When you use this command, the tool applies application option settings that affect synthesis, optimization, placement, legalization, clock tree synthesis, routing, extraction, and timing analysis.

To review the required setting, use one of the following methods:

- Generate a Tcl script that contains the settings by using the `-output` option
- Report the settings by using the `-report_only` option
- Report only the settings that are not set to their required values by using the `-diff_only` option

3

Physical Synthesis

The Fusion Compiler tool synthesizes the RTL descriptions into optimized gate-level designs, optimizes them to achieve the best quality of results (QoR) in speed, area, and power, and allows early design exploration and prototyping.

Topics in this section include

- [Performing Unified Physical Synthesis](#)
- [Controlling Mapping and Optimization](#)
- [Performing Multibit Register Optimization](#)
- [Running Concurrent Clock and Data Optimization](#)
- [Performing Test Insertion and Scan Synthesis](#)
- [Specifying Settings for Performance, Power, and Area Improvement](#)
- [Performing Design Analysis](#)

Performing Unified Physical Synthesis

The Fusion Compiler tool performs unified physical synthesis. The benefits of unified physical synthesis include

- Reduced runtime

Unified physical synthesis streamlines the iterations of placement and optimization, and utilizes the initial placement and buffer trees built during synthesis to achieve the best runtime.

- Improved quality of results (QoR)

The unified preroute optimization framework uses consistent costs in optimization algorithms to improve QoR. The best technologies, such as logic restructuring, concurrent clock and data optimization, multibit mapping, and advanced legalizer, are shared throughout the preroute flow.

The Fusion Compiler compilation includes both top-down and bottom-up (hierarchical) compile strategies. In a top-down compile, all environment and constraint settings are

defined with respect to the top-level design. Top-down compile provides a push-button approach and handles interblock dependencies automatically, but it requires that all designs must reside in memory at the same time. In a hierarchical compile flow, you start optimization of design subblocks using a top-down approach and then continue incorporating the subblocks into higher-level blocks until the top-level design is complete.

The following topics describe how perform unified physical synthesis:

- [Using the `compile_fusion` Command](#)
- [Performing Prechecks](#)
- [Generating Verification Checkpoints During Compilation](#)

Using the `compile_fusion` Command

To perform unified physical synthesis, use the `compile_fusion` command.

The `compile_fusion` command consists of the following stages:

1. `initial_map`
During this stage, the tool maps the design and performs area optimization.
2. `logic_opt`
During this stage, the tool performs logic-based delay optimization.
3. `initial_place`
During this stage, the tool merges the clock-gating logic and performs coarse placement.
4. `initial_drc`
During this stage, the tool removes existing buffer trees and performs high-fanout-net synthesis and electrical DRC violation fixing.
5. `initial_opto`
During this stage, the tool performs physical optimization and incremental placement.
6. `final_place`
During this stage, the tool performs final placement to improve timing and congestion.
7. `final_opto`
During this stage, the tool performs final optimization and legalization to improve timing, power, and logical DRCs.

When you run the `compile_fusion` command, by default, the tool runs all stages. To run only some of these stages, use the `-from` option to specify the stage from which you want to begin and the `-to` option to specify the stage after which you want to end. If you do not specify the `-from` option, the tool begins from the `initial_place` stage. Similarly, if you do not specify the `-to` option, the tool continues until the `final_opto` stage is completed.

Breaking the `compile_fusion` command into stages allows you to perform different tasks between the stages, such as changing constraints or settings, generating reports for analysis, performing other implementation tasks such as design planning or DFT insertion, and so on. If you make manual changes to the design between stages, ensure the changes honor the expected state of the design. For example, the design is expected to be mapped after the `initial_map` stage. Therefore, do not introduce unmapped logic to the design after this stage.

You can repeat a stage; however, ensure that you run all stages in the correct sequence. The following example runs the `compile_fusion` command until it completes the `initial_opto` stage, performs analysis, changes an application option setting, and restarts by running the `initial_opto` stage again.

```
fc_shell> compile_fusion -to initial_opto
fc_shell> report_qor
fc_shell> set_app_option \
    -name opt.common.advanced_logic_restructuring_mode -value timing
fc_shell> compile_fusion -from initial_opto
```

In the following situations, the design can contain unmapped cells after the `compile_fusion` command:

- Missing the required cells in the library
- User-specified restrictions on the required library cells

When a design contains unmapped cells, the `compile_fusion` command continues the compilation and provides an estimated area for the unmapped cells.

Unmapped cells are marked with the `is_unmapped` attribute. To query unmapped cells in a netlist, use the following command:

```
fc_shell> get_cells -hierarchical -filter "is_unmapped==true"
```

Note the following limitations:

- DesignWare operators are not mitigated.
- Only placement information is stored in the database after mitigation; routing information is not saved.

Performing Prechecks

Fusion Compiler can perform compile prechecks to prevent compile failures and allow the `compile_fusion` command to quickly exit with error messages when incomplete (or inconsistent) setup or constraint issues are encountered. This capability is on by default, but these checks can also be enabled by the `compile_fusion -check_only` command.

The `compile_fusion` command and the `compile_fusion -check_only` command perform the prechecks listed in the following table and issue error messages accordingly. A check mark (X) indicates the precheck is performed.

Table 19 Prechecks Performed During the Compile Flow

Code	Message	Default compile prechecks	<code>compile_fusion -check_only</code>
OPT-3000	Design is not linked or has link issues.	X	X
OPT-3001	<i>Cell</i> is not resolved after linking.	X	X
OPT-3002	<i>Cell</i> is physical hierarchy and not allowed in compile.	X	X
OPT-3003	<i>Cell</i> is logical hierarchy and not allowed in compile.	X	X
OPT-3005	Design is not present.	X	X
OPT-3006	Current instance <i>hierarchy</i> is not the top.	X	X
OPT-2001	Could not initialize scenario settings.		X
OPT-2002	Missing parasitic information for corners or missing TLUPlus information.		X
OPT-4000	Design does not have a floorplan or a valid floorplan.	X	X
OPT-4001	Design does not have a valid die area.	X	X
OPT-4002	<i>Macro</i> does not have placement. information.	X	X
OPT-4004	Design <i>port</i> does not have placement information.	X	X
OPT-4010	<i>Pad</i> connected to <i>port</i> does not have placement information.	X	X
OPT-4008	Design has no site rows defined.	X	X
OPT-4009	Design has no tracks defined.	X	X
OPT-1008	<i>Layer</i> does not have a preferred direction.		X

Table 19 Prechecks Performed During the Compile Flow (Continued)

Code	Message	Default compile prechecks	compile_fusion -check_only
OPT-1006	Library analysis failure - cannot find buffers and inverters.	X	X
OPT-1007	Library analysis failure - No technology data.	X	X
OPT-4012	Design has 1 duplicate. Run <code>check_duplicates</code> for more details.		X

Compile Precheck Examples

The following two examples show the compile logs with and without compile prechecks. When the `-check_only` option is set, the `compile_fusion` command issues error messages for any failed precheck condition.

Example 14 Compile Log With Compile Prechecks

```
----- Begin compile flow -----
(FWL-3001)
->Compile in progress: 1.0% (FWL-3778)
Error: Layer M1 does not have a preferred direction (OPT-1008)
...
Error: Layer MRDL does not have a preferred direction (OPT-1008)
...
Warning: Technology layer 'M1' setting 'routing-direction' is not valid
(NEX-001)
...
Warning: Technology layer 'MRDL' setting 'routing-direction' is not valid
(NEX-001)
```

Example 15 Compile Log Without Compile Prechecks

```
----- End logic optimization -----
(FWL-3001)
->Compile in progress: 27.0% (FWL-3778)
->Compile in progress: 28.0% (FWL-3778)
->Compile in progress: 28.5% (FWL-3778)
Information: Corner max_corner: no PVT mismatches. (PVT-032)
Warning: Technology layer 'M1' setting 'routing-direction' is not valid
(NEX-001)...
Warning: Technology layer 'MRDL' setting 'routing-direction' is not valid
(NEX-001)
Information: Design Average RC for design top (NEX-011)
```

Generating Verification Checkpoints During Compilation

You can generate intermediate verification checkpoints during the `compile_fusion` command, which you can use in the Formality tool to provide a netlist snapshot of the synthesis design state and verify the intermediate netlists. Verification checkpointing allows the Fusion Compiler and Formality tools to synchronize on an intermediate netlist and in turn results in higher completion rates and better QoR.

To enable verification checkpointing, use the `set_verification_checkpoints` command as shown in the following example:

```
fc_shell> set_verification_checkpoints
Enabling checkpoint "ckpt_logic_opt"
Enabling checkpoint "ckpt_pre_map"
```

When you enable this feature, by default, the `compile_fusion` command generates verification checkpoints before mapping and during logic optimization, and the corresponding verification-checkpoint stages are named `ckpt_pre_map` and `ckpt_logic_opt`. You can enable verification checkpointing at only one stage as shown in the following example:

```
fc_shell> set_verification_checkpoints {ckpt_logic_opt}
Disabling checkpoint "ckpt_pre_map"
Enabling checkpoint "ckpt_logic_opt"
```

At each verification checkpoint, the tool automatically generates a checkpoint netlist and the `guide_checkpoint` command in the .svf file that is used by the Formality tool during verification.

Controlling Mapping and Optimization

The following topics describe how you can customize and control the mapping and optimization that is performed during the `initial_map`, `logic_opto`, and `initial_drc` stages of the `compile_fusion` command:

- [Ungrouping or Preserving Hierarchies During Optimization](#)
- [Controlling Boundary Optimization](#)
- [Controlling Datapath Optimization](#)
- [Controlling Mux Mapping](#)
- [Controlling Sequential Mapping](#)
- [Controlling Register Replication](#)
- [Controlling Register Merging](#)

- Selectively Removing or Preserving Constant and Unloaded Registers
- Reporting Cross-Probing Information for Optimized Registers
- Controlling High-Fanout-Net Synthesis

Ungrouping or Preserving Hierarchies During Optimization

Ungrouping merges subdesigns of a given level of the hierarchy into the parent cell or design. It removes hierarchical boundaries and allows Fusion Compiler to improve timing by reducing the levels of logic and to improve area by sharing logic. By default, the `compile_fusion` command automatically ungroups logical hierarchies that do not have a `dont_touch` attribute setting.

During optimization, the `compile_fusion` command performs the following types of automatic grouping:

- Area-based automatic ungrouping

Before initial mapping, the command estimates the area for unmapped hierarchies and removes small subdesigns. Because the command performs automatic ungrouping at an early stage, it has a better optimization context. Additionally, datapath extraction is enabled across ungrouped hierarchies. These factors improve the area and timing quality of results.

- Delay-based automatic ungrouping

The command ungroups hierarchies along the critical path and is used essentially for timing optimization.

For ungrouping of DesignWare and datapath cells, see [Datapath Implementation](#).

You can disable automatic ungrouping by setting the `compile.flow.autoungroup` application option to `false`. The default is `true`. However, doing so can affect the QoR. Therefore, it is better to limit ungrouping, rather than disabling it, using the following methods:

Controlling the Automatic Ungrouping of Specific Types of Hierarchies

To control the automatic ungrouping of specific types of hierarchies, use the `set_autoungroup_options` command.

The following example specifies that parent hierarchies of blocks with UPF and SDC constraints should be preserved and that automatic ungrouping of other hierarchies should begin from the third level:

```
fc_shell> set_autoungroup_options -keep_parent_hierarchies UPF
fc_shell> set_autoungroup_options -keep_parent_hierarchies SDC
fc_shell> set_autoungroup_options -start_level 3
```

The `set_autoungroup_options` command is cumulative and the settings are saved in the design library.

To report information about the hierarchies that are automatically ungrouped or preserved, use the `report_ungroup` command.

Controlling the Automatic Ungrouping of Specific Objects

To force or prevent specific cell instances or modules being ungrouped during optimization, use the `set_ungroup` command with the `true` or `false` setting. The `set_ungroup` command sets the `ungroup` attribute to the appropriate value for the specified objects. If you set the attribute on a module, all cells that reference the module are affected.

Note:

The `set_ungroup` command has a higher priority than the `set_autoungroup_options` command.

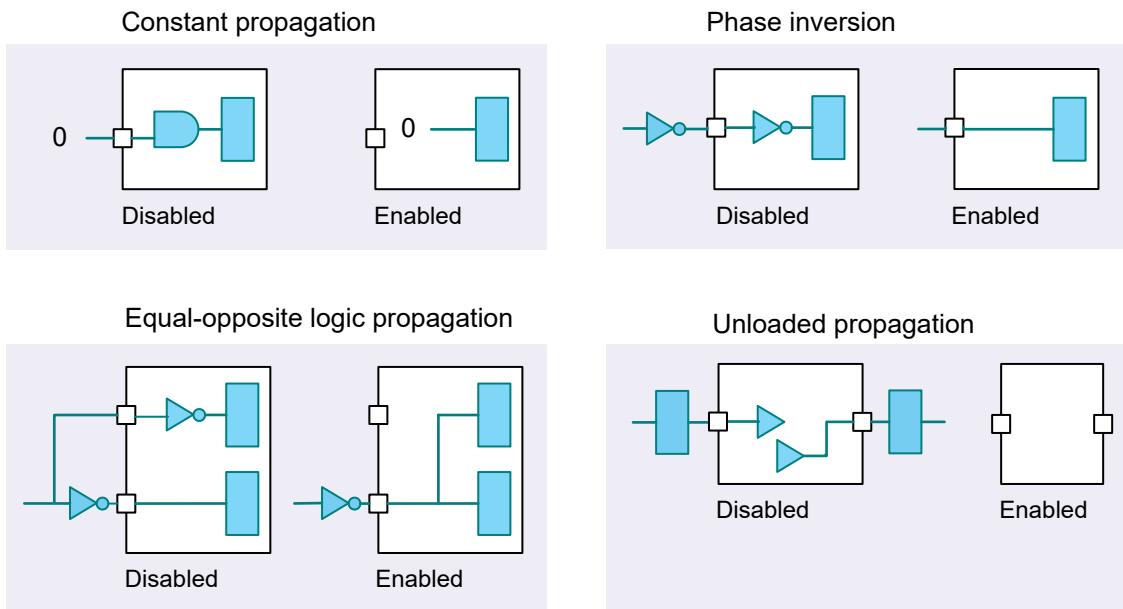
The following example script sets specific modules to be ungrouped and specific cell instances to be preserved during optimization:

```
# Set modules A and B to be ungrouped
set_ungroup [get_modules {A B}] true
# Set cells u_C and u_D to be preserved
set_ungroup [get_cells {u_C u_D}] false
# Query the ungroup attribute
get_attribute -name ungroup -objects [get_modules {A B}]
get_attribute -name ungroup -objects [get_cells {u_C u_D}]
compile_fusion
```

Controlling Boundary Optimization

You can control boundary optimization across user-specified logical boundaries and match the interface boundary of a physical block with the RTL module interface. Boundary optimization allows you to use RTL floorplanning DEF files during block implementation.

The following figure shows how constants, equal-opposite logic, logic phase, and unloaded ports are propagated through the logical boundaries without and with boundary optimization.



Disabled: without boundary optimization
Enabled: with boundary optimization

Global Control

By default, all types of boundary optimization are enabled during compile. To control boundary optimization globally, use the `compile.flow.boundary_optimization` application option.

When you set the `compile.flow.boundary_optimization` application option to `false`,

- The tool disables propagation of equal-opposite logic and phase inversion.
- Constants and unloaded ports continue to propagate across hierarchies.

To disable constant propagation and unloaded propagation globally, set the `compile.flow.constant_and_unloaded_propagation_with_no_boundary_opt` application option to `false`. The default is `true`. The setting does not affect object-level specifications.

Controlling Specific Types of Boundary Optimization

To control the level of boundary optimization for hierarchical cell pins, hierarchical cells, or blocks, use the `set_boundary_optimization` command. The tool uses the following order of precedence to determine the setting to use during optimization:

1. Setting applied on pins of hierarchical cell instances
2. Setting applied on hierarchical cell instances, which applies to its pins as well
3. Setting applied on modules, which applies to all its cells and pins as well

The following table shows the level of boundary optimization you can specify with the `set_boundary_optimization` command and the types of boundary optimization that are enabled for each level.

Table 20 Levels of Boundary Optimization and the Corresponding Types of Boundary Optimization Performed

Level of boundary optimization	Constant propagation	Unloaded propagation	Equal-opposite logic propagation	Phase inversion
all	Enabled	Enabled	Enabled	Enabled
auto	Enabled	Enabled	Disabled	Disabled
none	Disabled	Disabled	Disabled	Disabled

You can control the specific types of boundary optimization allowed for an object by using the `-constant_propagation`, `-unloaded_propagation`, `-equal_opposite_propagation`, and `-phase_inversion` options. These options are mutually exclusive with the boundary optimization level specified by using `all`, `auto`, or `none` and cannot be used on the same object in a single command invocation. If you use the `set_boundary_optimization` command multiple times on the same object, the effect is additive.

Based on the types of boundary optimization you enable or disable, the tool sets the `constant_propagation`, `unloaded_propagation`, `equal_opposite_propagation`, and `phase_inversion` read-only attributes of the specified object to `true` or `false`. If you

- Enable all four types, the tool sets the `boundary_optimization` read-only attribute to `true`
- Disable at least one type, the tool sets the `boundary_optimization` read-only attribute to `false`

To report boundary optimization settings, use the `report_boundary_optimization` command. To remove them, use the `remove_boundary_optimization` command.

The following example

- Disables all boundary optimization for the U1 cell instance
- Disables equal-opposite logic propagation and phase inversion for the U2 cell instance
- Disables constant propagation and phase inversion for the U3/in1 cell pin

```
fc_shell> set_boundary_optimization [get_cells U1] none
fc_shell> set_boundary_optimization [get_cells U2] auto
fc_shell> set_boundary_optimization [get_pins U3/in1] \
    -constant_propagation false
```

```
fc_shell> set_boundary_optimization [get_pins U3/in1] \
    -phase_inversion false
```

Controlling Phase Inversion

To prevent boundary optimization from moving inverters across hierarchical boundaries, set the `compile.optimization.enable_hierarchical_inverter` application option to `false`. By default, the application option is set to `true` to allow phase inversion. This application option is only effective when boundary optimization is enabled.

Controlling Datapath Optimization

Datapath design is commonly used in applications that contain extensive data manipulation, such as 3-D, multimedia, and digital signal processing (DSP). By default, when you run the `compile_fusion` command, datapath optimization is enabled and both standard and DesignWare Foundation libraries are automatically linked.

During datapath optimization, the tool performs the following tasks:

- Shares (or reverses the sharing) of datapath operators
- Uses the carry-save arithmetic technique
- Extracts the data path
- Performs high-level arithmetic optimization on the extracted data path
- Explores better solutions that might involve a different resource-sharing configuration
- Makes tradeoffs between resource sharing and datapath optimization

Topics in this section

- [Datapath Extraction](#)

Transforms arithmetic operators, such as addition, subtraction, and multiplication, into datapath blocks.

- [Datapath Implementation](#)

Uses a datapath generator to generate the best implementations for the extracted components.

- [Optimizing Datapaths](#)

Datapath Extraction

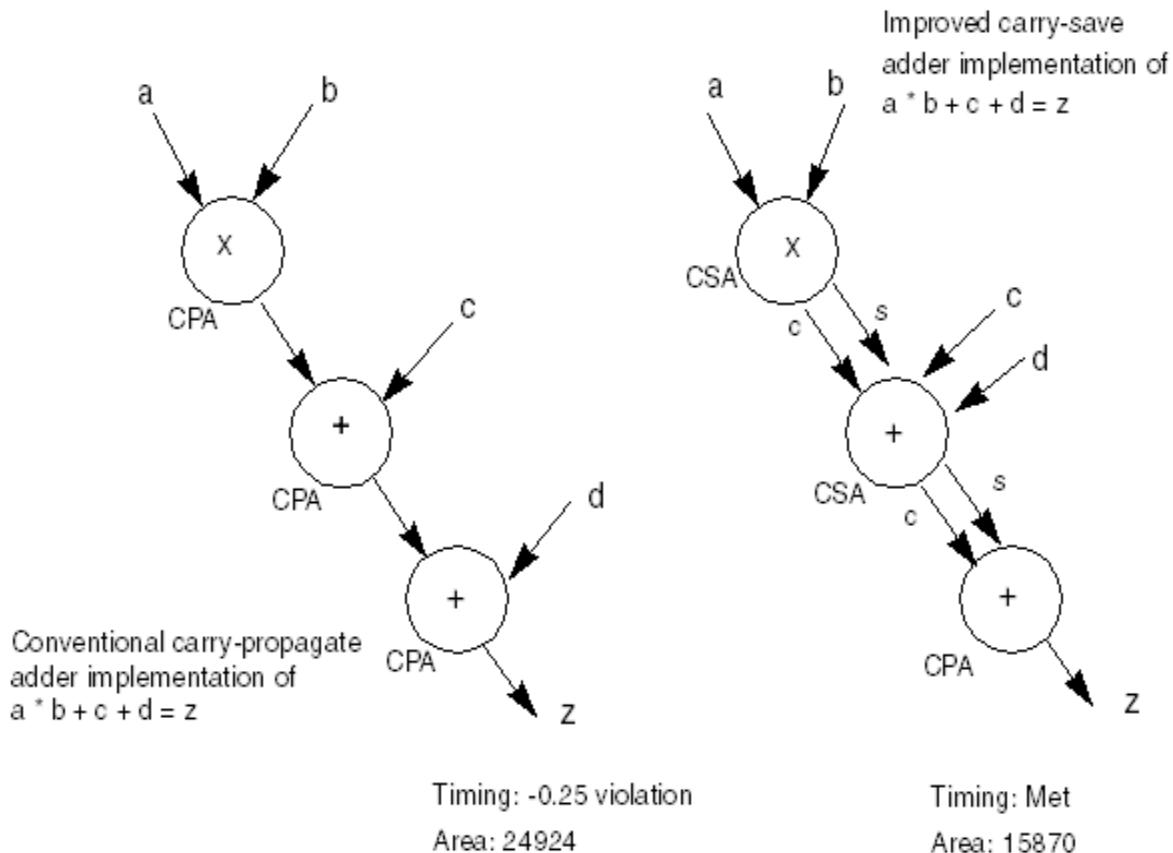
Datapath extraction transforms arithmetic operators, such as addition, subtraction, and multiplication, into datapath blocks to be implemented by a datapath generator. This transformation improves the quality of results (QoR) by using carry-save arithmetic.

Carry-save arithmetic does not fully propagate carries but stores results in an intermediate form. The carry-save adders are faster than the conventional carry-propagate adders because the carry-save adder delay is independent of bit-width. These adders use significantly less area than carry-propagate adders because they do not use full adders for the carry.

The following example shows the code for the $a * b + c + d = z$ expression. As shown in [Figure 29](#), the conventional implementation of the expression use three carry-propagate adders, whereas the carry-save arithmetic requires only one carry-propagate adder and two carry-save adders. The figure also shows the timing and area numbers for both implementations.

```
module dp (a, b, c, d, e);    input [15:0] a, b;
    input [31:0] c, d;
    output [31:0] Z;
assign Z = (a * b) + c + d;
endmodule
```

Figure 29 Conventional Carry-Propagate Adder Versus Carry-Save Adder



The tool supports extraction of the following components:

- Arithmetic operators that can be merged into one carry-save arithmetic tree
- Operators extracted as part of a datapath: *, +, -, >, <, <=, >=, ==, !=, and MUXes
- Variable shift operators (<<, >>, <<<, >>> for Verilog and sll, srl, sla, sra, rol, ror for VHDL)
- Operations with bit truncation

The datapath flow can extract these components only if they are directly connected to each other, that is, no nonarithmetic logic between components. Keep the following points in mind:

- Extraction of mixed signed and unsigned operators is allowed only for adder trees
- Instantiated DesignWare components cannot be extracted

Datapath Implementation

All the extracted datapath elements and the synthetic operators are implemented by the DesignWare datapath generator in Fusion Compiler. The DesignWare datapath generator uses the smart generation technology to perform the following tasks:

- Implement datapath blocks using context-driven optimizations
- Revisit high-level optimization decisions for a given timing context
- Consider logic library characteristics

Use these commands and application option for datapath implementation:

- `set_datapath_architecture_options`

This command controls the strategies used to generate the datapath cells for arithmetic and shift operators. The following example specifies to use Booth-encoded architectures to generate multiplexers:

```
fc_shell> set_datapath_architecture_options -booth_encoding true
```

- `set_implementation`

This command specifies the implementation to use for dividers and synthetic library cell instances with a specified name. The following example specifies to use the cla3 implementation for the instantiated DesignWare cell U1.

```
fc_shell> set_implementation cla3 U1
```

- `set_synlib_dont_use`

This command prevents using specific implementation. The following example specifies not to use the rpl implementation.

```
fc_shell> set_synlib_dont_use \
    {dw_foundation/DW_div/cla_standard/DW_div/rpl}
```

- `compile.datapath.ungroup`

By default, all DesignWare cells and datapath blocks are ungrouped during timing optimization. To disable the ungrouping, set the `compile.datapath.ungroup` application option to `false`, changing from its default of `true`. For example,

```
fc_shell> set_app_options \
    -name compile.datapath.ungroup -value false
```

Optimizing Datapaths

The following script disables the ungrouping of the DP_OP cell with datapath gating element `add_3`:

```
# Disable ungrouping of DP_OP with datapath gating
set_datapath_gating_options -ungroup false [get_cells add_3]
```

Analyzing Datapath Extraction

To get the best QoR results from datapath optimization, ensure that the biggest possible datapath blocks are extracted from the RTL code during compile. To examine how the datapath blocks will be extracted from your RTL before compile, use the `analyze_datapath_extraction` command. The command analyzes the arithmetic contents of the design and provides feedback, so you can improve the RTL code as needed.

The following example shows a datapath report:

```
Cell : DP_OP_123_2304_10297_J1
-----
Current Implementation : str(area,speed)
Contained Operations  : add_30(test.v:30) add_30_2(test.v:30)
                        mult_36(test.v:36)
Multiplier Arch       : benc_radix4

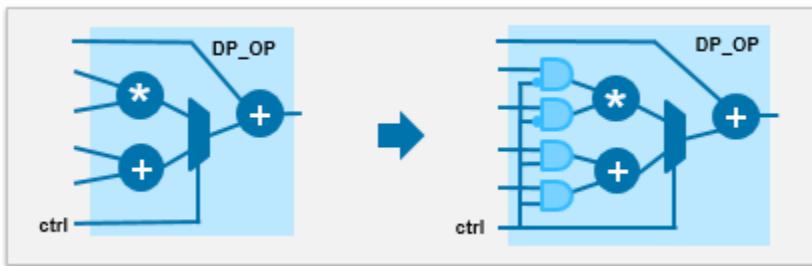
      Data
      Var   Type  Class   Width Expression
-----+
PI_0    PI    Unsigned   6
PI_1    PI    Unsigned   1
PI_2    PI    Unsigned  24
T16     IFO   Unsigned   7   PI_0 + PI_1  (test.v:30)
PO_0    PO    Unsigned  30   PI_2 * T16  (test.v:36)
```

To report the resources and datapath blocks used in the design, use the `report_resources` command. To report the implementation overview, specify the `-summary` with the `report_resources` command. For example,

```
fc_shell> report_resources -hierarchy
```

Controlling Datapath Gating

The following figure shows signal transitions of the inputs of a DP_OP cell that is generated during datapath gating optimization. To control datapath gating, use the `set_datapath_gating_options` command.



The following script disables datapath gating on the datapath element `add_3`:

```
# Disable datapath gating on particular operator
set_datapath_gating_options -enable false [get_cells add_3]
```

Controlling Mux Mapping

During logic synthesis, the tool implements multiplexing logic using one of the following two methods:

- AND, OR, and INV gates from the cell library
- MUX gates from the cell library

The tool evaluates the area effect of both methods and selects the solution with the smallest area.

You can specify that the tool uses MUX gates to implement specific multiplexing logic by applying the `map_to_mux` attribute on the corresponding SELECT_OP WVGTECH cells in the block, as shown in the following example script:

```
# Apply the map_to_mux attribute on the SELECT_OP cell named C10
set_attribute -objects [get_cells C10] -name map_to_mux -value true

# Query if the attribute is set as required on C10
get_attribute -objects [get_cells C10] -name map_to_mux

#Run the compile step
```

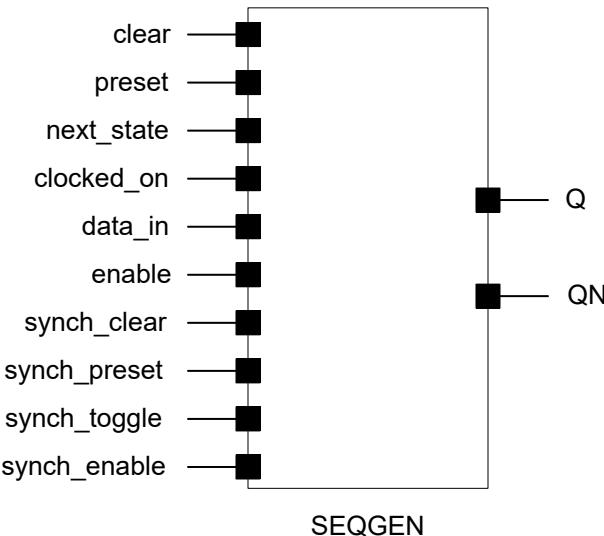
```
compile_fusion

#Identify the cell created using the map_to_mux attribute
get_cells -hier -filter "map_to_mux==true && !infer_mux_override"
```

Controlling Sequential Mapping

The sequential mapping phase consists of two steps: register inferencing and technology mapping. The term register refers to both edge-triggered registers and level-sensitive latches.

Register inferencing is the process by which the RTL description of a register is translated into a technology-independent representation called a SEQGEN. SEQGENs are created during elaboration and are usually mapped to flip-flops during compile. Technology mapping is the process by which a SEQGEN is mapped to gates of a specified target logic library. Both of these steps are performed by the `compile_fusion` command. Both registers and latches are represented by a SEQGEN cell, which is a technology-independent model of a sequential element as shown in the following figure:



This table lists the pins of a SEQGEN cell. Only a subset of these pins is used depending on the type of cell that is inferred. Unused pins are tied to zero.

Direction	Name	Description	Cell type
Input	clear	Asynchronous reset	Flip-flop or latch

Direction	Name	Description	Cell type
	preset	Asynchronous preset	Flip-flop or latch
	next_state	Synchronous data	Flip-flop
	clocked_on	Clock	Flip-flop
	data_in	Asynchronous data	Latch
	enable	Clock or asynchronous enable	Latch
	synch_clear	Synchronous reset	Flip-flop
	synch_preset	Synchronous preset	Flip-flop
	synch_toggle	Synchronous toggle	Flip-flop
	synch_enable	Synchronous enable	Flip-flop
Output	Q	Non-inverting output	Flip-flop or latch
	QN	Inverting output	Flip-flop or latch

During technology mapping, the tool uses the SEQGEN as the starting point for mapping. The sequential mapper checks the connections to the pins and the information present in the library cell descriptions when it maps to a logic library register.

Topics in this section

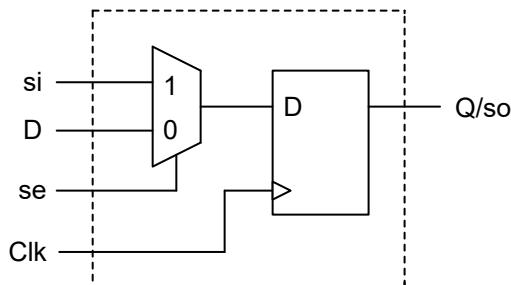
- [Mapping Sequential Elements Exactly as Described in RTL](#)
- [Mapping of Scan Cells](#)
- [Mapping of Synchronous Reset or Preset Registers](#)
- [Specifying Synchronous Input Types for Library Cell Pins](#)
- [Controlling Sequential Output Inversion](#)
- [Preventing Connections to Register QN Pins](#)
- [Mapping of Synchronous Enable Logic to a MUX](#)
- [Identification of Shift Registers](#)

Mapping Sequential Elements Exactly as Described in RTL

To specify that the tool maps sequential elements exactly as described in the RTL, set the `compile.seqmap.exact_map` application option to `true`. When this feature is enabled, the tool disables mapping to sequential library cells with inverted outputs or unused ports.

Mapping of Scan Cells

During test-ready compile, the tool replaces regular flip-flops with flip-flops that contain logic for testability. The following figure shows an example of how a D flip-flop is replaced with a scan register during test-ready compile. This type of architecture, a multiplexed flip-flop, incorporates a 2-input MUX at the input of the D flip-flop. The select line of the MUX enables two modes—functional mode (normal data input) and test mode (scanned data input). In this example, the scan-in pin is si, the scan-enable pin is se, and the scan-out pin, so, is shared with the functional output pin, Q.



When you run the `compile_fusion` command,

- The tool maps all sequential cells directly to scan registers in your library
- The scan pins, such as scan enable and scan out, are unconnected
- The scan output pin of a scan register is used for scan connections

Use the `seqmap.bind_scan_pins` application option to hold the scan enable pin inactive

If the library contains no scan registers, the tool uses nonscan cells for mapping and issues a warning. Set the `dont_use` attribute on all scan registers in the library to map to nonscan cells

Mapping of Synchronous Reset or Preset Registers

Fusion Compiler does not infer synchronous reset or preset flip-flops by default.

To infer registers with synchronous reset or preset pins, use the `sync_set_reset` Synopsys compiler directive in the RTL. HDL Compiler then connects these signals to the `synch_clear` and `synch_preset` pins on the SEQGEN WVGTECH cell to communicate to the mapper that these are the synchronous control signals and they should be kept as close to the register as possible.

To enable the tool to infer synchronous reset or preset flip-flops or preserve the synchronous reset or preset logic close to the flip-flop, use either of the following two methods:

- The `sync_set_reset` directive

Specify the `//synopsys sync_set_reset` directive in the RTL.

- The `hdlin.elaborate.ff_infer_sync_set_reset` application option

When the RTL does not contain the directive, set the

`hdlin.elaborate.ff_infer_sync_set_reset` application option to `true`. The default is `false`.

During compile, the tool performs the following mapping based on the registers in your library:

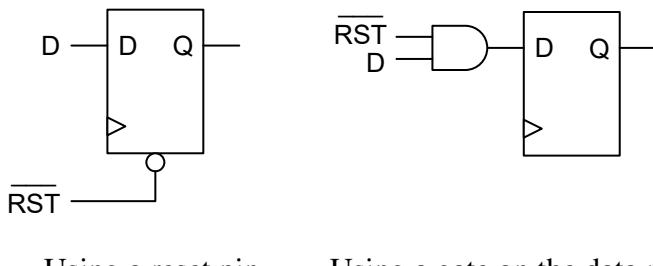
- The library containing registers with synchronous reset (or preset) pins

The tool connects the reset (or preset) net to the reset (or preset) pin of a register with a dedicated reset pin.

- The library containing no registers with synchronous reset (or preset) pins

The tool adds extra logic to the data input to generate the reset (or preset) condition on a register without a reset (or preset) pin. The tool also attempts to map the logic as close as possible to the data pin.

This figure shows examples of mapping to registers with and without a synchronous reset pin.



Using a reset pin

Using a gate on the data pin

If your library does have registers with synchronous reset (or preset) pins, you should still use the `sync_set_reset` directive in the RTL so that the tool can distinguish the reset (or preset) signals from other data signals and connect the reset signal as close to the register as possible.

Note:

Synchronous reset and preset signals are not inferred for level-sensitive latches. A synchronous reset or preset coding style on a latch always results

in combinational logic on the data signal even if the library has latches with synchronous reset or preset pins.

Mapping of Asynchronous Reset or Preset Registers

To enable the tool to map to registers with asynchronous reset or preset signals, you must

- Describe the registers correctly in the RTL
- Ensure the library contain the corresponding cells
- Ensure the `hdlin.elaborate.ff_infer_async_set_reset` application option is set to `true` (the default)

If the library does not contain this type of registers, the tool leaves the cells unmapped.

Specifying Synchronous Input Types for Library Cell Pins

When mapping sequential cells, the tool uses the `nextstate_type` attribute of a library cell pin to identify its synchronous pin type. The valid values for this attribute are `data`, `preset`, `clear`, `load`, `scan_in`, and `scan_enable`.

If this attribute is not specified or is incorrectly specified in the logic library, you can manually specify it by using the `set_attribute` command, as shown in the following example:

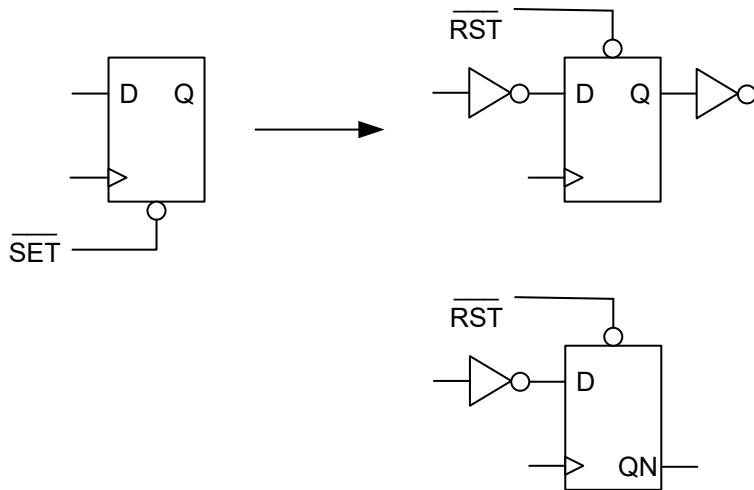
```
fc_shell> set_attribute [get_lib_pins my_lib/dffsrst/RN] \
    nextstate_type clear
```

Controlling Sequential Output Inversion

By default, the `compile_fusion` command can map a register to a sequential library cell with an output that is opposite to that of the register, if it improves QoR. This is known as sequential output inversion.

Sequential output inversion is also useful when mapping to a target library with sequential cells that have only one type of asynchronous inputs (either set or reset). In this case, the only way to map a register that uses the missing asynchronous inputs is to use a library cell with the opposite type of asynchronous input and invert the output of that cell, as shown in the following figure.

This figure shows an example of sequential output inversion during mapping.



Note:

Information about inverted registers is written to the verification guidance (.svf) file. You must include the .svf file in the Formality tool when verifying blocks with sequential output inversion.

To control sequential output inversion for

- The entire block, use the `compile.seqmap.enable_output_inversion` application option.
The default is `true`.
- Specific hierarchical or leaf cells, use the `set_register_output_inversion` command.

An instance-specific sequential output inversion setting specified with the `set_register_output_inversion` command overrides a block-specific setting specified with the `compile.seqmap.enable_output_inversion` application option.

The `compile_fusion` command allows the mapping of sequential elements in the design to sequential library cells whose output phase is inverted. In certain cases, the tool might infer a register that has one type of asynchronous control, but your library has the opposite type of pin. In such cases, the tool maps to the opposite type of register and inverts all the data inputs and outputs.

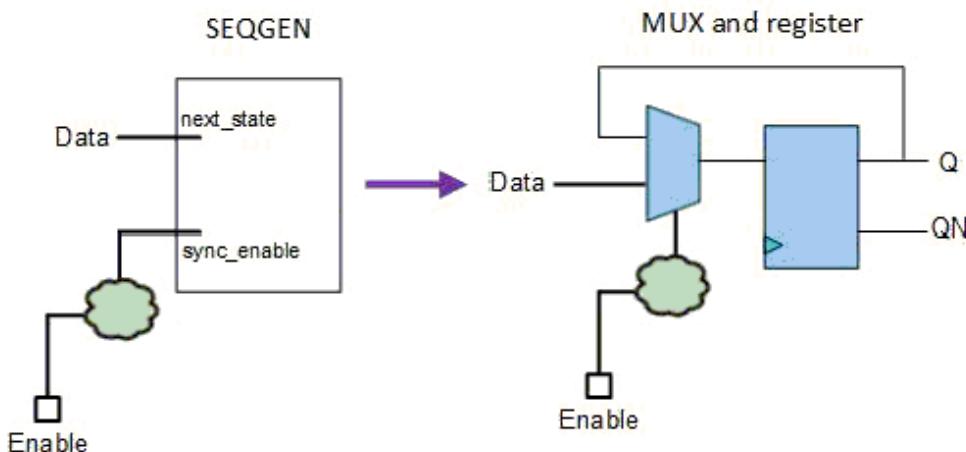
Preventing Connections to Register QN Pins

During sequential mapping, the tool can map registers to library cells that have both Q and QN pins, if doing so improves the QoR. However, you can prevent the tool from using the

QN pin of such registers by setting the `compile.seqmap.disable_qn_usage` application option to `true`.

Mapping of Synchronous Enable Logic to a MUX

When the RTL contains synchronous enable logic, the `compile_fusion` command maps the enable logic to library cells with a synchronous enable pin. When the library cells with a synchronous enable pin are not available, the `compile_fusion` command maps the pushed-out synchronous logic to a MUX, as shown in this figure.

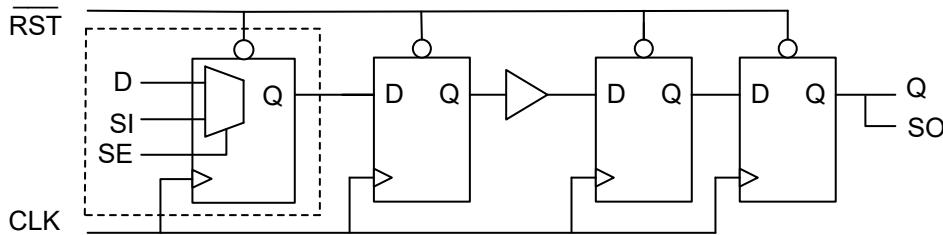


To enable the mapping to MUXs, set the `map_sync_enable_to_mux` attribute on the SEQGEN cells before compile. For example,

```
fc_shell> set_attribute "A1/B/reg[30] mid/sub1/Q_reg[0]" \
    -name map_sync_enable_to_mux -value true
fc_shell> set_attribute "mid3/apbif_reg/Q_reg[4]" \
    -name map_sync_enable_to_mux -value true
fc_shell> compile_fusion
```

Identification of Shift Registers

During compile, the tool can automatically identify shift registers in the design. The `compile_fusion -to initial_opto` command maps the first register to a scan cell and the other cells to nonscan cells, as shown in this figure.



This capability improves the sequential design area and reduces congestion because of fewer scan-signals for routing. To disable this capability, set the `compile.seqmap.identify_shift_registers` application option to `false`. The default is `true`.

Controlling Register Replication

To manually identify registers that you want the tool to replicate during optimization, use the `set_register_replication` command. This command sets the `register_replication` attribute on the specified registers. During optimization, the specified registers are replicated and the loads of the original registers are evenly distributed among the new replicated registers.

You can control the number of copies that are created by using one of the following methods:

- Specify the total number of registers after replication, including the original, by using the `-num_copies` option.
- Specify the maximum fanout of the replicated registers by using the `-max_fanout` option.

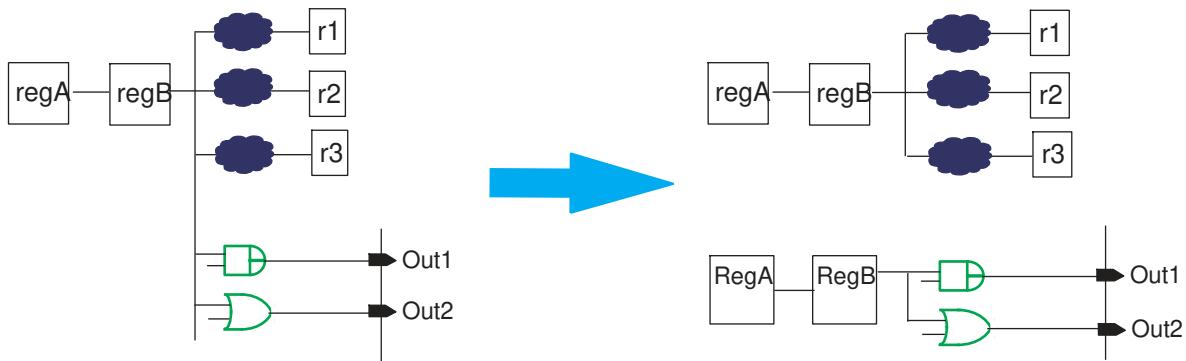
If you specify both options, the `-max_fanout` option is ignored.

The following example specifies that the tool should replicate the `regA` and `regB` registers by creating one additional copy of each register:

```
fc_shell> set_register_replication -num_copies 2 regA
fc_shell> set_register_replication -num_copies 2 regB
```

[Figure 30](#) shows the `regA` and `regB` registers before and after optimization.

Figure 30 Before and After Register Replication



You can include logic in the fanin or fanout of the specified register by using the `-include_fanin_logic` or `-include_fanout_logic` option. You can specify any startpoint in the fanin or endpoint in the fanout of the register.

The following example specifies that the tool should replicate the A_reg register and the logic in the path from this register to the U1 cell:

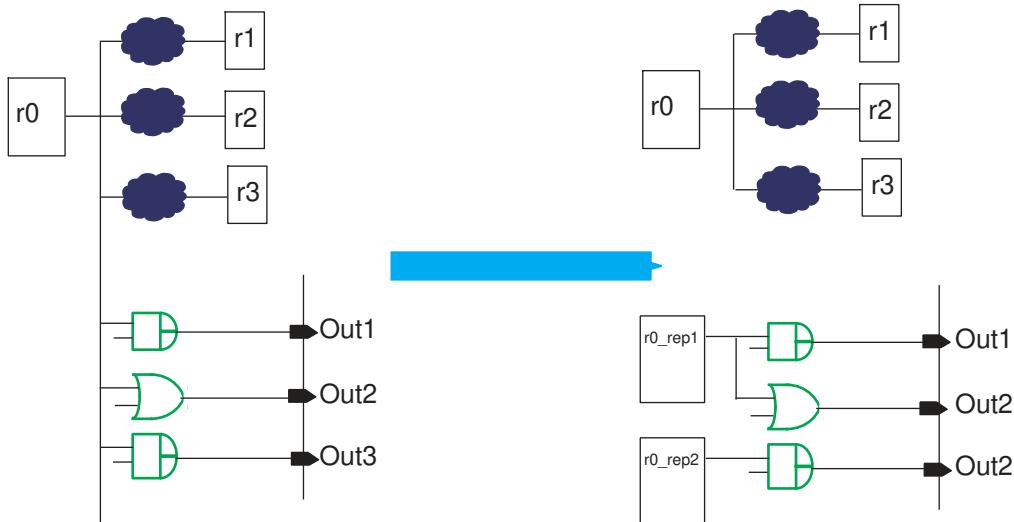
```
fc_shell> set_register_replication -num_copies 2 A_reg \
    -include_fanout_logic U1
```

To specify endpoint registers or primary output ports that must be driven by the original register after replication, as shown in Figure 31, use the `-driven_by_original_register` option.

The following example specifies that after replication, the r0 register must drive the r1, r2, and r3 registers:

```
fc_shell> set_register_replication -num_copies 3 r0 \
    -driven_by_original_register {r1 r2 r3}
```

Figure 31 Using the `-driven_by_original_register` Option



Controlling Register Merging

By default, the tool performs register merging during optimization. To disable register merging, set the `compile.seqmap.enable_register_merging` application option to `false`. The default is `true`. Information about which registers are being merged is stored in the `.svf` file, and the output log file shows information messages about register merging.

You can also control register merging based on the attribute setting by using the `set_register_merging` command. To enable register merging during optimization, specify a `true` value to set the register merging attribute on the specified object list. To disable register merging, specifies a `false` value for the specified object list. The objects can include leaf cells, hierarchical cells, and modules.

This example specifies not to merge registers that have the `u1/count_reg*` string in the instance names:

```
fc_shell> set_register_merging "u1/count_reg*" false
```

This example specifies not to merge all flip-flops of `u1` instance name:

```
fc_shell> set_register_merging "u1" false
```

Selectively Removing or Preserving Constant and Unloaded Registers

To optimize the design by reducing area, the tool can automatically detect and remove constant registers and unloaded registers. You can control this behavior as described in the following topics:

Removing Constant Registers

Certain registers in a design might never change state because they have constant values on one or more input pins. These constant values can either be directly at the input or result from the optimization of fanin logic that eventually leads to a constant input of the register. Eliminating such registers can improve area significantly.

By default, the `compile_fusion` command removes constant registers and prints an information message in the compile log file. This table lists cases in which a sequential element can be eliminated.

Type of register	Data	Preset	Reset
Simple, constant data	1 or 0		
Preset and constant data 1	1	X	
Constant preset	X	1	
Reset and constant data 0	0		X
Constant reset	X		1
Preset, reset, and constant data 1; reset always inactive	1	X	0
Preset, reset, and constant data 0; preset 0 always inactive		0	X
2-input XOR gate where one input is 0	X		
Retention register where D and Q pins are directly connected	1 or 0		

To disable constant register removal for the entire block you are synthesizing, set the `compile.seqmap.remove_constant_registers` application option to `false`, changing it from its default of `true`. To control constant register removal for a specific cell or module , use the `set_constant_register_removal` command.

For example, the following settings prevent the tool from removing the constant register named reg11:

```
fc_shell> set_constant_register_removal reg11 false
```

The following settings globally disables constant register removal for the entire block except for the module named sub5 :

```
fc_shell> set_app_option -name compile.seqmap.remove_constant_registers
-value false
```

```
fc_shell> set_constant_register_removal [get_module sub5] true
```

Information about removed constant registers is written to the verification guidance .svf file and the compile log file. To report the constant registers removed during compile, use the `report_constant_registers` command.

Removing Unloaded Registers

During optimization, the tool deletes registers with outputs that do not drive any load. The combinational logic cone associated with the input of the register can also be deleted if the cell is not used in the design. Register outputs can become unloaded because of redundancy in the circuit or as a result of constant propagation. In some designs where the registers have been instantiated, the outputs might already be unloaded.

By default, the `compile_fusion` command removes unloaded registers in the design and prints an information message in the compile log file. To disable this capability for the block you are synthesizing, set the `compile.seqmap.remove_unloaded_registers` application option to `false`, changing it from its default of `true`. To control unloaded register removal for a specific cell or module , use the `set_unloaded_register_removal` command.

For example, the following settings prevent the tool from removing the unloaded register named reg22:

```
fc_shell> set_unloaded_register_removal reg22 false
```

The following settings globally disables unloaded register removal for the entire block except for the module named sub5 :

```
fc_shell> set_app_option -name compile.seqmap.remove_unloaded_registers
-value false
```

```
fc_shell> set_unloaded_register_removal [get_module sub5] true
```

To report the unloaded registers that were removed during compile, use the `report_unloaded_registers` command.

Reporting Cross-Probing Information for Optimized Registers

The `compile_fusion` command merges registers and removes constant and unloaded registers during register optimization. To print the RTL file name and line number of these optimized register in the `compile_fusion` command log file and in the `report_transformed_registers` command output, set the `compile.seqmap.print_cross_probing_info_for_removed_registers` application option to `true`.

The following table shows the messages printed out during the different types of register optimization.

Table 21 Cross-Probing Information Messages Printed During Register Optimization

Register Optimization	Information Message
Constant register removal	Information: The register 'a/b_reg { {/user/test/design.v:49} }' is removed as constant 0 (SQM-4100)
Unloaded register removal	Information: The register 'a/c_reg{ {/user/test/design.v:193} }' is removed because it is unloaded. (SQM-4101)
Register merging	Information: The register 'a_reg[1] { {/user/testcases/top.v:113} }' is removed because it is merged to 'a_reg[0]'. (SQM-4102)

The following example output shows the cross probing information that is reported by the `report_transformed_registers` command:

Legend:

- c0r - Constant 0 Register Removed
- c1r - Constant 1 Register Removed
- ul - Unloaded Removed
- inv - Inverted
- rep - Replicated
- mrg - Merged register
- mb - Mutibit
- mbd - Mutibit debanked

Register	Transformation	Filename:
Line number		
-----	-----	-----
mega_shift_reg {{a/b/test1.v:112}}	c0r	
Carry_Flag_reg {{a/b/test2.v:117}}	ul	

```
mega_shift_reg[10]           mrg (mega_shift_reg[31])
{ {a/b/test3.v:121 } }
```

Controlling High-Fanout-Net Synthesis

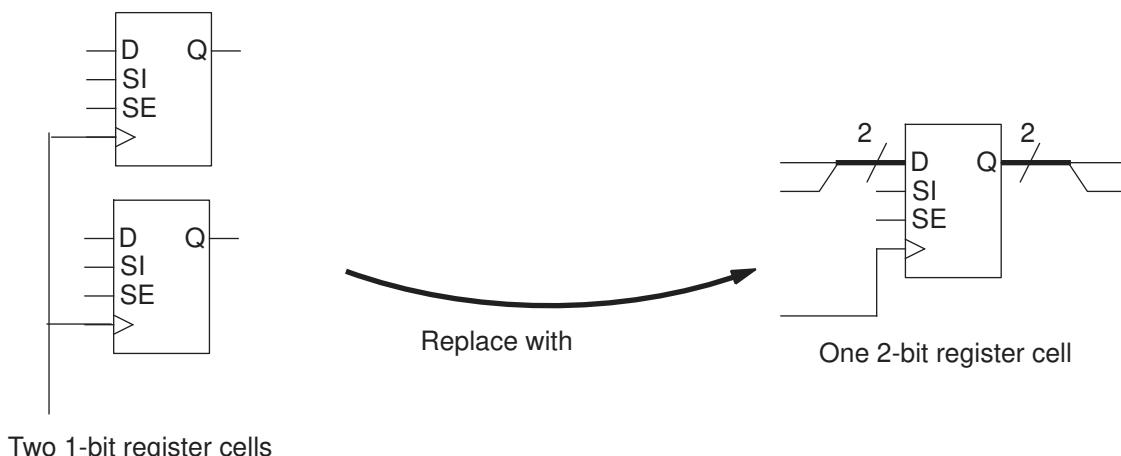
The tool performs high-fanout-net synthesis during the `initial_drc` stage of the `compile-fusion` command. To use globally routed nets to determine the topology for the buffers inserted during high-fanout synthesis, set the `compile.initial_drc.global_route_based` application option to `true`. After high-fanout synthesis, which is performed during the DRC fixing stage of the `compile_fusion` command, the global routes are deleted.

Global-route-based buffering benefits designs with high congestion in routing channels such as the channels between banks of macros. This feature increases the runtime of the `compile_fusion` command. Therefore, you should enable it only for designs with highly congested routing channels.

Performing Multibit Register Optimization

The Fusion Compiler tool can combine (bank) single-bit registers or smaller multibit registers and replace them with equivalent larger multibit registers. For example, the tool can combine eight 1-bit registers or four 2-bit register banks and replace them with one 8-bit register bank or two 4-bit register banks. The tool merges single-bit registers only if they have the same timing constraints, and it copies the timing constraints to the resulting multibit register.

Figure 32 Replacing Multiple Single-Bit Register Cells With a Multibit Register Cell



Replacing single-bit registers with multibit registers reduces

- Area due to shared transistors and optimized transistor-level layout
- The total clock tree net length
- The number of clock tree buffers and clock tree power

The tool can also split (debank) large multibit registers into smaller multibit registers or single-bit registers, if it improves the total negative slack.

Some logic libraries have mixed-drive-strength multibit registers where some bits have a higher drive strength than others. For designs that use such cells, if a violating path goes through a lower-drive-strength bit, the tool can rewire the mixed-drive-strength multibit cell such that the violating path goes through a higher-drive-strength bit.

Performing Integrated Multibit-Register Optimization

To perform integrated multibit register optimization during the `compile_fusion` command,

1. Enable multibit register optimization by setting the `compile.flow.enable_multibit` application option to `true`.

When you do so, by default, the tool performs the following multibit optimizations during different stages of the `compile_fusion` command:

- Combines single registers or smaller register banks at the RTL level to create larger register banks during the `initial_map` stage

To disable this feature, set the `compile.flow.enable_rtl_multibit_banking` application option to `false`.

- Splits multibit register banks created at the RTL level if the register bank includes registers on critical timing paths during the `logic_opto` stage

To disable this feature, set the `compile.flow.enable_rtl_multibit_debanking` application option to `false`.

- Creates multibit register banks considering placement during the `initial_opto` stage

To disable this feature, set the `compile.flow.enable_physical_multibit_banking` application option to `false`.

You can enable placement-based multibit banking during the `final_place` stage of the `compile_fusion` command, instead of the `initial_opto` stage, by setting the `compile.flow.enable_second_pass_multibit_banking` application option to `true` and the `compile.flow.enable_physical_multibit_banking` application option to `false`.

You should enable placement-based multibit banking either during the `initial_opto` stage or the `final_place` stage; not both.

- Splits multibit register banks, if it improves total negative slack (TNS), during the `initial_opto` and `final_opto` stages

To disable this feature, set the `compile.flow.enable_multibit_debanking` application option to `false`.

2. (Optional) .

3. Specify settings for multibit register banking by using the `set_multibit_options` command.

For example, the following command excludes the `reg[1]` cell from multibit optimization during the RTL-banking stage:

```
fc_shell> set_multibit_options -exclude [get_cells "reg[1]"] \
          -stage rtl
```

The following command specifies that during the physical-banking stage, the tool banks only registers that are connected to the same bus:

```
fc_shell> set_multibit_options -bus_registers_only \
          -stage physical
```

4. (Optional) Control multibit register optimization using application options as shown in the following table.

Table 22 General Application Options for Controlling Multibit Register Optimization

To do this	Set this application option
Ignore scan flip-flops that are not included in the SCANDEF information during banking	<code>multibit.banking.enable_strict_scan_check</code> to true
Consider multibit register library cells with multiple scan-in pins and internal scan-in pins during banking	<code>multibit.banking.lcs_consider_multi_si_cells</code> to true
Enable banking of registers driven by equivalent integrated-clock-gating (ICG) cells When you enable this feature, the tool might loose nondefault routing rules applied to the nets connected to the clock-gating cells. Therefore, check and reapply these nondefault routing rules after optimization.	<code>multibit.banking.across_equivalent_icg</code> to true

To do this	Set this application option
Enable timing-driven banking of power management cells The tool skips fixed cells and cells marked with <code>dont_touch</code> , <code>legalize_only</code> , and user-specified <code>size_only</code> attributes	<code>mv.cells.enable_multibit_pm_cells</code> to <code>true</code>
Include cells with user-specified <code>size_only</code> settings during physical banking and debanking	<code>multibit.common.exclude_size_only_cells</code> to <code>false</code>
Enable multibit register rewiring for the <code>compile_fusion</code> command	<code>compile.flow.enable_multibit_rewiring</code> to <code>true</code>
Consider the toggle rate when performing RTL-based multibit reordering at the <code>logic_opto</code> stage	<code>compile.flow.enable_toggle_aware_rtl_multibit_reordering</code> to <code>true</code>

5. (Optional) Create a multibit register bank from specific single-bit registers as described in [Creating a Multibit Register Bank From Specific Single-Bit Registers](#).
6. (Optional) Control the naming style used for multibit banking as described in [Specifying Naming Styles for Multibit Registers](#).
7. Perform optimization by using the `compile_fusion` command.
8. (Optional) Report multibit information.

To report the multibit register banks inferred by the tool, use the `report_transformed_registers -multibit` command, as described in [Reporting Multibit Registers](#).

To report multibit statistics such as the total number of single-bit and multibit registers, the multibit banking ratio, and so on, use the `report_multibit` command.

For more information about how to determine why multibit register banking is not performed for specific cells, see [Identifying Why Multibit Banking Is Not Performed](#).

9. (Optional) Analyze the multibit banking components in the GUI as described in [Viewing Multibit Components in the GUI](#).

Creating a Multibit Register Bank From Specific Single-Bit Registers

Before you run the `compile_fusion` command on an RTL design, you can create a multibit bus consisting of specific unmapped single-bit registers by using the `create_multibit` command. Then, when you run the `compile_fusion` command,

during the `initial_map` stage, the tool maps the single-bit cells of the bus created by the `create_multibit` command into a multibit register bank.

The single-bit registers you specify with the `create_multibit -sort` command must

- Be unmapped
- Be of the same type
- Be in the same logical hierarchy
- Not be part of an existing bus

In the following example, the single-bit cells are combined in alphanumeric descending order, which is reg4, reg3, reg2, and reg1:

```
fc_shell> set_app_option -name compile.flow.enable_multibit -value true
fc_shell> create_multibit {reg2 reg4 reg1 reg3} -sort descending
fc_shell> compile_fusion
```

In the following example the single-bit cells are combined in alphanumeric ascending order, which is reg1, reg2, reg3, and reg4:

```
fc_shell> set_app_option -name compile.flow.enable_multibit -value true
fc_shell> create_multibit {reg2 reg4 reg1 reg3} -sort ascending
fc_shell> compile_fusion
```

In the following example, the single-bit cells are combined in the specified order, which is reg2, reg4, reg1, and reg3:

```
fc_shell> set_app_option -name compile.flow.enable_multibit -value true
fc_shell> create_multibit {reg2 reg4 reg1 reg3} -sort none
fc_shell> compile_fusion
```

Note:

You must set the `compile.flow.enable_multibit` application option to `true` before you run the `create_multibit` command on unmapped single bit registers. If not, the tool issues the following error message:

```
Error: create_multibit command failed: multibit flow is
disabled. (MBIT-068)
```

Specifying Naming Styles for Multibit Registers

Fusion Compiler allows you to customize the naming styles for multibit registers. You can use the application options described in this topic to modify the default naming styles to match the requirements of your design flow or a third-party tool.

Specifying a Name Separator

When creating the name for a multibit register, the tool concatenates the names of the single-bit registers and uses the `_` character as the name separator. For example, when the tool combines the single-bit registers named `reg1` and `reg2`, the multibit register is named `reg1_reg2`.

To change the default name separator, use the

`multibit.naming.multiple_name_separator_style` application option.

Specifying a Name Prefix

To specify a prefix for the multibit register name, use the `multibit.naming.name_prefix` application option.

Compacting Hierarchical Names

When combining single-bit registers with hierarchical names, the tool uses the full hierarchical names of the single-bit registers for the concatenated multibit register name. For example, when the tool combines the single-bit registers named `A/B/P/reg1` and `A/B/Q/reg2`, the multibit register is named `A/B/P/reg1_A/B/Q/reg2`.

To make the multibit register name more compact, specify that common hierarchical names should not be repeated by setting the

`multibit.naming.compact_hierarchical_name` application option to `true`. If you do so, when the tool combines the single-bit registers named `A/B/P/reg1` and `A/B/Q/reg2`, the multibit register is named `A/B/P/reg1_Q/reg2`.

Specifying a Range Separator for Contiguous Bits of a Bused Register

When combining contiguous bits of a bused register, the tool uses the `:` character as the range separator. For example, when the tool combines single-bit bused registers named `regA[0]`, `regA[1]`, and `regA[2]`, the multibit register is named `regA[2:0]`.

To change the range separator, use the `multibit.naming.range_separator_style` application option.

Specifying an Element Separator for Noncontiguous Bits of a Bused Register

When combining noncontiguous bits of a bused register, the tool uses the `,` character as the element separator. For example, when the tool combines the single-bit bused registers named `regA[1]` and `regA[3]`, the multibit register is named `regA[3,1]`.

To change the element separator, use the `multibit.naming.element_separator_style` application option.

Specifying the Expanded Naming Styles When Combining Elements of a Multidimensional Register Array

To specify the expanded naming styles for the multibit registers created by combining elements of a multidimensional register array, use the `multibit.naming.expanded_name_style` application option. For more information on this application option, see the man page.

Reporting Multibit Registers

After you run the `compile_fusion` command, you can report the registers that are transformed due to multibit banking and debanking by using the `report_transformed_registers -multibit` command.

The following example shows a portion of a report generated by the `report_transformed_registers -multibit` command.

```
fc_shell> report_transformed_registers -multibit
...
*****
Attributes:
mb - multibit
mbd - multibit debanked
...
Register      Optimization
-----
MB_2_MB_1    mb (source:MB_2, MB_1)
MB_0_MB_3    mb (source:MB_0, MB_3)
MB_5, MB_6   mbd (source: MB_5_MB_6)
```

Identifying Why Multibit Banking Is Not Performed

To report a summary of the multibit banking ratio, a list of reasons for not banking or debanking cells, and the number of cells for each reason, use the `report_multibit` command, as shown in the following example:

```
fc_shell> report_multibit
*****
Report : report_multibit
...
...
*****
Total number of sequential cells:          4020
Number of single-bit flip-flops:           3149
Number of single-bit latches:              29
Number of multi-bit flip-flops:            842
Number of multi-bit latches:               0
```

Total number of single-bit equivalent cells:	853
(A) Single-bit flip-flops:	3149
(B) Single-bit latches:	29
(C) Multi-bit flip-flops:	5352
(D) Multi-bit latches:	0
Sequential cells banking ratio $((C+D) / (A+B+C+D))$:	62.74%
Flip-flop cells banking ratio $((C) / (A+C))$:	62.96%
BitsPer flop:	2.13

Reasons for sequential cells not mapping to multibit during RTL Banking:

Explanations:

- r12: Cell is single bit because its parent multibit cell was debanked due to improve timing (Number of cells: 91)
- r31: Cell cannot be banked to multibit because it is assigned to use single-bit lib cell (Number of cells: 63)

Reasons for multibit sequential cells not debanking to single bit cells during RTL Debanking:

Explanations::

- r45: Multibit cell cannot be debanked because it is not critical enough (Number of cells: 478)

To report all the cells that are ignored during banking and debanking , use the `-ignored_cells` option with the `report_multibit` command.

To report the compatible multibit and single-bit library cells that can be used for banking and debanking, use the `check_multibit_library` command.

The following example generates a compatible library cell report for RTL banking and debanking:

```
fc_shell> check_multibit_library -stage RTL \
-banking -debanking
*****
Report : check_multibit_library
Flow   : RTL BANKING
*****
-----
Single bit Lib cell          Compatibility        Multi bit Lib cell
-----
SB_Reg1           PIN ORDER MISMATCH  MB_Reg1
SB_Reg2           COMPATIBLE        MB_Reg2
-----
Singlebit with NO Multibit Equivalents
-----
SB_Reg3
```

```
*****
Report : check_multibit_library
Flow   : RTL DEBANKING
*****  

-----  

Multi bit Lib cell      Compatibility      Single bit Lib cell  

-----  

MB_reg2                  COMPATIBLE        SB_reg2  

MB_reg1                  PIN ORDER MISMATCH SB_reg1  

-----  

Multibit with NO Singlebit Equivalents  

-----  

MB_reg3
```

Improving the Banking Ratio

During the `initial_mapping` stage, the tool can prioritize the use of sequential library cells that have functionally equivalent multibit library cells. After you run the `compile_fusion` command, if the `report_multibit-ignored_cells` command indicates that many cells were not banked due to a lack of multibit cells in the library, you can enable this feature by setting the `compile.seqmap.prefer_registers_with_multibit_equivalent` application option to true and rerunning the `compile_fusion` command. However, enabling this feature might degrade the design QoR.

Viewing Multibit Components in the GUI

Fusion Compiler uses the WordView infrastructure to map multibit components. To view multibit components in the GUI, you must complete the following requirements:

- Ensure the library contains multibit components in .ndm format.
- Enable multibit component mapping by setting the `compile.flow.enable_multibit` application option as follows:

```
fc_shell> set_app_options -name compile.flow.enable_multibit \
    -value true
```

Follow these steps to view multibit components in WordView:

1. Read the RTL by using the `analyze` and `elaborate` commands.
2. Run the `compile_fusion -to initial_opto` command.
The tool maps the bused registers to multibit components during compile.
3. (Optional) Check the multibit component mapping information by using the `report_cells` command. For example,
The RTL contains the `reg_state` bused register.

```
module top (...);
output [2:0] reg_state;
always @(posedge clk)
    reg_state <= next_reg_state;
endmodule
```

The report shows that the reg_state bus register is mapped to the MB_DFF multibit component.

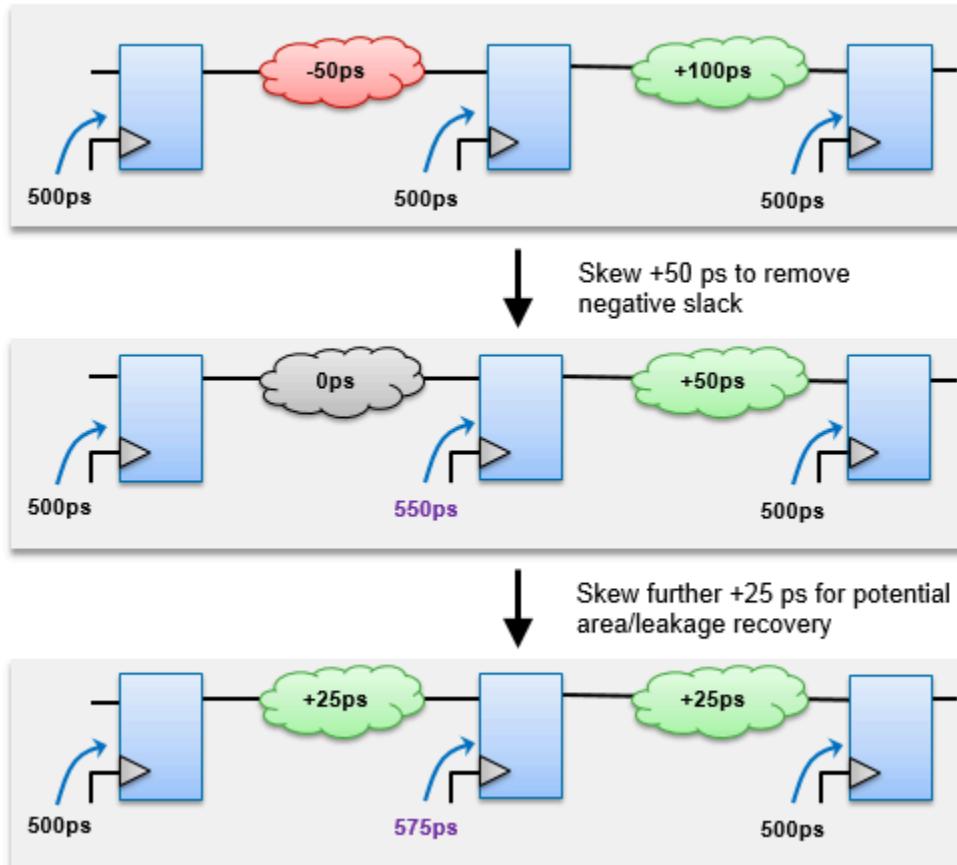
```
fc_shell> report_cells [get_cells *reg_state_reg* -hierarchical]
...
Cell           Reference      Library      Attributes
-----
reg_state_reg[2:0]   MB_DFF     my_mb_lib    n
```

Running Concurrent Clock and Data Optimization

Changing clock latencies can balance the slack in successive timing path stages to optimize clock and data paths. This capability is called concurrent clock and data (CCD) optimization, which can reduce total negative slacks (TNS), worst negative slacks (WNS), area, and leakage power. In addition, it can improve correlation between physical synthesis and place-and-route.

Useful Skew

This figure shows how concurrent clock and data optimization allows for clock skew to remove negative slack and potential area and leakage recovery.



Controlling Concurrent Clock and Data Optimization

By default, the tool performs concurrent clock and data optimization during the `compile_fusion` command. You can disable it by setting the `compile.flow.enable_ccd` application option to `false`. The default is `true`.

During concurrent clock and data optimization,

- The tool uses ideal clock tree, and it updates clock latencies and balance points for all active scenarios.
- The tool adjusts the register latencies within the delayed and advanced latency settings for the most critical sequential elements in the design.

By default, the maximum delayed latency is 100 ps, and the maximum advanced latency is 300 ps.

- All path groups are considered for useful skew computation.
- All boundary timing paths are considered during concurrent clock and data optimization.

Note:

In an incremental compile, enabling concurrent clock and data optimization has no effect.

For best results, you should also do the following:

- Enable the hold scenario even during compile to reduce hold violations that are caused by clock latency insertions during concurrent clock and data optimization.
The tool considers the hold scenario only for concurrent clock and data optimization, but not for other optimization steps.
- Enable concurrent clock and data optimization for the `clock_opt` command.

Controlling Clock Latencies, Path Groups, and Boundary Paths

To specify a maximum delayed latency for useful skew computation, set the `ccd.max_postpone` application option to a specific value, as shown in the following example:

```
fc_shell> set_app_options \
    ccd.max_postpone 50
```

To specify a maximum advanced latency for useful skew computation, set the `ccd.max_prepone` application option to a specific value, as shown in the following example:

```
fc_shell> set_app_options \
    ccd.max_prepone 100
```

To omit useful skew computation for specific path groups, specify one or more path groups with the `ccd.skip_path_groups` application option as follows:

```
fc_shell> set_app_options ccd.skip_path_groups path_group_list
```

To disable concurrent clock and data optimization for boundary timing paths, set the `ccd.optimize_boundary_timing` application to `false`, changing it from its default of `true`. All optimization on I/O paths and latencies for boundary registers will not be adjusted to improve register to register timing.

```
fc_shell> set_app_options ccd.optimize_boundary_timing false
```

Reducing Dynamic Voltage Drop

When running the `compile_fusion` command, you can reduce dynamic voltage drop across a block by enabling dynamic power shaping optimization by setting the `compile.flow.enable_dps` application option to `true`. The tool performs dynamic power shaping optimization during the `final_opto` stage of the `compile_fusion` command.

When you enable this feature, the tool performs power analysis and uses useful skew techniques to reduce dynamic voltage drop. Optionally, you can customize the power analysis performed during dynamic power shaping optimization by using the `ccd.dps.use_case` application option.

Specifying Optimization Targets at the Preroute Stage

When performing concurrent clock and data optimization using the `compile_fusion` command, you can give a higher priority to the WNS optimization of

- Certain path groups by specifying them by using the `ccd.targeted_ccd_path_groups` application option, as shown in the following example:

```
fc_shell> set_app_options -name ccd.targeted_ccd_path_groups \
    -value {PG1 PG2}
```

If a path group you specify with this application option is also specified as a path group to skip with the `ccd.skip_path_groups` application option, the path group is skipped during concurrent clock and data optimization.

- Certain endpoints by specifying a file containing the endpoints by using the `ccd.targeted_ccd_end_points_file` application option, as shown in the following example:

```
fc_shell> set_app_options \
    -name ccd.targeted_ccd_end_points_file \
    -value endpoint_targets.tcl
```

If you specify both path groups and endpoints, the tool optimizes only the specified endpoints that are in the specified path groups.

- The worst 300 timing paths by setting the `ccd.enable_top_wns_optimization` application option to `true`

Transferring to Back End

To transfer the clock latencies and balance points from the front-end tool to the back-end tool, use the `write_ascii_files` command. The command writes the `design.MCMM` directory, which contains timing contexts in Tcl files for . For example,

The `design.MCMM/scenario_scenario.tcl` file contains the following clock latency settings:

```
set_clock_latency -clock [get_clocks {CLK}] -0.3 [get_pins {d_reg[4]/CP}]
set_clock_latency -clock [get_clocks {CLK}] 0.1 [get_pins {d_reg[7]/CP}]
```

The `design.MCMM/mode_mode.tcl` file contains the following balance points:

```
set_clock_balance_points -clock [get_clocks {CLK}] \
    -balance_points [get_pins {d_reg[4]/CP}] \
    -consider_for_balancing true \
    -rise -delay 0.3 -corners [get_corners {default}]
set_clock_balance_points -clock [get_clocks {CLK}] \
    -balance_points [get_pins {d_reg[7]/CP}] \
    -consider_for_balancing true \
    -rise -delay -0.1 -corners [get_corners {default}]
```

Performing Test Insertion and Scan Synthesis

Fusion Compiler test synthesis solution enables transparent implementation of DFT capabilities into the Synopsys synthesis flow without interfering with functional, timing, signal integrity, or power requirements.

The test solution for Fusion Compiler consists of two stages:

- [Test Insertion Before Compile](#)
- [Scan Synthesis During Compile](#)
- [Minimizing Glitches on Asynchronous Set and Reset Lines of Scan Registers](#)

Benefits of two-stage test insertion include

- Better timing, area, power, and congestion QoR for logic and DFT synthesis at the same time
- Better integration of power intent and DFT logic
- Improved congestion optimization that considers the placement for compressor and decompressor (codec) test logic
- Eliminating the incremental compile step to synthesize and place the DFT logic

Test Insertion Before Compile

The tool supports these DFT logic solutions that enable you to implement test logic. When you run the `insert_dft` command before compile, the command generates the DFT logic according to the test solution that is configured.

DFT logic	Commands to configure
DFTMAX compression	<code>set_scan_compression_configuration</code>
DFTMAX Ultra compression	<code>set_scan_compression_configuration</code>
Core wrapping	<code>set_wrapper_configuration</code> <code>set_dft_signal</code> <code>set_boundary_cell</code>
User-defined mode	<code>set_dft_signal -type TestMode</code> <code>define_test_mode</code>
On-chip clocking	<code>set_dft_clock_controller</code> <code>set_dft_signal</code> <code>set_scan_path</code>
Pipeline scan data	<code>set_pipeline_scan_data_configuration</code>

DFT for Multivoltage Designs

For multivoltage designs, the supported flow is as follows:

```
load_upf
commit_upf
# Run DFT setup commands as needed
...
preview_dft
insert_dft
create_mv_cells
compile_fusion
```

You must follow these steps:

- Run the `commit_upf` command after the `load_upf` command.
- Run the `create_mv_cells` command after applying DFT setup and running the `insert_dft` command.

For details, see the *Fusion Compiler Design-for-Test User Guide*.

Scan Synthesis During Compile

During compile, the scan synthesis process tests and prepares the RTL design for test-ready compilation, synthesizes it, tests it again, performs scan insertion, and analyzes the post-DFT design.

- Scan synthesis includes three steps: scan architecture, scan chain stitching, and reordering.
- During compile, the `compile_fusion` command inserts scan structures for all modules in the design using the multiplexed flip-flop scan style.

Note:

You must create scan test and scan in and scan out ports before importing floorplan information. In addition, the floorplan must contain the physical locations of the test and scan ports.

Scan-Only Script Example

```
# Set up the library
...
# Read the design
...
# Scan chain setup
set_scan_configuration -chain_count sc
# DFT signals
# Clocks and asynchronous resets
set_dft_signal -view existing -type ScanClock \
    -port clk -timing [list 45 55]
set_dft_signal -view existing -type Reset \
    -port reset_n -active_state0
# Scan in, scan out, scan enable
set_dft_signal -view spec -type ScanDataIn -port SI
set_dft_signal -view spec -type ScanDataOut -port SO
set_dft_signal -view spec -type ScanEnable -port SE
create_test_protocol
compile_fusion -to initial_opto
# Perform post dft_drc
dft_drc -test_mode Internal_scan
# Generate post dft_drc verbose report
report_dft_drcViolations -test_mode Internal_scan -rule all
# Write out Scandef
write_scan_def design.scan.def
# Write out test model
write_test_model -output design.ctl
# Save output for TetraMAX
write_test_protocol -output design.scan.spf \
    -test_mode Internal_scan
write_verilog -hierarchy all design.v
```

Minimizing Glitches on Asynchronous Set and Reset Lines of Scan Registers

When a design switches between the functional and scan modes, glitches might occur on asynchronous set and reset lines of scan registers, which can reset the state of these registers. The tool can prevent this by adding gating logic to the asynchronous set and reset lines.

To enable this feature, you must specify a top-level port to control the gating logic by using the `set_register_async_gating_pin -port` command. This control port must be a predefined port in the RTL that is not connected to any other net. To get information about such gated scan registers, run the `report_async_gated_registers` command after physical synthesis.

The added gating logic can be an AND or OR gate, depending on the polarity of the asynchronous pins on the scan registers. When gating logic is added, the Formality tool requires additional information to verify the design and the Fusion Compiler tool generates an appropriate .svf file that disables the gating logic in the functional mode.

Note:

This feature is supported only for the in-compile DFT flow.

Specifying Settings for Performance, Power, and Area Improvement

The following topics provide information settings you can use to improve performance, power, and area (PPA) of a design:

- [Enabling High-Effort Timing Mode](#)
- [Enabling Enhanced Delay Optimization to Improve Total Negative Slack](#)
- [Enabling High-Effort Area Mode](#)
- [Enabling the Embedded Area Optimization Flow](#)

Enabling High-Effort Timing Mode

In most cases, you can achieve better timing QoR by enabling high-effort timing mode for compile. To enable this mode, set the `compile.flow.high_effort_timing` application option to 1.

Running high-effort timing mode during compile can increase runtime because the tool performs multiple passes of placement and optimization to obtain optimal timing results.

Enabling Enhanced Delay Optimization to Improve Total Negative Slack

You can improve the total negative slack of a block by using path groups to focus the tool on fixing more violating paths during optimization. However, using this technique can increase the runtime and degrade the area and power QoR.

To enable enhanced delay optimization techniques that improve total negative slack, set the `compile.flow.enhanced_delay_opto_for_pg` application option to 1. This feature has better runtime and better area and power QoR, as compared to the extensive use of path groups to improve total negative slack.

Enabling High-Effort Area Mode

On designs that need further reduction of area, set the `compile.flow.high_effort_area` application option to `true`.

Running high-effort area mode during compile might increase runtime because the tool performs multiple passes of placement and optimization to obtain optimal area results.

Enabling the Embedded Area Optimization Flow

During the `logic_opto` stage of the `compile_fusion` command, as an initial step, the tool performs timing optimization and DesignWare reselection and ungrouping. After this initial step, the tool can perform an optional embedded area optimization flow before moving on to other optimization techniques performed during the `logic_opto` stage.

The embedded area optimization flow consists of dedicated ungrouping, fast area-driven logic restructuring, and light timing optimization. To enable this feature, set the `compile.flow.areaResynthesis` application option to `true`.

This feature can reduce the area of blocks with many DesignWare hierarchies.

Performing Design Analysis

Use the reports generated by Fusion Compiler to analyze and debug your design. You can generate reports before and after you compile your design. Generate reports before compile to check that you have set attributes, constraints, and design rules properly. Generate reports after compile to analyze the results and debug your design.

This section includes the following topics:

- [Reporting Commands and Examples](#)
- [Measuring Quality of Results](#)

- Comparing QoR Data
- Reporting Logic Levels in Batch Mode
- Querying Specific Message IDs

Reporting Commands and Examples

This table shows some reporting commands to report QoR, timing, area, and so on.

Use this command	To do this
report_area	Report area information of the current design. - Area is based on physical libraries. - Numbers of ports and nets are not reported.
report_qor	Report QoR information and statistics for the current design. - Information is based on physical libraries.
report_timing	Report the timing information of the current design.
report_power	Calculate and report dynamic and static power of the design or instance.
report_clock_gating	Report total clock-gating statistics of the design.
report_logic_levels	Report logic levels of the design.

report_area Example

```
fc_shell> report_area -designware -hierarchy -physical
*****
Report : area
Design : test
...
*****
Information: Base Cell (com): cell XNOR2ELL, w=3300, h=6270 (npin=3)
Information: Base Cell (seq): cell LPSDFE2, w=10560, h=6270 (npin=5)
Number of cells: 8
Number of combinational cells: 4
Number of sequential cells: 4
Number of macros/black boxes: 0
Number of buf/inv 0
Number of references 2
Combinational area: 49.66
Buf/Inv area: 0.00
Noncombinational area: 231.74
Macro/Black box area: 0.00
Total cell area: 281.40
```

Hierarchical area distribution

Hierarchical cell	Global cell area		Local cell area		
	Absolute Total	Percent Total	Combi-national	Noncombi-national	Black-boxes
Design					
test	281.40	100.0	37.24	231.74	0.00
test					
mult_10	12.41	4.4	12.41	0.00	0.00
DW_mult_uns_J1_H3_D1					
Total		49.66	231.74	0.00	

Area of detected synthetic parts

Module	Implem.	Count	Perc.of cell area	
			Area	cell area
DW_mult_uns	pparch	1	12.41	4.4%
Total:		1	12.41	4.4%

Estimated area of ungrouped synthetic parts

Module	Implem.	Count	Estimated Perc. of cell area	
			area	cell area
DW_mult_uns	pparch	3	48.28	17.2%
Total:		3	48.28	17.2%

Total synthetic cell area: 60.69 21.6% (estimated)

Core Area:	1767
Aspect Ratio:	1.0902
Utilization Ratio:	0.1593

The above information was from the logic library.

The following information was from the physical library:

Total moveable cell area:	281.4
Total fixed cell area:	0.0
Total physical cell area:	281.4
Core area:	0.000, 0.000, 40.260, 43.890

report_qor Example

```
fc_shell> report_qor
*****
Report : qor
Design : top
...
*****
Scenario          'SC1'
Timing Path Group 'clk'
-----
Levels of Logic:           6
Critical Path Length:     0.28
Critical Path Slack:      -0.19
Critical Path Clk Period: 0.30
Total Negative Slack:     -0.80
No. of Violating Paths:   11
Worst Hold Violation:    -0.07
Total Hold Violation:    -0.13
No. of Hold Violations:  32.00
-----
Cell Count
-----
Hierarchical Cell Count:   2
Hierarchical Port Count:  24
Leaf Cell Count:          287
Buf/Inv Cell Count:       45
Buf Cell Count:           4
Inv Cell Count:           41
CT Buf/Inv Cell Count:   0
Combinational Cell Count: 208
Sequential Cell Count:   79
Macro Count:              1
-----
Area
-----
Combinational Area:        419.34
Noncombinational Area:    736.26
Buf/Inv Area:              67.60
Total Buffer Area:         9.66
Total Inverter Area:      57.94
Macro/Black Box Area:     2337.90
Net Area:                  0
Net XLength:               0
Net YLength:               0
-----
Cell Area (netlist):       3493.49
Cell Area (netlist and physical only): 3493.49
Net Length:                 0
-----
Design Rules
```

```
-----
Total Number of Nets: 363
Nets With Violations: 0
Max Trans Violations: 0
Max Cap Violations: 0
-----
```

report_transformed_registers Example

The `report_transformed_registers` command reports the following information about registers that are modified or removed during compile:

- Sequential output ports inversion
- Constant registers that are removed and the constant values
- Constant registers that are preserved by users
- Unloaded registers that are removed
- Unloaded registers that are preserved by users
- Register merging
- Register replication
- Shift registers
- Multibit registers
- Summary of register optimizations

Use the `report_transformed_registers` command only on a mapped design generated by `compile` or `incremental compile`. If you run the command on an unmapped design, the `report_transformed_registers` command reports no information. The information of transformed or optimized registers is stored in the design library, so you can query the same information by using the `report_transformed_registers` command in different sessions.

For example,

```
fc_shell> report_transformed_registers
*****
Report: report_transformed_registers
Version: ...
Date: ...
*****
Legend:
C0p - constant 0 register preserved
C1p - constant 1 register preserved
C0r - constant 0 register removed
C1r - constant 1 register removed
ulp - preserved unloaded register
```

```

ulr - removed unloaded register
mrg - merged register
srh - shift register head
srf - shift register flop
mb - multibit
rep - replicated
inv - inverted

```

Register	Optimization
I_RISC_CORE/I_DATA_PATH/PSWL_Carry_reg	C0r
I_RISC_CORE/I_STACK_TOP/I3_STACK_MEM/Stack_Mem_reg[0][0]	C0r
I_RISC_CORE/I_STACK_TOP/I3_STACK_MEM/Stack_Mem_reg[1][0]	C0r

To report the count of each type of register transformation, specify the `-summary` option as shown in the following example:

```

fc_shell> report_transformed_registers -summary
*****
Report: report_transformed_registers
Version: ...
Date: ...
*****
Legend:
C0p - constant 0 register preserved
C1p - constant 1 register preserved
C0r - constant 0 register removed
C1r - constant 1 register removed
ulp - preserved unloaded register
ulr - removed unloaded register
mrg - merged register
srh - shift register head
srf - shift register flop
mb - multibit
rep - replicated
inv - inverted

```

Register	Count
Constant Registers Deleted	12
Constant Registers Preserved	1
Unloaded Registers Deleted	101
Unloaded Registers Preserved	38
Shift Registers	170
Merged	12
Multibit	10
Inverted	15
Replicated	4

Measuring Quality of Results

To measure the quality of results (QoR) of the design in the current state and store the quality information in a set of report files, use the `create_qor_snapshot` command. You can capture the QoR information using different optimization strategies or at different stages of the design and compare the quality of results.

The `create_qor_snapshot` command measures and reports the quality of the design in terms of timing, design rules, area, power, congestion, and so on. It stores the quality information into a set of snapshot files in a directory named `snapshot` under the current working directory. To store the quality information in a different directory, use the `time.snapshot_storage_location` application option.

The following example generates a QoR snapshot named `r1` using zero wire load for timing paths in the `snapshot` directory:

```
fc_shell> create_qor_snapshot -name r1 -zero_wire_load
...
No. of scenario = 4
s1 = function
s2 = function_min
s3 = test
s4 = test_min
-----
WNS of each timing group:      s1      s2      s3      s4
-----
REGIN                      -0.34      -      -      -
fref_clk0                   2.34     3.25      -      -
pld_clk                     0.65     1.65      -      -
pll_fixed_clk_central       0.61     1.65      -      -
REGOUT                      0.39     0.44    1.18    1.72
...
-----
Setup WNS:                  -0.34     0.44    1.02    1.10   -0.34
Setup TNS:                 -130.82    0.00    0.00    0.00  -130.82
Number of setup violations: 1152        0        0        0      1152
Hold WNS:                   -0.55    -0.78    -0.30   -1.28   -1.28
...
-----
Area:                         1200539.606
Cell count:                   562504
Buf/inv cell count:           145920
Std cell utilization:         0.24
...
```

Querying QoR Snapshots

To retrieve and report the snapshots created by the `create_qor_snapshot` command, use the `query_qor_snapshot` command. The `query_qor_snapshot` command analyzes

the QoR timing reports previously generated and displays the results in HTML format according to the specified filters.

In this example, the `query_qor_snapshot` command analyzes the placeopt snapshot that is stored in the snapshot directory and then reports the timing paths with worst negative slack values between -2.0 ns and -1.0 ns:

```
fc_shell> create_qor_snapshot -name placeopt
fc_shell> query_qor_snapshot -name placeopt -filters "-wns -2.0,-1.0"
```

Reporting QoR Snapshots

To display QoR information and statistics for the current design, use the `report_qor` command. For example, the following command displays the r1 QoR snapshot that is stored in the snapshot directory:

```
fc_shell> report_qor_snapshot -name r1 -display
```

Removing QoR Snapshots

To remove an existing QoR snapshot from the directory specified by the `time.snapshot_storage_location` application option, use the `remove_qor_snapshot` command. For example, the following command removes the preroute QoR snapshot that is stored in the snapshot directory:

```
fc_shell> remove_qor_snapshot -name preroute
```

Comparing QoR Data

You can generate a web-based report to view and compare your QoR data with a QORsum report by performing the following steps:

1. (Optional) Configure the QoR data, which is captured and displayed in the subsequent steps, by using the `set_qor_data_options` command.

- To specify the most critical power scenarios in your design, use the `-leakage_scenario` and `-dynamic_scenario` options.

The tool uses the power scenarios you specify to generate the high-level summary of the power QoR in the QORsum report. If you do not specify these options, it uses the active power scenario with the highest total power for both the leakage and dynamic scenario for the power QoR summary.

These settings are only used for the power QoR summary. The tool uses the power information of all active power scenarios to capture and report the detailed power information in the QORsum report.

- To specify the most critical clock name and clock scenario, use the `-clock_name` and `-clock_scenario` options.

The tool uses the clock name and scenario you specify to generate the high-level summary of the clock QoR in the QORsum report. If you do not specify these options, the tool identifies the most critical clock and uses it for the clock QoR summary.

These settings are only used for the clock QoR summary. The tool uses all clocks to generate the detailed clock QoR information in the QORsum report.

- To specify a name to identify the run in the QORsum report, use the `-run_name` option.

By default, the tool names each run with a number, such as Run1, Run2, and so on. You can use this option to give a more meaningful name to each run. You can also specify the run name by using the `-run_names` option when you generate the QORsum report by using the `compare_qor_data` command in step 3. If you do so, the tool ignores the run name specified by the `set_qor_data_options -run_name` command.

The following example specifies the leakage-power scenario, dynamic-power scenario, clock scenario, and the clock to use for the corresponding summary in the QORsum report:

```
fc_shell> set_qor_data_options \
    -leakage_scenario S1 -dynamic_scenario S2 \
    -clock_scenario S3 -clock_name sys_clk
```

2. Collect the QoR data for your report by using the `write_qor_data` command. The `write_qor_data` command captures QoR data related to timing, power, runtime, and other metrics.

You can run the `write_qor_data` command multiple times, at each stage of the design flow at which you want to capture QoR data. Use the `-label` and specify a value to indicate at which stage of the flow you are capturing the data.

By default, the `write_qor_data` command captures data relevant to a placed and optimized design. You can specify the `-report_group` option with any of the five supported values (unmapped, mapped, placed, cts, routed) to tune the set of reports being captured, based on the state of the flow where you are capturing the data. For finer-grain control, use the `-report_list` option to explicitly specify each report you want to generate.

The `write_qor_data` command captures a standard set of QoR data to disk from most of the common tool reports like the `report_qor` and `report_power`. If there is additional custom data to capture that is not part of the command collection, use the `capture_qor_data` command to capture fully custom QoR data or GUI layout images to include in the web-based QORsum report. This can be used to supplement the standard QoR data captured by the `write_qor_data` command, or to create fully custom QORsum reports. Related commands include `define_qor_data_panel`,

that lets you create a new panel in the QORsum report for viewing QoR data, and `set_qor_data_metric_properties`, that lets you customize the properties of any metric column in any QORsum report panel, so you can control things like the style of coloring, or the thresholds for coloring when comparing data.

3. Generate the QORsum report by using the `compare_qor_data` command.

This command takes the data captured by one or more runs of the `write_qor_data` command and creates a web-based report for viewing and comparing those results.

You must specify the location of the output of each of the `write_qor_data` runs from the previous step by using `-run_locations` option. Each path location corresponds to the run result that you want to compare.

You can assign a specific name to identify the run in the QORsum report by using the `-run_names` option. If you do so, the tool ignores the run name specify by using the `set_qor_data_options -run_name` command. By default, the tool names each run by a number, such as `run1`, `run2`, and so on.

You can specify the output directory by using the `-output` option. By default, the tool writes the report to a directory named `compare_qor_data`. To overwrite exiting output data, use the `-force` option. By default, the tool does not overwrite existing output data.

4. View the generated QORsum report by using the `view_qor_data` command.

Figure 33 QORsum Report

QORsum												
	Checkpoint	Setup			Logical DRCs		Netlist			Power		
		WNS	TNS	NVE	TranV	CapV	Util	StdCellArea	StdCells	TotalPwr		
Flow: default (BASELINE)												
	compile_initial_opto	-0.22	-9.6	63	62	302	30.1	37024	10522	18.80		
	place_opt_final_opto	-0.21	-9.1	58	26	180	30.7	37695	10744	18.60		
	clock_opt_final_opto	-0.06	-1.2	125	24	185	34.2	42063	11782	25.40		
	route_detail	-0.20	-18.4	237	9	169	34.2	42063	11782	25.40		
	route_opt3	-0.06	-0.3	37	6	182	35.8	43917	12298	28.70		
Flow: no_ccd												
	compile_initial_opto	-0.27	-11.2	55	57	292	30.4	37328	10647	19.70		
	place_opt_final_opto	-0.28	-11.8	57	19	165	30.9	37966	10883	19.40		
	clock_opt_final_opto	-0.23	-6.0	69	18	165	35.6	43712	12294	26.10		
	route_detail	-0.26	-15.0	233	9	168	35.6	43712	12294	26.10		
	route_opt3	-0.26	-11.3	95	0	146	37.1	45525	12645	28.30		
Flow: no_atc												
	compile_initial_opto	-0.23	-9.9	81	79	320	30.1	36969	10492	18.40		
	place_opt_final_opto	-0.22	-9.4	73	16	175	30.6	37577	10733	18.40		
	clock_opt_final_opto	-0.03	-0.9	114	3	162	35.0	42986	11993	26.70		
	route_detail	-0.15	-15.2	215	2	145	35.0	42986	11993	26.70		
	route_opt3	-0.15	-0.4	29	6	170	35.8	43968	12374	28.00		

For more information about exploring the comparison data, see the following topics:

- [Setting Your Baseline Run](#)
- [Changing the QoR Display Style](#)
- [Sorting and Filtering the Data](#)
- [Exploring the Detailed Comparison Data](#)

Setting Your Baseline Run

By default, the first row of data is your baseline run against which all other runs are compared. The name of the baseline run is highlighted in gold to mark it as the “golden,” or baseline, result.

QoRsum		Runs	Metrics	Info	Settings								
QoR Summary	Run	Setup			Logical DRCs			Netlist			Routability		
		WNS	TNS	NVE	TranDRCs	CapDRCs	Util	StdCellArea	Overflow%				
Host Info	1 util60_aspect1-1_layerM7	-0.25	-19.6	186	0	7	64.6	2251	0.36				
App Option Details	2 util70_aspect1-1_layerM7	-0.25	-19.6	186	0	7	74.0	2253	0.14				
Timing / LDRC	3 util80_aspect1-1_layerM7	-0.25	-19.6	186	0	7	86.7	2241	0.27				
Timing Summary	4 util60_aspect1-2_layerM7	-0.25	-20.7	186	0	15	64.1	2255	0.28				
Path Group Details	5 util70_aspect1-2_layerM7	-0.25	-19.6	186	0	12	75.3	2252	0.28				
Logical DRCs	6 util80_aspect1-2_layerM7	-0.25	-19.6	186	0	10	85.6	2245	0.48				
	7 util60_aspect2-1_layerM7	-0.25	-19.3	186	0	10	64.0	2256	0.28				

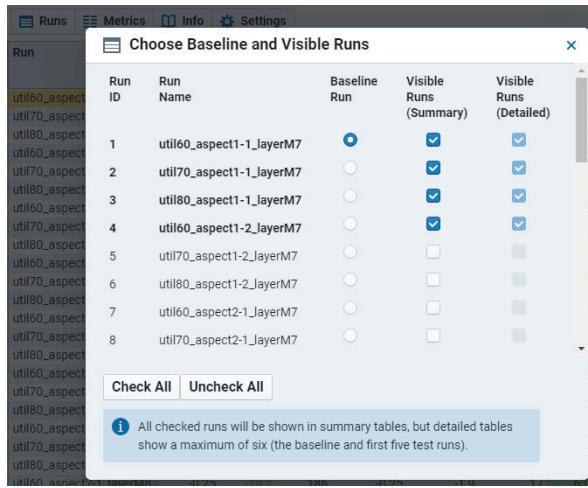
The shading of non-baseline cells indicates the direction and degree by which the data differs from the baseline:

- Red indicates a degradation compared to the baseline; green indicates an improvement compared to the baseline
- Lighter shading represents a smaller difference compared to the baseline; darker shading represents a larger difference compared to the baseline

To view the delta thresholds corresponding to each shade, hover the cursor over the column headers:

Timing				Power	
WNS	TNS	NVE	Total		
-0.47	-37			Color Thresholds: Light (0.5%), Medium (2%), Dark (10%) Skip coloring if delta is less than 0 Metric improves with closer-to-zero values	
6.4%				17.0%	20.0%

To change your baseline run, click the Runs button and select a new run as the baseline from the Baseline Run column, as shown in the following figure:



See Also

- [Changing the QoR Display Style](#)
- [Sorting and Filtering the Data](#)
- [Exploring the Detailed Comparison Data](#)

Changing the QoR Display Style

By default, the comparison tables show the QoR values for all runs. For example, the NVE number for the following run is 2374:

Flow	Timing		
	WNS	TNS	NVE
util0.60_ratio1_1_m9	-0.47	-375.54	2439
util0.70_ratio1_1_m9	-0.50	-409.84	2374

To cycle through different display styles of the data, right-click anywhere in the table. You can view the data

- as a percentage delta against the baseline
- as an absolute delta against the baseline (shown in *italic*)

For example, as a percentage delta, the NVE number shows 2.7% fewer failing endpoints than the baseline:

Flow	Timing		
	WNS	TNS	NVE
util0.60_ratio1_1_m9	-0.47	-375.54	2439
util0.70_ratio1_1_m9	6.4%	9.1%	-2.7%

As an absolute delta, the NVE number shows 65 fewer failing endpoints than the baseline:

Flow	Timing		
	WNS	TNS	NVE
util0.60_ratio1_1_m9	-0.47	-375.54	2439
util0.70_ratio1_1_m9	-0.03	-34.30	-65

See Also

- [Sorting and Filtering the Data](#)
- [Exploring the Detailed Comparison Data](#)

Sorting and Filtering the Data

You can sort and filter the run data to reveal patterns in the results and determine the parameters you want to explore further.

To sort and filter the data, see the following topics:

- [Sorting the Data](#)
- [Filtering Metrics](#)
- [Filtering Runs](#)
- [Example Analysis](#)

See Also

- [Exploring the Detailed Comparison Data](#)

Sorting the Data

Click any column header to sort the data using that metric. The first click performs an ascending sort, while the second click performs a descending sort.

For example, to show the worst TNS numbers at the top of the table and the best TNS numbers at the bottom, click the TNS column header:

Timing		
WNS	TNS	NVE
-0.76	-1106.56	7227
-0.76	-1066.94	6386
-0.76	-1057.45	6500
-0.76	-1018.50	6402
-0.76	-999.51	6459

Click the TNS column header again to show the best TNS numbers at the top and the worst at the bottom:

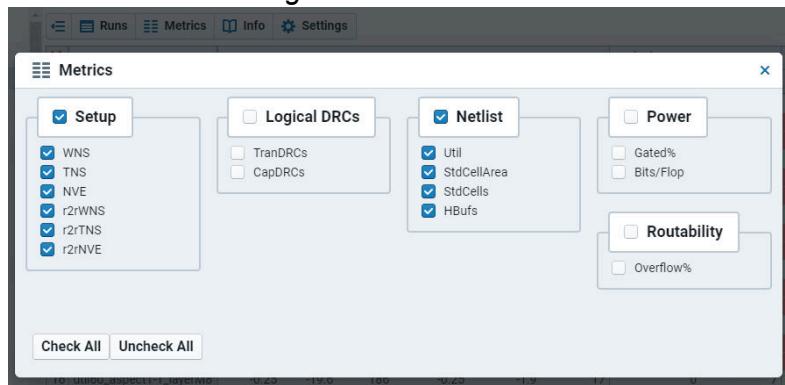
Timing		
WNS	TNS	NVE
-0.44	-328.70	2222
-0.44	-338.62	2405
-0.38	-347.69	2233
-0.40	-352.97	2209
-0.44	-354.87	2175
-0.47	-368.99	2548
-0.48	-371.02	2240
-0.47	-371.04	2593
-0.46	-378.48	2705
-0.49	-391.04	2610
-0.48	-407.98	2659
-0.45	-419.23	2655
-0.46	-420.94	2758
-0.51	-428.26	2719

Filtering Metrics

To control which metrics are displayed in the table, click the Metrics button and select or deselect the metrics from the Metrics dialog box accordingly.

For example, to show only the setup timing, netlist area, and cell counts, select the metrics as shown in [Figure 34](#).

Figure 34 Metrics Dialog Box



Filtering Runs

To control which runs are displayed in the table, click the Runs button and select or deselect the exploration runs accordingly from the Visible Runs (Summary) column.

For example, to display only the first four runs in a table, select the runs as shown in [Figure 35](#).

Figure 35 Choose Baseline and Visible Runs Dialog Box

Run ID	Run Name	Baseline Run	Visible Runs (Summary)	Visible Runs (Detailed)
1	util60_aspect1-1_layerM7	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
2	util70_aspect1-1_layerM7	<input type="radio"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3	util80_aspect1-1_layerM7	<input type="radio"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
4	util60_aspect1-2_layerM7	<input type="radio"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
5	util70_aspect1-2_layerM7	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>
6	util80_aspect1-2_layerM7	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>
7	util60_aspect2-1_layerM7	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>
8	util70_aspect2-1_layerM7	<input type="radio"/>	<input type="checkbox"/>	<input type="checkbox"/>

At the bottom are 'Check All' and 'Uncheck All' buttons. A note states: 'All checked runs will be shown in summary tables, but detailed tables show a maximum of six (the baseline and first five test runs).'

Example Analysis

The following example demonstrates how you might sort and filter your data to narrow down the runs you would like to explore further.

Suppose you open the comparison report shown in [Figure 36](#) and set the util60_aspect1%1_layerM9 run as the base run.

Figure 36 Comparison Report

		Flows	Metrics	Info				
QOR Summary		Flow	Congestion					
			CellsInCA	OverflowTotal	OverflowMax	GRCOverflow	GRCOverflow%	GRCMaxOverflow
Power Details		util85_aspect2%1_layerM9	22480	146020	12	104046	1.94	1
Congestion Details		util60_aspect2%1_layerM9	511	37708	11	30986	0.41	1
Timing		util80_aspect2%1_layerM9	7077	98132	9	68886	1.21	5
		util85_aspect1%2_layerM9	14993	122223	12	84815	1.58	1
Logic DRCs		util60_aspect2%3_layerM9	209	37370	8	30645	0.40	3
Scenario Details		util85_aspect3%2_layerM9	21941	138423	10	98868	1.84	2
Path Type Details		util85_aspect1%1_layerM9	23826	154202	11	110202	2.05	1
Path Group Details		util70_aspect3%2_layerM9	4113	76943	11	55868	0.86	1
Logic Level Details		util80_aspect3%2_layerM9	11362	107760	8	76758	1.35	7
Physical		util70_aspect1%1_layerM9	1819	71258	9	50779	0.78	1
		util60_aspect1%1_layerM9	428	38713	12	31753	0.42	1
Cell Count		util70_aspect2%3_layerM9	1295	69045	14	49055	0.75	1
		util80_aspect2%3_layerM9	15511	120778	9	87960	1.54	2

You could look at your TNS numbers first and sort the data from best TNS to worst TNS, as shown in the following figure:

Flow	Timing		
	WNS	TNS	NVE
util85_aspect2%1_layerM9	-0.44	-328.0	2222
util60_aspect2%1_layerM9	-0.38	-347.69	2233
util80_aspect2%1_layerM9	-0.40	-352.97	2209
util85_aspect1%2_layerM9	-0.44	-354.87	2175
util60_aspect2%3_layerM9	-0.47	-368.99	2548
util85_aspect3%2_layerM9	-0.48	-371.02	2240
util85_aspect1%1_layerM9	-0.46	-378.48	2705
util70_aspect3%2_layerM9	-0.49	-391.04	2610
util80_aspect3%2_layerM9	-0.48	-407.98	2659
util70_aspect1%1_layerM9	-0.45	-419.23	2655
util60_aspect1%1_layerM9	-0.46	-420.94	2758
util70_aspect2%3_layerM9	-0.51	-428.26	2719
util80_aspect2%3_layerM9	-0.52	-435.52	2668
util85_aspect2%3_layerM9	-0.51	-445.29	2879
util60_aspect3%2_layerM9	-0.52	-458.61	2985
util70_aspect2%1_layerM9	-0.47	-458.86	3146
util80_aspect1%1_layerM9	-0.62	-514.59	3243

Notice that the best TNS runs have the M9 top layer, and the worst have the M7 top layer. This suggests that restricting the metal layers significantly impacts timing.

You could then restrict your analysis to your M9 runs by turning off the visibility of your M7 runs, as shown in the following figure:

QORsum		Flows	Metrics	Info
QOR Summary	Flow	Timing		
		WNS	TNS	NVE
Power Details	util85_aspect2%1_layerM9	-0.44	-328.70	2222
Congestion Details	util80_aspect1%2_layerM9	-0.44	-338.62	2405
Timing	util60_aspect2%1_layerM9	-0.38	-347.69	2233
	util80_aspect2%1_layerM9	-0.40	-352.97	2209
Logic DRCs	util85_aspect1%2_layerM9	-0.44	-354.87	2175
Scenario Details	util60_aspect2%3_layerM9	-0.47	-368.99	2548
Path Type Details	util85_aspect3%2_layerM9	-0.48	-371.02	2240
Path Group Details	util60_aspect1%2_layerM9	-0.47	-371.04	2593
Logic Level Details	util85_aspect1%1_layerM9	-0.46	-378.48	2705
Physical	util70_aspect3%2_layerM9	-0.49	-391.04	2610
	util80_aspect3%2_layerM9	-0.48	-407.98	2659
Cell Count	util70_aspect1%1_layerM9	-0.45	-419.23	2655
Cell Area	util60_aspect1%1_layerM9	-0.46	-420.94	2758
	util70_aspect2%3_layerM9	-0.51	-428.26	2719
util70_aspect3%2_layerM9				
util70_aspect2%1_layerM9				
util70_aspect1%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				
util60_aspect2%3_layerM9				
util60_aspect3%2_layerM9				
util60_aspect2%1_layerM9				
util60_aspect1%1_layerM9				

visibility of your higher-utilization runs as shown in [Figure 37](#), leaving you with a manageable subset of exploration runs that better meet your timing and congestion goals.

Figure 37 Displaying Lower-Utilization Runs

QOR Summary		Flow	Congestion				
			CellsInCA	OverflowTotal	OverflowMax	GRCOverflow	GRCOverflow%
Power Details	util70_aspect1%2_layerM9	3889	76765	12	55927	0.86	
Congestion Details	util70_aspect3%2_layerM9	4113	76943	11	55868	0.86	
Timing	util70_aspect2%1_layerM9	3050	74431	11	53750	0.82	
	util70_aspect1%1_layerM9	1819	71258	9	50779	0.78	
Logic DRCs	util70_aspect2%3_layerM9	1295	69045	14	49055	0.75	
Scenario Details	util60_aspect1%2_layerM9	526	39237	11	32149	0.42	
Path Type Details	util60_aspect1%1_layerM9	428	38713	12	31753	0.42	
	util60_aspect2%1_layerM9	511	37708	11	30986	0.41	
Path Group Details	util60_aspect3%2_layerM9	274	37526	8	30894	0.41	
	util60_aspect2%3_layerM9	209	37370	8	30645	0.40	
Logic Level Details							

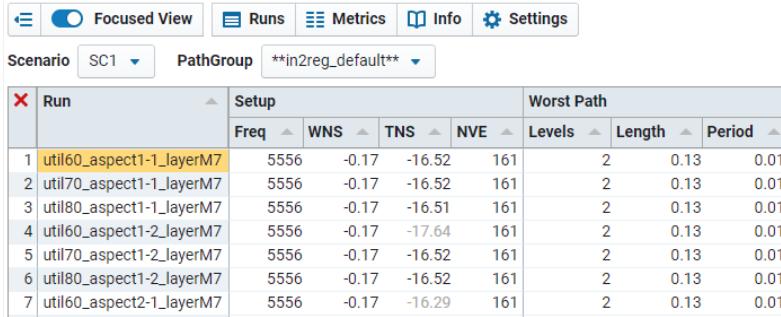
Exploring the Detailed Comparison Data

When you launch the QORsum report, the application opens the QOR Summary table, which summarizes high-level timing, power, and congestion metrics for each of your runs. This and other summary views help you sort and filter the data to pinpoint the runs you want to explore further.

After you have identified your best candidate runs using the summary tables, you can use the detailed tables to provide deeper insights into your data. While summary tables give a high-level overview of design QoR across all of your runs, such as the overall design timing (WNS, TNS, and NVE) for every run, the detailed table can show the path-group-based timing for all path groups in the design. Detailed tables allows you to view the timing of a specific path group for all your runs or view the timing of all path groups for up to six of your runs at once. All detailed tables follow the naming convention of ending with “Details.” For example, Path Type Details table shows information about your path-type-based timing.

By default, the detailed tables start in the Focused View, as shown in [Figure 38](#).

Figure 38 Focused View

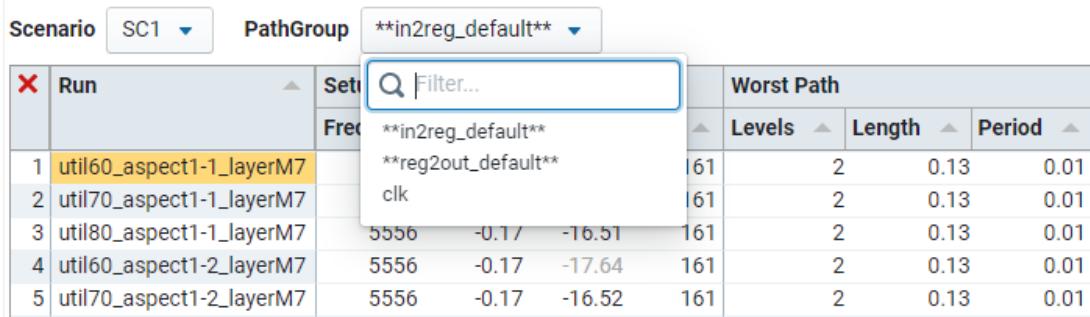


The screenshot shows a software interface for physical synthesis. At the top, there's a navigation bar with tabs: Focused View (which is selected and highlighted in blue), Runs, Metrics, Info, and Settings. Below the navigation bar, there are two dropdown menus: 'Scenario' set to 'SC1' and 'PathGroup' set to '**in2reg_default**'. The main area is a table with three sections: 'Run', 'Setup', and 'Worst Path'. The 'Run' section has a column for 'X' and 'Run' names. The 'Setup' section has columns for 'Freq', 'WNS', 'TNS', and 'NVE'. The 'Worst Path' section has columns for 'Levels', 'Length', and 'Period'. The data in the table is as follows:

X	Run	Setup				Worst Path		
		Freq	WNS	TNS	NVE	Levels	Length	Period
1	util60_aspect1-1_layerM7	5556	-0.17	-16.52	161	2	0.13	0.01
2	util70_aspect1-1_layerM7	5556	-0.17	-16.52	161	2	0.13	0.01
3	util80_aspect1-1_layerM7	5556	-0.17	-16.51	161	2	0.13	0.01
4	util60_aspect1-2_layerM7	5556	-0.17	-17.64	161	2	0.13	0.01
5	util70_aspect1-2_layerM7	5556	-0.17	-16.52	161	2	0.13	0.01
6	util80_aspect1-2_layerM7	5556	-0.17	-16.52	161	2	0.13	0.01
7	util60_aspect2-1_layerM7	5556	-0.17	-16.29	161	2	0.13	0.01

The Focused View shows you one path group at a time. To change your focus, use the drop-down menus at the top. For example, you can specify a scenario and path group as shown in [Figure 39](#).

Figure 39 Focused View Drop-down Menu



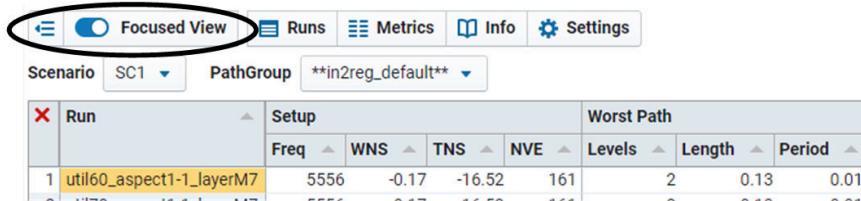
This screenshot is similar to Figure 38, but it includes a 'Filter...' dialog box overlaid on the 'Setup' section of the table. The dialog box has a search icon and a text input field containing the text '**in2reg_default**'. The rest of the table and interface elements are identical to Figure 38.

X	Run	Setup				Worst Path		
		Freq	WNS	TNS	NVE	Levels	Length	Period
1	util60_aspect1-1_layerM7	5556	-0.17	-16.51	161	2	0.13	0.01
2	util70_aspect1-1_layerM7	5556	-0.17	-16.52	161	2	0.13	0.01
3	util80_aspect1-1_layerM7	5556	-0.17	-17.64	161	2	0.13	0.01
4	util60_aspect1-2_layerM7	5556	-0.17	-16.52	161	2	0.13	0.01
5	util70_aspect1-2_layerM7	5556	-0.17	-16.52	161	2	0.13	0.01

The Focused View focuses on a specific property for all runs. However, you can also look at all of the properties for a smaller number of runs.

For example, you can identify the worst path group by viewing all path groups simultaneously. To view the data from the traditional detailed view, click the Focused View toggle button at the top menu, as shown in [Figure 40](#).

Figure 40 Focused View Toggle Button



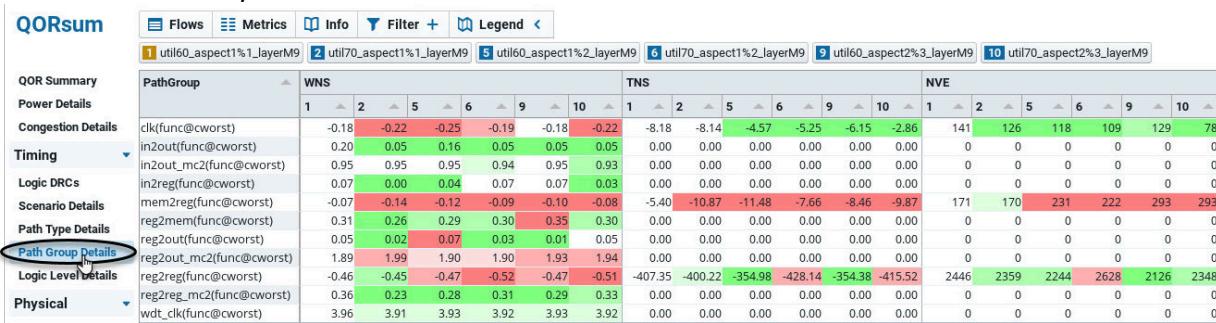
This screenshot shows the same software interface as Figure 38, but with a visual cue: the 'Focused View' toggle button in the top navigation bar is circled with a red oval. The rest of the interface, including the table data, is identical to Figure 38.

X	Run	Setup				Worst Path		
		Freq	WNS	TNS	NVE	Levels	Length	Period
1	util60_aspect1-1_layerM7	5556	-0.17	-16.52	161	2	0.13	0.01

To view detailed comparison data for a particular metric, click one of the detailed views in the panel on the left.

For example, to view the details of your path-group-based timing, select Path Group Details, as shown in [Figure 41](#).

[Figure 41 Path Group Details](#)



													WNS										TNS										NVE													
QOR Summary		Power Details		Congestion Details		Timing		Logic DRCS		Scenario Details		Path Type Details		Physical		1	2	5	6	9	10	1	2	5	6	9	10	1	2	5	6	9	10	1	2	5	6	9	10							
clk(func@cworst)	-0.18	-0.22	-0.25	-0.19	-0.18	-0.22	-8.18	-8.14	-4.57	-5.25	-6.15	-2.86	141	126	118	109	129	78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								
in2out(func@cworst)	0.20	0.05	0.16	0.05	0.05	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
in2out_mc2(func@cworst)	0.95	0.95	0.95	0.94	0.95	0.93	0.00	0.00	0.00	0.00	0.00	0.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
in2reg(func@cworst)	0.07	0.00	0.04	0.07	0.07	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
mem2reg(func@cworst)	-0.07	-0.14	-0.12	-0.09	-0.10	-0.08	-5.40	-10.87	-11.48	-7.66	-8.46	-9.87	171	170	231	222	293	293	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						
reg2mem(func@cworst)	0.31	0.26	0.29	0.30	0.35	0.30	0.00	0.00	0.00	0.00	0.00	0.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
reg2out(func@cworst)	0.05	0.02	0.07	0.03	0.01	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
reg2out_mc2(func@cworst)	1.89	1.99	1.90	1.90	1.93	1.94	0.00	0.00	0.00	0.00	0.00	0.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
reg2reg(func@cworst)	-0.46	-0.45	-0.47	-0.52	-0.47	-0.51	-407.35	-400.22	-354.98	-428.14	-354.38	-415.52	2446	2359	2244	2628	2126	2348	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
reg2reg_mc2(func@cworst)	0.36	0.23	0.28	0.31	0.29	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						
wdt_clk(func@cworst)	3.96	3.91	3.93	3.92	3.93	3.92	0.00	0.00	0.00	0.00	0.00	0.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						

Whereas each row in a summary table shows the results for a flow, each row in a detailed table shows detailed information such as path group names. The flows are shown side by side as columns under each metric. For example, the WNS column group (and the column group of each metric) shows the same six columns labeled 1, 2, 5, 6, 9, 10. These columns represent the results of each flow, and are given a flow ID number, rather than showing the full flow name, to prevent the columns from being too wide. The first column under each metric is the baseline, and the other five are the test flows being compared to that baseline. A legend is shown above the table with the flow ID number in a gold box (for the baseline) or a blue box (for the test flows), followed by the flow name. You can expand or collapse the legend by clicking the Legend button. You can also see the flow ID numbers in the “Choose Baseline and Visible Runs” dialog box, which is opened by clicking the Runs button.

The detailed table views can display up to six runs at one time (your baseline run, plus the first five other runs selected in the Choose Baseline and Visible Runs dialog box). To change the runs that are displayed,

1. Click the Runs button to open the Choose Baseline and Visible Runs dialog box.
2. From the Visible Runs (Summary) column, select or deselect the runs accordingly. This selects and deselects those runs from the Visible flows (Detailed) column.



Filtering the Detailed Data

Detailed views offer filters to focus on specific data. Some views have default filters that are shown automatically. For example, the following detailed views have default scenario and path group filters: Path Type Details, Path Group Details, and Logic Level Details.

You can modify the default filters by

- Removing the filter by clicking the X symbol before the filter name



- Changing the filter value

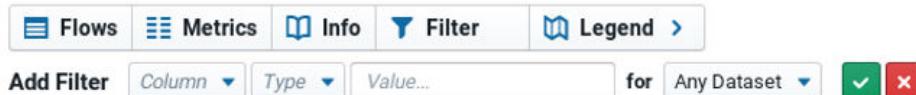


- Enable and disable the filter by clicking anywhere on the filter but the X symbol or value field



You can also apply custom filters to the detailed data. To create a custom filter,

1. Click the Filter button to display the Add Filter fields, which are shown in the following figure:



2. Define the filter by defining the filter criteria and selecting the datasets to include in the results.

To define the filter, select the column and comparison type, and then specify the comparison value.

- To perform a numeric comparison, select one of the following comparison types: =, !=, <, <=, >, or >= and specify a numeric value in the Value field.
- To perform a string comparison, select the contains comparison type and specify a string value in the Value field.
- To perform a regular expression comparison, select the regexp comparison type and specify a regular expression in the Value field.
- To filter based on the available values of the selected column, select the enum comparison type, which populates the Value field with a drop-down menu that

contains the available values, and then enable one or more of the displayed values. When a value is enabled, a check mark is displayed before the value.

To enable or disable a value, highlight the value by clicking it or navigating to it by using the Up and Down arrows, and then press Enter, which toggles the value status. You can also type a string in the Value field to filter the available values in the drop-down menu. To dismiss selected values, click the X symbol.

3. Apply the filter by clicking the green check mark.

For example, to filter the Path Group Details view by displaying the path groups in the func@worst scenario, define the filter as shown in the following figure:

Reporting Logic Levels in Batch Mode

As an alternative to displaying logic-level histograms in the GUI, you can report logic levels by using the `report_logic_levels` command on the command line or in a script (batch mode).

Note:

You must have a mapped design to run logic-level reporting.

You can use the command to report logic levels of

- The entire design or a selected path group
- All types of paths, including same-clock paths, cross-clock paths, infeasible paths, and multicycle paths in all scenarios

The report contains logic distribution and timing information in the summary, logic-level distribution, and logic-level path report sections. By default, buffers and inverters are not reported.

To report logic levels,

1. (Optional) Set a specific logic-level threshold other than the default by using the `set_analyze_rtl_logic_level_threshold` command.

By default, the tool derives the logic-level threshold based on the delay values required for the paths.

- To set a logic-level threshold for the entire design, enter

```
fc_shell> set_analyze_rtl_logic_level_threshold threshold_value
```

- To set a logic-level threshold for a specific path group, enter

```
fc_shell> set_analyze_rtl_logic_level_threshold \  
-group path_group threshold_value
```

The path group setting overrides the global threshold of the entire design.

- To reset all threshold values, enter

```
fc_shell> set_analyze_rtl_logic_level_threshold -reset
```

- To reset a specific group setting, enter

```
fc_shell> set_analyze_rtl_logic_level_threshold \
           -group path_group -reset
```

2. (Optional) Specify the fields in the summary section of the report by setting the `shell.synthesis.logic_level_report_summary_format` application option. The default contains five fields: "group period wns num_paths max_level". You can specify up to eight fields. For example,

```
fc_shell> set_app_options \
           -name shell.synthesis.logic_level_report_summary_format \
           -value "group num_paths max_level min_level avg_level"
...
```

3. (Optional) Specify the fields in the path group section of the report by setting the `shell.synthesis.logic_level_report_group_format` application option. The default contains five fields: "slack ll llthreshold startpoint endpoint". You can specify up to ten fields. For example,

```
fc_shell> set_app_options \
           -name shell.synthesis.logic_level_report_group_format \
           -value "slack ll ll_buf_inv clockcycles startclk endclk"
...
```

4. Report logic levels by using the `report_logic_levels` command. The following example generates a logic-level report of the `clk_i` path group:

```
fc_shell> report_logic_levels -group clk_i
```

Querying Specific Message IDs

To query specific information, warning, or error message IDs that occurred during the previous command step, you can use the `check_error` command.

Follow these steps to set up and query the message IDs:

1. Specify the message IDs to query by using the global scope application option, `shell.common.check_error_list`.

```
fc_shell> set_app_options -name shell.common.check_error_list \
    -value {DES-001 CMD-005}
shell.common.check_error_list {DES-001 CMD-005}
```

2. Use the `link` command as an example for the previous command step.

```
fc_shell> link
Error: Current block is not defined. (DES-001)
```

3. Query the specified message IDs that were issued during the `link` command step by using the `check_error` command.

```
fc_shell> check_error -verbose
{DES-001}
1
```

The result shows that the DES-001 message ID was issued during the `link` command step. If none of the specified message IDs was issued, the `check_error` command returns 0.

4

Clock Gating

RTL clock gating is a power optimization feature provided by the Fusion Compiler tool. This is a high-level optimization technique that can save a significant amount of power by adding clock gates to registers that are not always enabled and have synchronous load-enable or explicit feedback loops. The tool gates flip-flops by extracting common enable signals shared by the flip-flops either in the same hierarchy or across hierarchies. This technique greatly reduces dynamic power consumption by reducing the switching activity on the clock inputs to registers and eliminating the multiplexers. It can also result in a smaller chip area.

Clock gating occurs by default during the `compile_fusion` command using specified clock-gating settings. If you do not specify any clock-gating constraints, the default constraints are used. While you cannot disable clock gating, you can use the clock-gating commands in this section to prevent insertion of clock gates in specific blocks or the whole design.

The tool identifies preexisting clock-gating cells during the `analyze` and `elaborate` commands. The tool inserts one more level of clock-gating cells at the leaf level.

For more information, see the following topics:

- [Introduction to Clock Gating](#)
- [Clock-Gating Prerequisite Conditions](#)
- [Setting Up Clock Gating](#)
- [Clock Gating Flows](#)
- [Replacing Clock Gates](#)
- [Controlling Clock-Gate Latencies](#)
- [Controlling the Number of Clock-Gate Levels](#)
- [Merging Clock Gates](#)
- [Setting Clock Gating Transformations](#)
- [Setting Routing Rules for Clock Gates](#)
- [Clock Gating and Multibit Registers](#)

- Placement-Aware Clock Gating
- Fanin-Based Sequential Clock Gating
- Reporting Clock-Gating Results
- Self-Gating Optimization

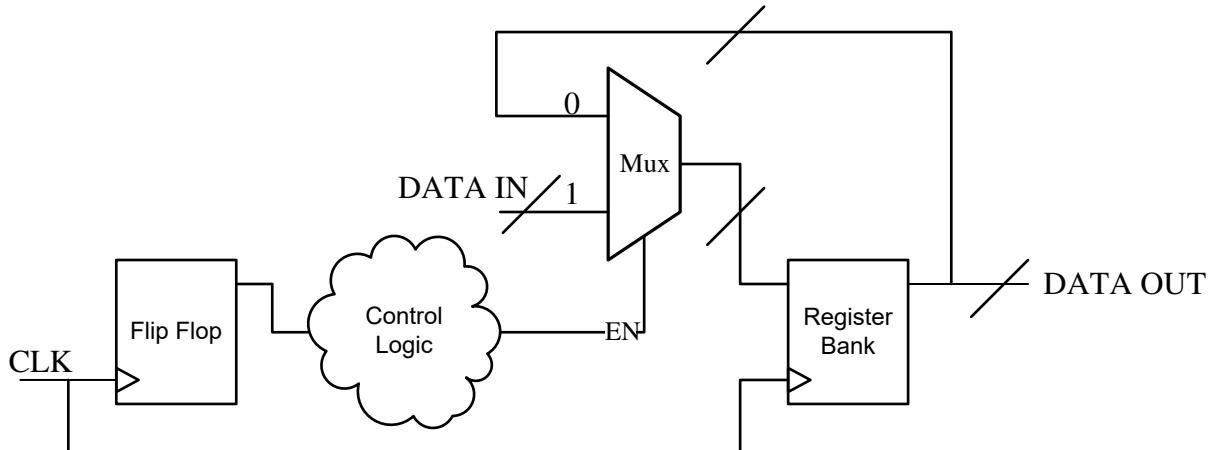
Introduction to Clock Gating

Clock gating applies to synchronous load-enable registers, which are flip-flops that share the same clock and synchronous control signals. Synchronous control signals include synchronous load-enable, synchronous set, synchronous reset, and synchronous toggle.

Synchronous load-enable registers are represented by a register with a feedback loop that maintains the same logic value through multiple cycles. Clock gating applied to synchronous load enable registers reduces the power needed when reloading the register banks.

[Figure 42](#) shows a simple register bank implementation using a multiplexer and feedback loop.

Figure 42 Synchronous Load-Enable Register With Multiplexer



When the synchronous load enable signal (EN) is at logic state 0, the register bank is disabled. In this state, the circuit uses the multiplexer to feed the Q output of each storage element in the register bank back to the D input. When the EN signal is at logic state 1, the register is enabled, allowing new values to load at the D input.

These feedback loops can unnecessarily use power. For example, if the same value is reloaded in the register throughout multiple clock cycles (EN equals 0), the register bank and its clock net consume power while values in the register bank do not change. The multiplexer also consumes power.

Clock gating eliminates the feedback net and multiplexer shown in Figure 42 by inserting a gate in the clock net of the register.

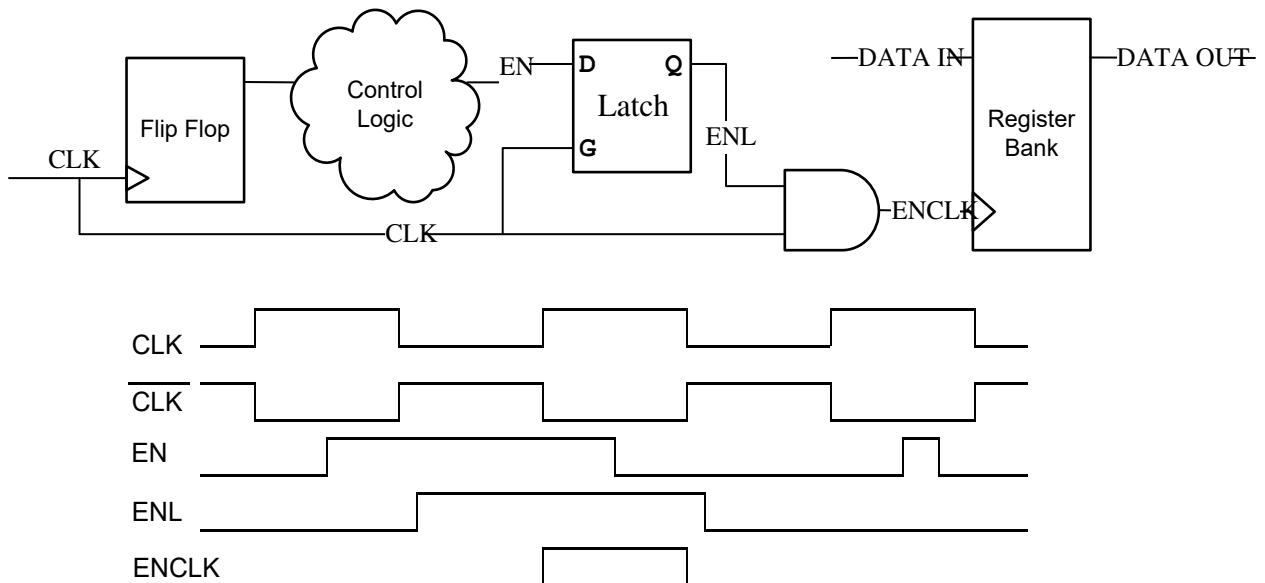
Note:

While applying the clock-gating techniques, the tool considers generated clocks similar to defined clocks.

The clock-gating cell selectively prevents clock edges, thus preventing the gated-clock signal from clocking the gated register.

Figure 43 shows a latch-based clock-gating cell and the waveforms of the signals are shown with respect to the clock signal, CLK.

Figure 43 Latch-Based Clock Gating



The clock input to the register bank, ENCLK, is gated on or off by the AND gate. ENL is the enabling signal that controls the gating; it derives from the EN signal on the multiplexer shown in Figure 42. The register bank is triggered by the rising edge of the ENCLK signal.

The latch prevents glitches on the EN signal from propagating to the register's clock pin. When the CLK input of the 2-input AND gate is at logic state 1, any glitching of the EN signal could, without the latch, propagate and corrupt the register clock signal. The latch eliminates this possibility because it blocks signal changes when the clock is at logic 1.

In latch-based clock gating, the AND gate blocks unnecessary clock pulses by maintaining the clock signal's value after the trailing edge. For example, for flip-flops inferred by HDL constructs of rising-edge clocks, the clock gate forces the gated clock to 0 after the falling edge of the clock.

By controlling the clock signal for the register bank, you can eliminate the need for reloading the same value in the register through multiple clock cycles. Clock gating inserts clock-gating circuitry into the register bank's clock network, creating the control to eliminate unnecessary register activity.

Clock gating does the following:

- Reduces clock network power dissipation
- Relaxes datapath timing
- Reduces congestion by eliminating feedback multiplexer loops

For designs that have large register banks, clock gating can save power and area by reducing the number of gates in the design. However, for smaller register banks, the overhead of adding logic to the clock tree might not compare favorably to the power saved by eliminating a few feedback nets and multiplexers.

Naming Convention for Tool-Inserted Clock Gates

Clock gates inserted by the Fusion Compiler tool use a naming convention that is consistent between runs. The naming convention is as follows:

prefix_base-name[_index]

- The prefix is `clock_gate` by default. To change the prefix, set the `compile.clockgate.clock_gate_name_prefix` application option.
- The base name is the most frequent instance name among the registers gated by the clock gate. The base name does not include bus bit indexes.
- If necessary to resolve a naming conflict between clock gates, the tool adds an underscore separator character and an integer index number beginning with 0.

For example, if the tool inserts a clock gate that gates a register named `out_reg`, the clock gate is named `clock_gate_out_reg`. If the tool inserts a second clock gate for a register named `out_reg` (perhaps on a different bit in the same bus), the clock gate is named `clock_gate_out_reg_0`.

However, if the tool inserts a clock gate for a register named `out_reg_1`, the inserted clock gate is named `clock_gate_out_reg_1_0` to avoid conflicts with the names of clock gates related to the `out_reg` register.

Clock Gating in the Compile Log File

The compile log file contains information about clock gating, which can help you debug simple setup issues.

By default, the clock-gating information is displayed after

- Clock-gate insertion in the `initial_map` stage
- Clock-gate restructuring in the `logic_opto` stage
- Clock-gate restructuring in the `initial_opto` stage
- Self-gating insertion in the `initial_opto` stage, if enabled

The reported information includes the following:

- The number of preexisting clock gates that have a `dont_touch` attribute, which prevents optimization and might lead to unsatisfactory results
- The number of clock-gating library cells that have a CTS purpose, which is required for insertion
- Clock-gating metrics such as the number of elements and the number of gated registers
- A summary of reasons for registers that could not be gated

If the self-gating or fanin-based sequential clock gating features are enabled, their metrics are displayed separately.

An example of the compile log for a run that includes self-gating is as follows:

```
Number of Pre-Existing Clock Gates with dont_touch Attribute: 0
Number of ICG library cells with CTS purpose: 12
-----
          Tool Gated Register Summary
-----
Clock Gating Type      | Number of Clock Gates | Register Count | Bitwidth
                         |                           |                | Equivalent
-----
Regular Clock Gating  |       1                  |       2          |       2
Self Gating           |       0                  |       0          |       0
-----
-----
          Regular Clock Gating Ungated Register Summary
-----
          Ungated Reason           |   Count    | Bitwidth
-----
Enable is Constant One |       2                  |       2
```

Self Gating Ungated Register Summary		
Ungated Reason	Count	Bitwidth
Register is clock gated, and interaction is set to none	2	2
The tool was not able to find enough registers that are compatible to this register in order to be gated.	2	2

Clock-Gate Levels and Stages

The tool uses the following definitions for clock-gate levels and stages in clock-gating commands and options:

- The clock-gate level is related to the fanin of a clock gate.
 - Level 0 refers to a clock source.
 - Level 1 refers to the first clock gate downstream from the clock source.
 - Level 2 refers to a clock gate driven by a clock gate of level 1.
 - Level N refers to a clock gate driven by a clock gate of level N-1.
- The clock-gate stage is related to the fanout of a clock gate.
 - Stage 0 refers to a register driven by a clock gate.
 - Stage 1 refers to a clock gate that only drives registers directly or across buffers, inverters, or hierarchical transitions. In addition, stage 1 is assigned to clock gates whose stage cannot be determined, such as unloaded clock gates.
 - Stage N refers to a clock gate that drives at least one clock gate of stage N-1, either directly or across buffers, inverters, or hierarchical transitions.

The tool keeps a separate count of levels and stages for all clock gates and for tool-inserted clock gates.

[Figure 44](#) shows a simple circuit with two preexisting clock gates (P1 and P2) and three tool-inserted clock gates (T1, T2, and T3). [Table 23](#) lists the levels and stages for each of the clock gates.

Figure 44 Design With Multiple Clock Gates

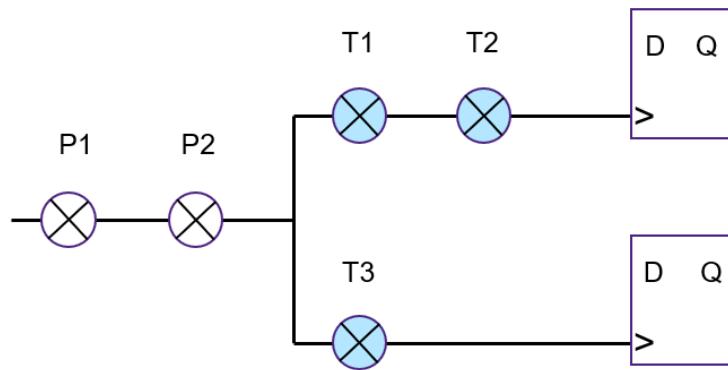


Table 23 Clock-Gate Levels and Stages

Object	Type	Level (all clock gates)	Level (tool-inserted clock gates)	Stage (all clock gates)	Stage (tool-inserted clock gates)
P1	Preexisting clock gate	1	0	4	2
P2	Preexisting clock gate	2	0	3	2
T1	Tool-inserted clock gate	3	1	2	2
T2	Tool-inserted clock gate	4	2	1	1
T3	Tool-inserted clock gate	3	1	1	1

Clock-Gating Prerequisite Conditions

Before gating the clock signal of a register, the Fusion Compiler tool checks to see if certain clock-gating conditions are satisfied. The tool inserts a clock gate only if all the clock-gating conditions are met:

- The circuit demonstrates synchronous load-enable functionality.
- The circuit satisfies the setup condition.
- The register bank or group of register banks satisfies the minimum number of bits you specify with the `set_clock_gating_options -minimum_bitwidth` command. The default minimum bitwidth is 3.

After clock gating is complete, the status of clock-gating conditions for gated and ungated register banks appears in the clock-gating report. For information about the clock-gating report, see [Reporting Clock-Gating Results](#).

Clock-Gating Conditions

The register must satisfy the following conditions for the tool to gate the clock signal of the registers:

- Enable condition

If the register bank's synchronous load-enable signal is a constant logic 1, reducible to logic 1, or logic 0, the condition is `false` and the circuit is not gated. If the synchronous load-enable signal is not a constant logic 1 or 0, the condition is `true` and the setup condition is checked. The enable condition is the first condition that the tool checks. For more information, see [Clock-Gating Enable Condition](#).

- Setup condition

This condition applies to latch-free clock gating only. The enable signal must come from a register that uses the same clock as the register being gated. The setup condition is checked only if the register satisfies the enable condition. For more information, see [Clock-Gating Setup Condition](#).

- Width condition

The width condition is the minimum number of bits for gating registers or groups of registers with equivalent enable signals. The default is 3. You can set the width condition by using the `-minimum_bitwidth` option of the `set_clock_gating_options` command. The width condition is checked only if the register satisfies the enable condition and the setup condition.

Clock-Gating Enable Condition

The enable condition of a register or clock gate is a combinational function of nets in the design. The enable condition of a register represents the states for which a clock signal must be passed to the register. The enable condition of a clock gate corresponds to the states for which a clock is passed to the registers in the fanout of the clock gate. The tool uses the enable condition of the registers for clock-gate insertion.

Enable conditions are represented by Boolean expressions for nets. For example:

```
module TEST (en1, en2, en3, in, clk, dataout);
    input en1, en2, en3, clk;
    input [5:0] in;
    output [5:0] dataout;
    reg [5:0] dataout;
    wire enable;
    assign enable = (en1 | en3) & en2;
    always @ (posedge clk) begin
        if (enable)
            dataout <= in;
        else
            dataout <= dataout;
    end
endmodule
```

In this example, the enable condition for the register bank `dataout_reg*` can be expressed as `en1 en2 + en3 en2`.

Clock-Gating Setup Condition

To perform clock gating, the tool requires that the enable signal of the register bank is synchronous with its clock. This is the setup condition.

For latch-based or integrated clock gating, the tool can insert clock gating irrespective of the enable signal's and the clock's clock domains. If the enable signal and the register bank reside in different clock domains, you must ensure that the two clock domains are synchronous and that the setup and hold times for the clock-gating cell meet the timing requirements.

An exception exists for primary input ports that use a clock specified with the `set_input_delay` command. In this case, the input port is synchronous with the clock and the setup condition is true.

Setting Up Clock Gating

The clock gating feature only supports latch-based integrated clock-gating cells. Tool-inserted clock-gating cells do not have a wrapper hierarchy.

Clock-gating optimization uses the following defaults:

- The maximum fanout is infinite.
- The minimum bitwidth is three.
- A latch-based precontrol cell for both positive-edge and negative-edge triggered flip-flops is used as a clock-gating cell. If only positive-edge or negative-edge clock-gating cells are available, the tool inserts inverters on both sides of the clock-gating cell.

You must load a design and libraries before you can set up clock-gating constraints. To set up clock gating, follow these steps:

1. Load the design and libraries
2. Set the style by using the `set_clock_gate_style` command
3. Set the options by using the `set_clock_gating_options` command
4. Specify objects to be included or excluded from clock gating by using the `set_clock_gating_objects` command
5. Allow other clock-gating cell types for optimization by using the following command:

```
fc_shell> set_lib_cell_purpose -include cts [get_lib_cells ...]
```

For more information, see the following topics:

- [Setting the Clock-Gating Style](#)
- [Setting Clock-Gating Options](#)
- [Enabling or Disabling Clock Gating on Design Objects](#)

Setting the Clock-Gating Style

Use the `set_clock_gate_style` command to select the type of integrated clock-gating cell you want to use. The following options are available:

Option	Description
<code>-test_point</code>	Specifies the test point position with respect to the latch within the integrated clock gate. Valid values are <code>none</code> , <code>before</code> (the default), and <code>after</code> .
<code>-observation_output</code>	Specifies that the library clock-gating cell must have an observation output pin. Valid values are <code>true</code> and <code>false</code> .

Option	Description
-target	Specifies the type of target sequential elements. Valid values are <code>pos_edge_flip_flop</code> and <code>neg_edge_flip_flop</code> . The default is both.
-objects	Objects to which the clock gate applies.

Setting Clock-Gating Options

Use the `set_clock_gating_options` command to define the netlist structure of the inserted clock-gating cells. The following options are available:

Option	Description
<code>-minimum_bitwidth</code>	Specifies the minimum number of accumulated bits that can be gated. The default is 3.
<code>-max_fanout</code>	Specifies the maximum number of cells gated by the same clock-gating cell. The default is infinite.
<code>-objects</code>	Specifies the list of objects to which the clock-gating options are applied. The design objects can be hierarchical instances, power domains, and designs.

Enabling or Disabling Clock Gating on Design Objects

You can enable or disable clock gating on certain design objects by overriding all necessary conditions set by the clock-gating style. The `set_clock_gating_objects` command specifies the design objects on which clock gating should be enabled or disabled during the `compile_fusion` command.

The following example excludes all registers in the subdesign ADDER, except the `out1_reg` bank. The `out1_reg` bank is clock gated according to the specified clock-gating style:

```
fc_shell> set_clock_gating_objects -exclude ADDER \
           -include ADDER/out1_reg[*]
```

The following example sets and then removes the inclusion and exclusion criteria specified by the `-include`, `-exclude`, and `-clear` options:

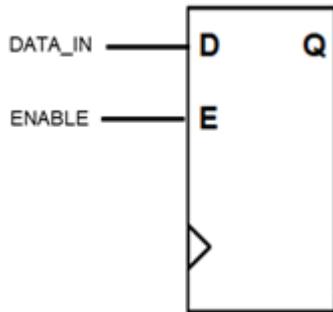
```
fc_shell> set_clock_gating_objects -include ADDER/out1_reg[*] \
           -exclude ADDER/out2_reg[*]
fc_shell> set_clock_gating_objects \
           -clear {ADDER/out1_reg[*] ADDER/out2_reg[*]}
```

Clock-Gating Enable Source Selection

By default, the tool extracts the clock-gating enable condition from one of these sources:

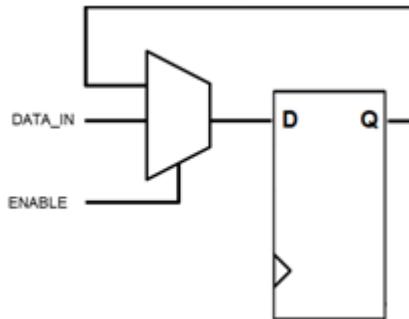
- Signals connected directly to the synchronous enable pin of the sequential element, as shown in [Figure 45](#)

Figure 45 Directly Connected Enable Signal



- The enable condition on the feedback loops of the input and output data pins of the sequential element, as shown in [Figure 46](#)

Figure 46 Enable Signal in a Feedback Loop



When using clock gating, you can specify the signal source of the enable logic. The `set_clock_gating_objects -enable_source` option applies to all objects specified with the `-force` or `-include` options in the same command. If you use the `-enable_source` option, you cannot use the `-clear`, `-reset`, or `-exclude` options.

The `-enable_source` option can be one of the following:

- `none`
- `enable_pin_only`
- `feedback_loop_only`
- `both` (**default**)

- prefer_enable_pin
- prefer_feedback_loop

For example, the following command inserts an always-enabled clock-gating cell:

```
set_clock_gating_objects -force [get_cells u_dfx/input_b_reg*] \
    -enable_source none
```

The following command uses the enable signal from the synchronous enable pin, if there is one. Otherwise, the tool uses the enable on the feedback loop of the register.

```
set_clock_gating_objects -include [get_cells u_dfx/input_c_reg*] \
    -enable_source prefer_enable_pin
```

Clock Gating Flows

The Fusion Compiler tool inserts clock-gating cells during the `compile_fusion` command. The following topics describe clock-gating flows:

- [Inserting Clock Gates in Multivoltage Designs](#)
- [Inserting Clock Gates in an RTL Design](#)

Inserting Clock Gates in Multivoltage Designs

In a multivoltage design, the different hierarchies of the design might have different operating conditions and use different target library subsets. When inserting clock-gating cells in a multivoltage design, the tool chooses the appropriate library cells based on the specified clock-gating style as well as the operating conditions that match the operating conditions of the hierarchical cell of the design. If you do not specify a clock-gating style, the tool uses a default style where the test point is `before` and the observation output is `false`.

If the tool does not find a library cell that suits the clock-gating style and the operating conditions, the tool issues a warning message and does not insert a clock-gating cell. To check whether there are integrated clock-gate cells available for clock-gate insertion, use the `check_clock_gate_library_cell_availability` command.

Inserting Clock Gates in an RTL Design

To insert clock gating logic in an RTL design and to synthesize the design with the clock-gating logic, follow these steps:

1. Read the RTL design.
2. (Optional) Use the `insert_dft` command to insert test cells into the design.

3. Use the `compile_fusion` command to compile the design.

During the compile process, the tool inserts clock gates on the registers qualified for clock-gating. By default, during clock-gate insertion, the `compile_fusion` command uses the clock gating default settings and also honors the setup, hold, and other constraints specified in the logic libraries. To override the setup and hold values specified in the library, use the `set_clock_gating_check` command before compiling the design. The default settings are suitable for most designs.

The `compile_fusion` command automatically connects the scan enable and test ports or pins of the integrated clock-gating cells, as needed.

4. Use the `report_clock_gating` command to report the registers and the clock-gating cells in the design. Use the `report_power` command to get information about the dynamic power used by the design after clock-gate insertion.

The following example illustrates a typical command sequence for clock using default settings:

```
fc_shell> read_verilog design.v
fc_shell> create_clock -period 10 -name CLK
fc_shell> compile_fusion
fc_shell> insert_dft
fc_shell> report_clock_gating
fc_shell> report_power
```

Replacing Clock Gates

If you want to replace all the technology-independent clock gates in your RTL with integrated clock-gating cells from the libraries, use the `replace_clock_gates` command before executing the `compile_fusion` command. The `replace_clock_gates` command identifies combinational elements in the clock network acting as clock gates and replaces them.

When you use the `replace_clock_gates` command, the tool honors the constraints specified by the `set_clock_gate_style`, `set_lib_cell_purpose`, and `set_target_library_subset` commands when it selects a library cell.

When the tool identifies combinational elements, these elements must have only two input pins (the clock pin and the enable signal pin) and only one output pin. If the cell does not meet these criteria, the cell is not a candidate for replacement. The following table lists the WVGTECH instances and their replacement cells.

Combinational Cell in RTL	Integrated Clock Gate Replacement
AND gate	Rising-edge triggered latch-based clock gate

Combinational Cell in RTL	Integrated Clock Gate Replacement
OR gate	Falling-edge triggered latch-based clock gate
NAND gate	Falling-edge triggered latch-based clock gate. Clock input pin is inverted.
NOR gate	Falling-edge triggered latch-based clock gate. Both input pins are inverted.

The new clock-gating cell is inferred with the following name:

```
rep_clock_gate_+<instance_name_of_combinational_cell>
```

The tool reports replaced clock gates as tool-inserted clock gates.

Controlling Clock-Gate Latencies

During synthesis, the Fusion Compiler tool assumes that clocks are ideal. An ideal clock incurs no delay through the clock network. This assumption is made because real clock-network delays are not known until after clock tree synthesis. In reality, clocks are not ideal and there is a nonzero delay through the clock network. For designs with clock gating, the clock-network delay at the registers is different from the clock-network delay at the clock-gating cell. This difference in the clock-network delay at the registers and at the clock-gating cell results in tighter constraints for the setup condition at the enable input of the clock-gating cell.

The tool can obtain clock-gate latencies in the following ways:

- Integrated latency estimation

By default, the tool estimates and updates clock-gate latencies throughout the flow. Estimated clock latencies are more accurate than user-specified clock latencies.

- User-specified latency

You can disable integrated latency estimation for specific instances and specify latency values manually.

Integrated Clock-Gate Latency Estimation

By default, the tool estimates and updates clock-gate latencies throughout the `compile_fusion` command flow. This ensures that the tool uses accurate clock-gate latencies during datapath optimization and useful-skew optimization performed before clock tree synthesis, when clocks are ideal.

If you plan to perform structural multisource clock tree synthesis for your design, ensure that the tool estimates clock-gate latencies that are appropriate for structural multisource clock subtrees by setting the `opt.clock_latency_estimation.estimate_smscts_subtrees` application option to true.

To improve the accuracy of the estimated clock-gate latencies, specify clock tree synthesis settings before you run the `compile_fusion` command. Such settings include the clock tree buffer library cell list, the clock tree inverter library cell list, and clock nondefault routing rules.

Estimated clock latencies are more accurate than user-specified clock latencies. In addition, the tool updates the estimated clock latencies after steps such as placement and multibit banking, while user-specified latencies are static values.

To prevent the tool from estimating the latency for a specific clock gate, set the `dont_estimate_clock_latency` attribute on its clock pin by using the `set_attribute` command, as shown in the following example:

```
fc_shell> set_attribute \
    [get_pins {ICG21/CK}] dont_estimate_clock_latency true
```

For best results, do not use the following features with integrated clock-gate latency estimation:

- Trial clock tree synthesis

To disable this feature, set the `compile.flow.trial_clock_tree` application option to false.

- Clock-gate optimization

To disable this feature, set the `compile.flow.optimize_icgs` application option to false.

- Placement-aware clock gating

The tool automatically disables this feature when clock-gate latency estimation is enabled regardless of the `compile.clockgate.physically_aware` application option setting.

- User-specified latencies

User-specified latencies are set by the `set_clock_latency`, `set_clock_gate_latency`, `set_clock_gating_check -setup`, or `set_path_margin -setup` commands.

The tool does not modify any `set_clock_latency` constraints that you specify on clock-gating cells. Tool-estimated clock-gate latency values are stored as offsets from the user-specified values. You can see the offset values by using the `write_script -format`

`icc2` command and checking for the `set_clock_latency -offset` commands in the generated output. If you do not set any latency values, the reported offsets are equal to the estimated latency values.

The estimated latency offset values are not captured in an SDC file generated by the `write_sdc` command.

Latency estimates are most accurate if the relative locations of the clock gates and their gated registers do not change during clock tree synthesis. To preserve the placement of integrated clock gates, enable the automatic relocation of clock network cells by setting the `cts.compile.enable_cell_relocation` application option to `auto` before clock tree synthesis.

User-Specified Clock-Gate Latency

You cannot disable general integrated clock-gate latency estimation. However, you can disable the feature for specific clock gates by setting a `dont_estimate_clock_latency` attribute on their clock pins. This topic describes how to manually specify latency values on those clock gates.

Specify the clock network latency by using either the `set_clock_latency` or `set_clock_gate_latency` command. The `set_clock_gate_latency` command can be used for both gate-level and RTL designs.

The clock latency specified using the `set_clock_gate_latency` command is annotated on the registers during the `compile_fusion` command when the clock-gating cells are inserted. However, if you modify the latency values on the clock gates after the compilation, you must manually apply the latency values on the existing clock-gating cells using the `apply_clock_gate_latency` command.

Note:

After you modify the clock-gate latency using the `set_clock_gate_latency` command, if you compile your design using the `compile_fusion` command, it is not necessary to use the `apply_clock_gate_latency` command to apply the latency values. The tool annotates the specified value during compilation.

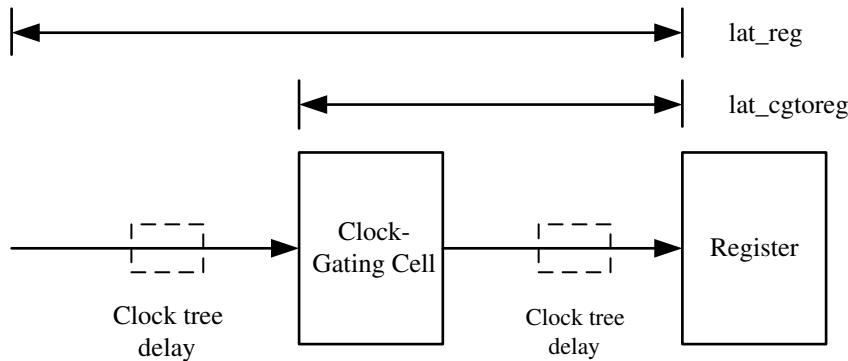
To remove clock latency information previously set on clock-gating cells with the `set_clock_gate_latency` or `apply_clock_gate_latency` commands, use the `reset_clock_gate_latency` command. This command removes the clock latency values on the specified clocks. If you do not specify the clock, the clock latency values on all the clock-gating cells are removed.

The `set_clock_latency` Command

Use the `set_clock_latency` command to specify clock network latency for specific clock-gating cells.

In Figure 47, `lat_cgtoreg` is the estimated delay from the clock pin of the clock-gating cell to the clock pin of the gated register and `lat_reg` is the estimated clock-network latency to the clock pins of the registers without clock gating.

Figure 47 Clock Latency With Clock-Gating Design



For all clock pins of registers (gated or ungated) in the design that are driven by a specific clock, use the `lat_reg` value for the `set_clock_latency` command. For clock pins of all the clock-gating cells, use the difference between the `lat_reg` and `lat_cgtoreg` values for the `set_clock_latency` command. Because the purpose of setting the latency values is to account for the different clock-network delays between the registers and the clock-gating cell, it is important to get a reasonably accurate value of the difference (`lat_cgtoreg`). The absolute values used are less important unless you are using these values to account for clock-network delay issues not related to clock gating.

The `set_clock_gate_latency` Command

When you use the `compile_fusion` command, clock gates are inserted during the compilation process. To specify the clock network latency before the clock-gating cells are inserted by the tool, use the `set_clock_gate_latency` command. This command lets you specify the clock network latency for the clock-gating cells as a function of the clock domain, clock-gating stage, and the fanout of the clock-gating cell. The latency that you specify is annotated on the clock-gating cells when they are inserted by the `compile_fusion` command. You can manually annotate the latency values on the existing clock-gating cells in your design using the `apply_clock_gate_latency` command.

The `set_clock_gate_latency` command takes the following arguments:

- `-clock clock_list`
- `-stage clock_gate_stage`
- `-fanout_latency fanout_list`

The `fanout_list` is a list of tuples specifying a fanout range and a delay decrement. In the following example, three fanout ranges are created: 1 to 5 with a latency value of 0.5, 6-15 with a latency value of 0.8, and 16 to infinity with a latency value of 0.6.

```
set_clock_gate_latency -stage 1 \
    -fanout_latency {{1-5 0.5} {6-15 0.8} {16-inf 0.6}}
```

To specify a clock latency value for clock-gated registers, use the `-stage` option with a value of 0. Because you are specifying the latency value for the clock-gated registers, the value for the `-fanout_latency` option should be `1-inf` (1 to infinity) and the latency is the absolute clock latency value for the registers, as shown in the following example:

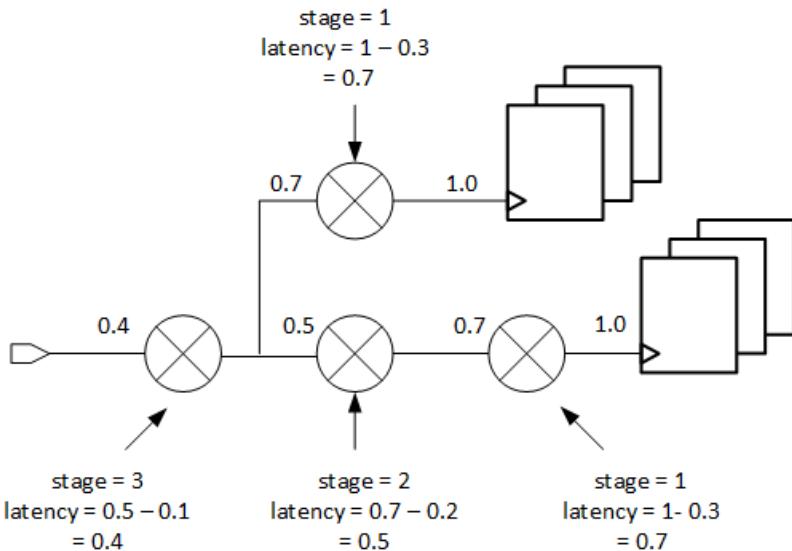
```
set_clock_gate_latency -clock CLK -stage 0 \
    -fanout_latency {1-inf 1.0}
```

If the `-clock` option is not specified, the setting applies to all clocks in the design.

Clock latencies using the `set_clock_latency` command have higher precedence and are not overwritten.

[Figure 48](#) shows an example of clock-gate stages and fanout.

Figure 48 Clock-Gating Stages and Latency Calculations

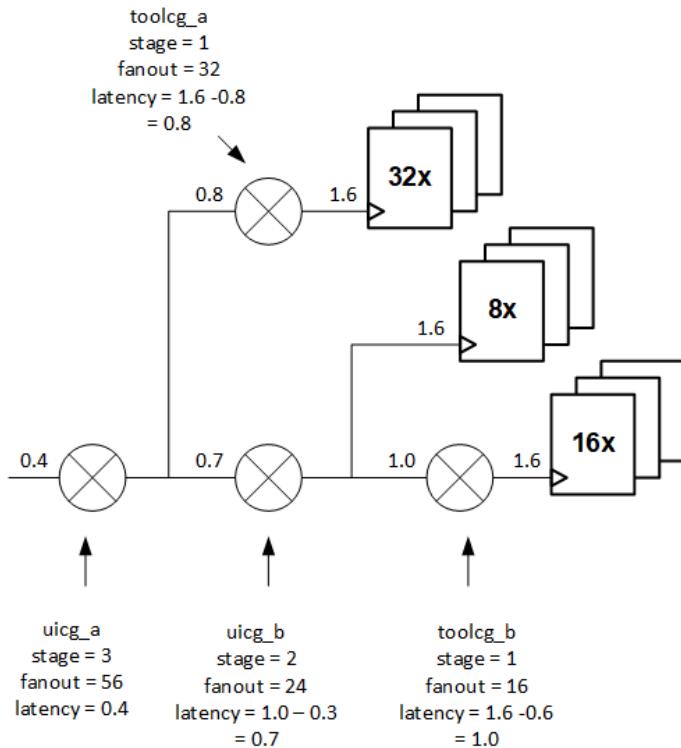


The following commands set the latency values for [Figure 48](#):

```
set_clock_gate_latency -stage 0 -fanout_latency {{1-inf 1.0}}
set_clock_gate_latency -stage 1 -fanout_latency {{1-inf 0.3}}
set_clock_gate_latency -stage 2 -fanout_latency {{1-inf 0.2}}
set_clock_gate_latency -stage 3 -fanout_latency {{1-inf 0.1}}
```

[Figure 49](#) shows another example of fanout and latency calculations.

Figure 49 Latency Calculations With Varying Fanout



The following commands specify the latency and fanout for Figure 49.

```
set_clock_gate_latency -stage 0 \
    -fanout_latency {{1-inf 1.6}}
set_clock_gate_latency -stage 1 \
    -fanout_latency {{1-20 0.6} {21-inf 0.8}}
set_clock_gate_latency -stage 2 \
    -fanout_latency {{1-20 0.2} {21-inf 0.3}}
set_clock_gate_latency -stage 3 \
    -fanout_latency {{1-30 0.1} {31-65 0.8} {66-inf 0.18}}
set_clock_latency "0.4" {uicg_a/CK}
```

Note that for the clock gate uicg_a, the latency value of 0.4 is assigned using the `set_clock_latency` command, which cannot be overwritten.

Controlling the Number of Clock-Gate Levels

The number of clock-gate levels is the count of clock gates between a clock source and a register. Standard clock-gate insertion provides one level of clock-gate insertion. However, designs often contain preexisting clock gates, in which case the design might contain multiple levels of clock gates.

You can modify the default clock-gate insertion operation by allowing additional levels of clock gating, by limiting the number of clock-gate levels, or both.

Clock-Gate Multilevel Expansion

It is difficult to determine the optimum number of clock-gate levels manually. The Fusion Compiler tool can analyze all clock gates and insert additional clock gates to take advantage of shared enable signals. Inserting additional clock gates might improve power, area, or other metrics, depending on the design. If six or more clock gates share an enable signal, the tool inserts an additional level of clock gates. The new clock gates are then considered for further level expansion. As a result, the tool might insert multiple levels of clock gates during this operation.

To allow the tool to expand the number of clock-gate levels, set the `compile.clockgate.enable_level_expansion` application option to `true` (the default is `false`) and set the `compile.clockgate.max_number_of_levels` application option to a nonzero value (the default is 0). In this case, the tool adds clock gates until either of the following conditions is met:

- All common enable signals are identified and used for multilevel clock-gate expansion.
- The user-specified maximum number of clock-gate levels is reached.

The tool names a newly-inserted clock gate as follows, based on the status of the set of clock gates whose enable functions are modified by the new clock gate:

- If at least one of the clock gates is a tool-inserted clock gate, the base name is `clock_gate_ml`. The name is appended with the register base name, similar to the naming convention for other tool-inserted clock gates.
- If all of the clock gates are preexisting clock gates, the tool uses the name of the preexisting clock gate with the largest fanout and appends `_ml_pe`.

The `compile.clockgate.max_number_of_levels` application option specifies the maximum number of allowed clock-gate levels. The default is 0, which allows an infinite number of levels. However, to use the clock-gate level expansion capability, you must set this application option to a nonzero value (in addition to setting the `compile.clockgate.enable_level_expansion` application option to `true`).

If you specify clock-gate level expansion, but the maximum number of allowed levels is low, the expansion operation cannot take full advantage of the tool's ability to analyze common enable signals.

To enable or disable clock-gate level expansion for specific clock gates, use the `set_clock_gate_transformations -expand_levels` command.

The following usage notes apply:

- New clock gates inherit the constraints set by the `set_clock_gate_transformations`, `set_clock_gate_style`, `set_dont_touch`, `set_size_only`, `set_lib_cell_purpose`, and `set_boundary_optimization` commands.
- Clock-gate latency estimations are updated after the insertion operation.
- When the insertion operation modifies the enable signal of a clock gate, the gate retains its original designation of being either preexisting or tool-inserted.
- Self-gating is restricted from adding another clock-gate level if the maximum number of levels is already reached.
- Integrated clock gates inside multibit cells are counted as part of the clock-gate level count. However, they are not candidates for clock-gate level expansion.
- New clock gates do not inherit timing exceptions.

Clock-Gate Collapsing

The default of the `compile.clockgate.max_number_of_levels` application option is 0, which allows an infinite number of levels.

You can set the `compile.clockgate.max_number_of_levels` application option without enabling clock-gate level expansion. In this case, the tool collapses clock-gate levels if the specified limit is less than the number of clock-gate levels that exist after the `initial_map` stage. The tool collapses clock-gate levels as follows:

- The tool first tries to collapse clock gates at levels higher than the specified maximum level, starting from the clock gates at the highest level (closest to the registers).
- If a specific clock gate cannot be collapsed, the tool tries to collapse upstream clock gates until it reaches the lowest level (closest to the source). The collapse can occur across hierarchies. The tool tries to preserve the parent (upstream) clock gate.
- The tool removes all remaining clock gates that do not meet the maximum number of levels.

Controlling Tool-Inserted Clock-Gate Levels

You can specify the maximum number of clock-gate levels that the tool should use for either all clock gates or only tool-inserted clock gates. If you specify a value for all clock gates, you must know how many clock-gate levels already exist in the RTL design. Alternatively, you can specify a value only for tool-inserted clock gates. For details about how the Fusion Compiler tool counts clock-gate levels, see [Clock-Gate Levels and Stages](#).

To specify the maximum number of clock-gate levels, use the `set_clock_gating_tree_options` command with one of the following mutually exclusive options:

- The `-max_total_levels` option specifies the maximum number of levels for all clock gates. The argument is an integer greater than or equal to 1.
- The `-max_tool_inserted_levels` option specifies the maximum number of levels for tool-inserted clock gates. The argument is an integer greater than or equal to 1.

To restrict the clock-level control to specific clock sources, use the `-clocks` option of the `set_clock_gating_tree_options` command, which takes a list of clocks as an argument.

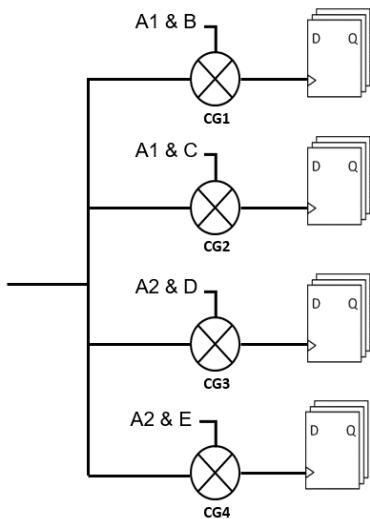
To allow clock-gate removal (also known as ungating) if some clock gates are not collapsible when the tool attempts to meet the maximum clock level constraint, use the `-ungate_if_not_collapsible` option.

Activity-Based Optimization

If you load a SAIF file, the Fusion Compiler tool analyzes the switching activity to evaluate whether to expand a tool-inserted clock gate. This analysis prevents the insertion of unnecessary clock gates.

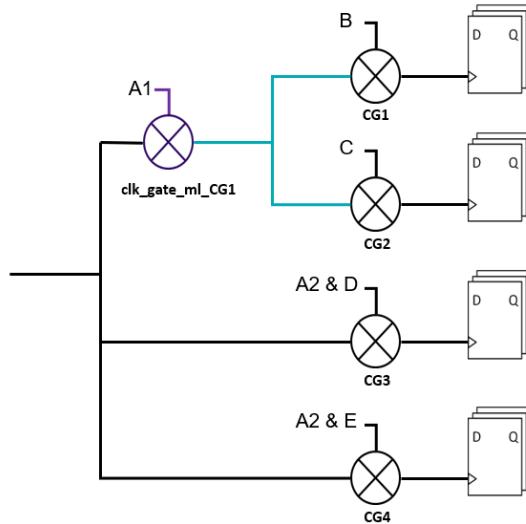
For example, in [Figure 50](#), clock gates A1 and A2 are each common to two paths and are therefore candidates for multilevel expansion.

Figure 50 Before Clock-Gate Level Expansion



However, if clock gate A2 is always enabled, it has a toggle rate of 0 and a static probability of 1. Adding another clock-gate level does not provide any benefit in this case. The final configuration is shown in [Figure 51](#).

Figure 51 After Clock-Gate Level Expansion



If clock-gate level expansion is enabled, the Fusion Compiler tool performs activity-driven analysis by default. If a SAIF file is not available, disable this feature by setting the `compile.clockgate.enable_activity_driven_level_expansion` application option to `false` (the default is `true`).

If clock-gate level expansion is disabled, the tool ignores the `compile.clockgate.enable_activity_driven_level_expansion` application option.

Merging Clock Gates

By merging clock gates, the tool attempts to minimize the number of clock gates while maximizing clock-gating coverage. The tool looks for clock-gating cells that share a common clock and enable signal. This feature is on by default.

The fanout of clock-gating cells in different hierarchies can be moved to a clock-gating cell in a common ancestor if there is no restriction preventing port punching. Note that the tool also checks clock-gating style and multivoltage constraints.

Clock-gating cells with the `dont_touch` or `size_only` constraints are not optimized.

To disable this feature, use the `set_clock_gate_transformations` command.

Setting Clock Gating Transformations

By default, clock gating optimization occurs automatically when you synthesize fhs design. You can, however, define settings to allow or prevent clock gating optimization on specific objects.

The tool makes no distinction among clock gates. Every clock gate is subject to optimization, whether it is a preexisting or tool-inserted clock gate.

The simplest way to restrict clock gating optimization is by using the `set_dont_touch` and `set_size_only` commands. These commands restrict all clock gating optimization by preventing removal, merging, and splitting of all clock gates.

For a more refined approach, use the `set_clock_gate_transformations` command. With this command, you can set specific constraints on each integrated clock gate and control which operations can be performed on the clock gate.

The `set_clock_gate_transformations` command provides the following options:

- `-split_fanout`

Setting this option to `false` prevents fanout splitting even if the fanout is larger than the maximum allowed. This setting also prevents any splitting performed by placement-aware clock gating.

- `-merge_equivalents`

Setting this option to `true` allows a clock gate to be merged with an equivalent clock gate.

- `-collapse_levels`

Setting this option to `false` prevents redundant clock gates from being collapsed

- `-ungate_fanout`

Setting this option to `true` specifies that the clock gate fanout remain ungated

- `-reset`

Resets all clock gate settings for the whole design

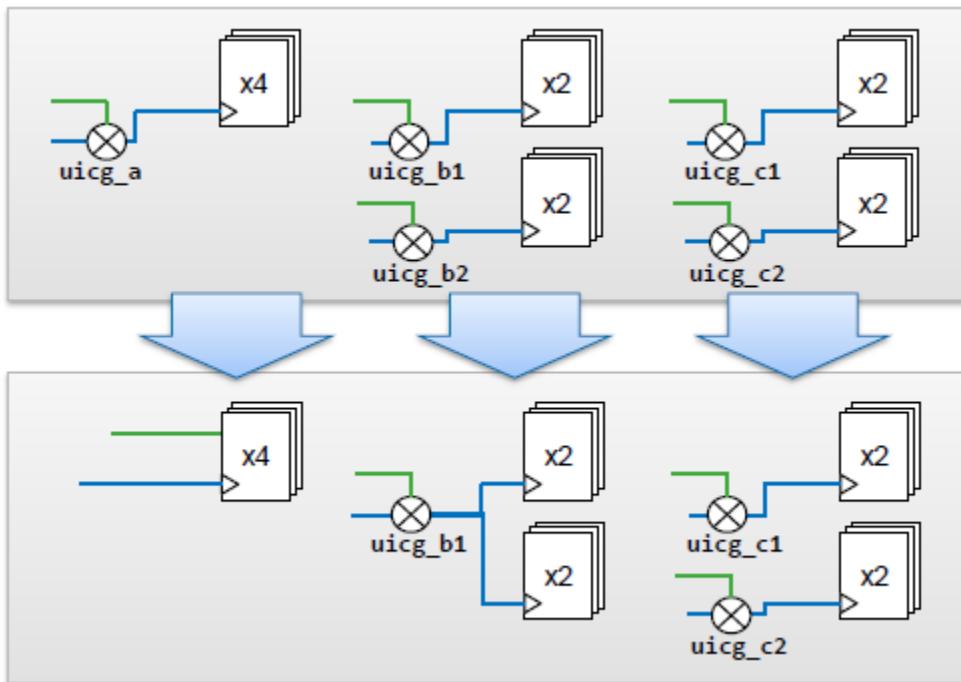
If the `size_only` or `dont_touch` attributes are set on a clock-gating instance or clock-gating library cell, these attributes take precedence over any of the constraints specified by the `set_clock_gate_transformations` command.

Example Using Clock Gating Transformations

For the example in [Figure 52](#), suppose you set the following restrictions:

```
set_clock_gating_options -minimum_bitwidth 6
set_clock_gate_transformations [get_cells uicg_b*] true \
    -ungate_fanout false
set_clock_gate_transformations [get_cells uicg_c*] false
```

Figure 52 Example of Clock Gating Restrictions



During optimization, the tool removes the `uicg_a` clock gate because of a minimum bitwidth violation (minimum bitwidth is 6). The `uicg_b1` and `uicg_b2` clock gates are merged (assuming they have an equivalent enable signal). Even though the minimum bitwidth is not met, ungating is not allowed, so the registers remain gated. The `uicg_c1` and `uicg_c2` clock gates do not allow any transformations, therefore no optimizations are performed for these clock gates.

Setting Routing Rules for Clock Gates

When you set constraints for physical implementation, you can set nondefault routing rules to define stricter (wider) wire width and spacing requirements for specific nets to improve timing and reduce crosstalk. To specify nondefault routing rules, use the `create_routing_rule` and `set_routing_rule` commands.

However, the `set_routing_rule` command does not allow annotation on the output nets of tool-inserted clock gates, because these nets do not exist until after the execution of the `compile_fusion` command.

To specify nondefault routing rules that are applied automatically on nets connected to the output pins of tool-inserted clock gates, use the `set_clock_gate_routing_rule` command. These rules are applied on clock gates after the `compile_fusion` command.

For example,

```
create_routing_rule cg_rule...
set_clock_gate_routing_rule -rule cg_rule
...
compile_fusion
...
report_routing_rules
```

Clock Gating and Multibit Registers

In the Fusion Compiler flow, clock gating works in conjunction with multibit register mapping. The multibit packing ratio has a higher precedence than clock gating.

The following command enables multibit mapping:

```
fc_shell> set_app_options -name compile.flow.enable_multibit -value true
```

Clock-gating cells are not inserted to gate preinstantiated multibit registers.

For more information on clock gating reports, see [Reporting Clock-Gating Results](#).

Placement-Aware Clock Gating

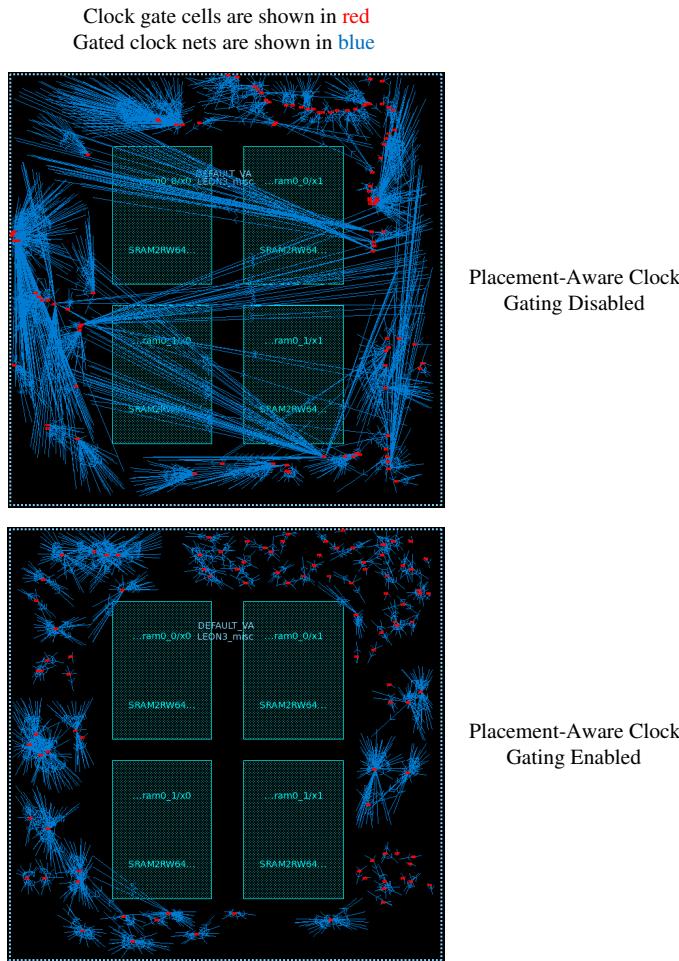
When you enable placement-aware clock gating, the tool performs the following optimizations:

- Replication of clock gates with timing-critical enables
- Adjustment of clock gates so they are placed closer to their gated registers
- Automatic clock network latency annotation for clock-gate cells

To enable this feature, set the `compile.clockgate.physically_aware` application option to `true`. (The default is `false`.)

The example in [Figure 53](#) shows the difference between using and not using physically-aware clock gating. In this layout view, you can see that the integrated clock gates (shown in red) are physically closer to their gated registers.

Figure 53 Placement-Aware Clock Gating



Fanin-Based Sequential Clock Gating

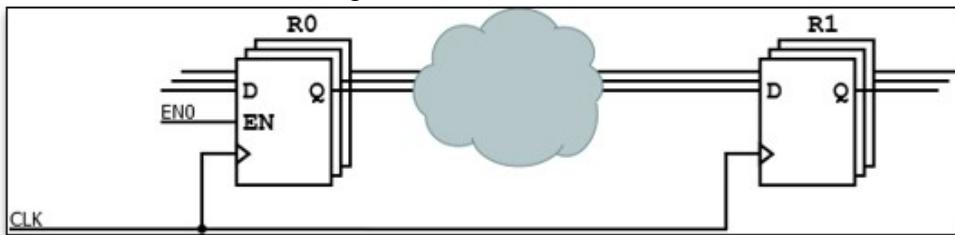
Sequential clock gating is the process of extracting and propagating enable conditions from upstream or downstream sequential elements in the data path for the purpose of enabling clock gate insertion for additional registers. The Fusion Compiler tool supports fanin-based sequential clock gating, in which a register can be gated during a given clock cycle if all the registers in the transitive fanin did not change state in the previous clock cycle.

To enable this feature, set the `comple.clockgate.fanin_sequential` application option to `true` (the default is `false`). This feature requires the FC-New-Tech-Power license.

[Figure 54](#) and [Figure 55](#) illustrate the general concept of fanin-based sequential clock gating. The circuit in [Figure 54](#) has a register bank R1 whose transitive fanin consists

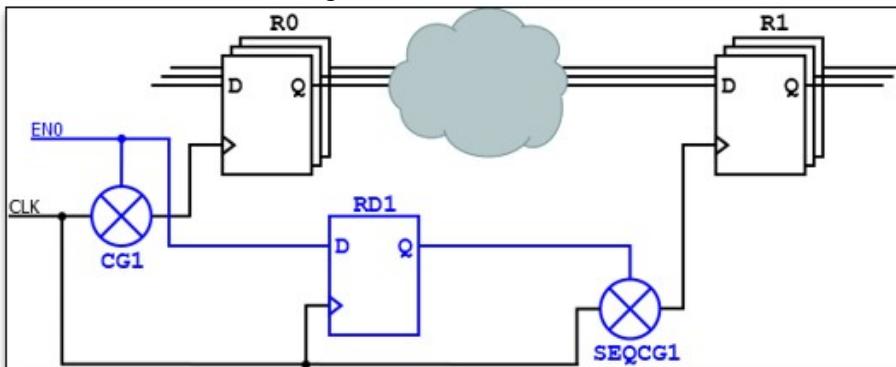
of register bank R0. All registers in bank R0 share the same enable condition EN0 and all registers in both register banks are clocked by clock CLK. If in one clock cycle the registers in bank R0 are disabled (in other words, enable signal EN0 is 0), those registers do not change their values. Therefore, in the next clock cycle, the registers in bank R1 do not change their values and consequently they are candidates for clock gating.

Figure 54 Before Clock Gating



[Figure 55](#) shows the clock-gate implementation. Clock gate CG1 is first inserted for register bank R0. The tool creates a new register RD1 to generate a delayed version of enable signal EN0. The delayed signal drives sequential clock gate SEQCG1 for register bank R1.

Figure 55 After Clock Gating



The Fusion Compiler tool implements several different clock-gating strategies as needed, based on analyzing the circuit. For example, if the fanin registers do not all have the same enable signals, the tool looks for common signals upstream from the enable signals and uses those signals as inputs to the inserted delay register.

Sequential clock gates are inserted by the `compile_fusion` command. During the `initial_map` stage, the tool analyzes the local hierarchy of the registers and inserts clock gates as needed. During the `logic_opto` stage, the tool performs analysis across hierarchical boundaries and inserts additional clock gates as needed.

The name of a clock gate inserted as a result of fanin-based sequential clock gating begins with the prefix `seq_clock_gate_fi_`. The rest of the name follows standard clock-

gate naming conventions. If the tool inserts a delay register as a result of this feature, the register name begins with the prefix `reg_seqcg_` followed by standard clock-gate naming conventions (in other words, the most common base name among the gated registers).

Setting Sequential Clock Gating Options

Sequential clock gates inserted by the tool honor the settings in the `set_clock_gating_style` command, which applies to all clock gates. However, the following commands are specific to fanin-based sequential clock gates:

- The `set_fanin_sequential_clock_gating_options` command

This command specifies the minimum bitwidth and maximum fanout that a fanin-based sequential clock gate is allowed to gate. The default minimum bitwidth is 3 and the default maximum fanout is unlimited. You can specify a list of designs, power domains, and hierarchical instances to which the settings apply.

- The `report_fanin_sequential_clock_gating_options` command

This command lists the settings for design objects.

- The `set_fanin_sequential_clock_gating_objects` command

This command allows you to specify registers, hierarchical cells, power domains, or modules to be included in or excluded from fanin-based sequential clock gate insertion.

- The `report_fanin_sequential_clock_gating_objects` command

This command lists the settings explicitly set with the `set_fanin_sequential_clock_gating_objects` command.

Regardless of any settings made with the

`set_fanin_sequential_clock_gating_options` and `set_fanin_sequential_clock_gating_objects` commands, the following conditions prevent fanin-based sequential clock-gate insertion:

- A register has asynchronous state changes.
- A register contains objects other than edge-triggered flip-flops in its transitive fanin. Disallowed objects include ports, latches, and registers with asynchronous state changes.
- The enable signal extracted from registers in the transitive fanin depends on their output pins.
- The toggle rate of a register's clock signal is so low that clock gating provides no benefit.

- Registers in the transitive fanin are controlled by a different base clock or triggered by the opposite clock edge.
- At least one of the registers in the transitive fanin is always enabled.
- The next state function of the register under analysis or of the registers in its transitive fanin depends on logic that contains a `dont_care` condition in the RTL.
- The register under analysis is multibit and its slices do not share the same enable signal.

Reporting Sequential Clock Gates

If sequential clock gating is enabled, the `report_clock_gating` command automatically displays a summary table, similar to the following:

Sequential Clock Gating Summary

		Bitwidth
No of sequential clock gates	2	
No of fanin-based sequential clock gates	2 (100.0%)	
No of sequentially-gated registers	7 (21.2%)	7 (21.2%)
No of fanin-based sequentially-gated regs	7 (21.2%)	7 (21.2%)
No of registers not sequentially gated	26 (78.8%)	26 (78.8%)
Total number of registers	33	33

Use the `report_clock_gating -ungated` command to report the reasons why sequential clock gating is not implemented. The report is similar to the following:

```
fc_shell> report_clock_gating -ungated
-----
          Not fanin-based sequentially gated registers
-----
      Register | Reason for not gating           | What's next?
-----
q2_reg[0]  | Register contains object different than edge-triggered
            | flip-flops in its transitive fanin | Check the register
            |                                     | transitive fanin
q2_reg[4]  | Register contains object different than edge-triggered
            | flip-flops in its transitive fanin | Check the register
            |                                     | transitive fanin
u/reg1_reg[2] | Register enable function is not a cover of the transitive
            | fanin enable function             | Check the register
            |                                     | enable condition
-----
      Total   7
```

Reporting Clock-Gating Results

After compiling your design, you can check the results using the `report_clock_gating` command. An example is shown in [Figure 56](#).

Figure 56 Example of a report_clock_gating Report

Clock Gating Summary	
Number of Clock gating elements	5046
Number of Tool-Inserted Clock gates	4818 (95.48%)
Number of Pre-Existing Clock gates	228 (4.52%)
Number of Gated registers	120449 (89.80%)
Number of Tool-Inserted Gated registers	117450 (87.56%)
Number of Pre-Existing Gated registers	99950 (74.51%)
Number of Ungated registers	13688 (10.20%)
Total number of registers	134137
Max number of Clock Gate Levels	5
Number of Multi Level Clock Gates	3997

The report provides a summary of the clock-gating statistics. The term "pre-existing clock gates" refers to user-instantiated clock gates. The preexisting category also includes registers that are gated by a tool-inserted clock gate. This includes registers that are gated by both preexisting and tool-inserted clock-gating cells.

By default, the tool reports multilevel statistics. This includes the maximum level of clock-gating cells in the design.

Clock-gate levels are counted from the clock source toward the register.

If multibit flip-flops are found in the design, multibit composition is automatically reported.

Figure 57 Example of a Multibit Composition Report

Clock Gating Multibit Composition		
	Actual Count	Single-bit Equivalent
Gated Registers		
1-bit	23183	23183
2-bit	1557	3114
4-bit	22562	90248
Total	47302	116545
Tool-Inserted Gated Registers		
1-bit	3895	3895
2-bit	1240	2480
4-bit	10995	43980
Total	16130	50355
Pre-Existing Gated Registers		
1-bit	21498	21498
2-bit	1233	2466
4-bit	20343	81372
Total	43074	105336
Ungated Registers		
1-bit	3356	3356
2-bit	443	886
4-bit	4533	18132
Total	8332	22374
Total Registers		
1-bit	26539	26539
2-bit	2000	4000
4-bit	27095	108380
Total	55634	138919

To report the details on ungated registers, use the `-ungated` option for `report_clock_gating`. For details on the gated registers, use the `-gated` option.

Reporting Clock-Gate Enable Signals

A clock gate often has an enable signal that is a Boolean expression comprised of one or more of the following reference (or invariant) points:

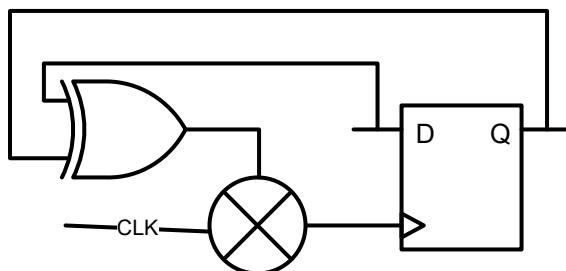
- Hierarchical ports
- Sequential output pins
- Black box output pins

To report the clock gate enable signal as a Boolean function, use the `report_clock_gating_enable_condition` command. You must specify a collection of clock gates to be reported.

Self-Gating Optimization

The tool supports self-gating optimization, which occurs at the `compile_fusion` command. This feature reduces dynamic power consumption by turning off the clock signal of certain registers during clock cycles when the data remains unchanged.

Figure 58 Example of a Self-Gating Cell



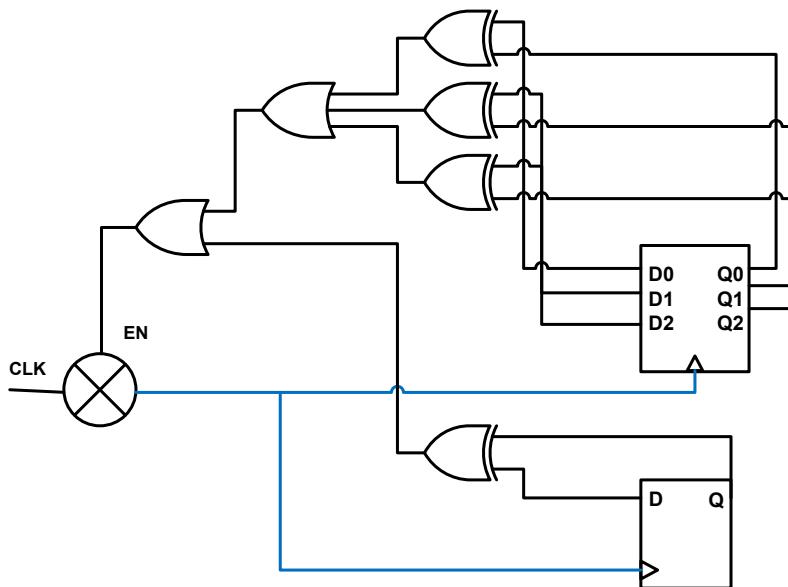
Registers with an enable condition that cannot be inferred from the existing logic can only be gated using the self-gating technique. They cannot be gated using traditional clock gating. By default, the tool supports self-gating only on registers that are not gated. You can use the `set_self_gating_options` command to allow self-gating on these registers. However, the time duration that the clock signal is turned off might increase for these registers.,

To ensure QoR improvements, the self-gating algorithm takes timing and power into consideration. A self-gating cell is inserted for the registers if

- There is enough timing slack available in the register's data pin. For designs with multiple scenarios, the algorithm considers the timing of the worst case among active scenarios enabled for setup.
- Internal dynamic power of the circuit is reduced. For designs with multiple scenarios, the algorithm uses the average internal dynamic power among active scenarios enabled for dynamic power.

To minimize the area and power overhead, a self-gating cell can be shared across a few registers by creating a combined enable condition with a tree of comparator cells. If the self-gated registers are driven by synchronous set or synchronous clear signals, these signals are also included in the construction of the enable signal so that the circuit remains functionally unchanged. [Figure 59](#) is an example of a self-gating cell that is shared across two registers (4 bits). Note that one of the self-gated registers is a multibit register and the other register is a single-bit register. The tool can also self-gate a group of multibit registers or a group of single-bit registers.

Figure 59 Self-Gating Tree



The tool does not support the following types of sequential cells for self-gate insertion:

- Level-sensitive sequential cells
- Level-sensitive scan design registers
- Master-slave flip-flops
- Retention registers

- Single-bit and multibit registers that belong to shift registers
- Multibit registers with multiple clock paths

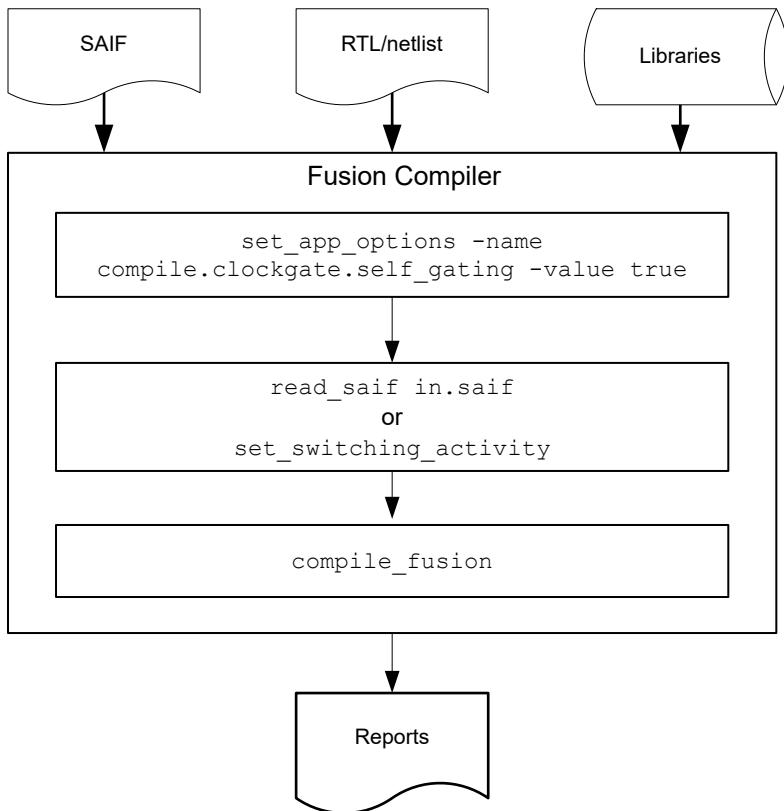
For more information, see the following topics:

- [Self-Gating Flow](#)
- [Library Requirements for Self-Gating](#)
- [Setting Up for Self-Gating](#)
- [Reporting and Querying Self-Gating Cells](#)

Self-Gating Flow

[Figure 60](#) illustrates the Fusion Compiler flow for inserting self-gates by using the `compile_fusion` command.

Figure 60 General Self-Gating Flow



Library Requirements for Self-Gating

To perform self-gating, the logic library should contain AND and OR gates to build the self-gating tree. The library should also contain XOR, NAND, and OR gates for the comparator gate. The integrated clock-gating cells in the library must have the following settings:

- Sequential cell: latch
- Control point: before
- Control signal: scan_enable
- Observation point: none

If the library does not contain cells with these configurations for the corresponding operating conditions, the tool does not insert self-gating cells. If an integrated clock gate compatible with self-gating is specified by the `set_clock_gate_style` command, the self-gating feature uses the same integrated clock gate or the clock gate that is most similar to the one specified.

Setting Up for Self-Gating

To set up the block for self-gating, perform the following steps before you run the `compile_fusion` command:

1. Enable self-gating by setting the `compile.clockgate.self_gating` application option to true.

```
fc_shell> set_app_options -name compile.clockgate.self_gating \
    -value true
```

2. Apply switching activity by using the `read_saif` or `set_switching_activity` command.
3. (Optional) Specify self-gating settings using the `set_self_gating_options` command. For more information, see [Setting Self-Gating Options](#).
4. (Optional) Override the tool's default behavior of selecting self-gating objects by using the `set_self_gating_objects` command. For more information, see [Controlling the Selection of Self-Gating Objects](#).

Setting Self-Gating Options

To specify self-gating settings, use the `set_self_gating_options` command with the following options:

- `-min_bitwidth bitwidth`

Controls the minimum size for self-gating banks. The default is 4.

- `-max_bitwidth bitwidth`
Controls the maximum size for self-gating banks. The default is 8.
- `-interaction_with_clock_gating interaction_type`
Registers gated by user-instantiated clock-gating cells are candidates for self-gating. Valid arguments are `none` (skip registers gated by tool-inserted clock gates), `insert` (the default; insert self-gates on gated registers), and `collapse` (collapse tool-inserted clock gates if they are in the same hierarchy).
- `-objects object_list`
Specifies that the options should be applied to the listed objects. Supported objects are hierarchical instances, power domains, and designs. By default, the settings apply to the entire design.
- `-tree_type combinational_logic_type`
Uses the specified logic type to build the gated enable tree. Valid arguments are `xor`, `nand`, `or`, and `auto` (the default).

To report self-gating options, use the `report_self_gating_options` command.

Controlling the Selection of Self-Gating Objects

To override the default behavior of selecting self-gating objects, use the `set_self_gating_objects` command with the following options:

- `-force object_list`
Directs the tool to perform self-gating on the listed objects, even if the objects do not pass internal checks.
- `-exclude object_list`
Prevents the tool from self-gating the listed objects, even if they pass the internal checks.
- `-include object_list`
Specifies that the listed objects should be self-gated according to the insertion rules set by the `set_clock_gating_options` command. This is the default setting for all registers in the design.
- `-clear object_list`
Removes all inclusion, exclusion, and forced criteria on the listed objects.
- `-reset`

Removes all configurations previously set by the `set_self_gating_objects` command.

- `-tree_type combinational_logic_type`

Specifies the type of combinational cells to use. Valid arguments are `xor` (the default), `or`, `nand`, and `auto`. When `auto` is specified, the tool automatically decides which comparison logic to use based on switching activity information. This might help to reduce the area while optimizing power. You must use this option with the `-include` or `-force` options.

To report self-gating objects, use the `report_self_gating_objects` command.

Reporting and Querying Self-Gating Cells

By default, the `report_clock_gating` command automatically detects the presence of self-gates in the design and displays the corresponding information at the bottom of the report, as shown in the following example report.

Clock Gating Summary		
Number of Clock gating elements	5	
Number of Tool-Inserted Clock gates	5 (100.00%)	
Number of Pre-Existing Clock gates	0 (0.00%)	
Number of Gated registers	48 (100.00%)	
Number of Tool-Inserted Gated registers	48 (100.00%)	
Number of Pre-Existing Gated registers	0 (0.00%)	
Number of Ungated registers	0 (0.00%)	
Total number of registers	48	
Max number of Clock Gate Levels	2	
Number of Multi Level Clock Gates	4	

Self Gating Summary		
Number of self-gating cells	3	
Number of self gated registers	20	(41.67%)
Number of registers not self-gated	28	(58.33%)
Total number of registers	48	

Self-gating cells are also counted as clock gates in the clock gating summary table. In this example, there are two clock gates and three self-gates, for a total of five clock-gating elements.

You can find all the self-gates in a design by using the `get_clock_gates` command. This command returns a collection of clock-gating or self-gating cells. For example, the following command returns the self-gating cells that gate registers clock by CLK:

```
fc_shell> get_clock_gates -clock CLK -type self-gate
```

5

Clock Tree Synthesis

To learn how to perform clock tree synthesis in the Fusion Compiler tool, see the following topics:

- [Prerequisites for Clock Tree Synthesis](#)
 - [Defining the Clock Trees](#)
 - [Verifying the Clock Trees](#)
 - [Setting Clock Tree Design Rule Constraints](#)
 - [Specifying the Clock Tree Synthesis Settings](#)
 - [Implementing Clock Trees and Performing Post-CTS Optimization](#)
 - [Implementing Multisource Clock Trees](#)
 - [Analyzing the Clock Tree Results](#)
-

Prerequisites for Clock Tree Synthesis

This topic details the prerequisites required before running the clock tree synthesis.

Before you run clock tree synthesis on a block, it should meet the following requirements:

- The clock sources are identified with the `create_clock` or `create_generated_clock` commands.
- The block is placed and optimized.

Use the `check_legality -verbose` command to verify that the placement is legal. Running clock tree synthesis on a block that does not have a legal placement might result in long runtimes and reduced QoR.

The estimated QoR for the block should meet your requirements before you start clock tree synthesis. This includes acceptable results for

- Congestion

If congestion issues are not resolved before clock tree synthesis, the addition of clock trees can increase congestion. If the block is congested, you can rerun the

`create_placement` command with the `-congestion` and `-congestion_effort` high options, but the runtime can be long.

- Timing
- Maximum capacitance
- Maximum transition time
- The power and ground nets are prerouted.
- High-fanout nets, such as scan enables, are synthesized with buffers.
- The active scenarios are defined.

By default, the Fusion Compiler tool synthesizes and optimizes all clocks in all active scenarios that are enabled for setup or hold analysis.

Defining the Clock Trees

This section details the information required to analyze and define each clock tree before running clock tree synthesis.

Before you run clock tree synthesis, analyze each clock tree in the block and determine

- What the clock root is
- What the required clock sinks and clock tree exceptions are
- Whether the clock tree contains preexisting cells, such as clock-gating cells
- Whether the clock tree converges, either with itself (a convergent clock path) or with another clock tree (an overlapping clock path)
- Whether the clock tree has timing relationships with other clock trees in the block, such as interclock skew requirements

Use this information to define the clock trees and validate that the tool has the correct clock tree definitions.

The Fusion Compiler tool derives the clock trees based on the clocks defined in the block. If the derived clock trees do not meet your requirements, you can define clock trees as described in the following topics:

- [Deriving the Clock Trees](#)
- [Defining Clock Tree Exceptions](#)
- [Restricting Optimization on the Clock Network](#)
- [Copying Clock Tree Exceptions Across Modes](#)

- [Deriving Clock Tree Exceptions From Ideal Clock Latencies](#)
- [Handling Endpoints With Balancing Conflicts](#)

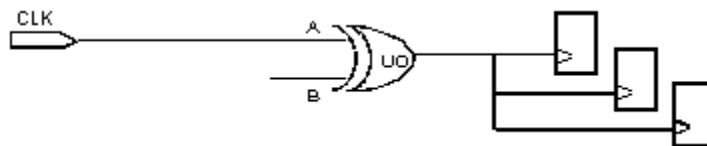
Deriving the Clock Trees

The Fusion Compiler tool derives the clock trees by tracing through the transitive fanout from the clock roots to the clock endpoints. In general, the tracing terminates when it finds a clock pin of a sequential cell or macro; however, the tool traces through sequential cells if they are integrated clock-gating (ICG) cells or their fanout drives a generated clock.

If the clock-gating logic uses a non-unate cell, such as an XOR or XNOR gate, the tool uses both the positive-unate timing arc and the negative-unate timing arc when tracing the clock path. If this does not correspond to the functional mode of the block, use the `set_case_analysis` command to hold all nonclock inputs of the cell at a constant value, which forces the cell into the required functional mode for clock tree synthesis.

For example, suppose a block has the gating logic shown in [Figure 61](#).

Figure 61 Non-Unate Gated Clock



To force the XOR gate (U0) into functional mode for timing analysis, use the following command:

```
fc_shell> set_case_analysis 0 U0/B
```

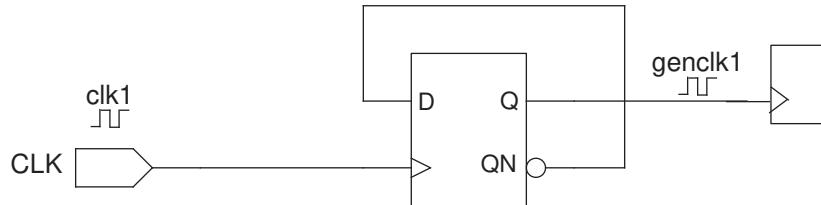
Identifying the Clock Roots

The Fusion Compiler tool uses the clock sources defined by the `create_clock` command, which can be either input ports or internal hierarchical pins, as the clock roots.

For nested clock trees with generated clocks, which are defined by the `create_generated_clock` command, the tool considers the master-clock source to be the clock root, and the clock endpoints of the nested clock tree are considered endpoints of the master-clock source.

For example, for the netlist shown in [Figure 62](#), the tool considers the CLK port to be the clock root for the genclk1 generated clock.

Figure 62 Nested Clock Tree With a Generated Clock



If the block contains generated clocks, ensure that the master-clock sources are correctly defined, as incorrect definitions can result in poor skew and timing QoR. In particular,

- If the tool cannot trace back to the master-clock source, it cannot balance the sink pins of the generated clock with the sink pins of its source.
- If the master-clock source is not a clock source defined by the `create_clock` or `create_generated_clock` command, the tool cannot synthesize a clock tree for the generated clock or its source.

Use the `check_clock_trees` command to verify that your master-clock sources are correctly defined, as described in [Verifying the Clock Trees](#).

Specifying the Clock Root Timing Characteristics

To get realistic clock tree synthesis results, you must ensure that the timing characteristics of the clock roots are correctly modeled.

- If the clock root is an input port without an I/O pad cell, you must accurately specify the driving cell of the input port.

If you specify a weak driving cell, the tool might insert extra buffers to try to meet the clock tree design rule constraints, such as maximum transition time and maximum capacitance.

If you do not specify a driving cell (or drive strength), the tool assumes that the port has infinite drive strength.

For example, if the CLK1 port is the root of the CLK1 clock tree, use the following command to set its driving cell as the CLKBUF cell in the mylib cell library:

```
fc_shell> set_driving_cell -lib_cell mylib/CLKBUF [get_ports CLK1]
```

- If the clock root is an input port with an I/O pad cell, you must accurately specify the input transition time of the input port.

For example, if the CLK1 port is the root of the CLK1 clock tree and the I/O pad cell has already been inserted, use the following commands to set its input transition time to 0.3 for rising delays and 0.2 for falling delays:

```
fc_shell> set_input_transition -rise 0.3 [get_ports CLK1]
fc_shell> set_input_transition -fall 0.2 [get_ports CLK1]
```

Identifying the Clock Endpoints

When deriving the clock trees, the tool identifies two types of clock endpoints:

- Sink pins

Sink pins are the clock endpoints that are used for delay balancing. The tool assigns an insertion delay of zero to all sink pins and uses this delay during delay balancing.

During clock tree synthesis, the tool uses sink pins in calculations and optimizations for both design rule constraints and clock tree timing (skew and insertion delay).

Sink pins are also referred to as balancing pins.

- Ignore pins

Ignore pins are clock endpoints that are excluded from clock tree timing calculations and optimizations. The tool uses ignore pins only in calculations and optimizations for design rule constraints.

During clock tree synthesis, the tool isolates ignore pins from the clock tree by inserting a guide buffer before the pin. Beyond the ignore pin, the tool never performs skew or insertion delay optimization, but does perform design rule fixing.

The tool identifies the following clock endpoints as sink pins:

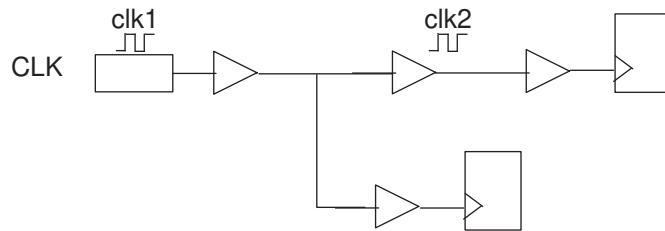
- A clock pin on a sequential cell (a latch or flip-flop), unless that cell drives a generated clock
- A clock pin on a macro cell

The tool identifies the following clock endpoints as ignore pins:

- Source pins of clock trees in the fanout of another clock

For example, in [Figure 63](#), the source pin of the driven clock (clk2) is an ignore pin of the driving clock (clk1). Sinks of the driven clock are not considered sinks of the driving clock.

Figure 63 Cascaded Clock With Two Source Clocks



- Nonclock input pins of sequential cells
- Three-state enable pins
- Output ports
- Incorrectly defined clock pins (for example, the clock pin does not have trigger edge information or does not have a timing arc to the output pin)
- Buffer or inverter input pins that are held constant by using the `set_case_analysis` command

Note:

The tool does not synthesize a clock tree if its source is held constant by using the `set_case_analysis` command.

- Input pins of combinational cells or integrated clock-gating cells that do not have any fanout or that do not have any enabled timing arcs

To verify that the tool has correctly identified the sink pins and ignore pins, examine the clock trees in the GUI, as described in .

If the default sink and ignore pins are correct, you are done with the clock tree definition. Otherwise, first identify any timing settings, such as disabled timing arcs and case analysis settings, that affect the clock tree traversal. To identify disabled timing arcs in the block, use the `report_disable_timing` command. To identify case analysis settings in the block, use the `report_case_analysis` command. Remove any timing settings that cause an incorrect clock tree definition.

If necessary, you can override the default balancing and ignore pin settings by using the `set_clock_balance_points` command, as described in [Defining Clock Tree Exceptions](#).

Defining Clock Tree Exceptions

Clock tree exceptions are user-defined changes to the default endpoints (balancing and ignore pins) derived by the tool for a specific clock.

The Fusion Compiler tool supports the following types of clock tree exceptions:

- User-defined sink pins

These exceptions define sink pins in addition to those derived by the tool for a clock tree. For example, you might define a tool-derived ignore pin as a clock sink.

For information about defining sink pins, see [Defining Sink Pins](#).

- User-defined insertion delay requirements

These exceptions specify special insertion delay requirements for a sink pin (either derived or user-specified).

For information about defining insertion delay requirements, see [Defining Insertion Delay Requirements](#).

- User-defined ignore pins

These exceptions exclude clock endpoints that were derived as sink pins by the tool. For example, you might define an ignore pin to exclude all branches of the clock tree that fan out from some combinational logic or to exclude a tool-derived sink pin.

For information about defining ignore pins, see [Defining Ignore Pins](#).

Defining Sink Pins

To define one or more pins as sink pins, use the following syntax:

```
set_clock_balance_points
  -clock clock
  -consider_for_balancing true
  -balance_points pins
  [-corners corners]
```

The `-clock` option is not required. If you do not use it, the sink pin applies to all clocks.

By default, the insertion delay for the user-specified sink pins is 0. For information about overriding the default insertion delay, see [Defining Insertion Delay Requirements](#).

When you define sink pins, by default, they apply to all corners. To apply the definition to specific corners, use the `-corners` option.

For example, to specify pin U2/A as a sink pin for the CLK clock in the current corner, use the following command:

```
fc_shell> set_clock_balance_points -clock [get_clocks CLK] \
    -consider_for_balancing true -balance_points [get_pins U2/A]
```

To report the user-defined sink pins, use the `report_clock_balance_points` command. This command reports both the user-defined sink pins and the user-defined ignore pins.

To remove the sink pin definition from a pin, use the `remove_clock_balance_points` command.

For example, to remove the sink pin definition from the U2/A pin for the CLK clock, use the following command:

```
fc_shell> remove_clock_balance_points -clock [get_clocks CLK] \
    -balance_points [get_pins U2/A]
```

Defining Insertion Delay Requirements

To override the default phase delay of zero for sink pins (derived or user-specified), use the `-delay` option with the `set_clock_balance_points` command to specify the insertion delay requirements. The tool adds the specified delay (positive or negative) to the calculated insertion delay up to the specified sink pin.

The syntax to define the insertion delay requirement for one or more sink pins is

```
set_clock_balance_points
    [-clock clock]
    -delay delay
    [-rise] [-fall] [-early] [-late]
    -balance_points pins
    [-corners corners]
```

The `-clock` option is not required. If you do not use it, the insertion delay requirement applies to all clocks.

Clock tree synthesis uses the specified delay value for the path delay calculations used to build the clock tree. By default, the specified delay value is used for both longest-path and shortest-path calculations for both rising-edge and falling-edge paths. To control when the specified delay value is used, use the `-rise`, `-fall`, `-early`, and `-late` options.

When you define insertion delay requirements, by default, they apply to all corners. To apply the definition to specific corners, use the `-corners` option.

For example, to specify that clock tree synthesis should use an insertion delay of 2.0 ns for rising-edge shortest-path calculations for the U2/CLK pin for the CLK clock in the current corner, use the following command:

```
fc_shell> set_clock_balance_points -clock [get_clocks CLK] \
    -rise -early -delay 2.0 -balance_points [get_pins U2/CLK]
```

To report the user-defined insertion delay for sink pins, use the `report_clock_balance_points` command. This command reports the user-defined sink pins, the derived sink pins with user-specified insertion delay, and the user-defined ignore pins.

Defining Ignore Pins

To define one or more pins as ignore pins, use the following syntax:

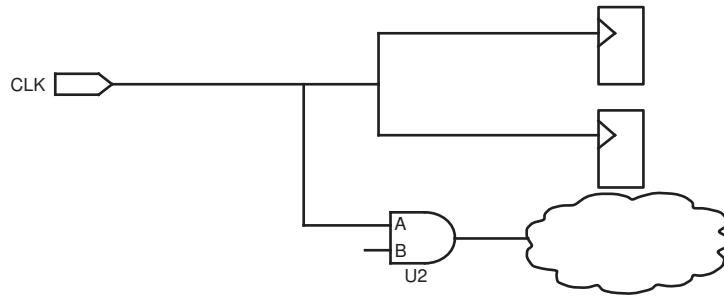
```
set_clock_balance_points
  [-clock clock]
  -consider_for_balancing false
  -balance_points pins
```

The `-clock` option is not required. If you do not use it, all user-defined ignore pins apply to all clocks.

For the CLK clock tree shown in [Figure 64](#), assume that you want to ignore all branches of the clock tree beyond the U2 cell. To do so, you would use the following command:

```
fc_shell> set_clock_balance_points -clock [get_clocks CLK] \
  -consider_for_balancing false -balance_points [get_pins U2/A]
```

Figure 64 User-Defined Ignore Pin



To report the user-defined ignore pins, use the `report_clock_balance_points` command. This command reports both the user-defined sink pins and the user-defined ignore pins.

During clock tree synthesis, the tool adds guide buffers and isolates ignore pins from the rest of the clock network. During subsequent data path optimization, the tool fixes any existing DRC violations beyond the guide buffers.

To remove the ignore pin definition from a pin, use the `remove_clock_balance_points` command.

For example, to remove the ignore pin definition from pin U2/CLK for the CLK clock, use the following command:

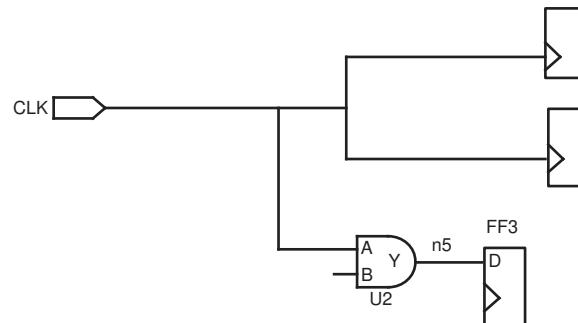
```
fc_shell> remove_clock_balance_points -clock [get_clocks CLK] \
    -balance_points [get_pins U2/CLK]
```

Ensuring Clock Tree Exceptions are Valid

To identify pins on the clock network, the tool checks if the `is_clock_is_used_as_clock` pin attribute is `true` during clock tree synthesis. If you use the `set_clock_balance_points` command to specify a clock tree exception on a pin for which the `is_clock_is_used_as_clock` attribute is `false`, the tool accepts the exception, but ignores it during clock tree synthesis.

Consider the clock network in the following figure.

Figure 65 Specifying Exceptions on Pins Beyond the Clock Network



By default, the tool sets

- The `is_clock_is_used_as_clock` attribute of the U2/A, U2/Y, and FF3/D pins to `false` and considers the network beyond the U2/A pin as part of the data network
- The U2/A pin as an ignore pin and excludes the network beyond this from clock tree synthesis

During clock tree synthesis, the tool adds a guide buffer and isolates the U2/A pin from the rest of the clock network. During subsequent data path optimization, the tool can fix any existing DRC violations beyond the guide buffer.

Assume you want clock tree synthesis to fix DRC violations up to the FF3/D pin using clock tree constraints. To do so, you must

1. Specify the FF3/D pin as an explicit ignore pin as follows:

```
fc_shell> set_clock_balance_points -clock [get_clocks CLK] \
    -consider_for_balancing false -balance_points [get_pins FF3/D]
```

2. Force clock tree synthesis to consider the FF3/D pin as a pin on the clock network by using the `set_sense -clock_leaf` command as follows:

```
fc_shell> set_sense -clock_leaf [get_pins FF3/D]
```

The `set_sense -clock_leaf` command sets

- The `is_clock_used_as_clock` attribute to `true` for all the pins of the clock branches that fanin to the pin specified as the clock leaf
- The `is_clock_used_as_clock` attribute to `false` for all the pins in the fanout from the pin specified as the clock leaf, and the tool does not propagate the clock beyond this pin

To check the value of the `is_clock_used_as_clock` pin attribute, use the `get_attribute` command, as shown in the following example:

```
fc_shell> get_attribute [get_pins FF3/D] is_clock_used_as_clock
```

Restricting Optimization on the Clock Network

You can restrict the optimization on portions of the clock network by specifying

- Don't touch settings

These settings identify parts of the clock tree on which clock tree synthesis and optimization is not performed. You can use don't touch settings to prevent the modification of a clock network beyond a specific pin, the buffering of specific nets, or the sizing of specific cells.

For information about defining don't touch settings, see [Setting Don't Touch Settings](#).

- Size-only settings

These settings identify clock tree cells for which the tool can perform only cell sizing. Note that these cells can be moved, unless they have a placement status of fixed.

For information about defining size-only settings, see .

Setting Don't Touch Settings

To set a don't touch setting on a clock tree, use the `set_dont_touch_network` command.

```
fc_shell> set_dont_touch_network -clock_only [get_pins pin_name]
```

The `-clock_only` option sets the don't touch setting only on the clock network. If you do not specify the `-clock_only` option, the don't touch setting is set on both the data path and the clock path.

To remove the don't touch setting from a clock tree, use the `-clear` option with the `set_dont_touch_network` command.

```
fc_shell> set_dont_touch_network -clock_only [get_pins pin_name] -clear
```

You can also set don't touch settings on specific objects in the clock network by using the `set_dont_touch` command.

The following commands set a don't touch setting on a cell and a net:

```
fc_shell> set_dont_touch [get_cells cell_name] true  
fc_shell> set_dont_touch [get_nets -segments net_name] true
```

To report all the don't touch settings set on the current block, use the `report_dont_touch` command.

```
fc_shell> report_dont_touch -all
```

Note:

If a cell has a placement status of fixed, it is treated like a don't touch cell during clock tree synthesis.

Setting Size-Only Settings

To set a size-only setting on a cell, use the `set_size_only` command.

The following command sets a size-only setting on a cell:

```
fc_shell> set_size_only [get_cells cell_name] true
```

To report all the size-only settings set on the current block, use the `report_size_only` command.

```
fc_shell> report_size_only -all
```

Copying Clock Tree Exceptions Across Modes

You can specify that the tool copies clock tree exceptions from one mode to one or more equivalent modes by using the `set_clock_tree_options` command with the `-copy_exceptions_across_modes`, `-from_mode`, and `-to_mode` options.

The following example specifies that during clock tree synthesis the clock tree exceptions in the mode name M1 should be copied to the modes name M2 and M3:

```
fc_shell> set_clock_tree_options -copy_exceptions_across_modes \
    -from_mode M1 -to_mode {M2 M3}
```

Deriving Clock Tree Exceptions From Ideal Clock Latencies

If you have set ideal clock latencies for specific sinks in your design, you can use the `derive_clock_balance_points` command to convert these ideal latency settings to clock tree exceptions.

The `derive_clock_balance_points` command converts ideal latencies specified with the `set_clock_latency` command on clock sink pins to `set_clock_balance_points` commands. Use the `derive_clock_balance_points` command before clock tree synthesis, when the design has ideal clocks.

By default, the `derive_clock_balance_points` command

- Generates balance point constraints for all ideal primary clocks of all active scenarios that are enabled for setup analysis, hold analysis, or both. It does not generate balance points for generated clocks.

To generate the balance points for

- Specific primary clocks, use the `-clocks` option.
 - All active scenarios of specific corners, use the `-corners` option.
 - Calculates a reference latency for each primary ideal clock, which is the sum of the ideal source and network latency specified on the clock with the `set_clock_latency` command.
- To specify a different reference latency value, use the `-reference_latency` option.
- Applies the derived clock balance point constraints to the design.

To generate an output file containing `set_clock_balance_points` commands, instead applying it to the design, use the `-output` option.

Balance points that the tool derives are clock and corner specific. Ensure that appropriate `set_clock_latency` constraints are set for all clocks for all modes used for clock tree synthesis in at least the worst corner.

The tool uses the following formula to derive the delay values for the corresponding `set_clock_balance_points` commands:

(Clock balance point delay value at the sink) =

(Reference latency of the clock) - (Total ideal clock latency at the sink)

The total ideal clock latency at the sink is the sum of the source latency of the clock and the ideal network latency for the sink.

For example, assume you have the following timing constraint settings:

```
fc_shell> set_clock_latency -source 0.5 [get_clocks clk]
fc_shell> set_clock_latency 1.0 [get_clocks clk]
fc_shell> set_clock_latency 1.5 [get_pins reg1/CK]
fc_shell> set_clock_latency 0.5 [get_pins reg2/CK]
```

If you run the `derive_clock_balance_points` command, the tool derives the following clock balance point constraints:

```
fc_shell> set_clock_balance_points -delay -0.5 \
    -balance_points [get_pins reg1/CK] -clock clk -corners worst
fc_shell> set_clock_balance_points -delay 0.5 \
    -balance_points [get_pins reg2/CK] -clock clk -corners worst
```

To avoid conflicts, do not manually apply any clock balance points for a clock that you derive balance points with the `derive_clock_balance_points` command.

Handling Endpoints With Balancing Conflicts

In some blocks, the clocks have endpoints that are structurally impossible to balance. The Fusion Compiler tool can automatically detect endpoints with balancing conflicts and derive an ignore pin to resolve the conflict. This capability is enabled by default for the `synthesize_clock_trees` and `clock_opt` commands. To disable this capability, set the `cts.common.enable_auto_exceptions` application option to `false`.

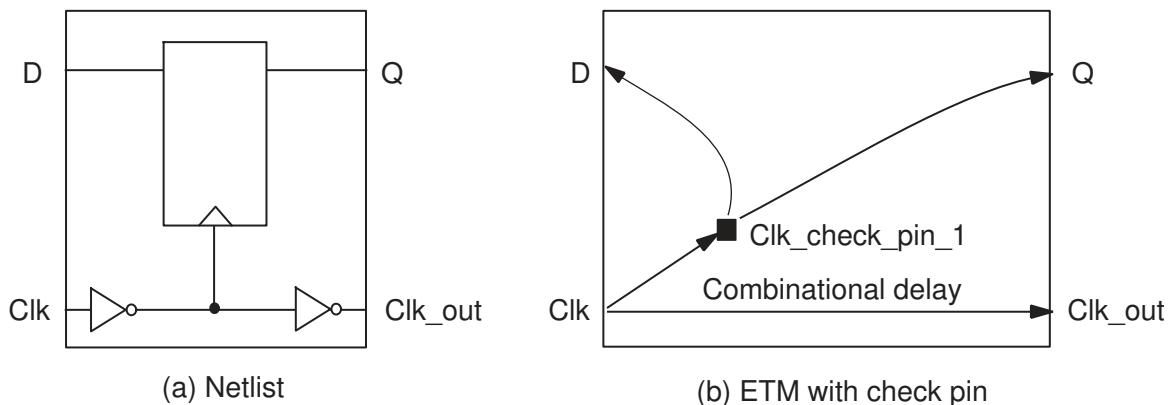
With this capability, the tool detects and fixes the following types of balancing conflicts:

- Internal sink pins of modules that have a combinational clock path

If a module contains both internal sink pins and combinational clock paths, it is impossible to balance the internal sink pins. To resolve the balancing conflict, the tool defines the internal sink pins as ignore pins.

For example, assume that you have a module that contains a combinational path in the clock network, as shown in figure (a) in [Figure 66](#), which is represented by the extracted timing model (ETM) shown in figure (b).

Figure 66 Module With Combinational Clock Path



In this case, the internal check pin inside the ETM has delays coming from the ETM timing library; these delays are considered by clock tree synthesis as sinks and are balanced with other sinks at the top level. If the path to the check pin is the shortest path, top-level clock tree synthesis cannot insert buffers on the shortest path and therefore leaves a large skew at the top level. Setting the internal check pin as an ignore pin resolves the skew issue coming from the ETM.

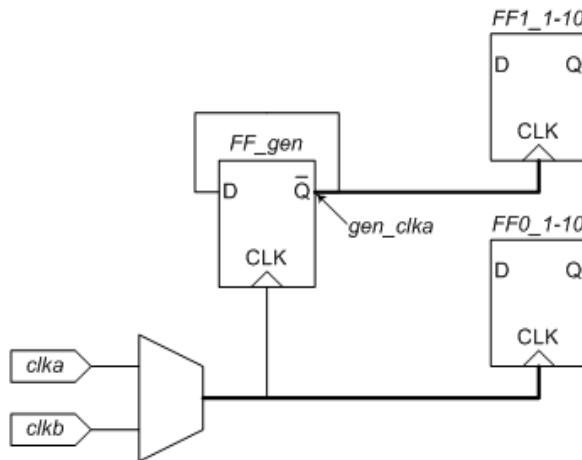
- Sink pins that cannot be balanced simultaneously

If a block contains two clocks that drive common sinks, one of which is a clock pin that is the parent of another sink, due to a generated clock, it is impossible to simultaneously balance the parent and child clock pins with another common sink. To resolve the balancing conflict, the tool defines the parent clock pin as an ignore pin.

For example, in [Figure 67](#), FF_gen/CLK and FF0_1-10/CLK are sinks of clkb, FF1_1-10/CLK and FF0_1-10/CLK are sinks of clka, and FF_gen/CLK is the parent of

FF1_1-10/CLK. In this case, the tool defines FF_gen/CLK as an ignore pin for clk_b to resolve the balancing conflict.

Figure 67 Clock Pins That Cannot Be Balanced Simultaneously



Verifying the Clock Trees

Before you synthesize the clock trees, use the `check_clock_trees` command to verify that the clock trees are properly defined. For example, to verify that the CLK clock tree is properly defined, use the following command:

```
fc_shell> check_clock_trees -clocks [get_clocks CLK]
```

If you do not specify the `-clock` option, the tool checks all clocks in the current block.

The `check_clock_trees` command checks for the following issues:

- Clock (master or generated) with no sinks
- Loops in the clock network
- Multiple clocks reach the same register because of overlapping clocks, but multiple-clocks-per-register propagation is not enabled
- Ignored clock tree exceptions
- Stop pin or float pin defined on an output pin
- Buffers with multiple timing arcs used in clock tree references
- Situations that cause an empty buffer list

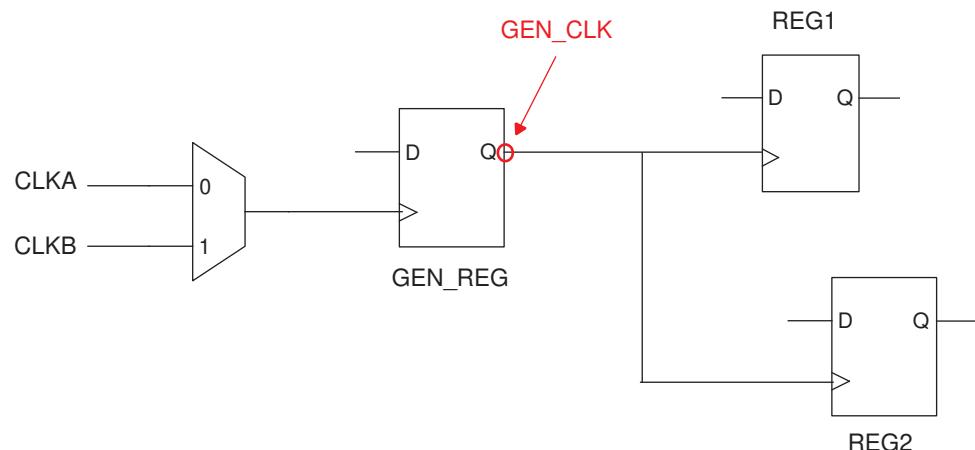
- Generated clock without a valid master clock source

A generated clock does not have a valid master-clock source in the following situations:

- The master clock specified in `create_generated_clock` does not exist
- The master clock specified in `create_generated_clock` does not drive the source pin of the generated clock
- The source pin of the generated clock is driven by multiple clocks, and some of the master clocks are not specified with `create_generated_clock`.

For example, in [Figure 68](#), the GEN_REG/Q pin is driven by both CLKA and CLKB. If only CLKA is specified as a master clock in a `create_generated_clock` command, GEN_CLK does not have a valid master clock source.

Figure 68 Generated Clock With Invalid Master Clock Source



For multicorner-multimode designs, the `check_clock_trees` command checks all active scenarios for the following issues:

- Conflicting per-clock exception settings
- Conflicting balancing settings

Before you implement the clock trees, you should manually fix the reported issues. Each message generated by the `check_clock_trees` command has a detailed man page that describes how to fix the identified issue. You can improve the clock tree and timing QoR by fixing all the issues identified by the `check_clock_trees` command.

Setting Clock Tree Design Rule Constraints

The Fusion Compiler tool supports the following design rule constraints for clock tree synthesis:

- Maximum capacitance

To specify maximum capacitance constraints for clock tree synthesis, use the `-clock_path` option with the `set_max_capacitance` command.

If you do not specify this constraint, the clock tree synthesis default is 0.6 pF.

- Maximum transition time

To specify maximum transition time constraints for clock tree synthesis, use the `-clock_path` option with the `set_max_transition` command.

If you do not specify this constraint, the clock tree synthesis default is 0.5 ns.

By default, these constraints apply to all corners associated with the current mode. To set the constraint for a specific mode, use the `-mode` option with the `get_clocks` command to identify the clocks. To set the constraint for specific corners associated with the specified mode, use the `-corners` option with the constraint command. Be careful to apply the correct constraints across all the modes and their associated corners.

For example, to set a maximum transition time of 0.20 ns on all pins in the CLK clock path for all corners of the current mode, use the following command:

```
fc_shell> set_max_transition 0.20 -clock_path [get_clocks CLK]
```

To set different maximum transition time constraints for different corners associated with a specific mode, use the `-mode` option with the `get_clocks` command to specify the mode and use the `-corners` option to specify the corner.

For example, to set the maximum transition time to 0.15 ns for corner1 in mode1 and 0.10 ns for corner2 in mode1, use the following commands:

```
fc_shell> set_max_transition 0.15 -corners corner1 \
           -clock_path [get_clocks -mode mode1]
fc_shell> set_max_transition 0.10 -corners corner2 \
           -clock_path [get_clocks -mode mode1]
```

To set the same maximum capacitance constraint for different corners associated with a specific mode, use the following command:

```
fc_shell> set_max_capacitance 0.6 \
           -clock_path [get_clocks -mode mode2] \
           -corners [get_corner \
                     [get_attribute [get_modes mode2] associated_corners]]
```

Specifying the Clock Tree Synthesis Settings

The clock tree synthesis options guide the implementation of the clock trees. The following topics describe how to set these options:

- [Specifying the Clock Tree References](#)
 - [Setting Skew and Latency Targets](#)
 - [Enabling Local Skew Optimization](#)
 - [Specifying the Primary Corner for Clock Tree Synthesis](#)
 - [Preventing Specific Clocks From Being Synthesized](#)
 - [Preserving Preexisting Clock Trees](#)
 - [Enabling Clock Tree Power Reduction Techniques](#)
 - [Reducing Electromigration](#)
 - [Handling Inaccurate Constraints During Clock Tree Synthesis](#)
 - [Defining Clock Cell Spacing Rules](#)
 - [Creating Skew Groups](#)
 - [Defining a Name Prefix for Clock Cells](#)
 - [Using the Global Router During Initial Clock Tree Synthesis](#)
 - [Specifying Constraints for Clock Nets](#)
 - [Reducing Signal Integrity Effects on Clock Nets](#)
 - [Specifying Settings for Clock Latency Adjustments](#)
 - [Reporting the Clock Tree Settings](#)
-

Specifying the Clock Tree References

The buffer and inverter cells that can be used to build a clock tree and the reference cells of the preexisting gates of the clock tree are referred to as *clock tree references*.

To specify the clock tree references, use the `set_lib_cell_purpose -include cts` command.

Ensure that the list of cells you specify as clock tree references meets the following criteria:

- The list contains at least one buffer or one inverter
- The list contains the library cells of the preexisting gates

If they are not included in the list, the tool is unable to resize these preexisting gates during clock tree synthesis and optimization. You can automatically derive equivalent cells for all the preexisting clock tree cells that are not buffers or inverters and specify them as clock references by using the `derive_clock_cell_references` command, as described in [Deriving Clock Tree References for Preexisting Gates](#).

- For multivoltage designs with always-on buffering requirements, the list contains always-on cells
- The library cells in the reference list do not have a `dont_touch` attribute

If library cells have the `dont_touch` attribute set on them, they are not used by clock tree synthesis even if you specify them as clock tree references.

To ensure that clock tree synthesis uses only the specified set of cells and that these cells are not used by any optimization step other than clock tree synthesis, run the following commands:

```
fc_shell> set cts_cells list_of_cells
fc_shell> set_lib_cell_purpose -exclude cts [get_lib_cells]
fc_shell> set_lib_cell_purpose -include none [get_lib_cells $cts_cells]
fc_shell> set_lib_cell_purpose -include cts [get_lib_cells $cts_cells]
```

Deriving Clock Tree References for Preexisting Gates

To ensure that the tool resizes preexisting gates in the clock network during clock tree synthesis and optimization, you must specify equivalent cells as clock references. Not specifying a complete list of equivalent cells for preexisting gates can affect clock tree QoR.

You can automatically derive equivalent cells for all the preexisting clock tree cells and specify them as clock references by using the `derive_clock_cell_references` command.

When you run this command, the tool sets the `valid_purpose` attribute to `true` on equivalent library cells of the preexisting clock tree cells that are not buffers or inverters. This command does not derive equivalent library cells for preexisting buffers and inverters,

Therefore, you must manually specify the buffers and inverters to use for clock tree synthesis, as shown in the following example:

```
fc_shell> set_lib_cell_purpose -include cts \
    {tech_lib/clk_buf* tech_lib/clk_inv*}
fc_shell> derive_clock_cell_references
fc_shell> synthesize_clock_trees
```

Instead of automatically specifying the equivalent library cells of the preexisting clock tree cells as clock tree references, you can generate a Tcl script that specifies the equivalent library cells of the preexisting clock tree cells as clock tree references. To do so, use the `-output` option.

```
fc_shell> derive_clock_cell_references -output cts_leq_cells.tcl
```

You can edit the output file, source it, and then run clock tree synthesis, as shown in the following example:

```
fc_shell> set_lib_cell_purpose -include cts \
    {tech_lib/clk_buf* tech_lib/clk_inv*}
fc_shell> source cts_leq_cells.tcl
fc_shell> synthesize_clock_trees
```

Restricting the Target Libraries Used

You can restrict the libraries used during clock tree synthesis for the top level or a lower level of the logical hierarchy of a design by using the `set_target_library_subset_clock` command. To enable the use of the target library subset, you must set the `opt.common.enable_target_library_subset_opt` application option to 1.

The following example specifies the buf1 and buf2 cells from the HVT_lib and LVT_lib libraries as clock tree references. However, it restricts the lower-level block named TSK_BLK to use only the cells from the LVT_lib library for its clock nets. Therefore, only buf1 and buf2 cells from the LVT_lib library are used as clock references for that block.

```
fc_shell> set_lib_cell_purpose -include cts \
    {HVT_lib/buf1 HVT_lib/buf2 LVT_lib/buf1 LVT_lib/buf2}
fc_shell> set_target_library_subset_clock {LVT_lib} \
    -objects [TOP/TSK_BLK]
fc_shell> set_app_options \
    -name opt.common.enable_target_library_subset_opt -value 1
```

Setting Skew and Latency Targets

By default, clock tree synthesis tries to achieve the best skew and latency for all clocks. However, this can lead to area, power, and runtime overhead for low frequency clocks, which have relaxed skew and latency targets.

- To specify a skew target, use the `-target_skew` option with the `set_clock_tree_options` command.
- To specify a latency target, use the `-target_latency` option with the `set_clock_tree_options` command.

By default, when you define skew and latency targets, they apply to all clocks in all corners. To define targets for specific clocks, use the `-clocks` option. To define targets for specific corners, use the `-corners` option.

To report the user-defined skew and latency targets, use the `report_clock_tree_options` command.

To remove user-defined skew or latency targets, use the `remove_clock_tree_options` command. To remove all skew and latency targets, use the `-all` option; otherwise, use the appropriate `-target_skew`, `-target_latency`, `-clocks`, and `-corners` options to remove the specific targets.

Enabling Local Skew Optimization

During clock tree synthesis, by default, the tool tries to minimize the global skew, which is the difference between the longest and shortest clock paths. If there is no timing path between the registers with the longest and shortest clock paths, optimizing global skew does not improve the timing QoR of the design. However, by optimizing the local skew, which is the worst skew between launch and capture registers of timing paths, you can improve the timing QoR of the design. During local skew optimization, the tool works on improving local skew of all violating setup and hold timing paths.

To enable local skew optimization during the clock tree synthesis and clock tree optimization stages of the `synthesize_clock_trees` and `clock_opt` commands, set the `cts.compile.enable_local_skew` and `cts.optimize.enable_local_skew` application options to `true`.

```
fc_shell> set_app_options -list {cts.compile.enable_local_skew true}
fc_shell> set_app_options -list {cts.optimize.enable_local_skew true}
```

When you enable local skew optimization using the previous settings, by default, the tool derives skew targets that help improve the timing QoR, and ignores the target skew you

specify with the `set_clock_tree_options -target_skew` command. To prevent the tool from deriving skew targets, use the following application option setting:

```
fc_shell> set_app_options \
    -name cts.common.enable_auto_skew_target_for_local_skew -value false
```

Specifying the Primary Corner for Clock Tree Synthesis

During initial clock tree synthesis, by default, the tool identifies the corner with the worst clock delays and inserts buffers to balance the clock delays in all modes of this corner. To identify a specific corner as the primary corner for initial clock tree synthesis, use the `cts.compile.primary_corner` application option.

Preventing Specific Clocks From Being Synthesized

By default, the tool synthesizes all clocks defined with the `create_clock` and `create_generated_clock` commands. To prevent the tool from synthesizing a specific clock, specify it by using the `cts.common.skip_cts_clocks` application option.

```
fc_shell> set_app_options \
    -name cts.common.skip_cts_clocks -value CK_B
```

Preserving Preexisting Clock Trees

When you perform clock tree synthesis, by default, the tool removes all preexisting clock buffers and inverters. You can prevent this by setting the `cts.compile.remove_existing_clock_trees` application option to `false`.

To preserve specific preexisting clock buffers or inverters, apply a don't touch or size-only exception on them.

Enabling Clock Tree Power Reduction Techniques

You can enable clock tree power reduction techniques by using the `cts.compile.power_opt_mode` application option as shown in the following table.

Table 24 Settings for the `cts.compile.power_opt_mode` Application Option

To do this	Use this setting
Relocate clock-gating cells closer to their drivers to reduce the length of <code>gate_relocation</code> input nets of the clock-gating cells, which have high switching activity	
Use clustering techniques to improve power	<code>low_power_targets</code>

Table 24 *Settings for the cts.compile.power_opt_mode Application Option (Continued)*

To do this	Use this setting
Use both clock-gate relocation and clustering techniques to improve power	all

Reducing Electromigration

Clock cells consume more power than cells that are not in the clock network. Clock cells that are clustered together in a small area increase the current densities for the power and ground rails, which increases the potential for electromigration problems. One way to avoid the problem is to set spacing requirements between clock cells to prevent local clumping of the clock cells along a standard cell power rail between the perpendicular straps.

To use this method to reduce electromigration in the block,

1. Define clock cell spacing rules for the inverters, buffers, and integrated clock-gating cells in the clock network by using the `set_clock_cell_spacing` command, as described in [Defining Clock Cell Spacing Rules](#).
2. Perform clock tree synthesis by using the `synthesize_clock_trees` command, as described in [Performing Standalone Clock Trees Synthesis](#), or the `clock_opt` command, as described in .

Note:

The clock cell spacing rules are also honored by the `balance_clock_groups` command. For information about using this command, see [Balancing Skew Between Different Clock Trees](#).

3. Check clock cell spacing rule violations by using the `check_legality -verbose` command.

You should not see any violations if you set the appropriate clock cell spacing constraints.

Handling Inaccurate Constraints During Clock Tree Synthesis

In the early stages of the implementation of a design, the constraints can be inaccurate or too restrictive. Running clock tree synthesis on such a design can lead to long runtime or poor clock tree QoR.

You can run clock tree synthesis on such a design, but still get usable results in a reasonable runtime. To do so, set the following application variable before you begin clock tree synthesis:

```
fc_shell> set_app_options \
    -name cts.common.enable_dirty_design_mode -value true
```

When you use this application option setting, the tool

- Ignores `dont_touch` and `dont_touch_network` attribute settings on clock nets.
- Removes maximum transition and maximum capacitance constraints if they are too tight.
- Removes the maximum net length constraint if it is less than 50 microns.
- Ignores clock cell spacing rules if the horizontal spacing requirement is more than three times the site height and the vertical spacing requirement is more than twenty times the site height.
- Ignores nondefault routing rules if the width plus the spacing requirement of the routing rule is more than ten times the default width plus the default spacing.
- Reports balance point delay values that are large enough to cause an increase in clock insertion delay.

Defining Clock Cell Spacing Rules

To define clock cell spacing rules, use the `set_clock_cell_spacing` command. At a minimum, you must specify the minimum spacing in the x-direction or the y-direction; typically you would specify the minimum spacing in both directions.

To specify the minimum spacing in microns in the

- X-direction, set the `-x_spacing` option to a nonzero value

This value is usually dependent on the library cell's drive strength and clock frequency.

- Y-direction, set the `-y_spacing` option to a nonzero value

This value is usually less than half a row height.

Note:

Using a large spacing value increases the skew because the cells are spaced further away from the intended clusters of sinks.

By default, the specified cell spacing requirements apply to all clock cells. To restrict the clock cell spacing rules to specific

- Library cells, use the `-lib_cells` option
- Clocks, use the `-clocks` option

If adjoining clock cells both have cell spacing rules for a given direction, the sum of the spacing values applies. For example, if two adjoining clock cells both have a minimum cell spacing of 5 microns in the x-direction, they are placed at least 10 microns apart in the x-direction. You can relax the adjoining cell spacing rule by setting the `cts.placement.cell_spacing_rule_style` application option to `maximum`, in which case the tool uses the larger of the spacing settings between two adjoining cells instead of the sum of the spacing settings. For example, the clock cells in the previous example are placed at least 5 microns apart in the x-direction, instead of 10 microns.

The clock cell spacing rules defined by the `set_clock_cell_spacing` command are honored by the `synthesize_clock_trees`, `balance_clock_groups`, and `clock_opt` commands.

To report clock cell spacing rules, use the `report_clock_cell_spacings` command.

To remove clock cell spacing rules, use the `remove_clock_cell_spacings` command. By default, this command removes all clock cell spacing rules. To remove specific clock cell spacing rules, use the `-lib_cells` option.

Creating Skew Groups

During clock tree synthesis, you might want to balance a group of sinks only among each other, and not the rest of the sinks. To do so, define a skew group by using the `create_clock_skew_group` command.

When you use this command, specify the sinks you want to group, by using the `-objects` option.

Optionally, you can specify the following:

- A clock or generated clock for which to create the skew group by using the `-clock` option.
- A mode for which to create the skew group by using the `-mode` option.
By default, the tool creates the skew group for the current mode.
- A name for the skew group by using the `-name` option.

The following example creates a skew group named sg1 consisting of sinks reg1/CP, reg2/CP, and reg3/CP:

```
fc_shell> create_clock_skew_group -name sg1 \
    -objects {reg1/CP reg2/CP reg3/CP}
```

To report skew groups, use the `report_clock_skew_groups` command. To remove skew groups, use the `remove_clock_skew_groups` command.

Defining a Name Prefix for Clock Cells

You can specify a name prefix for the cells added on the clock network during clock tree synthesis by using the `cts.common.user_instance_name_prefix` application option.

The following example specifies CTS_ as the name prefix for the cells added on the clock network during clock tree synthesis:

```
fc_shell> set_app_options \
    -name cts.common.user_instance_name_prefix -value "CTS_"
```

To specify a name prefix for the cells added on the data nets during optimization, including during the `clock_opt` command, use the `opt.common.user_instance_name_prefix` application option, as described in .

Using the Global Router During Initial Clock Tree Synthesis

By default, initial clock tree synthesis uses the virtual router to build the clock trees. For designs with complex floorplans, this can introduce congestion hotspots and degrade clock QoR after clock routing.

The tool can use the global router during the initial clock tree synthesis stage of the `synthesize_clock_trees` and `clock_opt` commands. To enable this feature, set the `cts.compile.enable_global_route` application option to `true`.

Specifying Constraints for Clock Nets

By default, the Fusion Compiler tool uses the default routing rule and any available routing layers to route the clock trees.

You can specify the minimum and maximum routing layers for clock nets as described in [Specifying the Routing Resources](#). These routing layer constraints are used during RC estimation, congestion analysis, and routing.

To reduce the wire delays in the clock trees, you can use wide wires and higher metal layers instead. Wide wires are represented by nondefault routing rules. To use nondefault

routing rules on the clock nets, you must first define the routing rules and assign them to the clock nets as described in [Using Nondefault Routing Rules](#).

Reducing Signal Integrity Effects on Clock Nets

The timing of a design can be improved by reducing signal integrity issues on clock nets. To do so, the tool can derive nondefault routing rules with larger spacing requirements and assign them to the clock nets in regions without routing congestion.

To enable this feature, use the following application option setting:

```
fc_shell> set_app_options \
    -name cts.optimize.enable_congestion_aware_ndr_promotion -value true
```

When the tool derives a new nondefault routing rule for a clock net, the new spacing requirement is dependent on the spacing requirement of the existing routing rule of that net. If the existing routing rule is

- A nondefault routing rule with a spacing requirement of less than or equal to two times the default spacing, the spacing requirement is doubled

The name of this new nondefault rule is the original name with an _ext_spacing postfix.
- A nondefault routing rule with a spacing requirement of more than two times the default spacing, the spacing requirement is increased by one default spacing

The name of this new nondefault rule is the original name with an _ext_spacing postfix.
- The default routing rule, the spacing requirement is doubled

The name of this new nondefault rule is default_rule_equivalent_ndr_double_spacing.

Specifying Settings for Clock Latency Adjustments

I/O ports are usually constrained using real or virtual clocks. After clock tree synthesis, the latencies of the clocks constraining the I/O ports need to be updated to ensure that the boundary constraints are accurate. When you run clock tree synthesis with the `clock_opt` command, the tool automatically updates the latencies of the clocks constraining the I/O ports. You can control these latency adjustments by using the `set_latency_adjustment_options` command.

The following example specifies that the clock latencies of the VCLK1 and VCLK2 clocks should be updated based on the latency of the PC_CLK clock, and that the latency of the VCLK3 clock should not be updated:

```
fc_shell> set_latency_adjustment_options \
    -reference_clock PC_CLK -clocks_to_update {VCLK1 VCLK2} \
    -exclude_clocks VCLK3
```

If you perform clock tree synthesis with the `synthesize_clock_trees` command, the tool does not automatically update the clock latencies. To do so, use the `compute_clock_latency` command after clock tree synthesis. This command honors the settings you specify with the `set_latency_adjustment_options` command.

Reporting the Clock Tree Settings

To report the settings that are used by clock tree synthesis, use the `report_clock_settings` command. By default, the command reports the clock tree configuration, including the design rule constraints, target skew, and target latency; the clock tree references; the clock cell spacing rules; and the nondefault routing rules for all clocks in the current block.

To report the settings for specific clocks, use the `-clock` option to specify the clocks of interest. To report specific information, use the `-type` option with the appropriate keyword.

- To report the clock tree design rule constraints, target skew, and target latency, use `-type configurations`.
- To report the clock tree references, use `-type references`.
- To report the clock cell spacing rules, use `-type spacing_rules`.
- To report the nondefault routing rule used for the clock nets, use `-type routing_rules`.

Implementing Clock Trees and Performing Post-CTS Optimization

Before you perform clock tree synthesis, you should save the block. This allows you to refine the clock tree synthesis goals and rerun clock tree synthesis with the same starting point, if necessary.

The following topics describe the different methods available for implementing clock trees:

- [Performing Standalone Clock Trees Synthesis](#)
- [Synthesizing, Optimizing, and Routing Clock Trees With the `clock_opt` Command](#)
- [Controlling Concurrent Clock and Data Optimization](#)
- [Splitting Clock Cells](#)
- [Balancing Skew Between Different Clock Trees](#)
- [Performing Global-Route-Based Optimization Using Machine Learning Data](#)
- [Routing Clock Trees](#)

- Inserting Via Ladders During Clock Tree Synthesis, Optimization, and Clock Routing
 - Marking Clocks as Propagated After Clock Tree Synthesis
 - Performing Postroute Clock Tree Optimization
 - Performing Voltage Optimization
 - Marking Clock Trees as Synthesized
 - Removing Clock Trees
-

Performing Standalone Clock Trees Synthesis

To perform standalone clock tree synthesis, use the `synthesize_clock_trees` command, as shown in the following example:

```
fc_shell> synthesize_clock_trees
```

When you use this command, the tool performs the following tasks:

1. Clock tree synthesis

Before clock tree synthesis, the tool performs virtual routing of the clock nets and uses RC estimation to determine the clock net timing.

2. Clock tree optimization

Before clock tree optimization, the tool performs global routing on the clock nets and uses RC extraction to determine the clock net timing. During clock tree optimization, the tool performs incremental global routing on clock nets modified during optimization.

Note:

This command does not detail route the clock trees.

By default, this command works on all clocks in all active scenarios that are enabled for setup or hold analysis. You can specify the clocks to be built by using the `-clock` option. For example, to build only the CLK clock tree, use the following command:

```
fc_shell> synthesize_clock_trees -clocks [get_clocks CLK]
```

After clock tree synthesis, the tool sets the clocks in all modes of all active scenarios as propagated.

Synthesizing, Optimizing, and Routing Clock Trees With the `clock_opt` Command

To synthesize the clock trees, route the clock nets, and further optimize the design by using a single command, use the `clock_opt` command.

The `clock_opt` command consists of the following stages:

1. The `build_clock` stage, during which the tool synthesizes and optimizes the clock trees for all clocks in all modes of all active scenarios. After clock tree synthesis, the tool sets the synthesized clocks as propagated.
2. The `route_clock` stage, during which the tool details routes the synthesized clock nets.
3. The `final_opto` stage, during which the tool performs further optimization, timing-driven placement, and legalization. It then global routes the block and performs extensive global-route-based optimization, which includes incremental legalization and route patching.

You can limit the `clock_opt` command to one or more contiguous stages by using the `-from` option to specify the stage from which you want to begin and the `-to` option to specify the stage after which you want to end. If you do not specify the `-from` option, the tool begins from the `build_clock` stage. Similarly, if you do not specify the `-to` option, the tool continues until the `final_opto` stage is completed.

For example, to synthesize and optimize the clock trees and detail route the clock nets only, limit the execution to the `build_clock` and `route_clock` stages by use the following command:

```
fc_shell> clock_opt -to route_clock
```

Because the tool performs global routing during the `final_opto` stage, you should specify all router related settings, such as routing rules, route guides, application option settings required for the technology node, and so on, before you run the `clock_opt` command.

You should run the `final_opto` only one time for each block. After this stage is completed, all the signal routes in the block are global routed. Therefore, when subsequently run any routing command, such as the `route_auto`, `route_global`, or `route_group` command, the tool skips global routing. To avoid errors during subsequent track assignment and detail routing, do not change any global route shapes after you run the `clock_opt` command.

Considering Voltage Drop Information During Clock Tree Synthesis

By default, the `clock_opt` command does not consider voltage drop information when inserting clock buffers. Therefore, the tool might place clock buffers in locations with

high PG resistance, which increases the voltage drop. To prevent this, you can use the RedHawk Fusion feature to analyze the PG network and use this information with the `clock_opt` command, as shown in the following flow:

1. Use the RedHawk Fusion feature to analyze the PG network by using the `analyze_rail -min_path_resistance` command.
2. Load the RedHawk Fusion results by using the `open_rail_result` command.
3. Enable voltage-drop-aware clock tree synthesis by setting the `clock_opt.flow.enable_voltage_drop_aware` application option to `true`.
4. Run the `clock_opt` command.

Using Nondefault Routing Rules for Critical Nets During Optimization

To improve timing QoR, the tool can use nondefault routing rules on timing critical nets during preroute optimization. In addition, the tool can guide the router to honor these nondefault rule assignments as soft constraints.

To enable this capability for the `clock_opt` command, set the `clock_opt.flow.optimize_ndr` application option to `true`.

```
fc_shell> set_app_options -name clock_opt.flow.optimize_ndr \
    -value true
```

Performing Concurrent Clock and Data Optimization During the `clock_opt` Command

Applying useful skew techniques during datapath optimization to improve the timing QoR by taking advantage of the positive slack and adjusting the clock arrival times of registers is referred to as concurrent clock and data (CCD) optimization.

By default, the tool performs concurrent clock and data optimization during the `clock_opt` command.

To disable concurrent clock and data optimization for the `clock_opt` command, set the `clock_opt.flow.enable_ccd` application option to `false`.

You can change the default behavior of concurrent clock and data optimization during the `clock_opt` command by performing any of the following optional steps:

1. (Optional) Limit the latency adjustment values for concurrent clock and data optimization as described in [Latency Adjustment Values for Concurrent Clock and Data Optimization](#).
2. (Optional) Control the latency adjustment of boundary registers for concurrent clock and data optimization as described in [Excluding Boundary Paths](#).

3. (Optional) Ignore specific path groups during concurrent clock and data optimization as described in [Excluding Specific Path Groups](#).
4. (Optional) Ignore specific scenarios during concurrent clock and data optimization as described in [Excluding Specific Scenarios](#).
5. (Optional) Ignore specific sinks during concurrent clock and data optimization as described in [Excluding Specific Sinks](#).
6. (Optional) Control the effort of timing optimization performed during concurrent clock and data optimization as described in [Controlling Timing Optimization Effort](#).
7. (Optional) Control the effort of hold timing optimization performed during concurrent clock and data optimization as described in [Controlling Hold Time Optimization Effort](#).
8. (Optional) Control the adjustment of I/O clock latencies performed during concurrent clock and data optimization as described in .
9. (Optional) Analyze the effects of concurrent clock and data optimization as described in .

Controlling Multibit Optimization Performed During the `clock_opt` Command

The following topics describe how you can control the multibit optimization performed during the `clock_opt` command:

- [Enabling the Rewiring of Mixed-Drive-Strength Multibit Cells](#)
- [Enabling Post-Clock-Tree-Synthesis Multibit Debanking](#)

Enabling the Rewiring of Mixed-Drive-Strength Multibit Cells

Some logic libraries have mixed-drive-strength multibit cells where some bits have a higher drive strength than others. For designs that use such cells, if a violating path goes through a lower-drive-strength bit, the tool can rewire the mixed-drive-strength multibit cell such that the violating path goes through a higher-drive-strength bit. To enable this feature for the `clock_opt` command, set the `clock_opt.flow.enable_multibit_rewiring` application option to `true`.

Enabling Post-Clock-Tree-Synthesis Multibit Debanking

To improve the timing QoR, the `clock_opt` command can debank multibit registers during the `final_opt` stage, if doing so does not introduce hold timing violations. To enable this feature, set the `clock_opt.flow.enable_multibit_debanking` application option to `true` before you run the `final_opt` stage of the `clock_opt` command.

Performing Power or Area Recovery on the Clock Network

If you enable concurrent clock and data optimization for the `clock_opt` command, the tool performs clock power recovery on clock cells and registers during the `final_opt` stage. If you do not enable concurrent clock and data optimization, you can enable clock power recovery by using the following application option setting:

```
fc_shell> set_app_options \
    -name clock_opt.flow.enable_clock_power_recovery \
    -value power
```

Before you perform power recovery using this technique, you must

1. Enable scenarios for dynamic, leakage, or total power optimization by using the `-dynamic_power` and `-leakage_power` options of the `set_scenario_status` command
2. (Optional) Provide a switching activity by using the `read_saif` command

If you do not provide switching activity, the tool uses default switching activity for dynamic power recovery.

Instead of power recovery, you can enable area recovery on the clock cells and registers during the `final_opt` stage `clock_opt` command by using the following application option setting:

```
fc_shell> set_app_options \
    -name clock_opt.flow.enable_clock_power_recovery \
    -value area
```

Performing IR-Drop-Aware Placement During the `clock_opt` Command

During placement, the tool can use the voltage (IR) drop values of cells to identify areas of high power density and spread the cells with high voltage drop values, which reduces the power density of such areas.

To perform IR-drop-aware placement during the `clock_opt` command, use the following steps:

1. Run the `clock_opt` command through the route-clock stage by using the `clock_opt -to route_clock` command.
2. Set up for RedHawk Fusion and perform static or dynamic voltage drop analysis by using the `analyze_rail -voltage_drop` command as shown in the following example:

```
fc_shell> source redhawk_setup.tcl
fc_shell> analyze_rail -voltage_drop static -nets {VDD VSS}
```

For more information, see [Performing Voltage Drop Analysis](#).

3. Enable IR-drop-aware placement by setting the `place.coarse.ir_drop_aware` application option to `true`.
4. (Optional) Specify additional settings for IR-drop-aware placement, as described in [Controlling IR-Drop-Aware Placement](#).
5. Run the final optimization stage of the `clock_opt` command by using the `clock_opt -from final_opt` command.

Controlling Concurrent Clock and Data Optimization

Applying useful skew techniques during datapath optimization to improve the timing QoR by taking advantage of the positive slack and adjusting the clock arrival times of registers is referred to as concurrent clock and data (CCD) optimization.

Concurrent clock and data optimization is enabled by default for the `compile_fusion` and `clock_opt` commands. To enable it for the `route_opt` command, set the `route_opt.flow.enable_ccd` application option to `true`.

You can change the default behavior of concurrent clock and data optimization by

- [Limiting the Latency Adjustment Values](#)
- [Excluding Boundary Paths](#)
- [Excluding Specific Path Groups](#)
- [Excluding Specific Scenarios](#)
- [Excluding Specific Sinks](#)
- [Controlling Timing Optimization Effort](#)
- [Controlling Hold Time Optimization Effort](#)
- [Controlling the Adjustment of I/O Clock Latencies](#)
- [Performing Dynamic-Voltage-Drop-Driven Concurrent Clock and Data Optimization During the route_opt Command](#)
- [Specifying Optimization Targets at the Preroute Stage](#)
- [Specifying Optimization Targets at the Postroute Stage](#)
- [Enabling Buffer Removal at the Postroute Stage](#)
- [Reporting Concurrent Clock and Data Timing](#)

- Enabling Automatic Zero Balance Point on Macros
- Skewing Latch and Discrete Clock Gates

Limiting the Latency Adjustment Values

You can limit the latency adjustment values for concurrent clock and data optimization performed during the `place_opt`, `clock_opt`, and `route_opt` commands as follows:

- Limit the amount that the clock latencies are advanced by using the `ccd.max_prepone` application option.
- Limit the amount that the clock latencies are delayed by using the `ccd.max_postpone` application option.

There is no default for these application options. Specify these values in the library timing units. The following example sets a limit of 0.2 for advancing and 0.1 for delaying clock latencies:

```
fc_shell> set_app_options -list {ccd.max_prepone 0.2}
fc_shell> set_app_options -list {ccd.max_postpone 0.1}
```

Excluding Boundary Paths

By default, the tool performs concurrent clock and data optimization on all paths in a block. However, you might want to exclude paths that are connected to boundary registers, which are registers in the transitive fanout of input ports and transitive fanin of output ports. To exclude boundary paths from concurrent clock and data optimization, set the `ccd.optimize_boundary_timing` application option to `false`.

When you do so, you can selectively ignore the boundary paths of some ports during boundary path exclusion by using the `ccd.ignore_ports_for_boundary_identification` application option. The following example disables concurrent clock and data optimization for all boundary paths except for those connected to the ports named `IN_A` and `OUT_A`:

```
fc_shell> set_app_options -name ccd.optimize_boundary_timing \
    -value false
fc_shell> set_app_options \
    -name ccd.ignore_ports_for_boundary_identification \
    -value {IN_A OUT_A}
```

Even when you disable concurrent clock and data optimization on boundary registers, the tool can still optimize the clock tree fanin cone of these boundary registers, when doing so improves the timing QoR of other internal registers that share the same clock paths. To prevent the clock tree fanin of boundary registers from changing, set the `ccd.optimize_boundary_timing_upstream` application option to `false`. However, doing so heavily restricts the scope of concurrent clock and data optimization.

These application option settings affect concurrent clock and data optimization performed during the `place_opt`, `clock_opt`, and `route_opt` commands.

Excluding Specific Path Groups

To exclude a specific path group from concurrent clock and data optimization, use the `ccd.skip_path_groups` application option and specify the name of the path group you want ignored.

You can ignore a path group for all scenarios or a specific scenario. For example, to ignore the path group named CLK1 for all scenarios and CLK2 for the scenario named scnA, use the following command:

```
fc_shell> set_app_options -name ccd.skip_path_groups \
    -value {CLK1 {CLK2 scnA}}
```

This setting affects concurrent clock and data optimization performed during the `place_opt`, `clock_opt`, and `route_opt` commands.

Excluding Specific Scenarios

To exclude a specific scenarios from concurrent clock and data optimization, use the `ccd.ignore_scenarios` application option and specify the name of the scenarios you want to ignore.

For example, to ignore the scenario named scnB, use the following command:

```
fc_shell> set_app_options -name ccd.ignore_scenarios \
    -value {scnB}}
```

This setting affects concurrent clock and data optimization performed during the `place_opt`, `clock_opt`, and `route_opt` commands.

Excluding Specific Sinks

To exclude a specific sink from concurrent clock and data optimization,

1. Apply a `cts_fixed_balance_pin` attribute on the sink pin by using the `set_attribute` command as shown in the following example:

```
fc_shell> set_attribute -objects reg21/CK \
    -name cts_fixed_balance_pin -value true
```

2. Set the `ccd.respect_cts_fixed_balance_pins` application option to `true`.

These setting prevents the tool from adjusting the latencies of the sinks that have a `cts_fixed_balance_pin` attribute set to `true`. However, changes in the clock tree due to concurrent clock and data optimization performed on other sinks can change the latencies of the sinks that have a `cts_fixed_balance_pin` attribute set to `true`. To prevent the tool from making any changes to the clock paths between the sinks

that have a `cts_fixed_balance_pin` attribute set to `true` and the clock root, set the `ccd.respect_cts_fixed_balance_pins` application option to `upstream` instead of `true`.

This application option setting affects concurrent clock and data optimization performed during the `place_opt`, `clock_opt`, and `route_opt` commands.

Controlling Timing Optimization Effort

By default, the tool focuses on improving the timing and power QoR during concurrent clock and data optimization. You can change the effort level for timing optimization by setting the `ccd.timing_effort` application option to `low` or `high`. The default is `medium`.

This setting affects concurrent clock and data optimization performed during the `final_opt` stage of the `clock_opt` command and the `route_opt` command.

Controlling Hold Time Optimization Effort

By default, the tool focuses on fixing setup violations during concurrent clock and data optimization. However, if your design has hold violations that are difficult to fix, you can increase the effort for hold fixing by setting the `ccd.hold_control_effort` application option to `medium`, `high`, or `ultra`. The default is `low`.

Increasing the priority of hold violations reduces the number of setup violations that are fixed. Therefore, the hold priority should be increased only if the hold timing is critical. This setting affects concurrent clock and data optimization performed during the `final_opt` stage of the `clock_opt` command and the `route_opt` command.

Controlling the Adjustment of I/O Clock Latencies

By default, the tool adjusts the latencies of the I/O clocks, which are the clocks that constrain the boundary ports, when it performs concurrent clock and data optimization during the `clock_opt` command. The tool also uses concurrent clock and data optimization techniques to adjust the I/O clock latencies when you subsequently run the `compute_clock_latency` command.

The tool does not update the I/O clock latencies

- For the entire block when you disable this feature by setting the `ccd.adjust_io_clock_latency` to `false` or when you disable concurrent clock and data optimization for boundary paths by setting the `ccd.optimize_boundary_timing` application option to `false`
- For specific I/O paths when you exclude the corresponding path groups from concurrent clock and data optimization by using the `ccd.skip_path_groups` application option
- For specific I/O clocks when you disable latency adjustment those I/O clocks by using the `set_latency_adjustment_options -exclude_clocks` command

Performing Dynamic-Voltage-Drop-Driven Concurrent Clock and Data Optimization During the `route_opt` Command

The concurrent clock and data optimization performed during the `route_opt` command can reduce the peak dynamic voltage drop without affecting the timing QoR. To enable this feature, set the `route_opt.flow.enable_voltage_drop_opt_ccd` application option to `true`.

When you enable this feature, during the `route_opt` command, the tool uses the RedHawk Fusion feature to identify dynamic-voltage-drop hotspots and cells rows where the cells from the hotspots can be relocated. Then, during optimization, the tool relocates or downsizes the cells in the hotspots to reduce the peak dynamic voltage drop without hurting the timing QoR.

You can specify one of the following thresholds for selecting cells for optimization:

- A threshold for the cell voltage drop, as a percentage of the supply voltage, by using the `ccd.voltage_drop_voltage_threshold` application option.
Any cell with a voltage drop that exceeds this threshold is selected for optimization.
- A threshold for the number of cells, as a percentage of the total number of cells, by using the `ccd.voltage_drop_population_threshold` application option.
Cells are selected starting with the worst violator.

During postroute optimization, the tool can perform RedHawk dynamic voltage drop analysis to identify cells involved in voltage drop violations, and then use optimization techniques on those cells to improve the dynamic voltage drop.

You can further improve the dynamic voltage drop by enabling IR-driven sizing, which uses the RedHawk dynamic voltage drop analysis results to identify cells involved in voltage drop violations, and then tries to replace those cells with cells having smaller leakage current. To enable this feature, set the `route_opt.flow.enable_irdrivenopt` application option to `true`, in addition to setting the `route_opt.flow.enable_voltage_drop_opt_ccd` application option to `true`.

Specifying Optimization Targets at the Preroute Stage

When performing concurrent clock and data optimization using the `place_opt` or `clock_opt` command, you can give a higher priority to the WNS optimization of

- Certain path groups by specifying them by using the `ccd.targeted_ccd_path_groups` application option, as shown in the following example:

```
fc_shell> set_app_options -name ccd.targeted_ccd_path_groups \
    -value {PG1 PG2}
```

If a path group you specify with this application option is also specified as a path group to skip with the `ccd.skip_path_groups` application option, the path group is skipped during concurrent clock and data optimization.

- Certain endpoints by specifying a file containing the endpoints by using the `ccd.targeted_ccd_end_points_file` application option, as shown in the following example:

```
fc_shell> set_app_options -name ccd.targeted_ccd_end_points_file \
    -value endpoint_targets.tcl
```

If you specify both path groups and endpoints, the tool optimizes only the specified endpoints that are in the specified paths groups.

- The worst 300 timing paths by setting the `ccd.enable_top_wns_optimization` application option to `true`

Specifying Optimization Targets at the Postroute Stage

At the final stages of your design flow, you can use the `route_opt` command to perform concurrent clock and data optimization on specific critical path groups or endpoints by using the following steps:

1. Enable targeted concurrent clock and data optimization for the `route_opt` command by setting the `route_opt.flow.enable_targeted_ccd_wns_optimization` application option to `true`.
2. Specify the optimization target by using one or both of the following settings:
 - Specify the path groups to optimize by using the `ccd.targeted_ccd_path_groups` application option

Note:

If a path group you specify with this application option is also specified as a path group to skip by using the `ccd.skip_path_groups` application option, the path group is skipped during targeted concurrent clock and data optimization.

- Specify the endpoints to optimize by using the `ccd.targeted_ccd_end_points_file` application option

You must specify at least one of these settings. If you specify both, the tool optimizes the specified endpoints of the specified paths groups.

3. (Optional) Specify the type of optimization to perform by using the `ccd.targeted_ccd_select_optimization_moves` application option.

The valid values are

- `auto`, the default, which enables all optimization types, including buffering
- `size_only`, which enables sizing only
- `equal_or_smaller_sizing`, which enables sizing only to cells that are the same size or smaller
- `footprint_sizing`, which enables sizing only to cells with the same footprint

4. (Optional) Specify an optimization effort by using the `ccd.targeted_ccd_wns_optimization_effort` application option.

The valid values are `high` (default), `medium`, and `low`. The higher the effort, the more timing QoR improvement you will see in the targeted paths, but at the expense of the timing QoR of other paths.

5. (Optional) Specify a threshold beyond which the slack for the path groups that are not targeted can be degraded by using the `ccd.targeted_ccd_threshold_for_nontargeted_path_groups` application option.

By default, if the path groups that are not targeted have positive slack, the tool can degrade this slack until it reaches zero.

For the `route_opt` command, you can enable both regular concurrent clock and data optimization by using the `route_opt.flow.enable_ccd` application option and targeted concurrent clock and data optimization by using the `route_opt.flow.enable_targeted_ccd_wns_optimization` application option. If you enable both, the tool performs regular concurrent clock and data optimization first, followed by targeted concurrent clock and data optimization.

The following script performs targeted concurrent clock and data optimization on a block that has already undergone routing and postroute optimization:

```
open_lib design1
open_block blk.post_route

#Disable regular CCD optimization, which was previously performed
set_app_options \
    -name route_opt.flow.enable_ccd -value false

#Enable targeted CCD optimization
set_app_options \
    -name route_opt.flow.enable_targeted_ccd_wns_optimization \
    -value true

#Specify the path group and endpoints to target
```

```

set_app_options -name ccd.targeted_ccd_path_groups -value clkA
set_app_options -name ccd.targeted_ccd_end_points_file \
    -value clkA_ep.txt

#Specify the optimization type and effort
set_app_options \
    -name ccd.targeted_ccd_select_optimization_moves -value size_only
set_app_options -name ccd.targeted_ccd_wns_optimization_effort \
    -value medium

# Perform the targeted CCD optimization
route_opt

```

Enabling Buffer Removal at the Postroute Stage

To improve QoR by removing buffers or pairs of inverters when concurrent clock and data optimization is performed during `route_opt` command, set the `ccd.post_route_buffer_removal` application option to `true`, as shown in the following example:

```

fc_shell> set_app_options -name route_opt.flow.enable_ccd \
    -value true
fc_shell> set_app_options -name ccd.post_route_buffer_removal \
    -value true
fc_shell> route_opt

```

Reporting Concurrent Clock and Data Timing

You analyze the timing effects of concurrent clock and data optimization by using the `report_ccd_timing` command. By default, it reports the setup and hold slack of the worst capture (D-slack) and launch (Q-slack) paths of the five most critical endpoint registers in the block, as shown in the following example report:

```

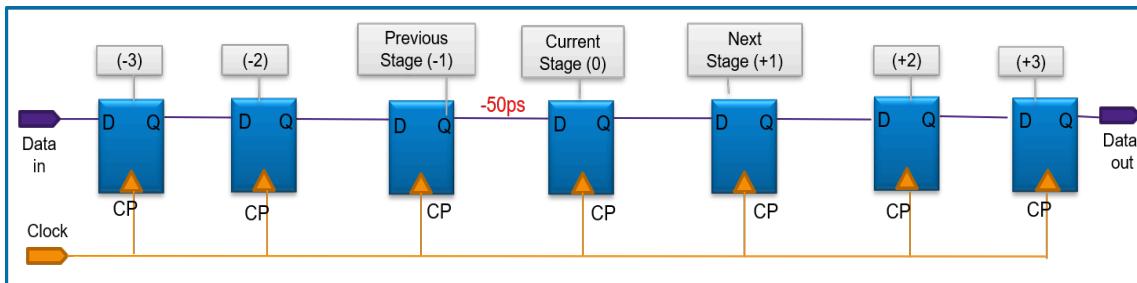
fc_shell> report_ccd_timing
...
...
-----
      Setup          Hold
-----
    D-slack    Q-slack    D-slack    Q-slack      Pin
-----
-0.567710  -0.173974    0.524202   0.167434  sink2/CP
-0.175268   0.125309    0.168582   0.077625  sink3e/CP
-0.173974   0.126836    0.167434   0.076277  sink3a/CP
-0.033271   -0.175268    0.068856   0.168582  sink3d/CP
  0.017364   -0.033271    -0.022138   0.068856  sink3c/CP

```

To control the type of report generated, use the `-type` option as follows:

- To report information for the paths in the fanin or fanout of the most critical endpoints, use the `-type fanin` or `-type fanout` option.
- To report information for the previous, current, and next path stages of the most critical endpoints, use the `-type stage` option.
- To report information for all the previous, current, and next path stages of the most critical endpoints, use the `-type chain` option. [Figure 69](#) shows the previous, current, and next stages for an endpoint register.

Figure 69 Previous, Current, and Next Path Stages



By default, the command reports setup slacks at D and Q pins of registers. To report the hold slacks at these pins, use the `-hold` option.

By default, the command analyzes timing paths across all active scenarios and reports the worst D and Q slacks across scenarios. To analyze and report the slack for specific scenarios use the `-scenarios` option.

To report the information for specific path endpoints, specify the corresponding clock pins by using the `-pins` option.

You can analyze the effects latency adjustments performed during concurrent clock and data optimization by increasing or decreasing the clock arrival of a specific endpoint using the `-prepone` or `-postpone` option with the `-pins` option as shown in the following example:

```
fc_shell> report_ccd_timing -pins Reg11/clk -postpone 0.02
```

You can also see the effects of changing specific cell and net delay values by using the `-annotate_cell_delay` and `-annotate_net_delay` options, as shown in the following example:

```
fc_shell> report_ccd_timing -annotate_cell_delay {{U1/a U1/y 0.2}} \
-pins I2/d -type stage
```

The delay values you specify with these options are used only for analyzing and reporting the concurrent clock and data timing. They are not saved and used during subsequent concurrent clock and data optimization.

Enabling Automatic Zero Balance Point on Macros

To enable the automatic derivation of zero balance points, set the `ccd.skew_opt_default_balance_point_macros` application option to `true` before `compile_fusion` or `place_opt final_opto`. By default, the application option is set to `false`. Setting the `ccd.skew_opt_default_balance_point_macros` application option to `true` adds a balance point of zero automatically on macro clock sink pins with internal delay. This automates the integrated structural multisource clock tree synthesis flow, and avoids manual setting of such balance points for consistent delays between clock tree synthesis and concurrent clock and data optimization function.

```
fc_shell> set_attribute
      - name ccd.skew_opt_default_balance_point_macros -value true
```

Skewing Latch and Discrete Clock Gates

To skew latches and discrete clock gates together, set the `ccd.skew_opt_merge_dcg_cells_by_driver` application option to `true` before `compile_fusion` or `place_opt final_opto`. By default, the application option is set to `false`. Setting the `ccd.skew_opt_merge_dcg_cells_by_driver` application option to `true` enhances the concurrent clock and data optimization to skew latches and gating element by the same offset. This ensures the latch and gate are at the same level and improves hold timing of discrete clock gates.

```
fc_shell> set_attribute
      -name ccd.skew_opt_merge_dcg_cells_by_driver -value true
```

Splitting Clock Cells

You can manually split clock cells that have DRC violations by using the `split_clock_cells -cells` command, as shown in the following example:

```
fc_shell> split_clock_cells -cells [get_cells U1/ICG*]
```

The tool does not split the specified cells if

- They do not have DRC violations
- They have don't-touch, size-only, or fixed-placement attribute settings
- It is necessary to punch ports on the boundaries of power domains or blocks that have been identified with the `set_freeze_ports` command

After splitting a cell, the tool

- Names the new cells using the <original_cell_name>_split_<integer> naming convention
- Copies all the settings and constraints from the original cell to the newly created cells

Instead of specifying the cells to split, you can specify one or more collection of loads that are driven by the same driver by using the `-loads` option, as shown in the following example:

```
fc_shell> set loads1 [get_pins I1/reg*/CK]
fc_shell> set loads2 [get_pins I2/reg*/CK]
fc_shell> split_clock_cells -loads [list $load1 $load2]
```

After splitting, each set of loads is driven by a newly created driver.

Balancing Skew Between Different Clock Trees

The Fusion Compiler tool can automatically balance the skew between a group of clocks. The set of clocks considered during delay balancing is referred to as a *clock balance group*. You can define multiple clock balance groups. For each clock balance group, you can define a delay offset between the clocks. Together, the clock balance groups and their delay offset settings are referred to as *interclock delay balancing constraints*.

Note:

The tool cannot balance skew between a generated clock and other clocks.

To balance multiple clocks, perform the following steps:

- Generate the interclock delay balancing constraints either by
 - Manually defining the clock balance groups and their delay offsets, as described in [Having the tool generate them as described in Generating Interclock Delay Balancing Constraints Automatically](#)
- Balance the interclock delays as described in [Running Interclock Delay Balancing](#).

Defining the Interclock Delay Balancing Constraints

To define interclock delay balancing constraints, use the `create_clock_balance_group` command. At a minimum, you must specify a name for the clock balance group and the clocks in the group.

- To specify a name for the clock balance group, use the `-name` option.
- To specify the clock trees in the group, use the `-objects` option with the `get_clocks` command. By default, the clock balance group is defined for the clocks in the current mode. To define a clock balance group for the clocks of a specific mode, use the `-objects` option with the `get_clocks -mode` command.

For example, to define a clock balance group named `group1` that contains the clocks named `clk1` and `clk2` in the current mode, use the following command:

```
fc_shell> create_clock_balance_group -name group1 \
    -objects [get_clocks {clk1 clk2}]
```

To define a clock balance group named `group2` that contains all the clocks in the mode named `m2`, use the following command:

```
fc_shell> create_clock_balance_group -name group2 \
    -objects [get_clocks -mode m2]
```

By default, the tool has a goal of zero delay offset between clocks. All clocks are balanced with the same insertion delay which usually is the longest insertion delay among the clocks. To specify a delay offset between clocks in the group, use the `-offset_latencies` option. By default, the delay offset applies to the current corner. To apply the delay offset to a specific corner, use the `-corner` option.

For example, assume the design has three clocks named `clk1`, `clk2`, and `clk3`. If you want the insertion delay of `clk1` and `clk2` to be the same and the insertion delay of `clk3` to be 100 less than that of `clk1` and `clk2`, use the following command:

```
fc_shell> create_clock_balance_group -name group3 \
    -objects [get_clocks {clk1 clk2 clk3}] \
    -offset_latencies {0 0 -100}
```

Reporting Clock Balance Groups

To report clock balance groups, use the `report_clock_balance_groups` command. The report lists the clock balance groups for all modes in all active scenarios.

Removing Clock Balance Groups

To remove clock balance groups, use the `remove_clock_balance_groups` command. You can either remove specific clock balance groups by specifying the clock balance groups or all clock balance groups by using the `-all` option. If you specify the clock balance groups to remove, they are removed from the current mode. However, if a specified group

does not exist in the current mode, but does exist in another mode, it is removed from that mode. If you use the `-all` option, the clock balance groups are removed from all modes.

For example, to remove a previously defined clock balance group named group1 from the current mode, use the following command:

```
fc_shell> remove_clock_balance_groups group1
```

To remove all clock balance groups from all modes, use the following command:

```
fc_shell> remove_clock_balance_groups -all
```

Generating Interclock Delay Balancing Constraints Automatically

The Fusion Compiler tool can automatically generate the interclock delay balancing constraints based on the timing relationships between the clocks. To automatically generate the interclock delay balancing constraints, use the following command:

```
fc_shell> derive_clock_balance_constraints
```

This command identifies clocks that have interclock timing paths and places them in the same balance group. After running this command, use the `report_clock_balance_groups` command to report the generated interclock delay balancing constraints, as described in [Reporting Clock Balance Groups](#). If necessary, you can modify the clock balance groups as described in .

By default, the tool considers all timing paths when identifying the timing relationships between the clocks. To consider only those timing paths with slack less than a specified value, use the `-slack_less_than` option with the `derive_clock_balance_constraints` command.

For example, to generate interclock balancing constraints only for paths with slack less than -0.2 ns, use the following command:

```
fc_shell> derive_clock_balance_constraints -slack_less_than -0.2
```

Assume you run this command on a block for which clock A has a timing relationship only with clock B and the worst negative slack (WNS) of this group of timing paths is -0.1 ns and clock C has a timing relationship only to clock D and the WNS of this group of timing paths is -0.3 ns. The command considers only those timing paths with slack less than -0.2 ns, so it defines a single balance group that contains clocks C and D. Clocks A and B are not constrained because the timing paths between them have slack greater than -0.2 ns.

Running Interclock Delay Balancing

Before you perform interclock delay balancing, you must generate the interclock delay balancing constraints as described in .

To perform interclock delay balancing, use the `balance_clock_groups` command. For multicorner-multimode designs, the tool performs interclock delay balancing on all active scenarios.

```
fc_shell> balance_clock_groups
```

Performing Global-Route-Based Optimization Using Machine Learning Data

For designs with poor timing correlation between the preroute and postroute stages, you can improve the correlation by performing optimization after clock tree synthesis using machine learning (ML) data derived after detail routing.

The machine learning concept uses the following terminology:

- Labels, which are the outputs you are trying to predict
- Features, which are the inputs that affect the outputs
- Model, which is the relationship between the features (inputs) and the labels (outputs)
- Training, which is the process of learning the relationship between the features (inputs) and the labels (outputs), based on the data collected

To perform global-route-based optimization using machine learning (ML) data derived from the postroute stage,

1. Collect machine learning features and labels, and train the model by performing the following steps in a specific iteration, iteration N:
 - a. Perform clock tree synthesis and clock routing by completing the `build_clock` and `route_clock` stages of the `clock_opt` command.
 - b. Specify the output directory for the machine learning data by using the new `est_delay.ml_delay_opto_dir` application option.


```
fc_shell> set_application_option \
    -name est_delay.ml_delay_opto_dir -value ./ML
```
 - c. Enable integrated feature extraction by setting the new `est_delay.ml_delay_gre_mode` application option to `feature`.


```
fc_shell> set_application_option \
    -name est_delay.ml_delay_gre_mode -value feature
```
 - d. Perform global-route-based optimization by using the `clock_opt -from final_opt` command.
 - e. Perform detail routing.

- f. Collect labels associated with the features that were previously collected during the `clock_opt -from final_opt` command by using the `estimate_delay -training_labels` command, as shown in the following example:

```
fc_shell> estimate_delay -training_labels "label_1" \
           -output_dir ./ML
```

The labels must be collected after detail routing, but before postroute optimization.

- g. Train the machine learning model for the collected features and label by using the `estimate_delay -train_model` command as shown in the following example:

```
fc_shell> estimate_delay -train_model "features_1 label_1" \
           -output_dir ./ML
```

- h. Continue with postroute optimization and the rest of the steps in the implementation flow.

2. Use the machine learning data to improve correlation by performing the following steps in the subsequent iteration, iteration N+1:

- a. Perform clock tree synthesis and clock routing by completing the `build_clock` and `route_clock` stages of the `clock_opt` command.

- b. Specify the output directory for the machine learning data by using the new `est_delay.ml_delay_opto_dir` application option.

```
fc_shell> set_application_option \
           -name est_delay.ml_delay_opto_dir -value ./ML
```

- c. Enable the machine learning model, which was derived in the previous iteration, by setting the new `est_delay.ml_delay_gre_mode` application option to `enable`.

```
fc_shell> set_application_option \
           -name est_delay.ml_delay_gre_mode -value enable
```

- d. Perform global-route-based optimization with the machine learning data by using the `clock_opt -from final_opt` command.

- e. Perform detail routing.

- f. Disable the machine learning model after you complete all the detail routing steps by using the `estimate_delay -disable_model` command, as shown in the following example:

```
fc_shell> estimate_delay -disable_model -output_dir ./ML
```

- g. Continue with postroute optimization and the rest of the steps in the implementation flow.

When using this feature, ensure that

- The same scenarios are active at each step and each iteration of the flow
- The machine learning model is re-created if there are changes to the design, its environment, or the flow

Routing Clock Trees

After synthesizing and optimizing the clocks, you can detail route the clock nets by using the `route_group` command, as shown in the following example:

```
fc_shell> route_group -all_clock_nets -reuse_existing_global_route true
```

When you set the `-reuse_existing_global_route` option of the `route_group` command to `true`, the detail router uses the existing clock global routes, which ensures better correlation.

Alternatively, you can detail route the clock nets by using the `clock_opt` command, as shown in the following example:

```
fc_shell> clock_opt -from route_clock -to route_clock
```

Inserting Via Ladders During Clock Tree Synthesis, Optimization, and Clock Routing

A via ladder is a stacked via that starts from the pin layer and extends into an upper layer where the router connects to it. Via ladders reduce the via resistance, which can improve performance and electromigration robustness.

The via ladder insertion flow during clock tree synthesis, optimization, and clock routing consists of the following steps:

1. Ensure that the via ladder rules are defined as described in [Defining Via Ladder Rules](#).
2. Specify the via ladders that can be used for specific library pins by using the `set_via_ladder_candidate` command, as described in [Specifying Via Ladder Candidates for Library Pins](#).
3. Define the via ladder constraints by using the `set_via_ladder_rules` command, as described in [Defining Via Ladder Constraints](#).

4. Enable high-performance and electromigration via ladder insertion for critical paths during the `clock_opt` command by setting the `opt.common.enable_via_ladder_insertion` application option to `true`.
 5. Perform clock tree synthesis and optimization by using the `clock_opt -to build_clock` command.
 6. Insert the via ladders by using the `insert_via_ladders` command, as described in [Inserting Via Ladders](#).
 7. Route the clock nets by using the `clock_opt -from route_clock` command.
-

Marking Clocks as Propagated After Clock Tree Synthesis

If you run the `synthesize_clock_trees` or `clock_opt` command, after clock tree synthesis, the tool removes the ideal setting from all the synthesized clocks of all active scenarios and sets all the corresponding register clock pins as propagated.

If your block has scenarios that were not set as active before clock tree synthesis, set these scenarios as active by using the `set_scenario_status -active true` command and mark all clocks as propagated by using the `synthesize_clock_trees -propagate_only` command, as shown in the following example:

```
fc_shell> set_scenario_status -active true [all_scenarios]
fc_shell> synthesize_clock_trees -propagate_only
```

The `synthesize_clock_trees -propagate_only` command removes the ideal setting from the clocks; it does not perform clock tree synthesis.

Performing Postroute Clock Tree Optimization

When you detail route the clock nets of a block, its clock tree QoR can degrade due to the differences between the clock global routes used during clock tree synthesis and the clock detail routes. Clock tree QoR can further degrade when you detail route the signal nets due to coupling capacitance and crosstalk effects.

You can perform clock tree optimization on a postroute design by using the `synthesize_clock_trees -postroute` command. When you do so, you should specify the type of routing performed on the design by using the `-routed_clock_stage` option.

For example, to perform clock tree optimization on a design that has completed clock routing, use the following command:

```
fc_shell> synthesize_clock_trees -postroute \
    -routed_clock_stage detail
```

To perform clock tree optimization on a design that has completed both clock and signal routing, use the following command:

```
fc_shell> synthesize_clock_trees -postroute \
    -routed_clock_stage detail_with_signal_routes
```

Note:

When you enable concurrent clock and data optimization, including power or area recovery that uses this technique, the tool does not optimize the clock latency or skew during postroute clock tree optimization; it only fixes the logical DRC violations on the clock network.

Performing Voltage Optimization

The voltage level of a design is determined by the technology node (at device level) and the performance requirement. The Fusion Compiler tool can perform voltage optimization, which tries to achieve better performance, power, and area (PPA) at a lower voltage level.

To perform voltage optimization, you need

- Scaled library panes

Voltage optimization works on the fundamental principle of scaling the voltage, Therefore, scaled library panes with different voltages are required.
- A signoff methodology with a scaled library setup

The voltage optimization flow consists of the following high-level steps:

1. Perform physical synthesis at the original design voltage using the `compile_fusion` command.
2. Perform clock tree synthesis and optimization at the original design voltage by using the `clock_opt` command.
3. Associate a set of related library panes to form a scaling group by using the `define_scaling_lib_group` command.
4. Specify voltage ranges for specific corners by using the `set_vopt_range` command, as shown in the following example:

```
fc_shell> set_vopt_range -corner C1 \
    -low_voltage 0.81 -high_voltage 1.0
fc_shell> set_vopt_range -corner C2 \
    -low_voltage 0.81 -high_voltage 1.0
```

5. Specify a total negative slack (TNS) and power target and tolerance for voltage optimization by using the `set_vopt_target` command.

The following example specifies a total negative slack target for voltage optimization and specifies that I/O paths should be excluded from the total negative slack costing:

```
fc_shell> set_vopt_target -tns -1.0 -excludeIO
```

6. Perform voltage optimization by using the `voltage_opt` command.
7. Perform routing at the optimized voltage by using the `route_auto` command.
8. Perform postroute optimization at the optimized voltage by using the `route_opt` command.

Marking Clock Trees as Synthesized

To prevent the Fusion Compiler tool from modifying them, you can mark existing clock trees in your design as synthesized clock tree by using the `mark_clock_trees` command as shown in [Table 25](#).

Table 25 Using the `mark_clock_trees` Command

To do this	Use this option
Mark only the clock trees of specific clocks By default, the command marks all clocks defined by the <code>create_clock</code> and <code>create_generated_clock</code> commands in all modes of all active scenarios	<code>-clocks clock_list</code>
Mark the clock tree as synthesized This is the default behavior	<code>-synthesized</code>
Remove the synthesized attribute settings from the clock trees	<code>-clear</code>
Apply the <code>dont_touch</code> attribute setting on the clock trees	<code>-dont_touch</code>
Remove the <code>dont_touch</code> attribute settings from the clock trees	<code>-clear -dont_touch</code>
Propagate the nondefault clock routing rules specified by the <code>set_clock_routing_rules</code> command	<code>-routing_rules</code>
Propagate the clock cell spacing rules specified by the <code>set_clock_cell_spacing</code> command	<code>-clock_cell_spacing</code>
Mark the clock sinks as fixed	<code>-fix_sinks</code>
Freeze the routing of the clock nets	<code>-freeze_routing</code>

The tool traverses the clock trees and marks the clock trees as specified. Clock tree traversal continues until it finds an exception pin or a default sink pin.

Removing Clock Trees

To remove the buffers and inverters on a clock tree, use the `remove_clock_trees` command. The `remove_clock_trees` command traverses the clock tree from its root to its sinks and removes all buffers and inverters, except those with `dont_touch` or `size_only` attributes. Cells beyond clock tree exceptions are considered part of the clock tree, but cells beyond a clock-to-data pin are considered part of the data path and are not removed. In addition to removing the buffers and inverters, the `remove_clock_trees` command resets the attributes related to clock tree synthesis.

Note:

The `remove_clock_trees` command does not support clock mesh nets.

By default, this command removes the buffers and inverters from all clock trees in all modes. To remove the buffers and inverters from specific clock trees, use the `-clocks` option to specify the clock trees. By default, when you use the `-clocks` option, the clocks are selected from the current mode. To select clocks from a specific mode, use the `get_clocks -mode` command to select the clocks.

For example, to remove only the clock tree in the current mode named `my_clk`, use the following command:

```
fc_shell> remove_clock_trees -clocks [get_clocks my_clk]
```

[Table 26](#) shows how clock tree removal is affected by the structure of the clock tree.

Table 26 Clock Tree Removal Behavior

Object	Impact on clock tree removal
Boundary cell	Removed.
Cells on don't touch net	Preserved.
Don't touch cell	Preserved.
Fixed cell	Preserved.
Generated clock	Preserved, if generated clock is defined on buffer/inverter pin. Traversal and clock tree removal continue past the generated clock.
Guide buffer	Removed.
Integrated clock-gating (ICG) cell	Preserved. Traversal (and clock tree removal) continues past the integrated clock-gating cell.

Table 26 Clock Tree Removal Behavior (Continued)

Object	Impact on clock tree removal
Block abstraction model	Preserved. Traversal (and clock tree removal) continues past the block abstraction model.
Inverter	Removed in pairs only. If a clock tree contains a single inverter, it is not removed.
Isolation cell	Preserved.
Level shifter	Preserved.
Three-state buffer	Preserved. Traversal (and clock tree removal) stops at the three-state buffer.
Buffer or inverter added beyond exception	Removed.

By default, the `remove_clock_trees` command removes the detail route shapes of the clock nets it removes. However, you can preserve the route shapes of the clock nets by setting the `shape_use` attribute of the clock nets to `user_route`.

Implementing Multisource Clock Trees

The following topics introduce the different types of multisource clock trees and provides detailed information about the different multisource clock tree implementation flows and steps:

- [Introduction to Multisource Clock Trees Structures](#)
- [Implementing a Regular Multisource Clock Tree](#)
- [Implementing a Regular Multisource Clock Tree Using Integrated Tap Assignment](#)
- [Implementing a Regular Multisource Clock Tree With an H-Tree-Only Global Clock Tree Structure](#)
- [Implementing a Structural Multisource Clock Tree](#)
- [Implementing a Structural Multisource Clock Tree Using Integrated Subtree Synthesis](#)
- [Inserting Clock Drivers](#)
- [Synthesizing the Global Clock Trees](#)
- [Creating Clock Straps](#)

- [Routing to Clock Straps](#)
 - [Analyzing the Clock Mesh](#)
 - [Performing Automated Tap Insertion and H-Tree Synthesis](#)
 - [Specifying Tap Assignment Options and Settings](#)
 - [Building the Local Clock Subtree Structures](#)
-

Introduction to Multisource Clock Trees Structures

A multisource clock tree is a custom clock structure that has more tolerance to on-chip variation and has better performance across corners than traditional clock tree structures.

These custom clock trees consist of

- A global clock structure, which includes
 - The clock root
 - A global clock tree, which is usually an H-tree structure
 - Clock mesh drivers
 - A clock mesh
- Local subtrees that can be driven
 - By a predefined set of drivers, called tap drivers, that are connected to the clock mesh, as shown in [Figure 70](#).

Because the subtrees are built using regular clock tree synthesis commands such as the `synthesize_clock_trees` or `clock_opt` command, such a structure is called a **regular multisource clock tree**.

- Directly from multiple points of the clock mesh, as shown in [Figure 71](#).

Because the subtrees are built preserving the user-defined structure and optimized by merging and splitting the clock cells, such a structure is called a **structural multisource clock tree**.

Figure 70 Regular Multisource Clock Tree

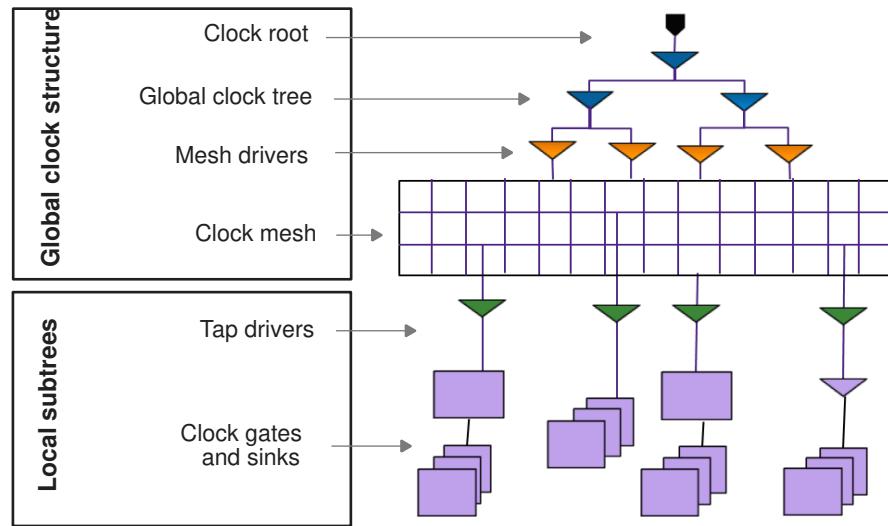
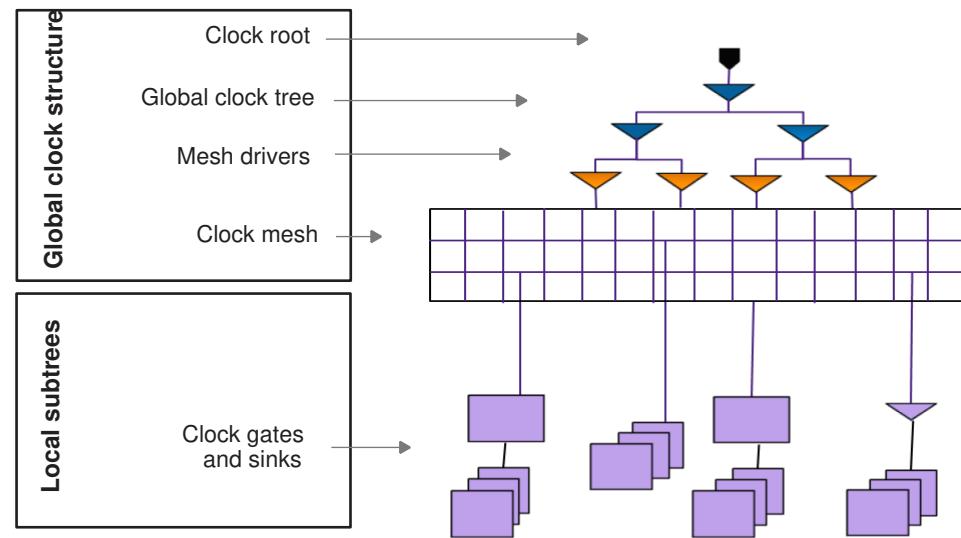


Figure 71 Structural Multisource Clock Tree



Implementing a Regular Multisource Clock Tree

To implement a regular multisource clock tree, use the following steps:

1. Specify your clock tree constraints and settings.
2. Perform the following steps for multivoltage designs:
 - a. Enable multivoltage support by setting the `cts.multisource.enable_full_mv_support` application option to `true`.
 - b. Enable the addition of physical feedthrough cells, which belong to a voltage area physically but not logically, by setting the `opt.common.allow_physical_feedthrough` application option to `true`.
 - c. Run the `check_mv_design` command and fix all multivoltage issues.
3. Enable cell electromigration fixing by setting the `cts.multisource.cell_em_aware` application option to `true`.
4. Insert the tap drivers by using the `create_clock_drivers` command, as described in [Inserting Clock Drivers](#).
5. Create the clock mesh by using the `create_clock_straps` command, as described in [Creating Clock Straps](#).
6. Insert the mesh drivers by using the `create_clock_drivers` command, as described in [Inserting Clock Drivers](#).
7. Build the global clock tree structure by using the `synthesize_multisource_global_clock_trees` command, as described in .
8. Route the connections between the clock mesh and the tap drivers by using the `route_clock_straps` command, as described in [Routing to Clock Straps](#).
9. Analyze the clock mesh by using the `analyze_subcircuit` command, as described in .
10. Specify options and settings for tap assignment as described in .
11. Perform tap assignment by using the `synthesize_multisource_clock_taps` command, which
 - Merges equivalent clock cells to remove any artificial boundaries between clusters of sinks
 - Assigns endpoints to the closest tap driver and split cells along the path honoring any sink groups defined
 - Copies the UPF and SDC constraints and the user-specified attributes onto the newly created cells across the active scenarios

12. Synthesize the entire clock tree, from the clock root, by using the following command:

```
fc_shell> clock_opt -from build_clock -to route_clock
```

During this step, the tool builds the local subtrees that are driven by the tap drivers. It also fixes DRC violations beyond ignore exceptions of the multisource clock tree, if applicable. You can also synthesize other clocks in the design that are not synthesized.

For more information about the `clock_opt` command, see [“clock_opt” Command](#).

13. Analyze the clock tree results as described in [“Analyzing the Clock Tree Results”](#).

Implementing a Regular Multisource Clock Tree Using Integrated Tap Assignment

A regular multisource clock tree has tap drivers that are driven by a clock mesh and drive clock-gating cells and sinks. If you insert these tap drivers before you run the `compile_fusion` command, the tool can assign sinks to these drivers such that it improves the overall timing QoR of the block. It also allows the tool to improve the timing of clock-gating paths.

To perform regular multisource clock tree synthesis integrated with placement and optimization, use the following steps:

1. Specify the placement and optimization constraints and settings.
2. Specify the clock tree constraints and settings.
3. Perform the following steps for multivoltage designs:
 - a. Enable multivoltage support by setting the `cts.multisource.enable_full_mv_support` application option to `true`.
 - b. Enable the addition of physical feedthrough cells, which belong to a voltage area physically but not logically, by setting the `opt.common.allow_physical_feedthrough` application option to `true`.
 - c. Run the `check_mv_design` command and fix all multivoltage issues.
4. Enable cell electromigration fixing by setting the `cts.multisource.cell_em_aware` application option to `true`.
5. Insert the tap drivers by using the `create_clock_drivers` command, as described in [“Inserting Clock Drivers”](#).
6. Create the clock mesh by using the `create_clock_straps` command, as described in [“Creating Clock Straps”](#).

7. Insert the mesh drivers by using the `create_clock_drivers` command, as described in [Inserting Clock Drivers](#).
8. Build the global clock tree structure by using the `synthesize_multisource_global_clock_trees` command, as described in [.](#)
9. Route the connections between the clock mesh and the tap drivers by using the `route_clock_straps` command, as described in [Routing to Clock Straps](#).
10. Analyze the clock mesh by using the `analyze_subcircuit` command, as described in [.](#)
11. Specify options and settings for tap assignment as described in [.](#)
12. Enable integrated tap assignment by setting the `compile.flow.enable_multisource_clock_trees` application option to `true` and run the `compile_fusion -from initial_place` command.

```
fc_shell> set_app_options \
    -name compile_flow.flow.enable_multisource_clock_trees -value true
fc_shell> compile_fusion -initial_place
```
13. Synthesize the entire clock tree, from the clock root, by using the following command:

```
fc_shell> clock_opt -from build_clock -to route_clock
```

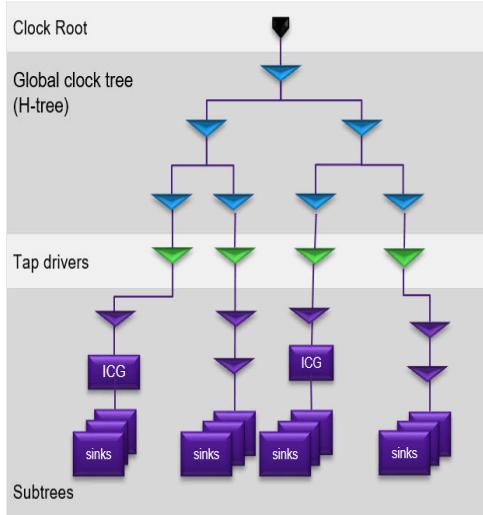
During this step, the tool builds the local subtrees that are driven by the tap drivers. It also fixes DRC violations beyond ignore exceptions of the multisource clock tree, if applicable. You can also synthesize other clocks in the design that are not synthesized.

For more information about the `clock_opt` command, see [.](#)
14. Analyze the clock tree results as described in [Analyzing the Clock Tree Results](#).

Implementing a Regular Multisource Clock Tree With an H-Tree-Only Global Clock Tree Structure

The following figure shows a regular multisource clock where the global clock structure does not have a clock mesh. The global clock structure consists only of an H-tree, which directly drives the tap drivers.

Figure 72 Regular Multisource Clock Tree With an H-Tree-Only Global Clock Structure



To implement a regular multisource clock tree structure with an H-tree-only global clock tree structure,

1. Specify your clock tree constraints and settings.
2. Perform the following steps for multivoltage designs:
 - a. Enable multivoltage support by setting the `cts.multisource.enable_full_mv_support` application option to `true`.
 - b. Enable the addition of physical feedthrough cells, which belong to a voltage area physically but not logically, by setting the `opt.common.allow_physical_feedthrough` application option to `true`.
 - c. Run the `check_mv_design` command and fix all multivoltage issues.
3. Enable cell electromigration fixing by setting the `cts.multisource.cell_em_aware` application option to `true`.
4. Specify the required settings by using the `set_regular_multisource_clock_tree_options` command and perform tap insertion and global clock tree (H-tree) synthesis by using the `synthesize_regular_multisource_clock_trees` command, as described in .
5. Specify options and settings for tap assignment, as described in .

6. Perform tap assignment by using the `synthesize_multisource_clock_taps` command, which
 - Merges equivalent clock cells to remove any artificial boundaries between clusters of sinks
 - Assigns endpoints to the closest tap driver and splits cells along the path honoring any sink groups defined
 - Copies the UPF and SDC constraints and the user-specified attributes onto the newly created cells across the active scenarios
7. Synthesize the entire clock tree, from the clock root, by using the following command:

```
fc_shell> clock_opt -from build_clock -to route_clock
```

During this step, the tool builds the local subtrees that are driven by the tap drivers. It also fixes DRC violations beyond ignore exceptions of the multisource clock tree, if applicable. You can also synthesize other clocks in the design that are not synthesized.

For more information about the `clock_opt` command, see .

8. Analyze the clock tree results, as described in [Analyzing the Clock Tree Results](#).

Implementing a Structural Multisource Clock Tree

In a structural multisource clock tree, the subtrees are driven directly by the clock mesh or through predefined drivers. These drivers are in turn driven by global clock distribution structure such as a clock mesh and H-tree. The local subtrees of a structural multisource clock tree are optimized by clock cell merging, splitting, sizing, and relocation, while preserving the levels of the user-defined clock tree structure.

To implement a structural multisource clock tree, perform the following steps:

1. Specify your clock tree constraints and settings.
2. For multivoltage designs,
 - Enable multivoltage support by using the following application option setting:


```
fc_shell> set_app_options \  
-name cts.multisource.enable_full_mv_support -value true
```
 - Enable the addition of physical feedthrough cells, which belong to a voltage area physically but not logically, by using the following application option setting:


```
fc_shell> set_app_options \  
-name opt.common.allow_physical_feedthrough -value true
```
 - Run the `check_mv_design` command and fix all multivoltage issues

3. Enable cell electromigration fixing by setting the `cts.multisource.cell_em_aware` application option to `true`.
4. Create the clock mesh by using the `create_clock_straps` command, as described in [Creating Clock Straps](#).
5. Insert the mesh drivers by using the `create_clock_drivers` command, as described in [Inserting Clock Drivers](#).
6. Build the global clock tree structure by using the `synthesize_multisource_global_clock_trees` command, as described in .
7. Model the delays of the mesh nets using `set_annotation_delay` and `set_annotation_transition` commands.

The loads of the clock mesh are not finalized until the local subtrees are synthesized in step 6 and routed to the mesh in step 7. Therefore the clock mesh can only be analyzed after you complete those steps. However, to prevent the tool from seeing a very large delay through the clock mesh when synthesizing the local subtrees, annotate a realistic delay and transition value on the clock mesh net.

8. Build the local subtrees by using the `synthesize_multisource_clock_subtrees` command, as described in .
9. Route the connections between the clock mesh and the local subtrees by using the `route_clock_straps` command, as described in [Routing to Clock Straps](#).
10. Analyze the clock mesh by using the `analyze_subcircuit` command, as described in .
11. Synthesize the entire clock tree, from the clock root, by using the `clock_opt -from build_clock -to route_clock` command.

During this step, the tool fixes DRC violations beyond ignore exceptions of the multisource clock tree, if applicable. You can also synthesize other clocks in the design that are not synthesized.

12. Analyze the clock tree results as described in [Analyzing the Clock Tree Results](#).

Implementing a Structural Multisource Clock Tree Using Integrated Subtree Synthesis

In a structural multisource clock tree, the subtrees are driven directly by the clock mesh or through predefined drivers. These drivers are, in turn, driven by a global clock distribution structure such as a clock mesh or an H-tree. By using the `compile_fusion` command to synthesize and optimize the local subtrees of a structural multisource clock tree, you can improve the timing QoR of the design.

To implement a structural multisource clock tree using the integrated subtree synthesis capabilities, perform the following steps:

1. Specify your clock tree constraints and settings.
2. For multivoltage designs,
 - Enable multivoltage support by using the following application option setting:

```
fc_shell> set_app_options \
    -name cts.multisource.enable_full_mv_support -value true
```
 - Enable the addition of physical feedthrough cells, which belong to a voltage area physically but not logically, by using the following application option setting:

```
fc_shell> set_app_options \
    -name opt.common.allow_physical_feedthrough -value true
```
 - Run the `check_mv_design` command and fix all multivoltage issues
3. Enable multicorner optimization by setting the `cts.multisource.enable_multi_corner_support` application option to `true`. The default is `false`.
 By default, the tool uses the worst corner associated with the mode in which the subtree options are defined. When you enable this feature, the tool considers all corners associated with the mode in which the subtree options are defined.
4. Enable cell electromigration fixing by setting the `cts.multisource.cell_em_aware` application option to `true`.
5. Create the clock mesh by using the `create_clock_straps` command, as described in [Creating Clock Straps](#).
6. Insert the mesh drivers by using the `create_clock_drivers` command, as described in [Inserting Clock Drivers](#).
7. Build the global clock tree structure by using the `synthesize_multisource_global_clock_trees` command, as described in .
8. Model the delays of the mesh nets by using the `set_annotated_delay` and `set_annotated_transition` commands.
9. Specify settings for local subtree synthesis by using the `set_multisource_clock_subtree_options` command.
 You must specify
 - The clock for synthesizing the subtrees by using the `-clock` option
 - The drivers of the subtrees to be synthesized by using the `-driver_objects` option

Optionally you can

- Prevent the tool from merging specific clock tree cells by using the `-dont_merge_cells` option
- Balance the levels of the local subtree by using the `-balance_levels true` option and specify a target number of levels, which applies to all sinks of the local subtree, by using the `-target_level` option

Optionally, you can apply a different target number of levels to specific sink pins and clock balance points by using the `set_multisource_clock_subtree_constraints` command with the `-pins` and `-target_level` options.

- Specify a maximum total wire delay from any subtree driver to any of its sinks by using the `-max_total_wire_delay` option and the corner it applies to by using the `-corner` option
- Enable the reordering of clock-gating cells by using the `-enable_icg_reordering true` option

When you enable this feature, the tool swaps the position of a clock-gating cell with the buffer that is driving the clock-gating cell. Then, the buffer is gated by the clock-gating cell, thereby reducing the dynamic power consumption.

To perform reordering, you must

- Enable the scenarios used for structural multisource clock tree synthesis
- Annotate the switching activity for the enable pins of the clock-gating cells

You can prevent specific clock-gating cells from being reordered by using the `set_multisource_clock_subtree_constraints` command with the `-cells` and `-ignore_for_icg_reordering` options.

To report the settings you specified, use the

`report_multisource_clock_subtree_options` command. To remove the settings you specified, use the `remove_multisource_clock_subtree_options` command.

10. Enable integrated structural multisource clock tree synthesis by setting the `compile.flow.enable_multisource_clock_trees` application option to `true`, and build the local subtree by running the `compile_fusion -from initial_place` command.

```
fc_shell> set_app_options \
    -name compile_flow.flow.enable_multisource_clock_trees -value true
fc_shell> compile_fusion -from initial_place
```

11. Route the connections between the clock mesh and the local subtrees by using the `route_clock_straps` command, as described in [Routing to Clock Straps](#).
 12. Analyze the clock mesh by using the `analyze_subcircuit` command, as described in [Analyzing the Clock Tree Results](#).
 13. Enable integrated structural multisource clock tree synthesis by setting the `clock_opt.flow.enable_multisource_clock_trees` application option to `true`, and synthesize the entire clock tree from the clock root by using the `clock_opt -from build_clock -to route_clock` command.
- ```
fc_shell> set_app_options \
 -name clock_opt.flow.enable_multisource_clock_trees -value true
fc_shell> clock_opt -from build_clock -to route_clock
```
- During this step, the tool incrementally optimizes the existing subtrees.
14. Analyze the clock tree results as described in [Analyzing the Clock Tree Results](#).

## Inserting Clock Drivers

Before you insert clock drivers,

- Specify all clock tree synthesis settings, including clock routing rules
- Specify that the inserted clock drivers should be aligned with the clock straps of the mesh structure by setting the `cts.multisource.enable_clock_driver_snapping` application option to `true`

During multisource clock tree synthesis, you can use the `create_clock_drivers` command for the following:

- Insert mesh drivers for both regular or structural multisource clock trees.
- Tap drivers for regular multisource clock trees.

When you insert mesh or tap drivers,

- Specify the loads to drive by using the `-loads` option. The load can be a net or a set of pins or ports connected to the same net.
- Specify where to add the clock drivers as exact locations by using the `-location` option.

Alternatively, you can add the clock drivers in an X by Y grid pattern over the entire core area by using the `-boxes` option. You can limit the X by Y grid pattern to a specific area by using the `-boundary` option with the `-boxes` option.

- Specify a list of cells that can be used as clock drivers by using the `-lib_cells` option. You can specify both single-rail and dual-rail library cells.

Alternatively, you can select the existing driver of the load net as a template for the clock drivers by using the `-template` option. If the template cell you specify is a buffer or inverter, you can also specify different versions of this buffer or inverter as clock drivers by using the `-lib_cells` option with the `-template` option.

When you insert mesh drivers, to specify that the outputs of all the clock drivers must be connected together to drive the mesh net, use the `-short_outputs` option.

The following example creates a grid of eight by eight mesh drivers, places them in a regular pattern that covers the full core area, shorts the outputs of the clock drivers and connects to the net named `clk_mesh`, and transfers preexisting routing shapes and vias from the net named `clk` to this net:

```
fc_shell> create_clock_drivers -loads [get_nets clk1_mesh] \
 -boxes {8 8} -lib_cells [get_lib_cells my_lib/CKBUF8X] \
 -short_outputs -output_net_name clk_mesh \
 -transfer_wires_from [get_nets clk]
```

The following example creates a grid of five by five tap drivers that are placed within the rectangle bounded at the lower-left by (200, 200) and upper right by (1000, 1000).

```
fc_shell> create_clock_drivers -loads [get_nets clkA] \
 -lib_cells [get_lib_cells my_lib/CKBUF8X] \
 -boxes {5 5} -boundary [list {{200 200} {1000 1000}}]
```

In this example, one of the tap drivers drives all the loads of the original clock net named `clkA`. To distribute the loads among all the clock drivers, you must subsequently perform automated multisource tap assignment by using the `synthesize_multisource_clock_taps` command, as described in .

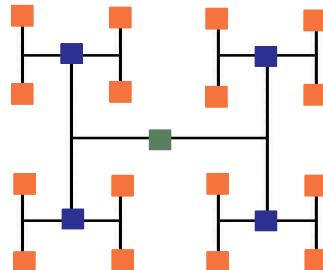
You can use the `create_clock_drivers` to add multiple levels of clock drivers by using the `-configuration` option and specifying the configuration of drivers at each level.

The following example inserts three levels of clock drivers. The first level consists of one buffer, the second level consists of four inverters in a 2x2 grid, and the third level consists of 16 inverters in a 4x4 grid.

```
fc_shell> create_clock_drivers -loads [get_nets clk] \
 -configuration [list \
 [list -level 1 -boxes {1 1} -lib_cells buf32x] \
 [list -level 2 -boxes {2 2} -lib_cells inv16x] \
 [list -level 3 -boxes {4 4} -lib_cells inv8x]]
```

For this example, because there is no bounding box specified at any level, the drivers at each level are evenly distributed in the core area. The buffer in the first level is placed at the center of the core area, the four buffers in the next level are placed at the center of the four quadrants of the core area, and so on, resulting in an evenly placed clock drivers that can be routed to form an H-tree structure as shown in [Figure 73](#)

**Figure 73** Clock Drivers Placed in an H-Tree Structure



After it inserts the clock drivers, the tool marks the clock drivers that it inserts as fixed and don't touch. The tool avoids overlapping the clock drivers with other fixed cells, blockages, and macros. However, the clock drivers can overlap with cells that are not fixed. The `create_clock_drivers` command does not legalize the design. To do so, run the `legalize_placement` command. If you run the `create_clock_drivers` command multiple times, to reduce runtime, run the `legalize_placement` command one time, after you complete all the `create_clock_drivers` command runs.

When routing the H-trees, the tool uses the highest routing layers available, based on the routing rules specified for the clock nets. These same layers can contain prerouted nets such as power and ground nets. To prevent placing clock drivers under these preroutes, which can cause pin accessibility issues, the `create_clock_drivers` command creates temporary placement blockages for the preroutes on the same layers it uses for routing. After it completes routing, it removes these temporary placement blockages, as shown in the following example output.

```
...
Net: clk; Rule: htree_ndr; Min Layer: M8; Max Layer: M9
...
Information: Using routing H/V layer pair 'M9'/'M8' for net 'clk'.
(CTS-659)
...
Converting metal shapes in horizontal layer M9 and vertical layer M8 into
placement blockages.
In total 703 placement blockages created.
...
Successfully deleted all temporary placement blockages and the cell map.
```

If you specify multirow-height cells with the `-lib_cell` option of the `create_clock_drivers` command, the tool might not be able to place them due to the temporary placement blockages it creates for the preroutes. If so, you can prevent the tool from creating the temporary placement blockages by setting the `cts.multisource.enable_pin_accessibility_for_global_clock_trees` application option to `false`.

To remove clock drivers inserted by the `create_clock_drivers` command, use the `remove_clock_drivers` command.

## Inserting Clock Drivers for Designs With Multiple Levels of Physical Hierarchy

To insert clock drivers from the top level of a design with multiple levels of physical hierarchy,

1. Enable the lower-level physical blocks for editing by using the `set_editability` command.
2. Enable clock driver insertion for multiple levels of physical hierarchy by setting the `cts.multisource.enable_mlph_flow` application option to `true`.
3. Insert clock drivers by using the `create_clock_drivers` command.
4. Route the inserted clock drivers by using the  
`synthesize_multisource_global_clock_trees -roots -leaves -use_zroute_for_pin_connections` command.

When inserting clock drivers to lower-level blocks, the tool reuses existing clock ports. If there are no existing clock ports, the tool creates new ports, if it is allowed. You can allow new ports to be added to a block by using the `set_freeze_ports` command.

## Synthesizing the Global Clock Trees

Before you insert clock drivers, ensure that all clock tree synthesis settings, including clock routing rules, are specified.

To perform clock tree synthesis and detail routing to build an H-tree style global clock tree, use the `synthesize_multisource_global_clock_trees` command. When building the H-tree, the tool tries to minimize the skew between the endpoints, which is essential for multisource clock tree synthesis.

When you use this command to synthesize and detail route a global clock tree, you must specify

- The clock net to synthesize by using the `-nets` option.
- The library cells to use by using the `-lib_cells` option. You can specify both single-rail and dual-rail library cells.

By default, the tool uses the Custom Router to route the H-tree structure and connect to the pins of the clock tree cells. If the Custom Router is unable to resolve all routing DRC violation when making pin connections, use the Zroute to make the pin connections by using the `-use_zroute_for_pin_connections` options. To stop the routes at the highest available metal layer close to the pin shape, use the `-skip_pin_connections` option.

The following example synthesizes and detail routes a global clock tree for the clock net named clkA.

```
fc_shell> synthesize_multisource_global_clock_trees \
 -nets [get_nets clkA] -lib_cells [get_lib_cells my_lib/CKBUF8X] \
 -use_zroute_for_pin_connections
```

You can also use the `synthesize_multisource_global_clock_trees` command to only perform clock detail routing for an H-tree style clock structure that has already been synthesized.

When doing so, you must specify

- The startpoint of the global clock structure, by using the `-roots` option.
- The endpoints of the global clock structure by using the `-leaves` option.

The following example detail routes an existing H-tree style clock structure. The startpoint is the port named clk1 and the endpoints are the inputs of a group of mesh drivers.

```
fc_shell> synthesize_multisource_global_clock_trees \
 -roots [get_ports clk1] -leaves [get_pins mesh_buf*/A] \
 -use_zroute_for_pin_connections
```

When routing the H-trees, the tool uses the highest routing layers available, based on the routing rules specified for the clock nets. These same layers can contain prerouted nets such as power and ground nets. To prevent placing clock drivers under these preroutes, which can cause pin accessibility issues, the `synthesize_multisource_global_clock_trees` command creates temporary placement blockages for the preroutes on the same layers it uses for routing. After it completes routing, it removes these temporary placement blockages, as shown in the following example output.

```
...
Net: clk; Rule: htree_ndr; Min Layer: M8; Max Layer: M9
...
Information: Using routing H/V layer pair 'M9'/'M8' for net 'clk'.
(CTS-659)
...
Converting metal shapes in horizontal layer M9 and vertical layer M8 into
placement blockages.
In total 703 placement blockages created.
...
Successfully deleted all temporary placement blockages and the cell map.
```

If you specify multirow-height cells with the `-lib_cell` option of the `synthesize_multisource_global_clock_trees` command, the tool might not be able to place them due to the temporary placement blockages it creates for the preroutes. If so, you can prevent the tool from creating the temporary placement blockages by setting the

`cts.multisource.enable_pin_accessibility_for_global_clock_trees` application option to `false`.

To remove a clock structure created by the `synthesize_multisource_global_clock_trees` command, use the `remove_multisource_global_clock_trees` command.

## Inserting Clock Drivers for Designs With Multiple Levels of Physical Hierarchy

To insert clock drivers from the top level of a design with multiple levels of physical hierarchy,

1. Enable the lower-level physical blocks for editing by using the `set_editability` command.
2. Enable clock driver insertion for multiple levels of physical hierarchy by setting the `cts.multisource.enable_mlph_flow` application option to `true`.
3. Insert clock drivers by using the `create_clock_drivers` command.

When inserting clock drivers to lower-level blocks, the tool reuses existing clock ports. If there are no existing clock ports, the tool creates new ports, if it is allowed. You can allow new ports to be added to a block by using the `set_freeze_ports` command.

## Creating Clock Straps

You can create clock straps, which are straight metal shapes in a single routing layer, by using the `create_clock_straps` command.

You can use this command to implement

- A clock mesh, which is a two-dimensional grid in a horizontal and a vertical layer, where the straps are connected by vias at the intersection points, as shown in [Figure 74](#).
- A clock spine, which can be either a one- or two-dimensional structures.

One-dimensional spines are straps in a single direction. Two-dimensional spines consists of one-dimensional spines connected to multiple stripes in the orthogonal direction. Stripes connected to one spine do not connect to stripes of a different spine and the minimum distance between the stripes of different spines is called the backoff, as shown in [Figure 75](#)

Figure 74 Clock Mesh Structure

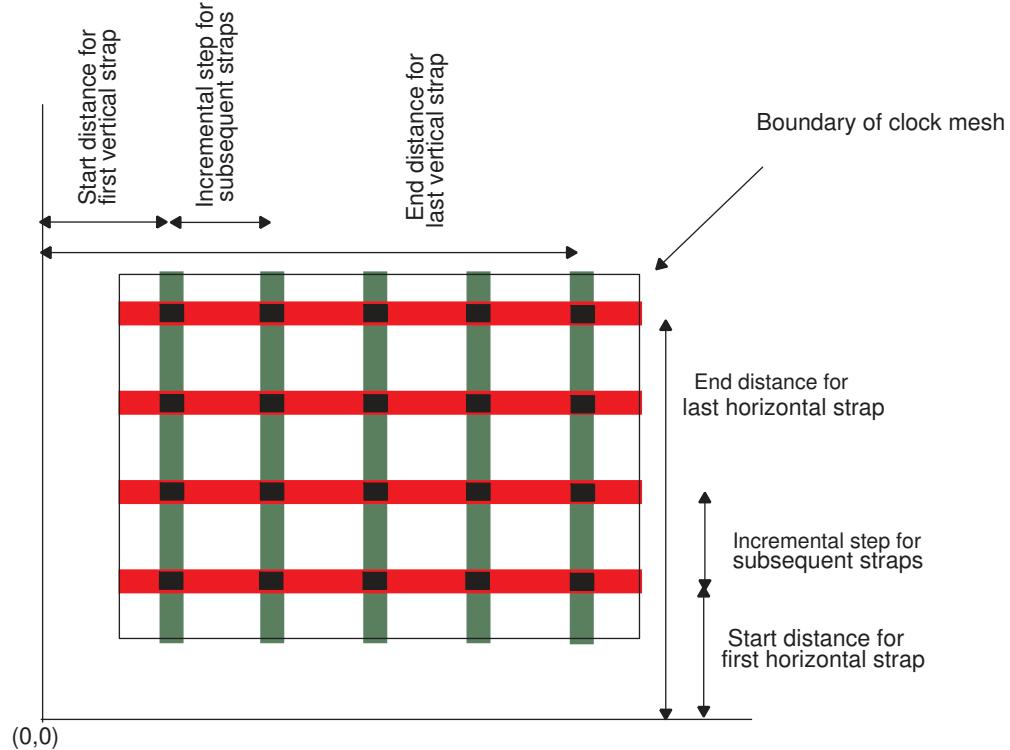
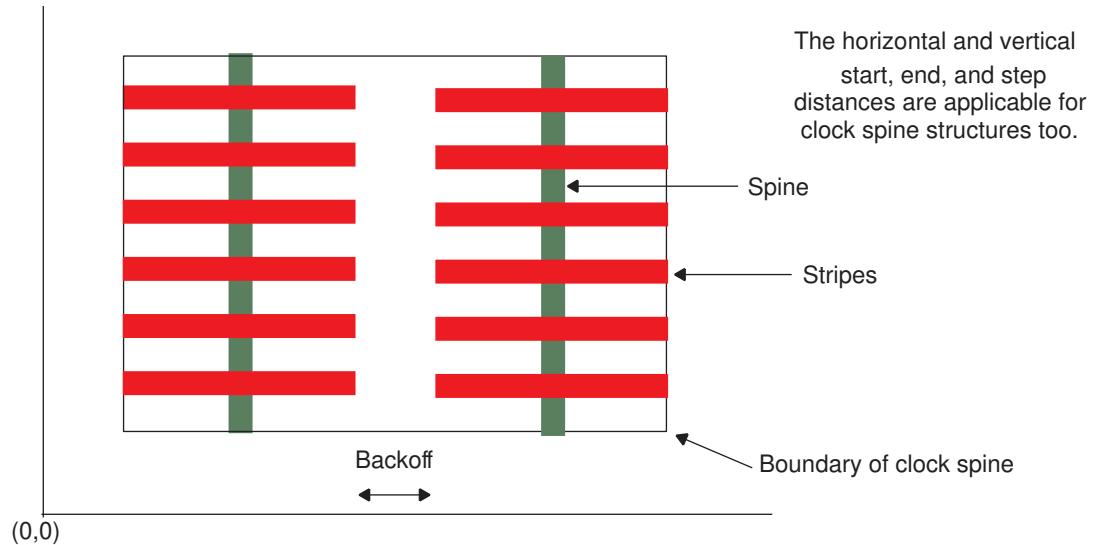


Figure 75 Clock Spine Structure



When you use the `create_clock_straps` command,

- To specify the clock net for which to create the structure, use the `-net` option.
- To specify the bounding box to confine the structure, use the `-boundary` option.
- To specify keepouts, use the `-keepouts` option.

If part of a strap is over a keepout, by default, the tool splits the strap and creates the portions that are outside the keepouts. However, you can disable splitting and specify that the tool not create a strap if it is over a keepout by using the `-allow_splitting false` option.

- To specify the layers on which create the straps, use the `-layers` option.

The tool determines the direction of the straps based on the metal layer you specify. To create straps in both directions, specify a layer for each direction.

- To specify the width of the straps, use the `-width` option.

- To specify where to create the straps, use the `-grid` option as follows:
  - To create a single horizontal or vertical strap, specify the distance from the x- or y-axis.
  - To create a multiple straps in a single direction, specify an iterator list consisting of the start distance to the first strap, the end distance to the last strap, and the incremental distance between straps.
  - To create a multiple straps in both the horizontal and vertical directions, specify an iterator list for each direction, starting with the horizontal direction.

Use the `-grid` option to create multiple straps for both the clock mesh and clock spine structures.

- To specify a margin within which the tool can move a strap from the position it derives based on the `-grid` option settings, use the `-margins` option.

If the tool cannot create a strap within the specified margin, due to obstructions or keepouts, it does not create the strap. By default, the tool uses a margin of zero and only creates the strap if it can do so at the exact position it derives.

- To create straps on the boundary specified with the `-boundary` option, use the `-create_ends` option.
- To allow straps that are unconnected to orthogonal straps, use the `-allow_floating_true` option.

By default, the tool does not allow orthogonal straps that are unconnected.

- To specify the type of straps to create, use the `-type` option and specify `user_route`, `stripe`, or `detect`.

When you create straps in both direction, specify a list of two values starting with the type for the horizontal direction.

You can specify `detect` as the type only when you are creating a clock spine structure and want the tool to detect and use existing spines. When you do so, you must specify a list consisting of the type for the orthogonal stripes and `detect`, with the type for the horizontal direction specified first in the list. In addition, when the tool detects existing spine, you can specify a minimum length for the spine, by using the `-detect_length` option.

- To specify the maximum length of the orthogonal stripes, when creating a clock spine structure, use the `-length` option.
- To specify the backoff distance between stripes of different spines, when creating a clock spine structure, use the `-backoff` option.

- To specify the direction of the spines, when creating a clock spine structure and the tool is not detecting and using existing spines, use the `-spine_direction` option.
- To shield the straps with power and ground nets, use the `-bias` option.
- To shield the straps with specific nets, use the `-bias_to_nets` with the list of nets.
- To specify the distance from the shielding nets, use the `-bias_margins` option.
- To remove the straps of a specific clock net, use the `-clear` option.

For example, the following command creates a clock mesh for the net named `clk1_mesh` that is bounded by coordinates (0,0) and (1200, 980). The straps are on layers M7 and M8 with a width of 2.4 units and of type stripe. The horizontal straps on layer M7 start at a distance of 20 units from the x-axis and repeat every 100 units, until they reach 1200 units from the x-axis. The vertical straps on layer M8 start at a distance of 60 units from the y-axis and repeat every 150 units, until they reach 980 units from the x-axis.

```
fc_shell> create_clock_straps -nets [get_nets clk1_mesh] \
 -layers {M7 M8} -widths {2.4 2.4} -types {stripe stripe} \
 -grids {{20 1200 100} {60 980 150}} -boundary {{0 0} {1200 980}}
```

The following example creates a two-dimensional spine structure that has spines on the vertical layer M8 with a width of 3.6 units and of type `user_route` and stripes on the horizontal layer M7 with a width of 2.4 units and of type `stripe`. The vertical spines on layer M8 start at a distance of 60 units from the y-axis and repeat every 150 units, until they reach 980 units from the x-axis. The horizontal stripes on layer M7 have a length of 120 units and they start at a distance of 20 units from the x-axis and repeat every 100 units, until they reach 1200 units from the x-axis. The minimum distance (backoff) between stripes of different spines is 5 units.

```
fc_shell> create_clock_straps -nets [get_nets clk1_mesh] \
 -layers {M7 M8} -widths {2.4 3.6} -types {stripe user_route} \
 -grids {{20 1200 100} {60 980 150}} -length 120 -backoff 5
```

## Routing to Clock Straps

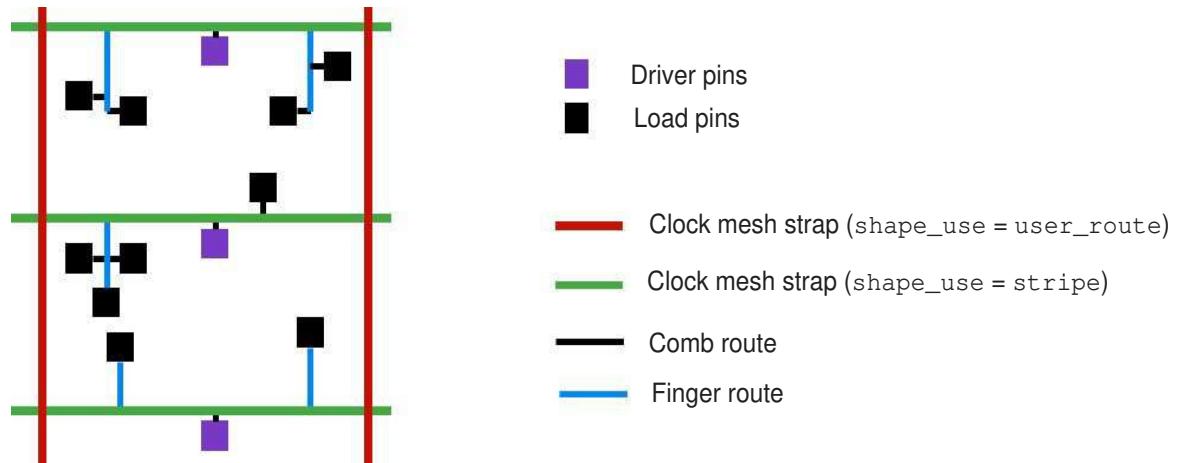
After you create clock straps for a clock net by using the `create_clock_straps` command, you can route the drivers and loads of the clock net to the clock straps by using the `route_clock_straps` command and specify the clock net name by using the `-nets` option. The tool connects the drivers and the loads only to clock straps with the `shape_use` attribute setting of `stripe`. It does not connect to clock straps with the `shape_use` attribute setting of `user_route`.

To specify a topology for connecting the clock drivers and sinks to the clock straps, use the `-topology` option as follows:

- To create a fishbone routing topology, the default topology, use the `-topology fishbone` option.

In a fishbone topology, each driver pin is individually connected to the nearest stripe. To minimize wire length, multiple load pins are connected using comb routing to a single finger, which is connected to the stripe, as shown in [Figure 76](#).

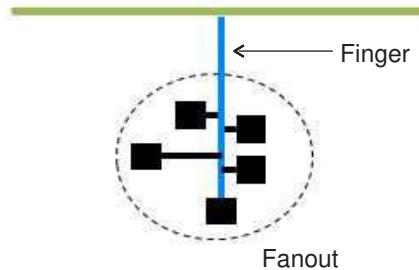
*Figure 76     Fishbone Topology*



If you are using a fishbone topology, you can specify

- A maximum fanout for the loads of a finger, as shown in [Figure 77](#), by using `-fishbone_fanout` option.

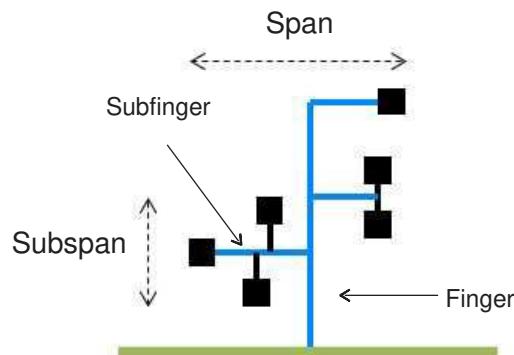
*Figure 77 Fanout of a Finger of the Fishbone Topology*



- A maximum span between any two loads connected to a finger, as shown in [Figure 78](#), by using `-fishbone_span` option.

The span is measured orthogonal to the direction of the finger.

*Figure 78 Span and Subspan of the Fishbone Topology*



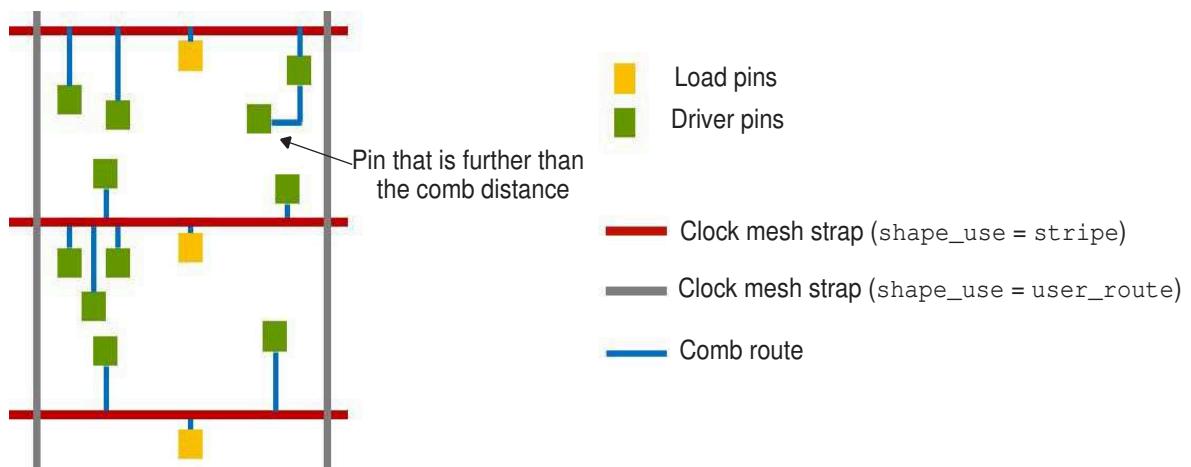
- A maximum subspan between any two loads connected to a subfinger, as shown in [Figure 78](#), by using `-fishbone_sub_span` option.

The subspan is measured orthogonal to the direction of the subfinger.

- The layers to use for routing the fingers and subfingers by using the `-fishbone_layers` option.
- To create a comb routing topology, use the `-topology comb` option.

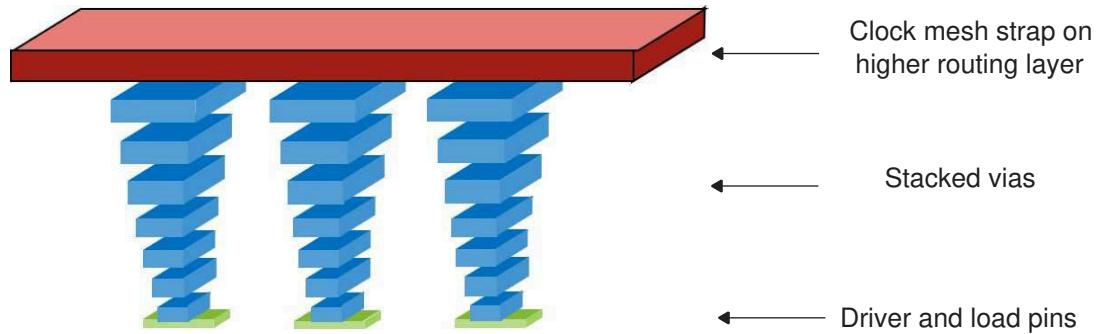
In a comb topology, each driver and load pin is directly routed to the nearest stripe. However, if the Manhattan distance from a pin to the nearest stripe is more than the comb distance, the tool routes the pin to the nearest net shape using a Steiner topology, as shown in [Figure 79](#). The default comb distance is two global routing cells. To change the comb distance, use the `route.common.comb_distance` application option.

*Figure 79 Comb Topology*



Comb routing is suitable when there are a large number of driver and load pins directly under the clock-mesh stripes. The tool uses stacked vias to connect the pins to the stripes, which are usually in the higher routing layers. However, this can contribute to physical DRC violations due to many adjacent stacked vias, as shown in [Figure 80](#).

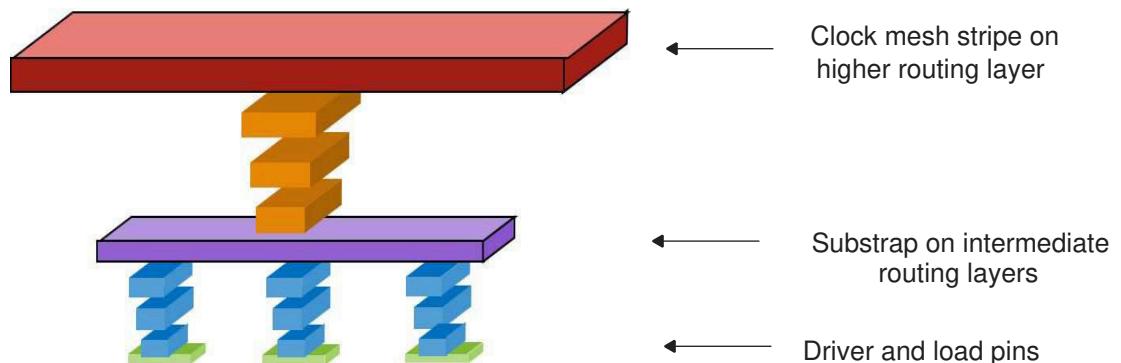
*Figure 80 Vias in the Comb Topology*



- To create a substrap routing topology, use the `-topology sub_strap` option.

In a substrap topology, additional straps that are parallel to the stripes are created on intermediate routing layers. This reduces the number of stacked vias, as compared to the comb topology.

*Figure 81 Vias in the Substrap Topology*



For the substrap topology, you

- Must specify the layers in which to create the substrap by using the `-sub_strap_layers` option
  - Can specify the maximum RC delay allowed for the substrap by using the `-sub_strap_max_delay` option
- The default is 2 ps. This delay constrains the length and the number of loads connected to the substrap.
- 

## Analyzing the Clock Mesh

To reduce skew variation, clock mesh structures require higher timing accuracy than traditional clock structures. Therefore, to analyze a clock mesh structures, use the `analyze_subcircuit` command, which performs transistor level circuit simulation for the clock mesh and back-annotates accurate timing information.

Before you run the `analyze_subcircuit` command, you must

- Detail route the clock mesh net.
- Have a circuit-level model for each of the gates in your clock tree and a transistor model for each of the transistors in the circuit-level models.
- Have access to a SPICE simulator such, as NanoSim, FineSim, or HSPICE.

For the `analyze_subcircuit` command, you must specify

- The clock mesh net you want to simulate by using the `-net` option.

Alternatively, you can specify the sinks of the mesh net by using the `-to` option.

If multiple clocks reach the net or the sinks you specify, use the `-clock` option to distinguish the clock you want to analyze.

- A name by using the `-name` option.

This name is used to construct the circuit elements. It is also used in the name of the output files and the directory where the output is stored.

Optionally you can specify an annotated transition on the clock port or the start point of your clock mesh. If you do so, the clock must be specified as propagated by using the `set_propagated_clock` command.

When you run the `analyze_subcircuit` command, the tool performs the following steps:

1. Performs RC extraction and generates parasitic files.

You can run extraction as a standalone step by using the `-extraction` option.

2. Generates SPICE files for simulating the clock mesh, using the parasitic files from the previous step as input.

You can generate the SPICE files as a standalone step by using the `-create_spice_deck` option. When you do so, you can use parasitic files generated by a different extraction tool. If these parasitic files have a different naming convention, you can specify the appropriate file suffix by using the `-spf_input_file_suffix` and `-rc_include_file_suffix` options.

3. Runs SPICE simulation, using the SPICE files generated in the previous step as input.

By default, the tool uses the NanoSim simulator. You can use the FineSim or HSPICE simulators by using the `finesim` or `hspice` setting with the `-simulator` option.

You must specify the location of the circuit-level and transistor-level models by using the `-driver_subckt_files` and `-spice_header_files` options. You customize these settings by specifying different files for the maximum and minimum conditions within each scenario by using the `-configuration` option.

You can run simulation as a standalone step by using the `-run_simulation` option.

4. Generates timing annotation files containing `set_disable_timing`, `set_annotated_delay`, and `set_annotated_transition` commands, using the simulation results as input.

For clock mesh nets, which have multiple drivers, the `set_disable_timing` command is used to disable all except one of the drivers, which is called the anchor driver. The annotated net delay arcs are defined from the anchor driver.

You can generate the timing annotation files as a standalone step by using the `-write_annotation` option.

5. Applies the annotation files generated in the previous step.

You can apply the annotation files as a standalone step by using the `-apply_annotation` option.

If you want to run some of the steps of the mesh analysis flow, such as extraction or SPICE simulation, using other tools, you can do so and run the rest of the steps of the flow using the standalone options of the `analyze_subcircuit` command, following the same sequence.

The following example analyzes the clock mesh net named `clk_mesh` using the HSPICE simulator. It customizes the simulation by using different files for the minimum and maximum conditions of each scenario.

```
fc_shell> analyze_subcircuit -net clk_mesh \
 -driver_subckt_files max_spice_model \
 -spice_header_files header_file \
 -configuration { \
```

```

{-scenario_name scenario1 \
 -max_driver_subckt_files max_file1 \
 -max_spice_header_files header_max1 \
 -min_driver_subckt_files min_file1 \
 -min_spice_header_files header_min1} \
{-scenario_name scenario2 \
 -max_driver_subckt_files max_file2 \
 -max_spice_header_files header_max2 \
 -min_driver_subckt_files min_file2 \
 -min_spice_header_files header_min2}}
-simulator hspice \
-name clk_mesh_analysis

```

## Performing Automated Tap Insertion and H-Tree Synthesis

To perform this task, you must

1. Specify the required settings by using the

`set_regular_multisource_clock_tree_options` command as follows:

- The clock name by using the `-clock` option
- The topology for the global clock tree by using the `-topology htree_only` option
- The number of rows and columns of tap cells to insert by using the `-tap_boxes` option
- A list of library cells to use as tap drivers by using the `-tap_lib_cells` option

The library cells you specify must be enabled for clock tree synthesis by using the `set_lib_cell_purpose -include cts` command.

- A list of library cells to use for synthesizing the global clock tree (H-tree) by using the `-htree_lib_cells` option

The library cells you specify must be enabled for clock tree synthesis by using the `set_lib_cell_purpose -include cts` command.

By default, the tool uses

- The net connected to the clock root to insert the tap drivers and build the clock structure
- You can specify a different net of the clock network by using the `-net` option.
- The locations of the sinks to determine the boundary within which to insert the tap cells

You can control this boundary by using the `-tap_boundary` option. You can also specify keepout areas within which tap cells should not be inserted by using the `-keepouts` option.

- The layers and routing rules specified by the clock tree synthesis settings to route the H-tree

You can specify the layers and routing rules for the H-tree by using the `-htree_layers` and `-htree_routing_rule` options.

You can report the options you specified by using the `report_regular_multisource_clock_tree_options` command, and remove them by using the `remove_regular_multisource_clock_tree_options` command.

2. (Optional) Enable flexible H-tree synthesis by setting the `cts.multisource.flexible_htree_synthesis` application option to `true`.

When you enable this feature, the tool identifies the best count, configuration, and placement of tap drivers for minimum clock tree insertion delay.

To specify that the tool derives a symmetric configuration of tap drivers, set the `cts.multisource.tap_selection` application option to `symmetric`. The default is `user`, and by default the tool uses the configuration specified by the `-tap_boxes` option of the `set_regular_multisource_clock_options` command for the tap drivers.

3. Insert the tap drivers and build the H-tree by using the `synthesize_regular_multisource_clock_trees` command, which consists of the following two stages:

- `tap_synthesis`, during which the tool inserts the tap drivers
- `htree_synthesis`, during which the tool builds the H-tree that drives the tap drivers

By default, the tool performs both stages. You can perform only one of these stages by using the `-to` or `-from` option.

The following example script inserts tap drivers in four columns and two rows and builds the H-tree:

```
Include only tap driver and H-tree library cell for CTS
set cts_references [get_lib_cells -filter valid_purposes=~*cts*]
set lib_cell_purpose -exclude cts [get_lib_cells $cts_references]
set lib_cell_purpose -include cts { LIB1/BUF1 LIB1/BUF2}

Set up and insert tap drivers and build H-tree
set_regular_multisource_clock_tree_options \
 -clock clk -topology htree_only -tap_boxes {4 2} \
 -tap_lib_cells [get_lib_cells */CKBUF*] \
 -htree_lib_cells [get_lib_cells */CKINV*] \
```

```
-htree_layers "m9 m10" -htree_routing_rule "htree_ndr"
synthesize_regular_multisource_clock_trees

Reapply CTS library cell list as required for the subsequent steps
set_lib_cell_purpose -include cts [get_lib_cells $cts_references]
```

---

## Specifying Tap Assignment Options and Settings

Before you can perform automated tap assignment by using the `synthesize_multisource_clock_taps` command, you must perform the following steps:

1. Remove `dont_touch`, `size_only`, `fixed`, and `locked` attributes setting on the clock cells that are not required. These settings prevent the tool from merging or splitting clock cells during tap assignment, if necessary.
2. Specify settings for automated tap assignment by using the `set_multisource_clock_tap_options` command.

You must specify

- The clock for tap assignment by using the `-clock` option.  
All the sinks reachable from the given clock are considered for redistribution.
- The tap drivers among which the sinks are redistributed by using the `-driver_objects` option.
- The number of taps that the sinks are redistributed among by using the `-num_taps` option.

Currently, the number of taps specified with this option must be equal to the number of drivers specified with the `-driver_objects` option.

To prevent the tool from merging specific clock tree cells, specify the instance names using the `-dont_merge_cells` option.

To report the settings you specify for automated tap assignment, use the `report_multisource_clock_tap_options` command. To remove the settings you specify for automated tap assignment, use the `remove_multisource_clock_tap_options` command.

3. (Optional) Create multisource sink groups for tap assignment by using the `create_multisource_clock_sink_group` command.

Skew groups defined for the clock can be distributed among different taps. To retain the objects of a skew group under the same tap, create a corresponding multisource sink group.

Use the following commands to remove, report, or manipulate multisource sink groups:

- `remove_multisource_clock_sink_groups`
- `report_multisource_clock_sink_groups`
- `get_multisource_clock_sink_groups`
- `add_to_multisource_clock_sink_group`
- `remove_from_multisource_clock_sink_group`

4. (Optional) Specify name prefixes and suffixes to use when merging and splitting clock cells during tap assignment by using the following application options:

- `cts.multisource.subtree_merge_cell_name_prefix`
- `cts.multisource.subtree_merge_cell_name_suffix`
- `cts.multisource.subtree_split_cell_name_prefix`
- `cts.multisource.subtree_split_cell_name_suffix`
- `cts.multisource.subtree_split_net_name_prefix`
- `cts.multisource.subtree_split_net_name_suffix`

## Building the Local Clock Subtree Structures

In a structural multisource clock tree, the local subtrees are directly driven by the clock mesh. You can synthesize these local subtrees by using the `synthesize_multisource_clock_subtrees` command.

Before you synthesize the local subtrees, you must

1. Build the global clock distribution structure.
2. Model the clock mesh net with a realistic annotated delay and transition values using the `set_annotationed_delay` and `set_annotationed_transition` commands.
3. Remove `dont_touch`, `size_only`, `fixed`, and `locked` attributes setting on the clock cells that are not required. These settings prevent the `synthesize_multisource_clock_subtrees` command from merging or splitting clock cells, if necessary.
4. Specify settings for local subtree synthesis by using the `set_multisource_clock_subtree_options` command.

You must specify

- The clock for synthesizing the subtrees by using the `-clock` option.
- The drivers of the subtrees to be synthesized by using the `-driver_objects` option.

The clock specified with the `-clock` option should pass through each drivers specified with the `-driver_objects` option.

Optionally you can

- Prevent the tool from merging specific clock tree cells by using the `-dont_merge_cells` option.
- Balance the levels of the local subtree by using the `-balance_levels true` option and specify a target number of levels, which applies to all sinks of the local subtree, by using the `-target_level` option.

Optionally, you can apply a different target number of levels to specific sink pins and clock balance points by using the `set_multisource_clock_subtree_constraints` command with the `-pins` and `-target_level` options.

- Specify a maximum total wire delay from any subtree driver to any of its sinks by using the `-max_total_wire_delay` option and the corner it applies to by using the `-corner` option.
- Enable the reordering of clock-gating cells by using the `-enable_icg_reordering true` option.

When you enable this feature, the tool swaps the position of a clock-gating cell with the buffer that is driving the clock-gating cell. Then, the buffer is gated by the clock-gating cell, thereby reducing the dynamic power consumption.

To perform reordering, you must

- Enable scenarios used for structural multisource clock tree synthesis for dynamic power optimization
- Annotate switching activity for the enable pins of the clock-gating cells

You can prevent specific clock-gating cells from being reordered by using the `set_multisource_clock_subtree_constraints` command with the `-cells` and `-ignore_for_icg_reordering` options.

To report the settings you specified, use the `report_multisource_clock_subtree_options` command. To remove the settings you specified, use the `remove_multisource_clock_subtree_options` command.

5. (Optional) Enable multicorner optimization by setting the `cts.multisource.enable_multi_corner_support` application option to `true`. The default is `false`.

By default, the tool uses the worst corner associated with the mode in which the subtree options are defined. When you enable this feature, the tool considers all corners associated with the mode in which the subtree options are defined.

6. (Optional) Ignore the maximum capacitance, maximum fanout, or both constraints of the clock drivers by using the `cts.multisource.ignore_drc_on_subtree_driver` application option.

When you synthesize multisource clock subtrees, by default, the tool considers the maximum transition, capacitance, and fanout constraints of the clock drivers you specify. However, the clock drivers typically have relaxed or no maximum capacitance or maximum fanout constraint, which can affect the multisource clock tree QoR.

The following example ignores both the maximum capacitance and maximum fanout constraints of the clock drivers:

```
fc_shell> set_app_options \
 -name cts.multisource.ignore_drc_on_subtree_driver \
 -value "max_fanout max_capacitance"
```

7. (Optional) Enable fishbone routing for the subtree clock nets by using the `cts.multisource.subtree_routing_mode` application option.

```
fc_shell> set_app_options \
 -name cts.multisource.subtree_routing_mode -value fishbone
```

8. (Optional) Specify name prefixes and suffixes to use when merging and splitting clock cells during local subtree synthesis by using the following application options:

- `cts.multisource.subtree_merge_cell_name_prefix`
- `cts.multisource.subtree_merge_cell_name_suffix`
- `cts.multisource.subtree_split_cell_name_prefix`
- `cts.multisource.subtree_split_cell_name_suffix`
- `cts.multisource.subtree_split_net_name_prefix`
- `cts.multisource.subtree_split_net_name_suffix`

When you run the `synthesize_multisource_clock_subtrees` command the tool performs the following steps:

1. Merges equivalent clock cells to remove any artificial boundaries between clusters of sinks.
2. Optimizes the subtree by splitting, sizing, and relocating the clock cells, and snaps the first level of clock cells to the clock straps.
3. Routes the clock nets.

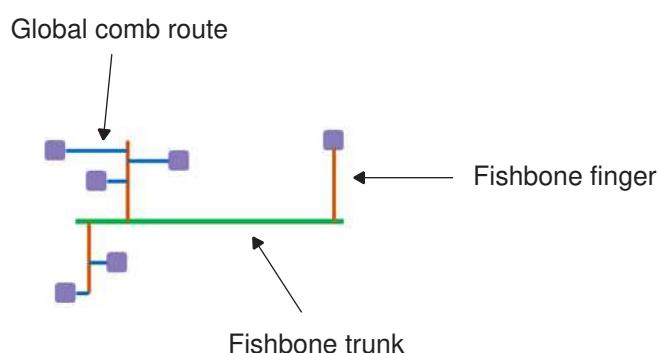
By default, it global routes the nets. However, if you enable fishbone routing by using the `cts.multisource.subtree_routing_mode` application option, the tool creates a mix of detail routed fishbone trunks and fingers, and global comb routes to connect the net loads to the fishbone fingers, as shown in [Figure 82](#).

You can control the fishbone routing by using the following application options:

- `cts.routing.fishbone_max_tie_distance`
- `cts.routing.fishbone_max_sub_tie_distance`
- `cts.routing.fishbone_bias_threshold`
- `cts.routing.fishbone_bias_window`
- `cts.routing.fishbone_horizontal_bias_spacing`
- `cts.routing.fishbone_vertical_bias_spacing`

4. Refines the clock trees by sizing and relocating cells based on the routing.

*Figure 82 Detail Fishbone Routes and Global Comb Routes*



You can run specific steps of the `synthesize_multisource_clock_subtrees` command by using the `-from` and `-to` options and specifying `merge`, `optimize`, `route_clock`, or `refine`.

Structural subtrees that you build by using the `synthesize_multisource_clock_subtrees` command are not changed or removed when you subsequently run any clock tree synthesis command.

## Analyzing the Clock Tree Results

After synthesizing the clock trees, analyze the results to verify that they meet your requirements. Typically the analysis process consists of the following tasks:

- Analyzing the clock tree QoR (as described in [Generating Clock Tree QoR Reports](#))
- Analyzing the clock tree timing (as described in [Analyzing Clock Timing](#))
- Verifying the placement of the clock instances, using the GUI (as described in [Analyzing Clock Trees in the GUI](#))

## Generating Clock Tree QoR Reports

To generate clock tree QoR reports, use the `report_clock_qor` command.

The `report_clock_qor` command can generate the following types of reports:

- Summary

By default, this command reports a summary of the clock tree QoR, which includes the latency, skew, DRC violations, area, and buffer count.

- Latency

To report the longest and shortest path for each clock, use the `-type latency` option.

- DRC violators

To report the maximum transition and capacitance constraint violators, use the `-type drc_violators` option.

- Robustness

To report the robustness of each sink, use the `-type robustness` option. The robustness of a sink is the ratio between its latency for the corner for which the report is being generated and its latency for the corner specified by the `-robustness_corner` option. The mode used for both latency values is the mode for which the report is being generated. When you use the `-type robustness` option, you must specify a corresponding robustness corner by using the `-robustness_corner` option.

- Clock-balance groups QoR

To report a clock QoR summary for each clock-balance group, use the `-type balance_groups` option. It also reports the clock latency offset values specified with

the `-offset_latencies` option of the `create_clock_balance_group` command and the actual latency offset values.

- Local skew

To report the QoR summary and the worst local skew for each clock or skew group, use the `-type local_skew` option. It also reports the five largest and five smallest local skew values and the corresponding endpoints.

- Clock tree power

To report a summary of the leakage, internal, sink, net switching, dynamic, and total power per clock per scenario, use the `-type power` option. If the `power.clock_network_include_clock_sink_pin_power` application option is set to `off`, the sink power is not reported.

When you use the `-type` option, you can generate the output as a comma separated values (CSV) file by using the `-csv` option and specifying the output file name using the `-output` option.

The `report_clock_qor` command can also generate the following types of histograms:

- Latency histogram

To report the latency of each sink in a histogram format, use the `-histogram_type latency` option.

- Transition histogram

To report the transition time of each sink in a histogram format, use the `-histogram_type transition` option.

- Capacitance histogram

To report the capacitance of each sink in a histogram format, use the `-histogram_type capacitance` option.

- Local skew histogram

To report the local skew in a histogram, use the `-histogram_type local_skew` option.

- Robustness histogram

To report the robustness of each sink, with respect to a robustness corner specified by using the `-robustness_corner` option, in a histogram format, use the `-histogram_type robustness` option.

- Wire delay fraction histogram

To report the ratio between the wire delay and the total delay for each logic stage of the clock tree in a histogram format, use the `-histogram_type wire_delay_fraction` option.

By default, the tool generates a report for all clock trees in all active modes and corners in all active scenarios. You can limit the report to specific

- Portions of a clock network by using the `-from`, `-to`, and `-through` options
- Clock trees by using the `-clock` option
- Skew groups by using the `-skew_group` option
- Modes by using the `-mode` option
- Corners by using the `-corner` option
- Scenarios by using the `-scenario` option

If you use the `-scenario` option, you cannot use the `-mode` and `-corner` options.

## Reporting Clock Tree Power

To report the power of a clock tree, use the `report_clock_power` command. By default, it reports the power for all clock trees in all modes and corners of all active scenarios. To report the power

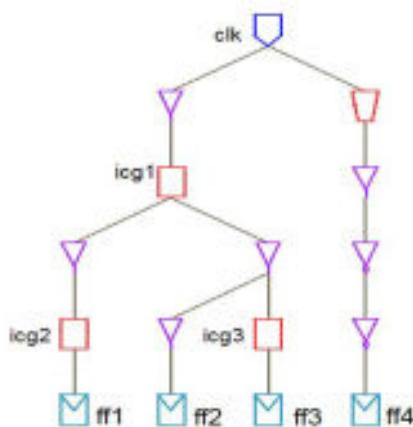
- Only for specific clocks, use the `-clocks` option
- Only for specific modes, use the `-modes` option
- Only for specific corners, use the `-corners` option
- Only for specific scenarios, use the `-scenario` option
- On a per-segment or per-subtree basis, use the `-type per_segment` or `-type per_subtree` option
  - A segment is the clock buffer tree from one integrated-clock-gating (ICG) cell to the next ICG cells or sinks in its fanout

For example, in the following figure, the buffer tree from the output of the ICG1 cell to the inputs of the ICG2, ICG3, and ff2 cells is considered a segment.

- A subtree is a clock tree from an ICG cell all the way to the sinks in its fanout.

For example, in the following figure, the clock tree from the output of the ICG1 cell to the inputs of the ff1, ff2, and ff3 cells is considered a subtree.

*Figure 83 Segments and Subtrees of a Clock Tree*



## Creating Collections of Clock Network Pins

You can create a collection of the pins on a clock network by using the `get_clock_tree_pins` command. By default, this command returns all the pins on all the clock networks of all scenarios.

You can limit the selection by using one of the following methods

- Limit it to specific clock by using the `-clocks` option.
- Limit it to specific scenario, scenarios of modes, or scenarios of corners by using the `-scenarios`, `-modes`, or `-corners` option.

You can further limit the selection by

- Considering only the clock paths that go from, through, and to specific pins by using the `-from`, `-through`, and `-to` options.
- Filtering the pins based on their attributes by using the `-filter` option.

For more information about all the available options and all the supported pin attributes, see the man page for the `get_clock_tree_pins` command.

The following example creates a collection named clk1\_icg\_pins, which consists of the pins of integrated-clock-gating cells that are on the clock network of the clock named clk1:

```
fc_shell> set clk1_icg_pin \
 [get_clock_tree_pins -filter is_on_ICG -clocks clk1]
```

---

## Analyzing Clock Timing

The timing characteristics of the clock network are important in any high-performance design. To obtain detailed information about the clock networks in the current block, use the `report_clock_timing` command.

You must use the `-type` option to specify the type of report to generate. The `report_clock_timing` command can generate the following types of reports:

- Single-clock local skew
  - To generate a single-clock local skew report, use the `-type skew` option.
- Interclock skew
  - To generate an interclock skew report, use the `-type interclock_skew` option.
- Latency
  - To generate a latency report, use the `-type latency` option.
- Transition
  - To generate a transition time report, use the `-type transition` option.

---

## Analyzing Clock Trees in the GUI

The Fusion Compiler GUI provides the Clock Tree Visual Mode to help you visualize and analyze the clock trees in your design. In this mode, you can overlay the clock tree information to the layout view, or view the clock tree structure in the schematic view.

For more information about analyzing clock tree information in the GUI, see the *Using Map and Visual Modes* topic in the *Fusion Compiler Graphical User Interface User Guide*.

# 6

## Routing and Postroute Optimization

---

This topic describes the routing capabilities of Zroute, which is the router for the Fusion Compiler tool. Zroute is architected for multicore hardware and efficiently handles advanced design rules for 45 nm and below technologies and design-for-manufacturing (DFM) tasks. It also describes the postroute optimization features supported by the Fusion Compiler tool.

To learn about routing and postroute optimization, see the following topics:

- [Introduction to Zroute](#)
- [Basic Zroute Flow](#)
- [Prerequisites for Routing](#)
- [Defining Vias](#)
- [Inserting Via Ladders](#)
- [Checking Routability](#)
- [Routing Constraints](#)
- [Routing Application Options](#)
- [Routing Clock Nets](#)
- [Routing Critical Nets](#)
- [Routing Secondary Power and Ground Pins](#)
- [Routing Signal Nets](#)
- [Shielding Nets](#)
- [Performing Postroute Optimization](#)
- [Analyzing and Fixing Signal Electromigration Violations](#)
- [Performing ECO Routing](#)
- [Routing Nets in the GUI](#)
- [Cleaning Up Routed Nets](#)

- [Analyzing the Routing Results](#)
  - [Saving Route Information](#)
  - [Deriving Mask Colors](#)
  - [Inserting and Removing Cut Metal Shapes](#)
- 

## Introduction to Zroute

Zroute has five routing engines: global routing, track assignment, detail routing, ECO routing, and routing verification. You can invoke global routing, track assignment, and detail routing by using task-specific commands or by using an automatic routing command. You invoke ECO routing and route verification by using task-specific commands.

Zroute includes the following main features:

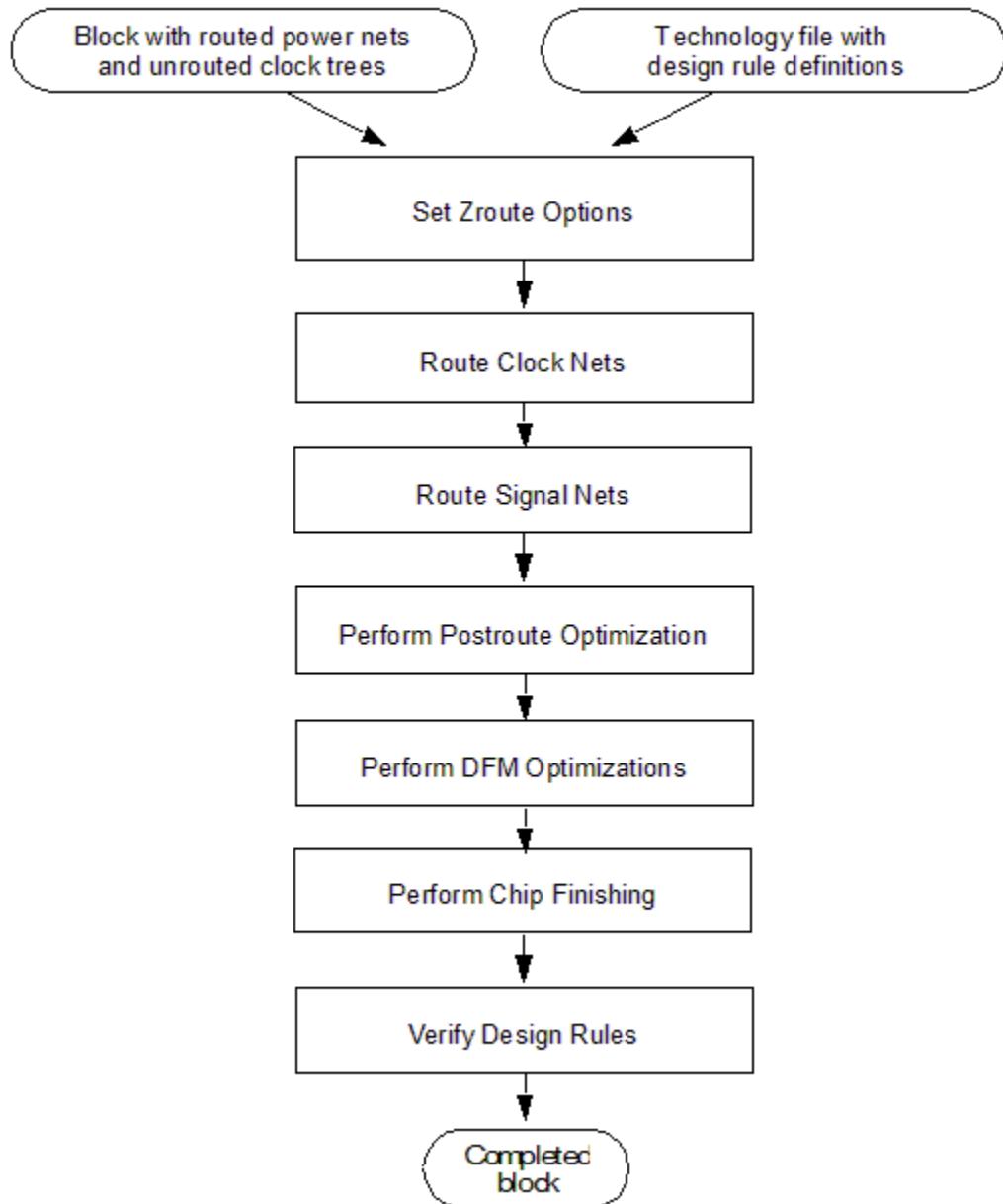
- Multithreading on multicore hardware for all routing steps, including global routing, track assignment, and detail routing
- A realistic connectivity model where Zroute recognizes electrical connectivity if the rectangles touch; it does not require the center lines of wires to connect
- A dynamic maze grid that permits Zroute to go off-grid to connect pins, while retaining the speed advantages of gridded routers
- A polygon manager, which allows Zroute to recognize polygons and to understand that design rule checks (DRCs) are aimed at polygons
- Concurrent optimization of design rules, antenna rules, wire optimization, and via optimization during detail routing
- Concurrent redundant via insertion during detail routing
- Support for soft rules built into global routing, track assignment, and detail routing
- Timing- and crosstalk-driven global routing, track assignment, detail routing, and ECO routing
- Intelligent design rule handling, including merging of redundant design rule violations and intelligent convergence
- Net group routing with layer constraints and nondefault routing rules
- Clock routing
- Route verification

- Optimization for DFM and design-for-yield (DFY) using a soft rule approach
- Support for advanced design rules, such as multiple patterning

## Basic Zroute Flow

Figure 84 shows the basic Zroute flow, which includes clock routing, signal routing, DFM optimizations, and route verification.

Figure 84 Basic Zroute Flow



---

## Prerequisites for Routing

Before you can run Zroute, you must ensure that the block and physical library meet the following requirements:

- Library requirements

Zroute gets all of the design rule information from the technology file; therefore, you must ensure that all design rules are defined in the technology file before you start routing.

For more information about the technology file and defining routing design rules, see the *Synopsys Technology File and Routing Rules Reference Manual*.

- Block requirements

Before you perform routing, your block must meet the following conditions:

- Power and ground nets have been routed after design planning and before placement.

For more information, see the *Fusion Compiler Design Planning User Guide*.

- Clock tree synthesis and optimization have been performed.

For more information, see [Clock Tree Synthesis](#).

- Estimated congestion is acceptable.

- Estimated timing is acceptable (about 0 ns of slack).

- Estimated maximum capacitance and transition have no violations.

To verify that your block meets the last three prerequisites, you can check the routability of its placement as explained in [Checking Routability](#).

## Defining Vias

The router supports the following types of via definitions:

- Simple vias and simple via arrays

A simple via is a single-cut via. It is specified by a cut layer and the height and width of the rectangular shapes on its cut and metal layers. A simple via array is a multiple-cut simple via. Simple vias and simple via arrays can be used for the following purposes:

- Clock or signal routing using nondefault routing rules
  - Redundant via insertion
  - Power and ground routing with advanced via rules
- Custom vias

A custom via is a multiple-cut, odd-shaped via that is created from an arbitrary collection of Manhattan polygons. Custom vias can be used only for redundant via insertion.

Via definitions can come from the following sources:

- The `ContactCode` sections of the technology file associated with the design library

The technology file can contain definitions for simple vias, simple via arrays, and custom vias. For information about defining vias in the technology file, see the *Synopsys Technology File and Routing Rules Reference Manual*.

- Via rule `GENERATE` statements in a LEF file

The via rule `GENERATE` statements define simple via arrays. For information about defining vias in a LEF file, see [Reading Via Definitions from a LEF File](#).

- User-defined via definitions

You can define simple vias, simple via arrays, and custom vias. For information about creating user-defined via definitions, see [Creating a Via Definition](#).

If you want Zroute to use a via definition for signal routing, ensure that it has the following attribute settings:

- `is_default` attribute is `true`
- `is_excluded_for_signal_routing` attribute is `false`

In addition, Zroute uses nondefault vias that are explicitly specified in a fat via table in the technology file or in a nondefault routing rule defined by the `create_routing_rule` command.

## Reading Via Definitions from a LEF File

To read via definitions specified by via rule GENERATE statements in a LEF file, use the `read_tech_lef` command. This command supports the following LEF syntax:

```
via_rule_name GENERATE [DEFAULT]
 LAYER lower_layer_name
 ENCLOSURE lower_overhang1 lower_overhang2
 LAYER upper_layer_name
 ENCLOSURE upper_overhang1 upper_overhang2
 LAYER cut_layer_name
 RECT l1x l1y urx ury
 SPACING x_spacing BY y_spacing
```

The WIDTH and RESISTANCE statements are not supported.

## Creating a Via Definition

To create a via definition, use the `create_via_def` command. After it is created, the via definition can be used anywhere in the design, similar to a `ContactCode` definition in the technology file. The user-specified via definition is stored in the design library, so it can be used only in that design.

The `create_via_def` command can define simple vias, simple via arrays, and custom vias. The following topics describe how to define these types of vias.

- [Defining Simple Vias](#)
- [Defining Custom Vias](#)

### Defining Simple Vias

To define a simple via or via array, use the following `create_via_def` syntax:

```
create_via_def
 -cut_layer layer
 -cut_size {width height}
 -upper_enclosure {width height}
 -lower_enclosure {width height}
 [-min_rows number_of_rows]
 [-min_columns number_of_columns]
 [-min_cut_spacing distance]
 [-cut_pattern cut_pattern]
 [-is_default]
 [-force]
 via_def_name
```

You do not need to specify the enclosure layers; the tool determines them from the technology file by getting the metal layers adjacent to the specified via layer.

By default, when you define a simple via, it is a single-cut via. To define a via array, specify the minimum number of rows and columns for the array (`-min_rows` and `-min_columns` options), as well as the minimum cut spacing (`-min_cut_spacing` option). By default, the cut pattern is a full array of cuts. To modify the cut pattern, use the `-cut_pattern` option.

To overwrite an existing via definition, use the `-force` option; otherwise, the command fails if the specified via definition already exists.

For example, to create a single-cut via definition named `design_via1_HV` for the VIA12 via layer, use the following command:

```
fc_shell> create_via_def design_via1_HV \
 -cut_layer VIA12 -cut_size {0.05 0.05} \
 -lower_enclosure {0.02 0.0} -upper_enclosure {0.0 0.2} \
```

To create a via definition with an alternating 2x2 cut pattern in which the lower-left cut is omitted, use the following command:

```
fc_shell> create_via_def design_via12 -cut_pattern "01 10"
```

To report information about the user-defined via definitions, use the `report_via_defs` command.

## Defining Custom Vias

To define a custom via, use the following `create_via_def` syntax:

```
create_via_def
 -shapes { {layer {coordinates} [mask_constraint]} ... }
 [-lower_mask_pattern alternating | uniform]
 [-upper_mask_pattern alternating | uniform]
 [-force]
 via_def_name
```

In the `-shapes` option, you must specify shapes for one via layer and two metal layers, which are the enclosure layers for the via. You can specify multiple shapes per layer.

For example, to create a custom via definition, use a command similar to the following:

```
fc_shell> create_via_def design_H_shape_via \
 -shapes { {VIA12 {0.035 -0.035} {0.100 0.035}}
 {VIA12 {-0.100 -0.035} {-0.035 0.035}}
 {METAL1 {-0.130 -0.035} {0.130 0.035}}
 {METAL2 {-0.100 -0.065} {-0.035 0.065}}
 {METAL2 {0.035 -0.065} {0.100 0.065}}
 {METAL2 {-0.100 -0.035} {0.100 0.035}} }
```

To overwrite an existing via definition, use the `-force` option; otherwise, the command fails if the specified via definition already exists.

If you are using double-patterning technology, you can assign mask constraints to the shapes or the enclosure layers, but not both.

- To assign mask constraints to the shapes, use the `mask_one`, `mask_two`, or `mask_three` keywords for the `mask_constraint` arguments when specifying the `-shapes` option. You would use this method if the mask constraints follow an arbitrary pattern. When you create a via with the `create_via` command, the via shapes inherit the mask constraints specified in the via definition. If the mask constraint you specify for an enclosure layer when you create a via differs from the mask constraint specified for the first shape in the via definition, the specified mask constraint determines the mask-shift used for all shapes on that layer.

For example,

```
fc_shell> create_via_def VIA12 \
 -shapes { {M1 {-0.037 -0.010} {-0.017 0.01} mask_two}
 {M1 {0.017 -0.010} {0.037 0.01} mask_one}
 {VIA1 {-0.037 -0.01} {0.037 0.01} mask_two}
 {M2 {-0.044 -0.010} {0.044 0.010} mask_one} }
```

```
fc_shell> create_via -net n1 -via_def VIA12 \
 -origin {100.100 200.320}
```

- To assign mask constraints to the enclosure layers, use the `-lower_mask_pattern` and `-upper_mask_pattern` options. You can specify either `uniform` or `alternating` as the mask pattern. In either case, you specify the mask constraint for the first shape when you create a via with the `create_via` command. If you specify `uniform`, all shapes on the layer use the specified mask constraint. If you specify `alternating`, the colors alternate from shape to shape after the first shape.

For example,

```
fc_shell> create_via_def VIA12 \
 -shapes { {M1 {-0.010 -0.030} {0.027 0.050}}
 {M1 {0.047 -0.030} {0.084 0.050}}
 {VIA1 {0.000 0.000} {0.074 0.020}}
 {M2 {-0.030 0.000} {0.104 0.020}} } \
 -lower_mask_pattern uniform
fc_shell> create_via -via_def VIA12 -origin {000.100 200.320}
 -lower_mask_constraint mask_two
```

To report information about the via definitions, use the `report_via_defs` command. This command reports the shapes that comprise the via definition but does not report the upper and lower mask patterns.

## Inserting Via Ladders

A *via ladder*, which is also referred to as a *via pillar*, is a stacked via that starts from the pin layer and extends into an upper layer where the router connects to it. Via

ladders reduce the via resistance, which can improve performance and electromigration robustness.

To use via ladders,

- Via ladder rules must be defined in the technology data for the design

For information about defining via ladder rules, see [Defining Via Ladder Rules](#).

- You must insert the via ladders before you perform global routing

The Fusion Compiler tool provides the following methods for inserting via ladders:

- Automatic insertion during preroute optimization, as described in [Specifying Automatic Via Ladder Insertion Settings for Preroute Optimization](#)
- Constraint-based insertion by using the `insert_via_ladders` command, as described in [Constraint-Based Via Ladder Insertion](#)
- Manual insertion by using the `create_via_ladder` command, as described in [Manual Via Ladder Insertion](#)

## See Also

- [Querying Via Ladders](#)
- [Removing Via Ladders](#)
- [Controlling Via Ladder Connections](#)

## Defining Via Ladder Rules

A via ladder rule defines the number of rows and the number of cuts in each row for each layer in a via ladder. Via ladder rules can be defined in the technology file or by using the `create_via_rule` command.

For example, each of the following methods creates an identical via ladder rule:

- Define the rule in the technology file.

```
ViaRule "VL1" {
 cutLayerNameTblSize = 2
 cutLayerNameTbl = (VIA1, VIA2) # via layer names
 cutNameTbl = (CUT1A, CUT2A) # cut names
 numCutRowsTbl = (2, 2) # rows in ladder
 numCutsPerRowTbl = (2, 2) # columns in ladder
 upperMetalMinLengthTbl = (L1, L2) # minimum length rule
 cutXMinSpacingTbl = (X1, X2) # X-direction spacing
 cutYMinSpacingTbl = (Y1, Y2) # Y-direction spacing
 maxNumStaggerTracksTbl = (S1, S2) # maximum staggering
 forHighPerformance = 0
```

```
 forElectromigration = 1
}
```

For details about defining via ladder rules in the technology file, see the "Via Ladder Rule" topic in the *Synopsys Technology File and Routing Rules Reference Manual*.

- Define the rule on the command line by using the `create_via_rule` command with a detailed specification and then specifying additional details by setting its attributes.

```
fc_shell> create_via_rule -name VL1 -cut_layer_names {VIA1 VIA2} \
 -cut_names {CUT1A CUT2A} -cut_rows {2 2} -cuts_per_row {2 2}
fc_shell> set_attribute [get_via_rules VL1] \
 upper_metal_min_length_table {L1 L2}
fc_shell> set_attribute [get_via_rules VL1] \
 cut_x_min_spacing_table {X1 X2}
fc_shell> set_attribute [get_via_rules VL1] \
 cut_y_min_spacing_table {Y1 Y2}
fc_shell> set_attribute [get_via_rules VL1] \
 max_num_stagger_tracks_table {S1 S2}
fc_shell> set_attribute [get_via_rules VL1] \
 for_high_performance false
fc_shell> set_attribute [get_via_rules VL1] \
 for_electro_migration true
```

- Define the rule on the command line by creating an empty via rule with the `create_via_rule` command and then specifying the details by setting its attributes.

```
fc_shell> create_via_rule -name VL1
fc_shell> set_attribute [get_via_rules VL1] \
 cut_layer_name_table_size 2
fc_shell> set_attribute [get_via_rules VL1] \
 cut_layer_name_table {via1 via2}
fc_shell> set_attribute [get_via_rules VL1] \
 cut_name_table {CUT1A CUT2A}
fc_shell> set_attribute [get_via_rules VL1] \
 num_cuts_per_row_table {2 2}
fc_shell> set_attribute [get_via_rules VL1] num_cut_rows_table {2 2}
fc_shell> set_attribute [get_via_rules VL1] \
 upper_metal_min_length_table {L1 L2}
fc_shell> set_attribute [get_via_rules VL1] \
 cut_x_min_spacing_table {X1 X2}
fc_shell> set_attribute [get_via_rules VL1] \
 cut_y_min_spacing_table {Y1 Y2}
fc_shell> set_attribute [get_via_rules VL1] \
 max_num_stagger_tracks_table {S1 S2}
fc_shell> set_attribute [get_via_rules VL1] \
 for_high_performance false
fc_shell> set_attribute [get_via_rules VL1] \
 for_electro_migration true
```

The tool treats the via rules the same, regardless of the method used to create them. You can perform the following tasks for via rules:

- Create a collection of via rules by using the `get_via_rules` command
- Remove via rules by using the `remove_via_rules` command
- Display detailed information about the via rules by using the `report_via_rules` command
- Set or query via rule attributes

To see the attributes supported on via rule objects, use the `list_attributes -application -class via_rule` command.

- Save the via rules in a technology file by using the `write_tech_file` command

#### See Also

- [Generating Via Ladder Rules for Electromigration Via Ladders](#)

## Generating Via Ladder Rules for Electromigration Via Ladders

Instead of explicitly defining via ladder rules for electromigration via ladders, you can use the `generate_via_ladder_template` command to generate via ladder rules and constraints based on a configuration file. The configuration file is an XML file that provides information about the via ladder rules, as well as the association between the via ladder rules and specific pins. You must specify the name of the configuration file by using the `-config_file` option. The command generates two script files:

- A script file that defines the via ladder rules, which is called the template script file

This script file contains a `create_via_rule` command for each template defined in the configuration file, as well as `set_attribute` commands to set the appropriate via rule attributes.

By default, the file is named `auto_gen_via_ladder_template.tcl`. To specify a different file name, use the `-template_file` option.

- A script file that defines the via ladder constraints, which is called the association script file

This script file contains `set_via_ladder_candidate` commands for the pins specified in the configuration file, as well as `set_attribute` commands to set the `is_em_via_ladder_required` attribute for the pins to true.

By default, the file is named `auto_gen_via_ladder_association.tcl`. To specify a different file name, use the `-association_file` option.

The configuration file has the following syntax:

```
<EmRule>
 <!-- template definition -->
 <Template name="rule_name">
 <Layer name="layer_name" row_number="cuts_per_row"
 [max_stagger_tracks="count"]
 [upper_cut_x_min_spacing="x_spacing"]
 [upper_cut_y_min_spacing="y_spacing"] />
 ... more layers
 </Template>
 ... more templates

 <!-- Pin to template association -->
 <Pin name="pin_name">
 <Template name="rule_name"/>
 </Pin>
 ... more associations
</EmRule>
```

Each `Template` section specifies the template for a via ladder rule. You can specify one or more `Template` sections. Within a template section, you specify the via ladder structure by specifying the number of cuts per row for each metal layer involved in the via ladder. The command uses the information in the configuration file and associated information from the technology file to generate the via ladder rules. The command derives the following information from the technology file

- The cut layers that connect the specified metal layers
- The cut names
- The minimum required length for each metal layer

**Note:**

You must open the block before running the `generate_via_ladder_template` command to ensure that the command has access to the technology data for the block.

Each `Pin` section specifies the template associated with a pin, where the pin name can include the asterisk wildcard character (\*). You can specify one or more `Pin` sections. The template specified in the `Pin` section's `Template` attribute must be one of the templates specified in the `Template` sections.

For example, assume you have a configuration file named `vl_config` that has the following contents:

```
<EmRule>
 <Template name="template_1_3231">
 <Layer name="M1" row_number="3"/>
 <Layer name="M2" row_number="2"/>
 <Layer name="M3" row_number="3"/>
```

```

<Layer name="M4" row_number="1"/>
</Template>
<Pin name="*/BCELLD5A11*/Z">
 <Template name="template_1_3231"/>
</Pin>
</EmRule>
```

To generate the script files to define the via ladder rules and constraints, use the following command:

```
fc_shell> generate_via_ladder_template -config_file vl_config
```

[Example 16](#) shows the template script file generated by this command. [Example 17](#) shows the association script file generated by this command.

#### *Example 16 Template Script File*

```

create_via_rule -name template_1_3231 \
 -cut_layer_names {VIA1 VIA2 VIA3} -cut_names {V1S V2S V3S} \
 -cut_rows {2 3 1} -cuts_per_row {3 2 3}
set viaRule [get_via_rules template_1_3231]
set_attribute $viaRule upper_metal_min_length_table {0.3 0.2 0.4}
set_attribute $viaRule for_electro_migration true
```

#### *Example 17 Association Script File*

```

foreach_in_collection pin [get_lib_pins -quiet */BCELLD5A11*/Z] {
 set_attribute -quiet $pin is_em_via_ladder_required true
 set_via_ladder_candidate $pin -ladder_name "template_1_3231"
}
```

#### See Also

- [Defining Via Ladder Rules](#)
- [Specifying Automatic Via Ladder Insertion Settings for Preroute Optimization](#)

## Generating Via Ladder Rules for Performance Via Ladders

Instead of explicitly defining via ladder rules for performance via ladders, you can use the `setup_performance_via_ladder` command to generate the via ladder rules and associations.

By default, this command

- Generates via ladder rules for all layers up to the M5 layer

To specify a different maximum layer, use the `-max_layer` attribute.

The command uses information from the technology file to generate the via ladder rules. The command derives the following information from the technology file:

- The cut layers that connect the specified metal layers
- The cut names
- The minimum required length for each metal layer

**Note:**

You must open the block before running the `setup_performance_via_ladder` command to ensure that the command has access to the technology data for the block.

The command outputs an XML file that provides information about the via ladder rules and a script file that defines the via ladder rules. For details about these files, see [Via Ladder Rule Files](#).

You can also generate these files by using the `generate_via_rules_for_performance` command.

- Associates via ladder rules with the pins of all library cells that do not have a `dont_use` attribute

To include the pins of library cells that have a `dont_use` attribute, use the `-dont_use` option.

To associate via ladder rules only with specific pins, use the `-lib_pins` option.

The command analyzes the information about each library cell pin and its shapes to determine the via ladder rules to associate with that pin and then outputs a script file that defines the associations. For details about this file, see [Via Ladder Association File](#).

You can also generate this file by using the `associate_performance_via_ladder` command.

- Runs the generated script files

**See Also**

- [Defining Via Ladder Rules](#)
- [Specifying Automatic Via Ladder Insertion Settings for Preroute Optimization](#)

## Via Ladder Rule Files

The command generates two via ladder rule files:

- An XML file that provides information about the via ladder rules

The XML file has the following syntax:

```
<EmRule>
 <!-- template definition -->
 <Template name="rule_name" for_electro_migration="true"
 for_high_performance="true">
 <Layer name="layer_name" row_number="cuts_per_row"
 ... more layers
 </Layer>
 ... more templates

</EmRule>
```

Each `Template` section specifies the template for a via ladder rule. Within a template section, the via ladder structure is specified by specifying the number of cuts per row for each metal layer involved in the via ladder.

By default, the generated XML file is named `auto_perf_via_ladder_rule.xml`. To specify a different name, use the `-xml_file` option.

- A script file that defines the via ladder rules

The script file contains a `create_via_rule` command for each template defined in the XML file, as well as `set_attribute` commands to set the appropriate via rule attributes.

By default, the generated script file is named `auto_perf_via_ladder_rule.tcl`. To specify a different file name, use the `-rule_file` option.

## Via Ladder Association File

The command generates a script file that defines the association between the library cell pins and the via ladder rules.

This script file contains `set_via_ladder_candidate` commands for the library cell pins.

By default, the generated script file is named `auto_perf_via_ladder_association.tcl`. To specify a different file name, use the `-association_file` option.

---

## Constraint-Based Via Ladder Insertion

To use the constraint-based method to insert via ladders,

1. Ensure that the via ladder rules are defined, as described in [Defining Via Ladder Rules](#).
2. Define the via ladder constraints by using the `set_via_ladder_rules` command, as described in [Defining Via Ladder Constraints](#).
3. Insert the via ladders by using the `insert_via_ladders` command, as described in [Inserting Via Ladders](#).
4. Verify the via ladders by using the `verify_via_ladders` command, as described in [Verifying Via Ladders](#).

---

## Defining Via Ladder Constraints

The via ladder constraints define the pins on which to insert via ladders and the via ladder templates that can be used for those pins. You can define both global and instance-specific via ladder constraints; if the constraints conflict, the instance-specific constraints override the global constraints.

- To define global via ladder constraints, which are referred to as via ladder rules, use the `set_via_ladder_rules` command, as described in [Defining Global Via Ladder Constraints](#).
- To define instance-specific via ladder constraints, use the `set_via_ladder_constraints` command, as described in [Defining Instance-Specific Via Ladder Constraints](#).

In addition, you can specify via ladder candidates for specific pins by using the `set_via_ladder_candidate` command. This command is used by preroute optimization to insert via ladders on timing-critical paths, as described in [Specifying Automatic Via Ladder Insertion Settings for Preroute Optimization](#).

To report the via ladder constraints, use the `report_via_ladder_constraints` command.

### See Also

- [Inserting Via Ladders](#)

## Defining Global Via Ladder Constraints

To define global via ladder constraints, which are referred to as via ladder rules, use the `set_via_ladder_rules` command to set via ladder constraints on library cell pins. You must specify the mapping between library cell pins and via ladder templates, as well as the pins to which to apply the mapping.

Use the following options to specify the mapping:

- `-master_pin_map` or `-master_pin_map_file`

These options specify the mappings for specific library pins using the following format:

```
{ {lib_cell/pin {via_ladder_list}} ... }
```

To specify the mapping on the command line, use the `-master_pin_map` option. To specify the mapping in an external file, use the `-master_pin_map_file` option to specify the mapping file name.

- `-default_ladders`

This option specifies the via ladder templates to use for all library pins not explicitly specified in the mapping.

Use the following options to specify the pins to which to apply the mapping:

- `-all_instances_of`

This option explicitly specifies library cell pins to which to apply the mapping.

- `-all_clock_outputs`

Set this option to `true` to apply the mapping to all clock output pins.

- `-all_clock_inputs`

Set this option to `true` to apply the mapping to all clock input pins.

- `-all_pins_driving`

This option applies the mapping to all pins that drive one of the specified ports.

To report the via ladder rules, use the `report_via_ladder_rules` command. To remove via ladder rules, use the `remove_via_ladder_rules` command.

## Defining Instance-Specific Via Ladder Constraints

To define instance-specific via ladder constraints, use the `set_via_ladder_constraints` command. You must specify the instance pins and the via ladder templates that can be used for those pins.

For example, to enable the use of the VL1, VL2, and VL3 via ladder templates for the u1/i1 and u2/i2 pins, use the following command:

```
fc_shell> set_via_ladder_constraints -pins {u1/i1 u2/i2} \
 {VL1 VL2 VL3}
```

To report the via ladder constraints, use the `report_via_ladder_constraints` command. To remove via ladder constraints, use the `remove_via_ladder_constraints` command.

## Inserting Via Ladders

To insert via ladders, use the `insert_via_ladders` command. By default, this command

- Does not remove existing via ladders in the block
- To remove the existing via ladders before via ladder insertion, use the `-clean true` option.
- Targets all pins specified in the via ladder constraints, whether on signal nets or clock nets
- To restrict via ladder insertion to specific nets, use the `-nets` option.
- Inserts a single via ladder on each target pin

If the constraints specify more than one via ladder template for a pin, the command inserts the via ladder using the first template that does not cause a DRC violation.

- By default, if all of the via ladder templates cause a DRC violation, the command inserts the via ladder with the lowest DRC cost.
- To ignore detail routing shapes when checking for DRC violations, use the `-ignore_routing_shape_drcs true` option.
- To ignore the line-end cut enclosure rule, use the `-relax_line_end_via_enclosure_rule true` option.
- To ignore metal spacing rules on the pin layer when the via ladder shape is fully enclosed within the pin, use the `-relax_pin_layer_metal_spacing_rules true` option.
- To prevent the insertion of via ladders that cause DRC violations, use the `-allow_drcs false` option.

### Note:

If a pin is too small to accommodate any of the via ladders specified for it, the `insert_via_ladders` command does not insert a via ladder for that pin.

- Uses via cuts other than those specified in the template if they improve compliance with fat via rules
- To require the command to use only those via cuts specified in the template, use the `-strictly_honor_cut_table true` option.
- Centers the via ladder cuts at the intersection of routing tracks between adjacent layers
- To shift the via ladder cuts on transition layers off the routing tracks for improved DRC compliance, use the `-shift_vias_on_transition_layers true` option.

- Honors nondefault width rules for the wires on all layers of the via ladder
- To use the default width for wires on lower layers of the via ladder and honor the nondefault width only for the top layer, use the `-ndr_on_top_layer_only true` option.
- Does not require the via inserted above the pin layer to be contained within the pin shape

To prevent changes to the pin shape boundary, you can require that the via inserted above the pin layer be fully contained within the pin shape boundary, which includes the pin shape and its extensions. To specify this requirement, use one or both of the following options:

- `-connect_within_metal true`

This option controls via enclosures for all types of via ladders: performance, electromigration, and pattern-must-join.

- `-connect_within_metal_for_via_ladder true`

This option controls the via enclosures only for performance and electromigration via ladders. By default, the tool uses the global setting specified by the `-connect_within_metal` option. To explicitly specify the behavior for performance and electromigration via ladders, set this option to `true` or `false`.

- Does not extend the via enclosures to meet the minimum length rule for the layer

To enable patching of the via enclosures to meet the minimum length rule and allow additional stacking of the via ladder, use the `-allow_patching true` option.

- Uses staggering only when it is defined for the via ladder rule

Via ladder staggering extends the row metal of a via ladder level in the preferred direction of the upper layer for the level to avoid obstructions above or near the pin, which can increase the success rate of via ladder insertion. In many cases, the via ladder rule specifies the maximum number of stagger tracks. To allow staggering if the maximum number of stagger tracks is not defined for the via ladder rule, use the `-auto_stagger true` option.

- Does not report on the insertion status

To report the via ladder insertion status for each target pin, use the `-verbose true` option.

To output detailed information about insertion failures, use the `-user_debug true` option. The detailed information reports the following types of violations:

- Internal DRC violations

These are DRC violations internal to a via ladder. These violations are never allowed.

- Hard DRC violations

These are DRC violations between a via ladder and a fixed shape, such as a pin, preroute, or obstruction. These violations are never allowed.

- Soft DRC violations

These are DRC violations between a via ladder and a detail routing shape, which can be rerouted. These violations are allowed if one or more of the following options are `true: -allow_drcs, -ignore_rippable_shapes, or -ignore_routing_shape_drcs`.

After inserting the via ladders, the command reports statistics on the inserted ladders and any DRC violations caused by the insertion.

## See Also

- [Defining Via Ladder Constraints](#)
- [Verifying Via Ladders](#)
- [Updating Via Ladders](#)
- [Querying Via Ladders](#)
- [Removing Via Ladders](#)

## Protecting Via Ladders

To prevent via ladders from being edited, set the `design.enable_via_ladder_protection` application option to `true`.

```
fc_shell> set_app_options -name design.enable_via_ladder_protection \
-value true
```

When the `design.enable_via_ladder_protection` application option is `true`, the following commands process only objects that do not belong to via ladders:

- `remove_shapes`
- `remove_vias`

- remove\_objects
- set\_attribute
- set\_via\_def
- add\_to\_edit\_group

If an object belongs to a via ladder, these commands issue a warning message and do not process the object.

The following example sets the `design.enable_via_ladder_protection` application option to `true` and attempts to remove a collection of shapes:

```
fc_shell> set_app_options -name design.enable_via_ladder_protection \
 -value true
...
fc_shell> sizeof_collection [get_shapes -of net1]
77

fc_shell> remove_shapes [get_shapes -of net1] -verbose
Warning: Cannot edit PATH_32_7403 because it is part of via_ladder.
(NDM-160)
76
```

---

## Verifying Via Ladders

To verify that the via ladders in the block match the via ladder constraints and are properly connected to pins, use the `verify_via_ladders` command. By default, this command checks all via ladders. To restrict the checking to specific nets, use the `-nets` option.

### Note:

If you use the `-shift_vias_on_transition_layers true` option when you insert the via ladders, you must also use this option when you verify the via ladders; otherwise, the command reports false violations.

A via ladder matches the via ladder constraints if the pattern of connected vias and wires for the via ladder structure satisfies any of the via ladder templates assigned to the connected pin. The command flags a violation in the following situations:

- A via ladder is connected to a pin that does not have a via ladder constraint
- A via ladder is connected to a pin with a via ladder constraint, but none of the templates correctly describe the current via ladder structure
- A pin has a via ladder constraint, but does not have a via ladder connected to it

A via ladder is properly connected to a pin if all the via enclosures at the base of the via ladder touch the same pin and that pin is logically connected to the same net as the via ladder. The command flags a violation in the following situations:

- An enclosure of the via ladder base touches a different pin, unless the pins are must-join pins that belong to the same must-join set
- An enclosure of the via ladder base does not touch any pin
- An enclosure of the via ladder touches a pin that is not of the same net as the via ladder

By default, the `verify_via_ladders` command reports the via ladder insertion status for each target pin, which can result in a very large report. To limit the number of via ladders reported for each category to 40, use the `-report_all_via_ladders false` option with the `verify_via_ladders` command.

## Updating Via Ladders

You can use the following methods to update existing via ladders in a design:

- Use the `refresh_via_ladders` command
- Enable automatic updates during the `route_group`, `route_auto`, and `route_eco` commands by setting the `route.auto_via_ladder.update_during_route` application option to `true`

Both of these methods perform the following tasks:

- Verifies the existing via ladders based on the current via ladder constraints
- Removes invalid via ladders
- Inserts via ladders where needed based on the settings of the `route.auto_via_ladder` application options

By default, the tool

- Updates all pins with via ladder constraints, whether on signal nets or clock nets

During the `route_group` command, you can restrict the via ladder updates to only those nets specified in the `route_group` command by setting the `route.auto_via_ladder.update_on_route_group_nets_only` application option to `true`.

- Does not require the via inserted above the pin layer to be contained within the pin shape

To prevent changes to the pin shape boundary, you can require that the via inserted about the pin layer be fully contained within the pin shape boundary, which includes

the pin shape and its extensions. To specify this requirement, set the following application options:

- `route.auto_via_ladder.connect_within_metal`

This application option controls via enclosures for all types of via ladders: performance, electromigration, and pattern-must-join. To prevent changes to the pin shape boundary, set this application option to `true`.

- `route.auto_via_ladder.connect_within_metal_for_via_ladder`

This application option controls the via enclosures only for performance and electromigration via ladders. By default, the tool uses the global setting specified by the `route.auto_via_ladder.connect_within_metal` application option. To explicitly specify the behavior for performance and electromigration via ladders, set this application option to `true` or `false`.

- Uses staggering only when it is defined for the via ladder rule

Via ladder staggering extends the row metal of a via ladder level in the preferred direction of the upper layer for the level to avoid obstructions above or near the pin, which can increase the success rate of via ladder insertion. In many cases, the via ladder rule specifies the maximum number of stagger tracks. To allow staggering if the maximum number of stagger tracks is not defined for the via ladder rule, set the `route.auto_via_ladder.auto_stagger` application option to `true`.

- Reports the via ladder insertion status for each target pin

To limit the number of via ladders reported for each category to 40, set the `route.auto_via_ladder.report_all_via_ladders` application option to `false`.

## See Also

- [Inserting Via Ladders](#)
- [Verifying Via Ladders](#)
- [Querying Via Ladders](#)
- [Removing Via Ladders](#)

## Manual Via Ladder Insertion

To manually insert a via ladder,

1. Create the shapes that compose the via ladder by using the `create_shape` and `create_via` commands.
2. Create the via ladder by using the `create_via_ladder` command.

At a minimum, you must specify the shapes that compose the via ladder by using the `-shapes` option.

The command creates a via ladder named `VIA_LADDER_n`, where *n* is a unique integer.

Use the following options with the `create_via_ladder` command to specify attributes of the created via ladder:

- `-via_rule`

This option specifies the via ladder rule associated with the created via ladder. It sets the `via_rule_name` attribute of the created via ladder.

- `-electromigration`

This option specifies that the via ladder is an electromigration via ladder. It sets the `is_electromigration` attribute of the created via ladder to `true`.

- `-high_performance`

This option specifies that the via ladder is a performance via ladder. It sets the `is_high_performance` attribute of the created via ladder to `true`.

- `-pattern_must_join`

This option specifies that the via ladder is a pattern-must-join via ladder. It sets the `is_pattern_must_join` attribute of the created via ladder to `true`.

- `-pin`

This option specifies the physical pin connected to the created via ladder. It sets the `pin` attribute of the created via ladder.

## See Also

- [Querying Via Ladders](#)
- [Removing Via Ladders](#)

---

## Querying Via Ladders

To query the via ladders in a block, use the `get_via_ladders` command. To report on the via ladders in a block, use the `report_via_ladders` command.

## Removing Via Ladders

To remove via ladders and their associated shapes, use the `remove_via_ladders` command. You must specify the via ladders to remove. To remove all via ladders from the block, use the asterisk (\*) wildcard character, as shown in the following example:

```
fc_shell> remove_via_ladders *
```

To remove via ladders only from specific nets, use the `-nets` option.

## Checking Routability

After placement is completed, you can use the `check_routability` command to check whether your block is ready for detail routing.

By default, this command checks for

- Blocked standard cell ports

A standard cell port is considered blocked if none of its physical pins is accessible.

A standard cell pin is considered accessible if the pin contains a via that extends to a neighboring layer, there is a path on the pin layer that is at least as long as the search range distance, or there is a shorter path on the pin layer that ends at a via to a neighboring layer. By default, the search range distance is two times the layer pitch. To change this distance, use the `-standard_cell_search_range` option to specify a different pitch multiplier, up to a maximum of 10. If you specify a value larger than 10, the command sets the value to 10.

By default, the tool does not check whether via connections are fully inside the standard cell pins. To enable this check, use the `-connect_standard_cells_within_pins true` option.

To disable the checking of blocked standard cell ports, use the `-check_standard_cell_blocked_ports false` option.

- Blocked top-level or macro cell ports

A top-level or macro cell port is considered blocked if none of its physical pins is accessible.

A top-level or macro pin is considered accessible if a legal path can be extended from the pin to a certain distance around it. This path can be just on the pin layer or can extend to a neighboring layer by a single via. By default, the distance is 10 times the layer pitch. To change this distance for same-layer paths, use the `-blocked_range` option to specify a different pitch multiplier, up to a maximum of 40. To change this distance for neighboring-layer paths through a via, use the `-blocked_range_via_side`

option to specify a different pitch multiplier, up to a maximum of 40. If you specify a value larger than 40 for either of these options, the command sets its value to 40.

By default, the command checks whether a pin is accessible in either the horizontal or vertical direction. To require that the pin is accessible in the preferred direction for its layer, use the `-obey_direction_preference true` option.

To disable the checking of blocked top-level or macro cell ports, use the `-check_non_standard_cell_blocked_ports false` option.

- Out-of-boundary pins

This check verifies that all pins are within the block boundary.

To disable the checking of out-of-boundary pins, use the `-check_out_of_boundary false` option.

- Minimum grid violations

This check verifies that all pins, including those within library cells, are on the minimum grid, as defined by the `gridResolution` attribute in the technology file.

To disable the checking of minimum grid violations, use the `-check_min_grid false` option.

- Incorrect via definitions

This check verifies that

- Uncolored via arrays do not have more than 20 rows or columns
- Custom or asymmetric simple via definitions do not have more than 1000 cuts
- The design does not contain more than 65535 via definitions

You cannot disable these checks.

- Invalid real metal via cut blockages

This check verifies that all real metal via cut blockages match a legal cut size, as defined by the `cutWidthTbl`, `cutHeightTbl`, and `minWidth` attributes in the technology file.

To disable this check, use the `-check_via_cut_blockage false` option.

- Minimum width settings

This check verifies that the nondefault minimum width and shield width settings are no larger than the maximum width defined in the technology file.

You cannot disable these checks.

The `check_routability` command also supports the following optional checks:

- Blocked power or ground ports

To enable this check, use the `-check_pg_blocked_ports true` option.

- Redundant power or ground shapes

To enable this check, use the `-check_redundant_pg_shapes true` option.

- Staggered power and ground vias that block the routing tracks

To enable this check, use the `-check_routing_track_space true` option.

- Blocked ports on frozen nets

To enable this check, use the `-check_frozen_net_blocked_ports true` option.

- Blocked unconnected pins

To enable this check, use the `-check_no_net_pins true` option.

- Real metal blockages that overlap library cell pins

To enable this check, use the `-check_real_metal_blockage_overlap_pin true` option.

- Invalid real metal via cut blockages in the library cells

To enable this check, use the `-check_lib_via_cut_blockage true` option.

- Shielding checks

These checks detect possible issues with shielding by checking for the following conditions:

- Signal net shapes with a `shape_use` attribute of `shield_route`.
- PG net shapes, which might be a PG strap or rail, but have a `shape_use` attribute of `detail_route`.
- Signal, clock, or PG nets that have a shielding nondefault rule but no associated shield shapes, which might be caused by inappropriate `shape_use` attributes.

To enable these checks, use the `-check_shield true` option.

- Via ladder checks

These checks detect possible issues with via ladder insertion by checking for the following conditions:

- Library cell pins whose top-layer terminals have multiple pin shapes but do not have a `pattern_must_join` attribute

This check applies only to signal and secondary PG pins.

- Missing performance or electromigration via ladder on a pin that has the `is_em_via_ladder_required` attribute
- Via ladder application options that do not have the required settings

The following application options must be set to `true` (their default is `false`):

- `route.auto_via_ladder.allow_patching`
- `route.auto_via_ladder.enable_em_to_performance_via_ladder_update`
- `route.auto_via_ladder.ignore_routing_shape_drcs`
- `route.auto_via_ladder.relax_line_end_via_enclosure_rule`
- `route.auto_via_ladder.relax_pin_layer_metal_spacing_rules`
- `route.auto_via_ladder.shift_vias_on_transition_layers`
- `route.auto_via_ladder.update_during_route`
- `route.common.via_ladder_top_layer_overrides_net_min_layer`

To enable these checks, use the `-via_ladder true` option.

The `check_routability` command supports the following additional pin connection controls that apply to all pin access checks:

- Pin access edges

During frame view extraction, the tool annotates the frame views with information about the pin access edges. By default, the `check_routability` command ignores the pin access edges and allows pin connections at any point. You can restrict pin connections to the defined access edges by setting the `-obey_access_edges` option to `true`. You can further restrict pin connections to the access edge mark, which can be a narrow rectangle, a short line, or even a single point, by setting the `-access_edge_whole_side` option to `true`. In addition, the command can report

unconnected pins that do not have an access edge defined. To enable this check, use the `-report_no_access_edge true` option.

- Via rotation

By default, pin connections can use rotated vias. To disallow pin connections that use rotated vias, set the `-allow_via_rotation` option to `false`.

By default, this command considers the following routing layer constraints when checking for blocked ports:

- The global minimum and maximum routing layer constraints set by the `-min_routing_layer` and `-max_routing_layer` options of the `set_ignored_layers` command

These constraints can cause blocked ports only if they are defined as hard constraints. By default, these constraints are soft constraints. They are hard constraints only if the `route.common.global_min_layer_mode` and `route.common.global_max_layer_mode` application options are set to hard.

- The net-specific minimum and maximum routing layer constraints set by the `-min_routing_layer` and `-max_routing_layer` options of the `set_routing_rule` command

These constraints can cause blocked ports only if they are defined as hard constraints. By default, the net-specific minimum layer constraint is a soft constraint and the net-specific maximum layer constraint is a hard constraint. They are hard constraints only if the `route.common.net_min_layer_mode` and `route.common.net_max_layer_mode` application options are set to hard.

- The clock minimum and maximum routing layer constraints set by the `-min_routing_layer` and `-max_routing_layer` options of the `set_clock_routing_rules` command

These constraints can cause blocked ports only if they are defined as hard constraints. By default, the clock minimum layer constraint is a soft constraint and the clock maximum layer constraint is a hard constraint. They are hard constraints only if the `route.common.net_min_layer_mode` and `route.common.net_max_layer_mode` application options are set to hard.

- The freeze layer constraints set by the `route.common.freeze_layer_by_layer_name` and `route.common.freeze_via_to_frozen_layer_by_layer_name` application options

To ignore the routing layer constraints during the blocked port checks, use the `-honor_layer_constraints false` option.

You can use the error browser to examine the errors detected by the `check_routability` command. By default, the error data generated by the `check_routability` command

is named `check_routability.err`; to specify a different name for the error data, use the `-error_data` option. The error data is saved in the design library when you save the block. For information about using the error browser, see the *Fusion Compiler Graphical User Interface User Guide*.

## Routing Constraints

Routing constraints provide guidance during routing. [Table 27](#) describes the types of guidance provided by the routing constraints.

*Table 27 Routing Constraints*

Guidance	Description
Controlling the layers used for routing	For information about specifying the routing layers, see <a href="#">Specifying the Routing Resources</a> .
Setting stricter minimum width and spacing rules	To set stricter minimum width and spacing rules, define a nondefault routing rule and apply it to the affected nets. For information about defining and applying nondefault routing rules, see <a href="#">Using Nondefault Routing Rules</a> .
Controlling routing through voltage areas	By default, routing can pass through any voltage areas. To restrict the voltage-area pass-throughs, define a voltage area routing rule, as described in <a href="#">Controlling Physical-Feedthrough Nets in Voltage Areas</a> .
Controlling the routing direction	You can specify the preferred routing direction for specific layers or for specific regions. <ul style="list-style-type: none"> <li>For information about specifying the preferred routing direction for specific layers, see <a href="#">Setting the Preferred Routing Direction for Layers</a>.</li> <li>To control the routing direction in a specific region, use either a preferred-direction-only routing guide or a switch-preferred-direction routing guide. For information about routing guides, see <a href="#">Defining Routing Guides</a>.</li> </ul>
Limiting the number of edges in the nonpreferred routing direction	To limit the number of edges in the nonpreferred routing direction, use a maximum-number-of-pattern routing guide. For information about routing guides, see <a href="#">Defining Routing Guides</a> .
Controlling off-grid routing	By default, off-grid routing is allowed for wires or vias unless the <code>onWireTrack</code> or <code>onGrid</code> attribute is set to 1 for the layer in the technology file.  You can prevent or discourage off-grid routing, as described in <a href="#">Controlling Off-Grid Routing</a> .
Preventing routing of specific nets	To prevent routing of specific nets, set the <code>physical_status</code> attribute of each net to <code>locked</code> , as described in <a href="#">Setting the Rerouting Mode</a> .
Preventing routing in specific region	To prevent routing in a specific region, define a routing blockage, as described in <a href="#">Defining Routing Blockages</a> .
Reserving space for top-level routing	To reserve space for top-level routing, create a corridor routing blockage, as described in <a href="#">Reserving Space for Top-Level Routing</a> .
Restricting routing to specific regions	To restrict routing to a specific region, use a routing corridor, as described in <a href="#">Routing Nets Within a Specific Region</a> .

**Table 27     *Routing Constraints (Continued)***

Guidance	Description
Prioritizing routing regions	To prioritize routing regions, use an access preference routing guide. For information about routing guides, see <a href="#">Defining Routing Guides</a> .
Controlling the routing density	To control the routing density for specific layers, use a utilization routing guide. For information about routing guides, see <a href="#">Defining Routing Guides</a> .
Encouraging river routing	To encourage river routing on specific layers, use a single-layer routing guide. For information about routing guides, see <a href="#">Defining Routing Guides</a> .
Improving routability of hard macro pins	To improve the routability of hard macro pins, use pin access routing guides, as described in <a href="#">Deriving Pin Access Routing Guides</a> .
Controlling routing along the perimeter of a block	To control detail routing around the perimeter of a block, insert constraint objects, as described in <a href="#">Controlling Routing Around the Block Boundary</a> .
Creating routing guides for the metal cut allowed and forbidden preferred grid extension rules	<p>You can use the following methods to create these routing guides:</p> <ul style="list-style-type: none"> <li>• Automatic creation during boundary cell insertion, as described in <a href="#">Creating Routing Guides During Boundary Cell Insertion</a></li> <li>• Manual creations by using the <code>derive_metal_cut_route_guides</code> command, as described in</li> </ul>
Controlling the pin connections	<p>You can control both the allowed types of pin connections and the pin tapering.</p> <ul style="list-style-type: none"> <li>• For information about must-join pins, see <a href="#">Routing Must-Join Pins</a>.</li> <li>• For information about controlling the allowed types of pin connections, see <a href="#">Controlling Pin Connections</a>.</li> <li>• For information about controlling pin tapering, see <a href="#">Controlling Pin Tapering</a>.</li> </ul>
Controlling via ladder connections	For information about controlling the via ladder connections, see <a href="#">Controlling Via Ladder Connections</a> .
Restricting the extent of rerouting	For information about restricting the extent of rerouting for specific nets, see <a href="#">Setting the Rerouting Mode</a> .

## Defining Routing Blockages

A routing blockage defines a region where routing is not allowed on specific layers. Zroute considers routing blockages to be hard constraints.

To define a routing blockage, use the `create_routing_blockage` command. At a minimum, you must define the boundary of the routing blockage and the affected layers.

- To create a rectangular routing blockage, use the `-boundary` option to specify the lower-left and upper-right corners of the rectangle using the following syntax: `{ {llx lly} {urx ury} }`.
- To create a rectilinear routing blockage, use the `-boundary` option to specify the coordinates of the polygon using the following syntax: `{ {x1 y1} {x2 y2} ... }`.

You can also create a rectilinear routing blockage by specifying the polygon as the combined area of a heterogeneous collection of objects with physical geometry, such as `poly_rects`, `geo_masks`, `shapes`, `layers`, and other physical objects. If you specify a layer, the resulting area includes the area of every shape on the layer. For all other objects, the resulting area includes the area of each object. If you use this method, the resulting area must resolve to a single, connected polygon; otherwise, the command fails.

- To specify the affected layers, use the `-layers` option.

Specify the routing layers by using the layer names from the technology file. The layers can be metal, via, or poly layers.

By default, the tool creates a routing blockage named `RB_objId` in the current block that prevents routing of all nets within the routing blockage boundary. The routing blockage is considered as real metal for RC extraction and DRC checking and the router must meet the minimum spacing requirements between the routing blockage boundary and the net shapes. Use the following options to change the default behavior:

- `-name blockage_name`

Specifies a name for the routing blockage.

- `-cell cell`

Creates the routing blockage in a different physical cell.

When you use this option, the tool creates the routing blockage in the cell's reference block using the coordinate system of the cell's top-level block.

- `-zero_spacing`

Disables the minimum spacing rule between the routing blockage boundary and the net shapes (zero minimum spacing). This option also prevents the routing blockage from being treated as real metal during extraction and DRC checking.

When you use this option, the net shapes can touch, but not overlap, the routing blockage boundary.

**Note:**

When you create a routing blockage to prevent via insertion, you must use the `-zero_spacing` option; otherwise, frame view extraction does not use the route guide to trim the via region.

- `-net_types list_of_types`

Applies the routing blockage only to the specified net types.

Specify one or more of the following net types: `analog_ground`, `analog_power`, `analog_signal`, `clock`, `deep_nwell`, `deep_pwell`, `ground`, `nwell`, `power`, `pwell`, `reset`, `scan`, `signal`, `tie_high`, and `tie_low`. To remove the net type settings and prevent routing of all nets in the routing blockage, use the `-net_types unset` setting.

To prevent signal routing on the M1 layer within the rectangle with its lower-left corner at (0, 0) and its upper-right corner at (100, 100), use the following command:

```
fc_shell> create_routing_blockage \
 -boundary { {0.0 0.0} {100.0 100.0} } \
 -net_types signal -layers M1
```

To prevent vias on the V1 layer within the rectangle with its lower-left corner at (0, 0) and its upper-right corner at (100, 100), use the following command:

```
fc_shell> create_routing_blockage \
 -boundary { {0.0 0.0} {100.0 100.0} } \
 -net_types signal -layers V1 -zero_spacing
```

To prevent PG routing on the M2 layer within the rectangle with its lower-left corner at (0, 0) and its upper-right corner at (100, 100), use the following command:

```
fc_shell> create_routing_blockage \
 -boundary { {0.0 0.0} {100.0 100.0} } \
 -net_types {power ground} -layers M2
```

## Reserving Space for Top-Level Routing

To reserve space for top-level routing, create a corridor routing blockage by using the `-reserve_for_top_level_routing` option when you create the routing blockage.

During block-level implementation, a corridor routing blockage acts as a regular blockage to prevent routing in the blockage area. During frame view extraction, the tool removes the corridor routing blockage to allow top-level routing in this area.

## Querying Routing Blockages

To find routing blockages, use the `get_routing_blockages` command. For example, to get all the routing blockages in a block, use the following command:

```
fc_shell> get_routing_blockages *
```

To find the routing blockages for specific nets, use the `-of_objects` option to specify the nets of interest. For example, to find the routing blockages for the n1 net, use the following command:

```
fc_shell> get_routing_blockages -of_objects [get_nets n1]
```

To find the routing blockages in a specific location, use the `get_objects_by_location -classes routing_blockage` command.

## Removing Routing Blockages

To remove routing blockages from the current block, use the `remove_routing_blockages` command.

- To remove specific routing blockages, specify the routing blockages, either as a list or collection, such as that returned by the `get_routing_blockages` command.
- To remove all routing corridors, specify the `-all` option.

## Defining Routing Guides

Routing guides provide routing directives for specific areas of a block. Some types of routing guides are user-defined; others are derived from the block data. This topic describes how to create user-defined routing guides.

### Note:

User-defined routing guides are honored by Zroute; however, they are not honored by the Advanced Route tool.

To define a routing guide, use the `create_routing_guide` command. When you define a routing guide, you must specify

- Its rectangular boundary

To specify the boundary, use the `-boundary` option to specify the lower-left and upper-right corners of the rectangle using the following syntax:

```
{ {lx ly} {urx ury} }
```

- Information specific to the purpose of the routing guide

The following table describes the user-defined routing guides.

**Table 28 User-Defined Routing Guides**

Purpose	Option
Route all nets in the preferred direction within the routing guide boundary. For details, see <a href="#">Using Routing Guides to Control the Routing Direction</a> .	<code>-preferred_direction_only</code>
Switch the preferred routing direction within the routing guide boundary. For details, see <a href="#">Using Routing Guides to Control the Routing Direction</a> .	<code>-switch_preferred_direction</code>
Limit the number of occurrences of a specific routing pattern within the routing guide boundary. For details, see <a href="#">Using Routing Guides to Limit Edges in the Nonpreferred Direction</a> .	<code>-max_patterns</code>
Control the routing density within the routing guide boundary. For details, see <a href="#">Using Routing Guides to Control the Routing Density</a> .	<code>-horizontal_track_utilization</code> <code>-vertical_track_utilization</code>
Prioritize regions within the routing guide boundary for routing. For details, see <a href="#">Using Routing Guides to Prioritize Routing Regions</a> .	<code>-access_preference</code>
Encourage river routing within the routing guide boundary. For details, see <a href="#">Using Routing Guides to Encourage River Routing</a> .	<code>-river_routing</code>

By default, the tool creates a routing guide named RD#*n* in the current block, where *n* is a unique integer. Use the following options to change the default behavior:

- To specify a name for the routing guide, use the `-name` option.
- To specify the routing layers affected by the routing guide, use the `-layers` option.

Specify the routing layers by using the layer names from the technology file.

- To create the routing guide in a different physical cell, use the `-cell` option.

When you use this option, the tool creates the routing guide in the cell's reference block using the coordinate system of the cell's top-level block.

## See Also

- [Deriving Routing Guides](#)
- [Querying Routing Guides](#)
- [Removing Routing Guides](#)

## Using Routing Guides to Control the Routing Direction

You can use a routing guide to control the routing direction within the routing guide boundary, either by requiring routes to be in the preferred direction within the routing guide boundary or by switching the preferred direction within the routing guide boundary.

- To force the router to route all nets in the preferred direction within the routing guide boundary, use the `-preferred_direction_only` option with the `create_routing_guide` command.

You can use this type of routing guide to prevent wrong-way jog wires on specific layers.

By default, this routing guide applies to all layers within the routing guide boundary. To require preferred direction routing only for specific layers, use the `-layers` option to specify the affected layers.

For example, to force the router to use only the preferred direction on the M4 layer within the rectangle with its lower-left corner at (0, 0) and its upper-right corner at (100, 100), use the following command:

```
fc_shell> create_routing_guide -boundary {{0.0 0.0} {100.0 100.0}} \
 -layers {M4} -preferred_direction_only
```

- To switch the preferred routing direction within the routing guide boundary, use the `-switch_preferred_direction` option with the `create_routing_guide` command.

You can use this type of routing guide to allow routing over macros, which might reduce congestion for a block that contains much detour routing.

By default, this routing guide applies to all layers within the routing guide boundary. To switch the preferred routing direction only for specific layers, use the `-layers` option to specify the affected layers.

For example, to switch the preferred routing direction for the M1 and M2 layers within the rectangle with its lower-left corner at (0, 0) and its upper-right corner at (100, 100), use the following command:

```
fc_shell> create_routing_guide -boundary {{0.0 0.0} {100.0 100.0}} \
 -layers {M1 M2} -switch_preferred_direction
```

### Note:

If a switch-preferred-direction routing guide overlaps with a preferred-direction-only routing guide, the switch-preferred-direction routing guide takes precedence.

## Using Routing Guides to Limit Edges in the Nonpreferred Direction

You can use a routing guide to limit the number of occurrences of a specific routing pattern within the routing guide boundary on specific layers. Zroute tries to route the nets within the routing guide to meet this constraint; however, if the number of edges in the nonpreferred direction exceeds the specified threshold, Zroute reports a violation.

To create this type of routing guide, use the `-max_patterns` option with the `create_routing_guide` command. Use the following syntax to specify the routing pattern and its threshold for each affected layer:

```
{ {layer pattern limit} ... }
```

In addition to using the `-max_patterns` option, you must also use the `-layers` option when you create this type of routing guide. Specify the same layers in the `-layers` option as those specified in the `-max_patterns` option.

If you do not specify a pattern threshold for a layer, the routing pattern has no limit for that layer.

Currently, the only supported pattern is `non_pref_dir_edge`, which represents the edges in the nonpreferred direction. For example, to limit the number of edges in the nonpreferred direction to two on M2 and to three on M4, with no limits on other layers, use the following command:

```
fc_shell> create_routing_guide -boundary {{50 50} {200 200}} \
 -max_patterns {{M2 non_pref_dir_edge 2} {M4 non_pref_dir_edge 3}} \
 -layers {M2 M4}
```

## Using Routing Guides to Control the Routing Density

By default, the maximum track utilization is 100 percent. You can use a routing guide to control the routing density within the routing guide boundary.

- To set the maximum track utilization for layers with a horizontal preferred direction, use the `-horizontal_track_utilization` option with the `create_routing_guide` command.
- To set the maximum track utilization for layers with a vertical preferred direction, use the `-vertical_track_utilization` option with the `create_routing_guide` command.

By default, when you create these routing guides, they apply to all layers within the routing guide boundary. To set the routing density only for specific layers, use the `-layers` option to specify the affected layers.

For example, to set a maximum track utilization of 50 percent for all layers with a horizontal preferred direction and a maximum track utilization of 30 percent for all layers

with a vertical preferred direction within the rectangle with its lower-left corner at (0, 0) and its upper-right corner at (100, 100), use the following command:

```
fc_shell> create_routing_guide -boundary {{0.0 0.0} {100.0 100.0}} \
 -horizontal_track_utilization 50 -vertical_track_utilization 30
```

## Using Routing Guides to Prioritize Routing Regions

You can use a routing guide to prioritize regions within the routing guide boundary for routing. The regions are called access preference areas. You prioritize the access preference areas by assigning strengths to them. Access preference areas with higher strengths are preferred for routing. You can define access preference areas for both wires and vias. When you define a via access-preference area, you are defining a preference area for the via surrounds on the specified metal layer for vias coming from both above and below that metal layer.

To create this type of routing guide, use the `-access_preference` option with the `create_routing_guide` command. Use the following syntax to specify the access preference areas and their strengths:

```
{layer [{wire_access_preference wire_rect wire_strength}] \
 [{via_access_preference via_rect via_strength}] ...}
```

You can define multiple access preference areas. To define the relative preference of the access preference areas, use strength values between 0 and 1. If access preference areas overlap, the stronger access preference area takes precedence over the weaker access preference area. To require routing in a specific access preference area, use a strength value of 1. When you define access preference areas with a strength of 1, all access preference areas with a strength less than 1 are ignored and Zroute treats the access preference routing guide as a hard constraint.

The following example creates an access preference routing guide whose boundary is a rectangle with its lower-left corner at (0, 0) and its upper-right corner at (300, 300). It contains one wire access-preference area with coordinates of (0, 0) and (2, 1) and two via access-preference areas, one with coordinates of (0, 0) and (5, 5) and one with coordinates of (40, 40) and (45, 45). The wire access-preference area is slightly preferred over areas outside of the access preference area because it has a strength of 0.2. The via access-preference area with coordinates at (40, 40) and (45, 45) is ignored, because routing is required in the via access-preference area with coordinates at (0, 0) and (5, 5), which has a strength of 1.

```
fc_shell> create_routing_guide -boundary {{0 0} {300 300}} \
 -access_preference {M1 {wire_access_preference {{0 0} {2 1}} 0.2} \
 {via_access_preference {{0 0} {5 5}} 1.0} \
 {via_access_preference {{40 40} {45 45}} 0.5}}
```

To report information about the access preference routing guides defined for your block, use the `report_routing_guides` command.

## Using Routing Guides to Encourage River Routing

River routing is a special routing topology that tries to minimize the space taken by the router by compressing the routes to follow a particular contour. River routing is useful if there are tight routing channels or you need to minimize the space taken by routing. You can use a routing guide to encourage river routing within the routing guide boundary.

To create this type of routing guide, use the `-river_routing` option with the `create_routing_guide` command. You must also use the `-layers` option to specify the affected layers.

For example, to encourage river routing on the M2 layer within the rectangle with its lower-left corner at (0, 0) and its upper-right corner at (100, 100), use the following command:

```
fc_shell> create_routing_guide -boundary {{0.0 0.0} {100.0 100.0}} \
 -river_routing -layers {M2}
```

## Querying Routing Guides

To find routing guides, use the `get_routing_guides` command. For example, to get all the routing guides in your block, use the following command:

```
fc_shell> get_routing_guides *
```

To find the routing guides in a specific location, use the `get_objects_by_location -classes routing_guide` command.

## Removing Routing Guides

To remove routing guides from the current block, use the `remove_routing_guides` command.

- To remove specific routing guides, specify the routing rules, either as a list or collection, such as that returned by the `get_routing_guides` command.
- To remove all routing guides, specify the `-all` option.

For example, to remove the `new_width_rule` routing rule, use the following command:

```
fc_shell> remove_routing_rules new_width_rule
```

## Deriving Routing Guides

Routing guides provides routing directives for specific areas of a block. Some types of routing guides are user-defined; others are derived from the block data. This topic described how to derive the following types of routing guides:

- [Deriving Pin Access Routing Guides](#)
- [Deriving Metal Cut Routing Guides](#)

## See Also

- [Defining Routing Guides](#)

## Deriving Pin Access Routing Guides

When you reuse hard macros, such as IP blocks, at a smaller technology node, it can cause routability issues to the macro pins. You can use routing guides and blockages to enable access to the hard macro pins.

To create these routing guides, which are referred to as *pin access routing guides*, and the associated blockages, use the `derive_pin_access_routing_guides` command. You must specify the following information:

- The hard macros around which to create the routing guides and routing blockages (the `-cells` option)
- The metal layers for which to create the routing guides and routing blockages (the `-layers` option)
- The width of the routing guides and blockages in the x- and y-directions (the `-x_width` and `-y_width` options)

For each specified hard macro, the command creates routing guides to enable routing to the pins on the specified layers. In addition, the command creates metal blockages that surround the macro on the specified layers, with cutouts to allow pin access, and via blockages that cover the macro on the via layers adjacent to the specified metal layers. Vertical routing guides and blockages have the specified x-width. Horizontal routing guides and blockages have the specified y-width. By default, the command creates pin cutouts only in the preferred routing direction. To also create pin cutouts in the nonpreferred direction, use the `-nonpreferred_direction` option. To extend the pin cutouts beyond the metal blockages, use the `-pin_extension` option to specify the extension distance in microns.

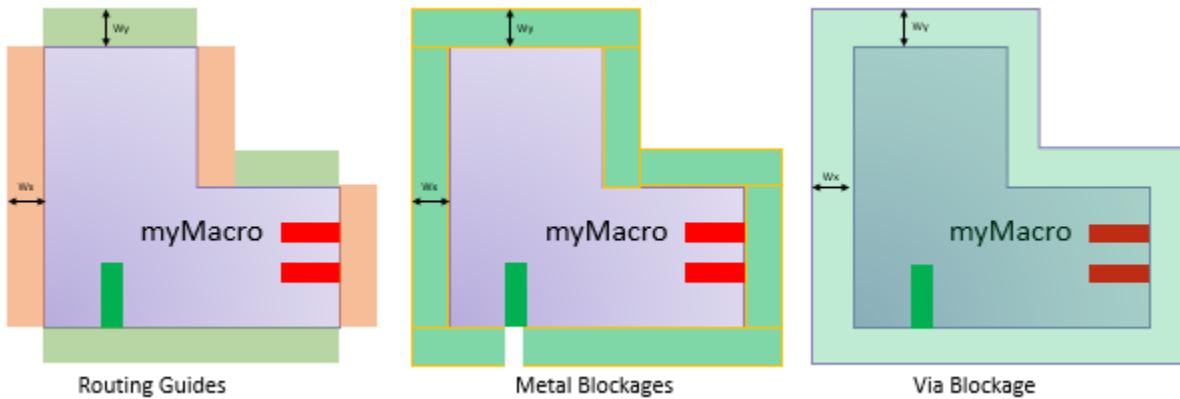
To enable proper routing to pins of different sizes, the command sets the `is_rectangle_only_rule_waived` attribute on the macro pins to `true` to waive the rectangle-only rule.

For example, the following command creates the routing guides and blockages shown in [Figure 85](#).

```
fc_shell> derive_pin_access_routing_guides -cells myMacro \
 -layers {M2 M3} -x_width 0.6 -y_width 0.7
```

In the figure, the two red pins are on M2 and the green pin is on M3. The figure shows the routing guides created on both the M2 and M3 layers, the metal blockages created on the M3 layer, and the via blockage created on the V2 and V3 layers. The tool also creates metal blockages on the M2 layer, which have cutouts for the M2 pins, and via blockages on the V1 and V2 layers.

Figure 85 Pin Access Routing Guides and Blockages



If a hard macro already has pin access routing guides and their associated blockages, the command removes these existing objects and creates new routing guides and blockages.

If you move a hard macro after deriving the pin access routing guides, you must regenerate the pin access routing guides. You cannot manually modify pin access routing guides.

## Deriving Metal Cut Routing Guides

If your technology file defines metal cut allowed and forbidden preferred grid extension rules and your block already contains boundary cells, you can create routing guides for these rules by using the `-add_metal_cut_allowed` option with the `derive_metal_cut_routing_guides` command. When you use this option, the `derive_metal_cut_routing_guides` command creates the following routing guides:

- Metal cut allowed routing guides, which cover the area taken up by all the placeable site rows reduced by the vertical shrink factor, which is 50 percent of the smallest site row height
- Forbidden preferred grid extension routing guides, which cover the remaining area up to the block boundary

To check the inserted routing guides, use the `-check_only` option with the `derive_metal_cut_routing_guides` command. By default, the checking results are stored in an error data file named `block_name.err`. To specify a different name for the error data file, use the `-error_view` option.

---

## Controlling Routing Around the Block Boundary

To control routing around the block boundary, use the `derive_perimeter_constraint_objects` command to place rectangular metal shapes,

routing guides, and routing blockages along the boundary edges, as described in the following topics:

- [Inserting Metal Shapes in the Preferred Direction](#)
- [Inserting Routing Guides Along the Nonpreferred-Direction Edges](#)
- [Inserting Routing Blockages Along the Boundary Edges](#)
- [Removing Perimeter Constraint Objects](#)

If you change the block boundary after creating the constraint objects, you must regenerate the objects. You cannot manually modify the constraint objects.

To check the perimeter constraint objects, use the `-check_only` option with the `derive_perimeter_constraint_objects` command. By default, the checking results are stored in an error data file named `block_name.err`. To specify a different name for the error data file, use the `-error_view` option

## See Also

- [Removing Perimeter Constraint Objects](#)

## Inserting Metal Shapes in the Preferred Direction

To insert rectangular metal shapes along the preferred-direction edges of the current block, use the `derive_perimeter_constraint_objects` command with the `-perimeter_objects metal_preferred` option.

When you specify this object type, the command creates one or more rectangular metal shapes parallel to each preferred-direction boundary edge. The metal shapes are floating; they are not connected to any logical nets.

- If your technology does not specify a maximum area for these floating shapes, you can insert continuous metal shapes along the preferred-direction boundary edges, as described in [Inserting Continuous Metal Shapes Parallel to Preferred-Direction Edges](#).
- If your technology specifies a maximum area for these floating shapes or you need more flexibility in placing the metal shapes, insert multiple metal shapes, called metal stubs, along the preferred-direction boundary edges, as described in [Inserting Metal Stubs Parallel to Preferred-Direction Edges](#).

In addition to inserting metal shapes along the preferred-direction edges, you can insert short metal shapes perpendicular to the nonpreferred-direction edges, as described in [Inserting Short Metal Shapes Perpendicular to Nonpreferred-Direction Edges](#).

### Inserting Continuous Metal Shapes Parallel to Preferred-Direction Edges

To insert continuous rectangular metal shapes along the preferred-direction edges of the current block, use the `derive_perimeter_constraint_objects` command with the `-perimeter_objects metal_preferred` option.

When you specify this option, the command creates a single rectangular metal shape parallel to each preferred-direction boundary edge. By default,

- The metal shapes are inserted only in the top-level block
 

To insert metal shapes in all soft macros in a hierarchical block, use the `-hierarchical` option.
- The metal shapes are inserted on all routing layers
 

To insert the metal shapes only on specific routing layers, use the `-layers` option.
- The metal shapes have the default metal width specified for the routing layer in the technology file
 

For information about specifying the default metal width for a layer, see the “Default Width Rule” topic in the *Synopsys Technology File and Routing Rules Reference Manual*.
- The metal shapes abut the nonpreferred-direction edges
 

To specify an offset from the nonpreferred-direction edges, use the `-spacing_from_boundary` option. Use the following format to specify the offset for each layer:

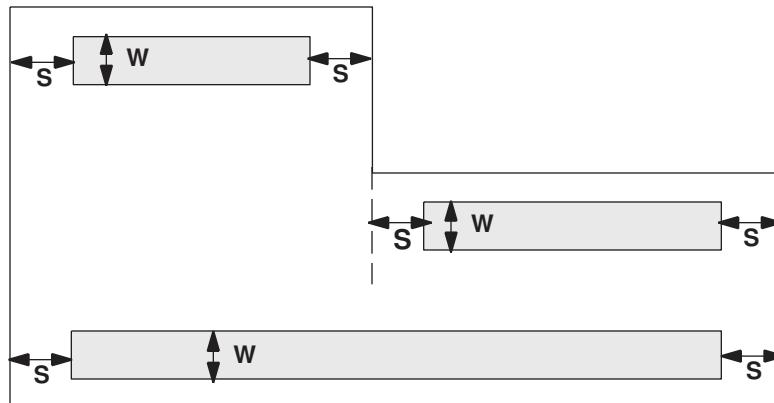
```
{ {layer_name offset_value} ... }
```

The offset value must be a positive value.

- The metal shapes are placed on the closest wire track to the preferred-direction edge

[Figure 86](#) shows the metal shapes inserted for an L-shaped block on a layer whose preferred direction is horizontal. Each metal shape is placed on the closest track to the edge. W is the default width for the layer and S is the spacing specified for the layer by the `-spacing_from_boundary` option.

Figure 86 Continuous Metal Shape Perimeter Constraint Objects



### Inserting Metal Stubs Parallel to Preferred-Direction Edges

To insert metal stubs along the preferred-direction edges of the current block, use the `derive_perimeter_constraint_objects` command with the `-perimeter_objects` `metal_preferred` and `-stub` options.

When you specify both of these options, the command creates rectangular metal shapes parallel to each preferred-direction boundary edge based on the parameters you specify with the `-stub` option. Use the following format to specify the shape parameters in microns:

```
{offset_value start_value length_value spacing_value}
```

By default,

- The metal shapes are offset from the preferred-direction edge by the value specified in the `offset_value` argument

The offset value is measured from the block edge to the center of the metal shape. It can be a positive value, zero, or a negative value. The absolute value of a negative value must be greater than the stub width. A positive value must be less than half of the smaller of the width or height of the block boundary.

- The first metal shape is offset from the nonpreferred-direction edge by the value specified in the `start_value` argument

The start value must be greater than or equal to 0 and less than the smaller of the width or height of the block boundary. For horizontal shapes, the metal shapes start at the left edge. For vertical shapes, the metal shapes start at the bottom edge.

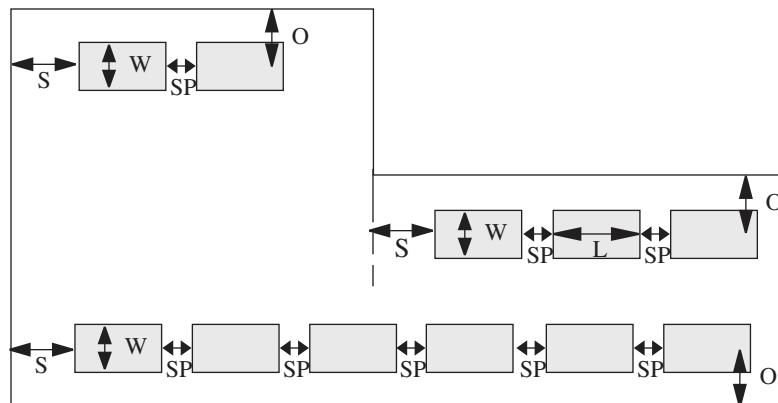
**Note:**

When you use the `-stub` option, the command uses the specified `start_value` and ignores the `-spacing_from_boundary` option.

- The metal shapes have the length specified by the `length_value` argument  
The length value must be greater than 0 and less than or equal to the smaller of the width or height of the block boundary.
- The metal shapes have the default metal width specified for the routing layer in the technology file  
For information about specifying the default metal width for a layer, see the “Default Width Rule” topic in the *Synopsys Technology File and Routing Rules Reference Manual*.
- The metal shapes are separated the space specified by the `spacing_value` argument  
The spacing value must be greater than or equal to 0.
- The metal shapes are inserted only in the top-level block  
To insert metal shapes in all soft macros in a hierarchical block, use the `-hierarchical` option.
- The metal shapes are inserted on all routing layers  
To insert the metal shapes only on specific routing layers, use the `-layers` option.

[Figure 87](#) shows the metal stubs inserted for an L-shaped block on a layer whose preferred direction is horizontal. Each metal shape is placed at the specified offset, O, from the horizontal (preferred-direction) edge, where O is measured from the boundary to the center of the shape. The first metal stub is placed at the specified offset, S, from the vertical (nonpreferred-direction) edge. Each metal shape has a width of W, the default width for the layer and the specified length, L. The spacing between metal stubs is the specified spacing, SP.

Figure 87 Metal Stub Perimeter Constraint Objects



### Inserting Short Metal Shapes Perpendicular to Nonpreferred-Direction Edges

To insert additional short metal shapes in the preferred direction perpendicular to the nonpreferred-direction boundary edges,

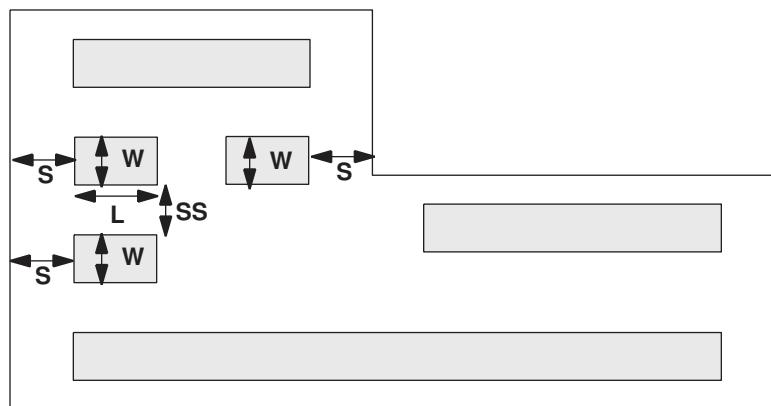
- Specify the length of the metal shapes by using the `-short_metal_length` option
- Specify the distance between the metal shapes by using the `-metal_spacing` option

For both of these options, you use the following format to specify the value for each layer:

```
 {{layer_name value} ...}
```

[Figure 88](#) shows the short metal shapes inserted for an L-shaped block on a layer whose preferred direction is horizontal. Each metal shape has a width of  $W$ , which is the default width for the layer, and a length of  $L$ , which is the length specified for the layer by the `-short_metal_length` option. The offset from the nonpreferred-direction edge is  $S$ , which is the spacing specified for the layer by the `-spacing_from_boundary` option. The distance between short metal shapes is  $SS$ , which is specified for the layer by the `-metal_spacing` option.

*Figure 88 Metal Shape Perimeter Constraint Objects With Additional Short Shapes*



### Inserting Routing Guides Along the Nonpreferred-Direction Edges

To insert maximum pattern routing guides along the nonpreferred-direction edges of the current block, use the `derive_perimeter_constraint_objects` command with the `-perimeter_objects mpdg_nonpreferred` option.

When you specify this object type, the command creates maximum pattern routing guides parallel to each nonpreferred-direction boundary edge. The maximum edge threshold for these routing guides is zero.

By default,

- The routing guides are inserted only in the top-level block

To insert routing guides in all soft macros in a hierarchical block, use the `-hierarchical` option.

- The routing guides are inserted on all routing layers

To insert the routing guides only on specific routing layers, use the `-layers` option.

- The routing guides have the default metal width specified for the routing layer in the technology file

To specify a different width in microns, use the `-width_nonpreferred` option.

For information about specifying the default metal width for a layer, see the “Default Width Rule” topic in the *Synopsys Technology File and Routing Rules Reference Manual*.

- The routing guides abut the preferred-direction edges

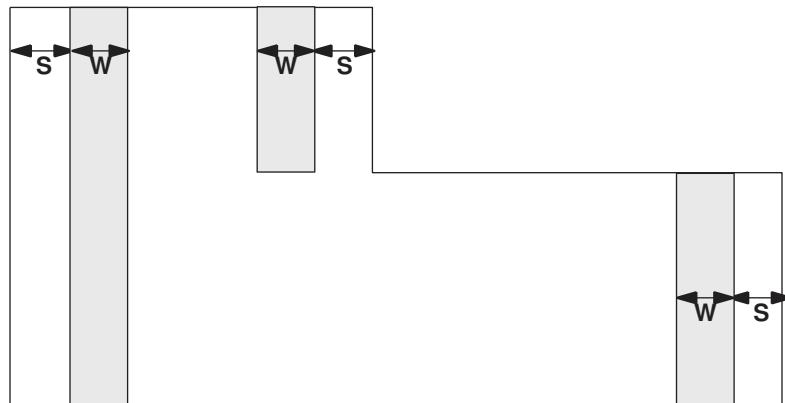
- The routing guides are offset from the nonpreferred-direction edges by the minimum spacing specified for the routing layer in the technology file

To specify a different offset in microns, use the `-spacing_nonpreferred` option.

For information about specifying the minimum spacing for a layer, see the “Minimum Spacing Rule” topic in the *Synopsys Technology File and Routing Rules Reference Manual*.

[Figure 89](#) shows the routing guides inserted for an L-shaped block on a layer whose preferred direction is horizontal. S is the spacing specified by the `-spacing_nonpreferred` option. W is the width specified by the `-width_nonpreferred` option.

*Figure 89 Routing Guide Perimeter Constraint Objects*



#### See Also

- [Using Routing Guides to Limit Edges in the Nonpreferred Direction](#)

### Inserting Routing Blockages Along the Boundary Edges

You can insert routing blockages along the preferred-direction edges, the nonpreferred-direction edges, or all edges.

- To insert routing blockages along the preferred-direction edges of the current block, use the `derive_perimeter_constraint_objects` command with the `-perimeter_objects route_blockage_preferred` option.
- To insert routing blockages along the preferred-direction edges of the current block, use the `derive_perimeter_constraint_objects` command with the `-perimeter_objects route_blockage_nonpreferred` option.
- To insert routing blockages along all edges of the current block, use the `derive_perimeter_constraint_objects` command with the `-perimeter_objects {route_blockage_preferred route_blockage_nonpreferred}` option.

When you specify this object type, the command creates routing blockages parallel to the boundary edges in the specified directions.

By default,

- The routing blockages are inserted only in the top-level block

To insert routing blockages in all soft macros in a hierarchical block, use the `-hierarchical` option.

- The routing blockages are inserted on all routing layers

To insert the routing blockages only on specific routing layers, use the `-layers` option.

- The routing blockages have the default metal width specified for the routing layer in the technology file

To specify a different width in microns for preferred-direction routing blockages, use the `-width_preferred` option. To specify a different width in microns for nonpreferred-direction routing blockages, use the `-width_nonpreferred` option.

For information about specifying the default metal width for a layer, see the “Default Width Rule” topic in the *Synopsys Technology File and Routing Rules Reference Manual*.

- The routing blockages are offset from the boundary edges by the minimum spacing specified for the routing layer in the technology file

To specify a different offset in microns for preferred-direction routing blockages, use the `-spacing_preferred` option. To specify a different offset in microns for nonpreferred-direction routing blockages, use the `-spacing_nonpreferred` option.

For information about specifying the minimum spacing for a layer, see the “Minimum Spacing Rule” topic in the *Synopsys Technology File and Routing Rules Reference Manual*.

- The routing blockages do not have pin cutouts

To generate pin cutouts, use the `-pin_cutout` option.

- Cut layer routing blockages are not created

To create via blockages, specify the via layers in the `-layers` option.

By default, the width of the cut layer routing blockages is the maximum width for the preferred-direction and nonpreferred-direction routing blockages.

- To specify the width for cut layer routing blockages along the horizontal edges, use the `-width_cut_layer_horizontal` option.
- To specify the width for cut layer routing blockages along the vertical edges, use the `-width_cut_layer_vertical` option.

By default, the cut layer routing blockages are offset from the boundary edges by the maximum spacing for the preferred-direction and nonpreferred-direction routing blockages

- To specify the spacing for cut layer routing blockages along the horizontal edges, use the `-spacing_cut_layer_horizontal` option.
- To specify the spacing for cut layer routing blockages along the vertical edges, use the `-spacing_cut_layer_vertical` option.

## Removing Perimeter Constraint Objects

The `derive_perimeter_constraint_objects` command places the inserted constraint objects in an edit group named `PC_Edit_Group_blockname_n`. Each time you run the `derive_perimeter_constraint_objects` command, it creates a unique edit group.

To remove a group of constraint objects, use the following command:

```
fc_shell> remove_objects [get_attribute \
 [get_edit_groups PC_Edit_Group_blockname_n] objects]
```

## Routing Nets Within a Specific Region

To route one or more nets within a specific region,

1. Define a routing corridor by using the `create_routing_corridor` command, as described in [Defining Routing Corridors](#).
2. Assign the nets and supernets to the routing corridor by using the `add_to_routing_corridor` command, as described in [Assigning Nets to a Routing Corridor](#).
3. Verify that the routing corridors are valid by using the `check_routing_corridors` command, as described in [Verifying Routing Corridors](#).  
If necessary, modify the routing corridors, as described in [Modifying Routing Corridors](#).
4. Route the nets by using the `route_group` command, as described in [Routing Critical Nets](#).
5. Remove the routing corridors, as described in [Removing Routing Corridors](#).

### See Also

- [Reporting Routing Corridors](#)

## Defining Routing Corridors

A routing corridor restricts Zroute global routing for specific nets to the region defined by a set of connected rectangles. In addition to specifying the region in which the routing occurs, you can also specify the minimum and maximum routing layers for each of the rectangles that comprise the routing corridor.

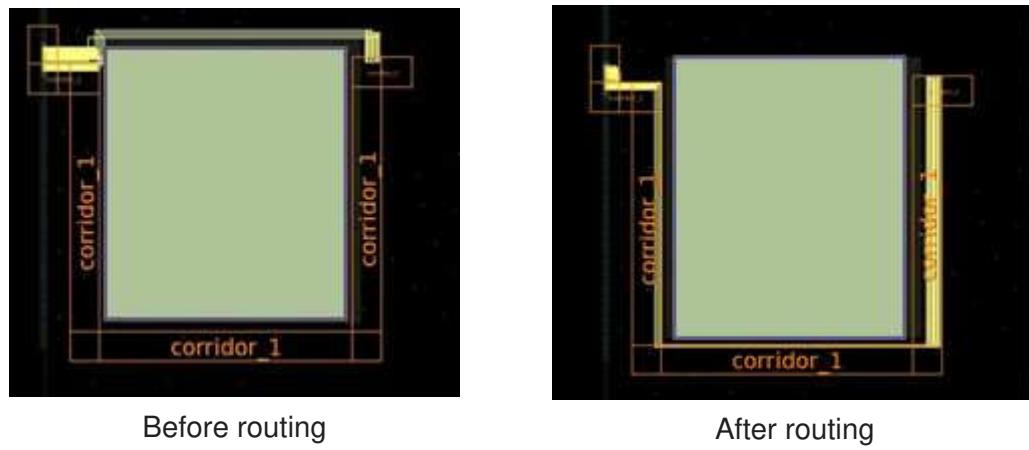
Routing corridors are intended to be used to route critical nets before signal routing. Zroute global routing considers routing corridors as a hard constraint, while track assignment and detail routing consider routing corridors as a soft constraint and might route nets slightly outside of the routing corridor to fix DRC violations.

**Note:**

If a routing guide overlaps with a routing corridor and its attributes conflict with the routing corridor, the routing corridor takes precedence.

For example, [Figure 90](#) shows a routing corridor named `corridor_1`, which is made up of six rectangles. This routing corridor is associated with the nets shown in yellow. The figure on the left shows the nets before routing, while the figure on the right shows the nets routed within the routing corridor.

*Figure 90 Using a Routing Corridor*



To define a routing corridor, use the `create_routing_corridor` command. At a minimum, you must define the boundary of the routing corridor.

- To create a rectangular routing corridor, use the `-boundary` option to specify the lower-left and upper-right corners of the rectangle using the following syntax: `{ {llx lly} {urx ury} }`.
- To create a rectilinear routing corridor, use the `-boundary` option to specify the coordinates of the polygon using the following syntax: `{ {x1 y1} {x2 y2} ... }`.

- To create a path-based routing corridor, use the `-path` option to specify the path and the `-width` option to specify the width in microns. Specify the path using the following syntax: `{ {x1 y1} {x2 y2} ... }`.

By default, path-based routing corridors have flush end caps at the start and end of the path. To change the end cap style, use the `-start_endcap` and `-end_endcap` options. Valid styles are `flush`, `full_width`, and `half_width`.

By default, the tool creates a routing corridor named `CORRIDOR_objId` in the current block that honors the existing minimum and maximum routing layer constraints (for information about specifying routing layer constraints, see [Specifying the Routing Resources](#)). Use the following options to change the default behavior.

- To specify a name for the routing corridor, use the `-name` option.
- To create the routing guide in a different physical cell, use the `-cell` option.

When you use this option, the tool creates the routing guide in the cell's reference block using the coordinate system of the cell's top-level block.

- To specify the minimum and maximum routing layers for the routing corridor, use the `-min_layer_name` and `-max_layer_name` options.

Specify the routing layers by using the layer names from the technology file.

## Assigning Nets to a Routing Corridor

To assign nets and supernets to a routing corridor, use the `add_to_routing_corridor` command. You must specify the routing corridor and the nets or supernets to add to it. You can assign a net or supernet to only one routing corridor and that routing corridor must cover all pins connected to the associated nets. In addition, the supernets must belong to the same block as the routing corridor.

For example, to define a routing corridor named `corridor_a` and assign the nets named `n1` and `n2` to this routing corridor, use the following commands:

```
fc_shell> create_routing_corridor -name corridor_a \
 -boundary { {10 10} {20 35} } \
 -min_layer_name M2 -max_layer_name M4
fc_shell> create_routing_corridor_shape -routing_corridor corridor_a \
 -boundary { {20 25} {40 35} } \
 -min_layer_name M2 -max_layer_name M4
fc_shell> create_routing_corridor_shape -routing_corridor corridor_a \
 -boundary { {40 10} {50 35} } \
 -min_layer_name M2 -max_layer_name M4
fc_shell> add_to_routing_corridor corridor_a [get_nets {n1 n2}]
```

**Note:**

You can also assign nets or supernets to the routing corridor by using the `-object` option when you use the `create_routing_corridor` command to create the routing corridor.

## Verifying Routing Corridors

To successfully route a net within a routing corridor, the routing corridor must meet the following requirements:

- It must be a contiguous region; all regions that comprise the routing corridor must be connected.
- It must contain the pins that connect to the nets and supernets associated with the routing corridor.

To verify that a routing corridor meets these requirements, use the `check_routing_corridors` command.

```
fc_shell> check_routing_corridors RC_0
```

You can view the errors detected by the `check_routing_corridors` command in the message browser.

## Modifying Routing Corridors

You can make the following modifications to an existing routing corridor:

- Add new shapes to the routing corridor.

To add new shapes, use the `create_routing_corridor_shape` command. Use the `-routing_corridor` option to specify the routing corridor that you want to update. You must specify the boundary of the routing corridor by using the `-boundary` or `-path` option (for details about these options, see [Defining Routing Corridors](#)). You can also specify the minimum and maximum routing layers for the shape by using the `-min_layer_name` and `-max_layer_name` options.

- Remove rectangles from the routing corridor.

To remove shapes, use the `remove_routing_corridor_shapes` command.

- Change the nets and supernets associated with the routing corridor.

To add nets or supernets to a routing corridor, use the `add_to_routing_corridor` command. To remove nets or supernets from a routing corridor, use the `remove_from_routing_corridor` command.

You can also modify routing corridors in the GUI by using the Create Route Corridor tool, the Move/Resize tool, or the Delete tool, or by editing the attributes in the Properties dialog box.

## Reporting Routing Corridors

To report the routing corridors in your block, use the `report_routing_corridors` command. By default, the command reports all routing corridors; to report specific routing corridors, specify the routing corridors to report.

By default, the command reports the following information for each routing corridor: its name; the shapes associated with the routing corridor, including their names, minimum routing layer, maximum routing layer, and boundary; the connectivity of the routing corridor shapes; and the nets and supernets associated with the routing corridor. To output a Tcl script that re-creates the routing corridors, use the `-output` option.

To report the routing corridor for a specific net or supernet, use the `get_routing_corridors` command. When you run this command, you must use the `-of_objects` option to specify the nets of interest.

## Removing Routing Corridors

To remove routing corridors from the current block, use the `remove_routing_corridors` command.

- To remove specific routing corridors, specify the routing corridors, either as a list or collection, such as that returned by the `get_routing_corridors` command.
- To remove all routing corridors, specify the `-all` option.

You can also remove routing corridors in the GUI by using the Delete tool.

---

## Using Nondefault Routing Rules

Zroute supports the use of nondefault routing rules, both for routing and for shielding.

- For routing, you can use nondefault routing rules to define stricter wire width and spacing rules, to define the pin tapering distance, to specify the vias used when routing nets with nondefault routing rules, and to specify multiple-patterning mask constraints.
- For shielding, you can use nondefault routing rules to define the minimum width and spacing rules.

For information about working with nondefault routing rules, see the following topics:

- [Defining Nondefault Routing Rules](#)
- [Reporting Nondefault Routing Rule Definitions](#)
- [Removing Nondefault Routing Rules](#)
- [Modifying Nondefault Routing Rules](#)

- [Assigning Nondefault Routing Rules to Nets](#)
- [Reporting Nondefault Routing Rule Assignments](#)

## Defining Nondefault Routing Rules

To define a nondefault routing rule, use the `create_routing_rule` command. When you define a nondefault routing rule, you must specify a name for the nondefault routing rule. You use the name to assign the nondefault routing rule to nets or clocks.

The following topics describe how to create nondefault routing rules for various purposes:

- [Defining Minimum Wire Width Rules](#)
- [Defining Minimum Wire Spacing Rules](#)
- [Defining Minimum Via Spacing Rules](#)
- [Specifying Nondefault Vias](#)
- [Specifying Mask Constraints](#)
- [Defining Shielding Rules](#)
- [Reporting Nondefault Routing Rule Definitions](#)
- [Removing Nondefault Routing Rules](#)
- [Modifying Nondefault Routing Rules](#)

You can create a single routing rule that serves multiple purposes. In addition, you can assign multiple nondefault routing rules to a net.

### Defining Minimum Wire Width Rules

You can define nondefault minimum wire width rules that are stricter than the minimum width rules defined in the technology file. Nondefault minimum width rules are hard constraints, which must be met during routing.

#### Note:

If you specify a nondefault width that violates the `signalRouteMinWidth` setting in the technology file, the tool ignores the nondefault width.

The minimum width defined in a nondefault routing rule applies to all metal segments, including via enclosures. To avoid DRC violations, ensure that the enclosures for nondefault vias meet the minimum width rule.

To define a minimum wire width rule, use the `create_routing_rule` command. You can specify the minimum width by specifying a multiplier that is applied to the default width for each layer, by specifying the minimum width in microns for each layer, or both.

- To use a multiplier to specify the minimum width, use the following syntax:

```
create_routing_rule rule_name
 [-default_reference_rule | -reference_rule_name ref_rule]
 -multiplier_width multiplier
```

The multiplier value must be between 0.001 and 2000. The default width for each layer is determined from the reference rule, which is either the default routing rule or the reference rule specified in the `-reference_rule_name` option.

For example, to define a nondefault routing rule named `new_width_rule` that uses the default routing rule as the reference rule and defines the nondefault width as two times the default width, use the following command:

```
fc_shell> create_routing_rule new_width_rule -multiplier_width 2.0
```

- To specify the minimum width values for each layer, use the following syntax:

```
create_routing_rule rule_name
 [-default_reference_rule | -reference_rule_name ref_rule]
 -widths {layer1 width1 layer2 width2 ... layern widthn}
```

Specify the routing layers by using the layer names from the technology file. You can specify a single width value per layer. Zroute uses the wire width from the reference rule, which is either the default routing rule or the reference rule specified in the `-reference_rule_name` option, for any layers not specified in the `-widths` option.

For example, to define a nondefault routing rule named `new_width_rule2` that uses the default routing rule as the reference rule and defines nondefault width rules for the M1 and M4 layers, use the following command:

```
fc_shell> create_routing_rule new_width_rule2 \
 -widths {M1 0.8 M4 0.9}
```

- If you specify both the `-multiplier_width` option and the `-widths` option, the tool uses the `-widths` option to determine the base width, and then applies the multiplier to that value to determine the minimum width requirement.

For example, to define a nondefault routing rule named new\_width\_rule3 that uses the default routing rule as the reference rule and defines the nondefault width as 0.8 for the M1 layer, 0.9 for the M4 layer, and two times the default width for all other layers, use the following command:

```
fc_shell> create_routing_rule new_width_rule3 \
 -multiplier_width 2.0 -widths {M1 0.4 M4 0.45}
```

### Defining Minimum Wire Spacing Rules

You can define nondefault minimum wire spacing rules that are stricter than the rules defined in the technology file. Nondefault wire spacing rules can be defined as hard constraints, which must be met, or as soft constraints, which Zroute tries to meet.

#### Note:

The spacing rules defined in the technology file are always considered hard constraints.

By default, Zroute checks the nondefault spacing rules between signal nets and other signal nets, PG nets, and blockages, but not between shapes of the same signal net or between signal nets and shield wires for PG nets. For information about modifying these checks, see [Configuring Nondefault Spacing Checks](#).

To define a minimum wire spacing rule, use the `create_routing_rule` command. You can specify the minimum spacing by specifying a multiplier that is applied to the default spacing for each layer, by specifying the minimum spacings in microns for each layer, or both.

- To use a multiplier to specify the minimum spacing, use the following syntax:

```
create_routing_rule rule_name
 [-default_reference_rule | -reference_rule_name ref_rule]
 -multiplier_spacing multiplier
```

The multiplier value must be between 0.001 and 2000. The default wire spacing for each layer is determined from the reference rule, which is either the default routing rule or the reference rule specified in the `-reference_rule_name` option.

For example, to define a nondefault routing rule named new\_spacing\_rule that uses the default routing rule as the reference rule and defines the nondefault spacing as two times the default spacing, use the following command:

```
fc_shell> create_routing_rule new_spacing_rule \
 -multiplier_spacing 2.0
```

- To specify the minimum spacing values for each layer, use the following syntax:

```
create_routing_rule rule_name
 -spacings { layer1 {spacing11 spacing12 ... spacing1n} }
```

```

layer2 {spacing21 spacing22 ... spacing2n}
...
layern {spacingn1 spacingn2 ... spacingnn} }
-spacing_weight_levels { layer1 {weight11 weight12 ... weight1n}
 layer2 {weight21 weight22 ... weight2n}
...
 layern {weightn1 weightn2 ... weightnn} }

```

Specify the routing layers by using the layer names from the technology file. You can define multiple spacing values per layer. Zroute uses the spacing values from the reference rule, which is either the default routing rule or the reference rule specified in the `-reference_rule_name` option, for any layers not specified in the `-spacings` option.

If you specify more than one spacing value per layer, you must assign a weight to each spacing value by using the `-spacing_weight_levels` option. The valid weight values are `low`, `medium`, `high`, and `hard`. When you assign a weight level other than `hard`, the spacing rule is a soft spacing rule. By default, Zroute does not fix soft routing rule violations. To fix soft routing rules, you must map the weight levels to routing effort levels, as described in [Specifying the Routing Effort for Soft Spacing Violations](#).

For example, to define a nondefault routing rule named `new_spacing_rule2` that uses the default routing rule as the reference rule and defines nondefault spacing rules for the M1 and M4 layers, use the following command:

```
fc_shell> create_routing_rule new_spacing_rule2 \
 -spacings { M1 {0.12 0.24} M4 {0.14 0.28} } \
 -spacing_weight_levels { M1 {hard medium} M4 {hard medium} }
```

- If you specify both with `-multiplier_spacing` option and the `-spacings` option, the tool uses the `-spacings` option to determine the base spacing, and then applies the multiplier to that value to determine the minimum wire spacing requirement.

To limit spacing to one side of the net, use the `-single_side_spacing` option with the `create_routing_rule` command.

For example, the following command creates single-side spacing rules on the M2, M3, and M4 metal layers METAL2, METAL3, and METAL4:

```
fc_shell> create_routing_rule new_spacing_rule3 \
 -spacings {M2 1.300 M3 1.400 M4 1.500 } \
 -single_side_spacing \
 -widths {M1 0.230 M2 0.280 M3 0.280 M4 0.280 M5 0.280 M6 0.440}
```

## Configuring Nondefault Spacing Checks

You can configure the checking of nondefault spacing rules by enabling or disabling checks between signal nets and other objects. In addition, you can ignore violations of nondefault spacing rules for short parallel distances.

- To honor nondefault spacing rules between shapes of the same signal net, set the `route.detail.var_spacing_to_same_net` application option to `true`.
- To honor nondefault spacing rules between signal nets and shield wires for PG nets
  - For all signal nets, set the `route.common.ignore_var_spacing_to_shield` application option to `false`
  - For specific signal nets, use the `-ignore_spacing_to_shield false` option when you define the nondefault spacing rule with the `create_routing_rule` command
- To ignore nondefault spacing rules between signal nets and PG nets
  - For all signal nets, set the `route.common.ignore_var_spacing_to_pg` application option to `true`
  - For specific signal nets, use the `-ignore_spacing_to_pg true` option when you define the nondefault spacing rule with the `create_routing_rule` command

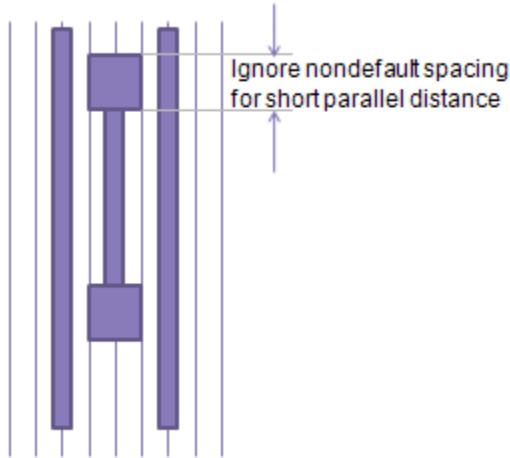
### Note:

When you ignore nondefault spacing rules between signal nets and PG nets, the rules are also ignored between the signal nets and the shield wires regardless of the setting of the `route.common.ignore_var_spacing_to_shield` and `create_routing_rule -ignore_spacing_to_shield` options.

- To ignore nondefault spacing rules between signal nets and blockages
  - For all signal nets, set the `route.common.ignore_var_spacing_to_blockage` application option to `true`
  - For specific signal nets, use the `-ignore_spacing_to_blockage true` option when you define the nondefault spacing rule with the `create_routing_rule` command

You can relax the nondefault spacing checks for specific nondefault routing rules by ignoring violations for short parallel distances, as shown in [Figure 91](#). This technique increases the flexibility of DRC convergence without adversely affecting crosstalk.

*Figure 91 Ignoring Nondefault Spacing Rule Violations*



To define the length thresholds within which to ignore the nondefault spacing violations, use the `-spacing_length_thresholds` option when you create the nondefault routing rule with the `create_routing_rule` command. The length threshold values are in microns and must have a one-to-one correspondence with the spacing entries specified in the `-spacings` option.

For example,

```
fc_shell> create_routing_rule new_rule \
 -spacings { M1 {0.09 0.15 0.2}
 M2 {0.09 0.15 0.2}
 M3 {0.09 0.15 0.2} } \
 -spacing_weight_levels { M1 {hard medium low}
 M2 {hard medium low}
 M3 {hard medium low} } \
 -spacing_length_thresholds { M1 {0.01 0 0}
 M2 {0.01 0 0}
 M3 {0.01 0 0} }
```

### Specifying the Routing Effort for Soft Spacing Violations

By default, Zroute does not fix soft spacing violations. To fix soft spacing violations, you must assign a routing effort to each of the soft weight levels. These assignments apply to all soft spacing rules.

To assign a routing effort for each weight level, use the following syntax to set the `route.common.soft_rule_weight_to_effort_level_map` application option:

```
set_app_options
 -name route.common.soft_rule_weight_to_effort_level_map
 -value { {weight effort} ... }
```

Each `weight` argument can be one of `low`, `medium`, or `high`. You should specify each weight level only one time; if you specify a weight level multiple times, the router uses the last specification. Each `effort` argument can be one of the following values:

- `off` (the default)

Zroute does not fix the soft spacing rule violations.

- `low`

Zroute uses a small number of rip-up and reroute passes to resolve soft spacing rule violations.

- `medium`

Zroute uses a medium number of rip-up and reroute passes to resolve soft spacing rule violations. Note that you cannot specify this effort level for the `low` weight level.

- `high`

Zroute treats soft spacing rule violations the same as regular design rule violations during rip up and reroute. Note that you cannot specify this effort level for the `low` weight level.

For example, to assign low routing effort to low-weight soft spacing rules, medium routing effort to medium-weight soft spacing rules, and high routing effort to high-weight soft spacing, use the following command:

```
fc_shell> set_app_options \
 -name route.common.soft_rule_weight_to_effort_level_map \
 -value { {low low} {medium medium} {high high} }
```

### Defining Minimum Via Spacing Rules

You can define nondefault minimum via spacing rules that are stricter than the rules defined in the technology file. Nondefault via spacing rules are hard constraints that must be met.

To define a minimum via spacing rule, use the `-via_spacings` option with the `create_routing_rule` command using the following syntax:

```
create_routing_rule rule_name
 -via_spacings { {layer1 layer2 spacing} ... }
```

Specify the routing layers by using the layer names from the technology file. You can define a single width value per layer pair.

The minimum via spacing for vias between any unspecified layer combinations is determined from the technology file. For vias on the same layer, the router uses the minimum spacing defined in the `Layer` section for the via layer. For vias on different layers, the router uses the minimum spacing defined in the `DesignRule` section for the via layer combination.

For example, to define a nondefault routing rule named `via_spacing_rule` that defines nondefault spacing rules between vias on the V1 layer and between vias on the V1 and V2 layers, and uses the minimum spacing rules defined in the technology file between vias on all other layer combinations, use the following command:

```
fc_shell> create_routing_rule via_spacing_rule \
 -via_spacings {{V1 V1 2.3} {V1 V2 3.2}}
```

### Specifying Nondefault Vias

By default, when routing nets with nondefault routing rules, Zroute selects vias based on the design rules.

To specify the vias to use when routing nets with nondefault routing rules, use the `create_routing_rule` command. You can define the nondefault vias by using either the `-cuts` option or the `-vias` option.

- When you use the `-cuts` option, the tool determines the suitable via definitions from the technology file based on the specification in the `-cuts` option and the rules defined in the technology file.
- When you use the `-vias` option, you explicitly specify the nondefault via definitions, including the allowed cut numbers and rotation for each via definition.

### Specifying Nondefault Vias Using the `-cuts` Option

The syntax for specifying nondefault vias using the `-cuts` option is

```
create_routing_rule rule_name
 -cuts { {cut_layer1 {cut_name1, ncuts} {cut_name2, ncuts} ...}
 {cut_layer2 {cut_name1, ncuts} {cut_name2, ncuts} ...}
 ...
 {cut_layerN {cut_name1, ncuts} {cut_name2, ncuts} ...}
 }
```

The `cut_layer` arguments refer to the via layer names in the technology file and the `cut_name` arguments refer to the cut names defined in the `cutNameTbl` attribute in the associated `Layer` section in the technology file. You can specify multiple cut names per layer. For each cut name, you must specify the minimum number of cuts, which must be an integer between 1 and 255.

The tool searches for the vias defined in the `ContactCode` section of the technology file that meet the rules defined in the technology file for the specified cut name, such as the `cutWidthTbl`, `cutHeightTbl`, and `fatTblFatContactNumber` rules. In addition, the width of the via enclosure must meet the nondefault width and the `xLegalWidthTbl` and `yLegalWidthTbl` rules defined in the technology file for the adjacent metal layers. If the fat metal contact rule is not defined for a via layer, the tool searches for the default vias that meet the cut width, cut height, and via enclosure width requirements.

The minimum number of cuts required is the larger of the `ncuts` value in the `-cuts` option and the value defined in the `fatTblFatContactMinCuts` attribute. For the selected vias, the tool always allows both the rotated and unrotated orientations for the via.

For example, assume the following information is defined in the technology file:

```
Layer "VIA1" {
 fatTblThreshold = (0, 0.181, 0.411)
 fatTblFatContactNumber = ("2,3,4,5,6 ","5,6,20", "5,6,20")
 fatTblFatContactMinCuts = ("1,1,1,1,1", "1,1,1", "2,2,2")

 cutNameTbl = (Vsq, Vrect)
 cutWidthTbl = (0.05, 0.05)
 cutHeightTbl = (0.05, 0.13)
 ...
}

ContactCode "VIA12_LH" {
 contactCodeNumber = 5
 cutWidth = 0.13
 cutHeight = 0.05
 ...
}
ContactCode "VIA12_LV" {
 contactCodeNumber = 6
 cutWidth = 0.05
 cutHeight = 0.13
 ...
}
ContactCode "VIA12_P" {
 contactCodeNumber = 20
 cutWidth = 0.05
 cutHeight = 0.05
 ...
}
```

If you use the following command,

```
fc_shell> create_routing_rule cut_rule -cuts {VIA1 {Vrect 1}}
```

The tool selects the following vias: {VIA12\_LH 1x1 R}, {VIA12\_LH 1x1 NR}, {VIA12\_LV 1x1 R}, {VIA12\_LV 1x1 NR}, {VIA12\_LH 1x2 R}, {VIA12\_LH 1x2 NR}, {VIA12\_LH 2x1

R}, {VIA12\_LH 2x1 NR}, {VIA12\_LV 1x2 R}, {VIA12\_LV 1x2 NR}, {VIA12\_LV 2x1 R}, and {VIA12\_LV 2x1 NR}.

### Specifying Nondefault Vias Using the `-vias` Option

The syntax for specifying nondefault vias using the `-vias` option is

```
create_routing_rule rule_name
 -vias { {via_type1 cut_number1 orientation1}
 {via_type2 cut_number2 orientation2}
 ...
 {via_typen cut_numbern orientationn} }
```

You can specify multiple via types per layer; each via type must be a via definition defined in the technology file or a via definition created by the `create_via_def` command. For each via type, you must explicitly specify the allowed cut numbers and orientation. To specify the orientation, use `NR` to indicate that the via is not rotated or `R` to indicate that the via is rotated. The order of via specification is not important; during routing, Zroute selects the lowest cost nondefault via.

For example, to specify the vias selected by the `-cuts` option in the previous example, use the following command:

```
fc_shell> create_routing_rule via_rule \
 -vias { {VIA12_LH 1x1 R} {VIA12_LH 1x1 NR} {VIA12_LV 1x1 R}
 {VIA12_LV 1x1 NR} {VIA12_LH 1x2 R} {VIA12_LH 1x2 NR}
 {VIA12_LH 2x1 R} {VIA12_LH 2x1 NR} {VIA12_LV 1x2 R}
 {VIA12_LV 1x2 NR} {VIA12_LV 2x1 R} {VIA12_LV 2x1 NR} }
```

### Specifying Mask Constraints

If you are using the precolored design flow for a design that uses multiple-patterning technology, you can set mask constraints on timing-critical nets, such as clock nets, by defining a precoloring rule and applying it to the nets. For details about the mask constraints, see [Mask Constraints](#).

To define a precoloring rule, use the `create_routing_rule` command using the following syntax:

```
create_routing_rule rule_name
 -mask_constraints
 {layer1 constraint1 layer2 constraint2 ... layern constraintn}
```

where `constraint` is one of `same_mask`, `mask1_soft`, or `mask2_soft`.

For example, to define a precoloring routing rule that sets `mask_one` constraints on the M4 and M5 layers, use the following command:

```
fc_shell> create_routing_rule clock_mask1 \
 -mask_constraints {M4 mask_one M5 mask_one}
```

### Note:

To ensure DRC convergence, you should set double-patterning mask constraints only on a very few timing-critical nets.

### Defining Shielding Rules

To define shielding rules, use the `create_routing_rule` command using the following syntax:

```
create_routing_rule rule_name
 -shield_widths {layer1 width1 layer2 width2 ... layern widthn}
 -shield_spacings {layer1 spacing1 layer2 spacing2 ... layern spacingn}
 [-snap_to_track]
```

Specify the routing layers by using the layer names from the technology file. You can define a single width and spacing value per layer. Zroute uses the default wire width for any layers not specified in the `-shield_widths` option and the default spacing for any layers not specified in the `-shield_spacings` option.

By default, shielding wires are not snapped to the routing tracks. To snap shielding wires to the routing tracks, use the `-snap_to_track` option when you define the nondefault routing rule.

For example, to specify a shielding rule that uses spacing of 0.1 microns and a width of 0.1 microns for M1 through M5 and spacing of 0.3 microns and a width of 0.3 microns for M6, use the following command:

```
fc_shell> create_routing_rule shield_rule \
 -shield_widths {M1 0.1 M2 0.1 M3 0.1 M4 0.1 M5 0.1 M6 0.3} \
 -shield_spacings {M1 0.1 M2 0.1 M3 0.1 M4 0.1 M5 0.1 M6 0.3}
```

### Reporting Nondefault Routing Rule Definitions

To report the nondefault routing rules defined by the `create_routing_rule` command, use the `report_routing_rules -verbose` command. By default, this command reports all of the nondefault routing rules for the current block. To limit the report to specific nondefault routing rules, specify the rule names as an argument to the command.

```
fc_shell> report_routing_rules {rule_names}
```

To output a Tcl script that contains the `create_routing_rule` commands used to define the specified nondefault routing rules, use the `-output` option when you run the `report_routing_rules` command.

### Removing Nondefault Routing Rules

To remove nondefault routing rules from the current block, use the `remove_routing_rules` command. When you remove a nondefault routing rule, the rule

is removed from all nets to which it is applied and the rule definition is removed from the design library.

- To remove specific routing rules, specify the routing rules.
- To remove all routing rules, specify the `-all` option.

For example, to remove the `new_width_rule` routing rule, use the following command:

```
fc_shell> remove_routing_rules new_width_rule
```

### Modifying Nondefault Routing Rules

To change the definition for an existing rule, you must use the `remove_routing_rules` command to remove the rule and then use the `create_routing_rule` command to redefine the rule. The tool issues an error message if you try to redefine an existing routing rule.

## Assigning Nondefault Routing Rules to Nets

The Fusion Compiler tool provides two commands for assigning nondefault routing rules to nets:

- `set_clock_routing_rules`

This command assigns nondefault routing rules to clock nets before clock tree synthesis. During clock tree synthesis and optimization, the tool propagates the nondefault routing rules to the newly created clock nets.

- `set_routing_rule`

This command assigns nondefault routing rules to signal nets and to clock nets after clock tree synthesis.

The following topics describe how to assign nondefault routing rules to nets:

- [Assigning Nondefault Routing Rules to Clock Nets](#)
- [Assigning Nondefault Routing Rules to Signal Nets](#)
- [Reporting Nondefault Routing Rule Assignments](#)

### Assigning Nondefault Routing Rules to Clock Nets

To assign a clock routing rule to a clock net, use the `-rule` option with the `set_clock_routing_rules` command. The specified rule must be a rule that you previously defined with the `create_routing_rule` command.

To reset a clock routing rule to the default routing rule, use the `-default_rule` option. To change a clock routing rule, first reset it to the default routing rule and then use the `-rule` option after resetting the assignment to the default routing rule.

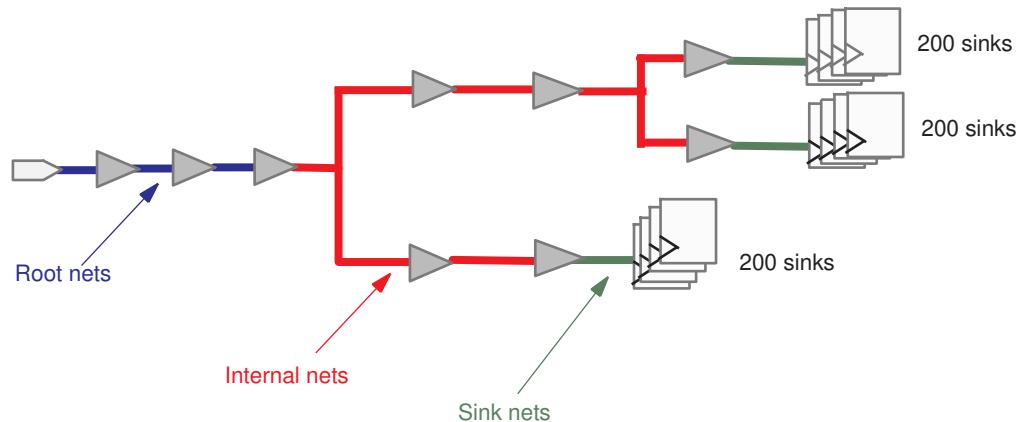
By default, the `set_clock_routing_rules` command assigns the specified clock routing rule to all clock trees in the block. [Table 29](#) shows the options used to restrict the routing rule assignment.

*Table 29      Restricting Clock Routing Rule Assignments*

To assign a nondefault routing rule to	Use this option
Specific clock trees	<code>-clocks clocks</code>
Nets connected to the clock root <sup>2</sup>	<code>-net_type root</code>
Nets connected to one or more clock sinks <sup>2</sup>	<code>-net_type sink</code>
Internal nets in the clock tree (all nets except the root and sink nets) <sup>2</sup>	<code>-net_type internal</code>
Specific clock nets	<code>-nets nets</code>

[Figure 92](#) shows the root, internal, and sink nets of a clock tree after clock tree synthesis. By default, the root-net routing rule is applied to all the single-fanout clock nets starting from the clock root up to the point where the clock tree branches out to a fanout of more than one. Internal-net routing rules are applied to the nets from this point until the sink nets.

*Figure 92      Root, Internal, and Sink Clock Net Types*



2. You can use this option with the `-clocks` option to further restrict the assignment. This option is not valid with the `-nets` option.

The following example specifies routing rules for the root, internal, and sink nets:

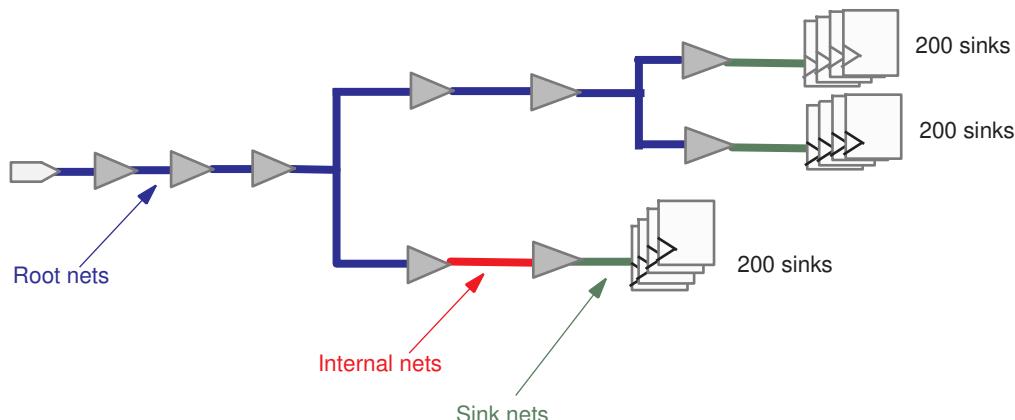
```
fc_shell> set_clock_routing_rules -rules NDR1 -net_type root
fc_shell> set_clock_routing_rules -rules NDR2 -net_type internal
fc_shell> set_clock_routing_rules -rules NDR3 -net_type sink
```

To specify a transitive fanout limit to use when identifying root nets, use the `set_clock_tree_options -root_ndr_fanout_limit` command. For example, to specify that any clock net with a transitive fanout of more than 300 be considered as a root net, use the following command:

```
fc_shell> set_clock_tree_options -root_ndr_fanout_limit 300
```

[Figure 93](#) shows the root, internal, and sink nets of the same clock tree when a transitive fanout limit of 300 is used for identifying the clock root nets.

*Figure 93 Using a Fanout Limit for Selecting Root Nets*



When calculating the transitive fanout of clock nets for the purpose of identifying root nets, the tool includes only the valid clock sinks; It does not include the ignore pins. If a net identified as a root net is less than 10 microns, the tool uses internal-net routing rules for that net.

**Note:**

Specifying a smaller value with the `set_clock_tree_options -root_ndr_fanout_limit` command increases the number of clock nets that are assigned the root-net routing rule, which can increase routing congestion.

During clock tree synthesis and optimization, the tool also honors nondefault routing rules set by using the `set_routing_rule` command. However, the tool does not propagate these routing rules to any new clock nets it creates.

If a net is assigned more than one nondefault routing rule, the tool uses the following priority to determine the effective routing rule:

1. Nondefault routing rule set by the `set_routing_rule` command
2. Net-specific clock routing rule set by the `set_clock_routing_rules -nets` command
3. Clock-specific clock routing rule set by the `set_clock_routing_rules -clocks` command
4. Global clock routing rule set by the `set_clock_routing_rules` command

### Assigning Nondefault Routing Rules to Signal Nets

To assign a nondefault routing rule to a net, use the `-rule` option with the `set_routing_rule` command. The specified rule must be a rule that you previously defined with the `create_routing_rule` command. You must specify the nets to which to assign the nondefault routing rule. You can assign multiple nondefault routing rules to a net.

To change the routing rule assignment for one or more nets,

- Use the `-default_rule` option to reset the nets to the default routing rule.  
 To assign different nondefault routing rules to the nets, use the `-rule` option after resetting the nets to the default routing rule.
- Use the `-no_rule` option to remove all routing rules from the nets and allow the tool to automatically assign a routing rule to them.
- Use the `-clear` option to remove all routing rules and net-specific layer constraints from the nets.

For example, to assign a nondefault routing rule called WideMetal to the CLK net, use the following command:

```
fc_shell> set_routing_rule -rule WideMetal [get_nets CLK]
```

To reset the routing rule for the CLK net to the default routing rule, use the following command:

```
fc_shell> set_routing_rule -default_rule [get_nets CLK]
```

## Reporting Nondefault Routing Rule Assignments

The Fusion Compiler tool provides commands to report the nondefault routing rules assigned by the `set_routing_rule` command and the clock routing rules assigned by the `set_clock_routing_rules` command.

- To report the nondefault routing rules assigned by the `set_routing_rule` command, use the `-of_objects` option with the `report_routing_rules` command.

```
fc_shell> report_routing_rules -of_objects [get_nets *]
```

- To report the clock routing rule assignments, use the `report_clock_routing_rules` command. This command reports only the routing rules assigned by the `set_clock_routing_rules` command.

## Controlling Off-Grid Routing

You can prevent off-grid routing of wires or vias by requiring them to be aligned to the wire track or discourage off-grid routing of vias by increasing the cost associated with off-grid routing.

### Preventing Off-Grid Routing

By default, wires and vias need to be aligned to the wire track grid for a metal layer only if the `onWireTrack` or `onGrid` attribute is set to 1 in its `Layer` section in the technology file.

To override the technology file settings, set the following application options:

- `route.common.wire_on_grid_by_layer_name`  
This option controls off-grid routing for metal layers.
- `route.common.via_on_grid_by_layer_name`  
This option controls off-grid routing for via layers.

Use the following syntax to set these options:

```
{ {layer true|false} ... }
```

Specify the layers by using the layer names from the technology file. Specify `true` to forbid off-grid routing and `false` to allow off-grid routing.

If you use either of these options, the tool ignores all settings for the `onWireTrack` and `onGrid` attributes in the technology file and uses only the settings specified by these options. If you do not specify a layer in these options, off-grid routing is allowed on that layer, regardless of the setting in the technology file.

For example, to prevent off-grid routing for wires on the M2 and M3 metal layers and for vias on the V2 via layer, regardless of the settings in the technology file, use the following commands:

```
fc_shell> set_app_options \
 -name route.common.wire_on_grid_by_layer_name \
 -value {{M2 true} {M3 true}}
fc_shell> set_app_options \
 -name route.common.via_on_grid_by_layer_name \
 -value {{V2 true}}
```

## Discouraging Off-Grid Routing for Vias

To discourage off-grid routing for vias, you can increase the cost of routing the via enclosure metal shapes off the wire track grid. To specify the extra cost multiplier for the metal layers on which the via enclosures are routed, set the `route.common.extra_via_off_grid_cost_multiplier_by_layer_name` application option.

Use the following syntax to set this option:

```
{ {layer multiplier} ... }
```

Specify the layers by using the layer names from the technology file. The cost multiplier must be a value between 0.0 and 20.0.

When you specify this option, the effective cost is the base cost times (1+multiplier). For example, assume that the technology file defines the VIA12 layer between the M1 and M2 metal layers and the VIA23 via layer between the M2 and M3 metal layers. To set the extra cost multiplier for the via enclosures on the M2 metal layer (and therefore the vias on the VIA12 and VIA23 via layers) to 0.5 (for an effective via cost of 1.5 times the base cost), use the following command:

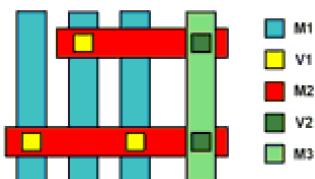
```
fc_shell> set_app_options \
 -name route.common.extra_via_off_grid_cost_multiplier_by_layer_name \
 -value {{M2 0.5}}
```

## Routing Must-Join Pins

When a pin is defined as a must-join pin, the router connects all terminals of the pin as a single net. The terminals of a must-join pin are specified with the `must_join_port` library pin attribute.

By default, Zroute connects must-join pins using a random structure, as shown in the following figure:

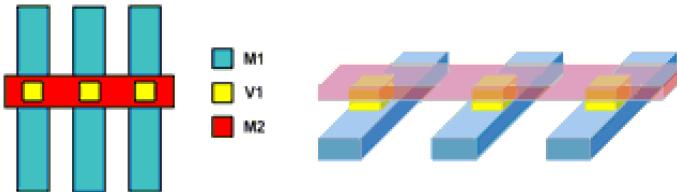
*Figure 94 Default Must-Join Pin Connection*



To achieve better electromagnetic results, Zroute can use the via ladder insertion capabilities to connect the must-join pins with a simplified structure, which is referred to as a pattern-based must-join connection.

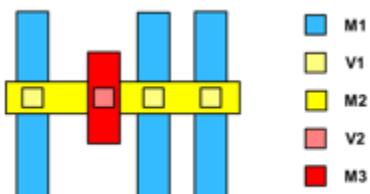
The following figure shows the simplified structure:

*Figure 95 Single-Level Pattern-Based Must-Join Pin Connection*



The single-level structure shown in [Figure 95](#) is the default structure used for a pattern-based must-join connection. To use a multi-level structure, which is shown in [Figure 96](#), set the `route.auto_via_ladder.pattern_must_join_over_pin_layer` application option to 2.

*Figure 96 Multi-Level Pattern-Based Must-Join Pin Connection*



Zroute automatically uses a pattern-based must-join pin connection when a library cell pin has a `pattern_must_join` attribute and you use one of the following commands to perform the routing: `route_auto`, `route_group`, or `route_eco`.

To query cells with the `pattern_must_join` attribute, use the following command:

```
fc_shell> get_attribute \
 [get_lib_pins -all -of_objects */*/frame] pattern_must_join
```

To get a list of pins that are marked with the `pattern_must_join` attribute, use the following command:

```
fc_shell> get_lib_pins -all \
 -of_objects */*/frame -filter "pattern_must_join==true"
```

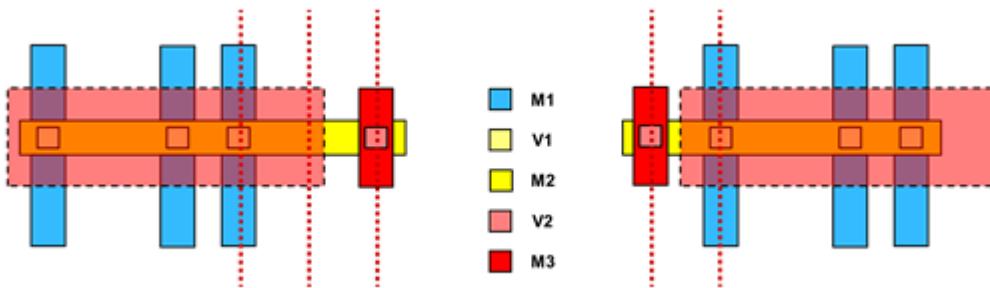
By default, a via ladder is not inserted if there are fixed shapes on higher layers that block the insertion. To increase the insertion rate, you can allow staggering of the vias on each level of the via ladder to avoid the fixed shapes. To specify the maximum number of tracks allowed for staggering on each level, set the `route.auto_via_ladder.pattern_must_join_max_number_stagger_tracks` application option. You can specify a value between 0 and 9 for each level; the default is 0, which disables staggering.

For example, to allow staggering of up to three tracks on the first level, use the following command:

```
fc_shell> set_app_options -name \
 route.auto_via_ladder.pattern_must_join_max_number_stagger_tracks \
 -value {3 0}
```

[Figure 97](#) shows the possible solutions to avoid a blockage on the M3 layer when you allow staggering of up to three tracks.

*Figure 97 Pattern-Based Must-Join Pin Connection With Staggering*



**Note:**

To ensure that it can successfully create pattern-must-join pin connections, Zroute considers the following application options to have a value of `true` when creating these connections, regardless of their actual setting:

```
route.auto_via_ladder.allow_patching
route.auto_via_ladder.ignore_routing_shape_drcs
route.auto_via_ladder.relax_line_end_via_enclosure_rule
route.auto_via_ladder.relax_pin_layer_metal_spacing_rules
```

By default, the pattern-based must-join connections are updated each time you run the `route_auto`, `route_group`, or `route_eco` command.

- During the `route_group` command, you can restrict the updates to only those nets specified in the `route_group` command by setting the `route.auto_via_ladder.update_on_route_group_nets_only` application option to `true`.
- To disable all updates during each of these routing commands, set the `route.auto_via_ladder.update_pattern_must_join_during_route` application option to `false`.

If Zroute cannot create a pattern-based must-join connection, it reports a “Needs pattern must join pin connection” DRC violation.

## See Also

- [Inserting Via Ladders](#)

## Controlling Pin Connections

By default, Zroute connects a signal route to a pin by using wires or vias anywhere on the pin. To restrict the allowed types of pin connections on a per-layer basis, use the following syntax to set the `route.common.connect_within_pins_by_layer_name` application option:

```
set_app_options
 -name route.common.connect_within_pins_by_layer_name
 -value { {layer mode} ... }
```

Valid values for the *mode* argument are

- `off` (the default)

There are no restrictions on pin connections.

- `via_standard_cell_pins`

Only the connections to standard cell pins by using a via are restricted. When using a via connection, the via's metal enclosure must be contained within the pin shape.

There are no restrictions on signal routes connected to macro cell and pad cell pins by using a via or to any pins by using wires.

- `via_wire_standard_cell_pins`

The connections to standard cell pins by using a via or a wire are restricted. When using a via connection, the via's metal enclosure must be contained within the pin shape. When using a wire, the wire must be contained within the pin shape.

- `via_all_pins`

The connections to any pins (standard cell, macro cell, or pad cell) by using a via are restricted. When using a via connection, the via's metal enclosure must be contained within the pin shape.

There are no restrictions on signal routes connected to any pins by using wires.

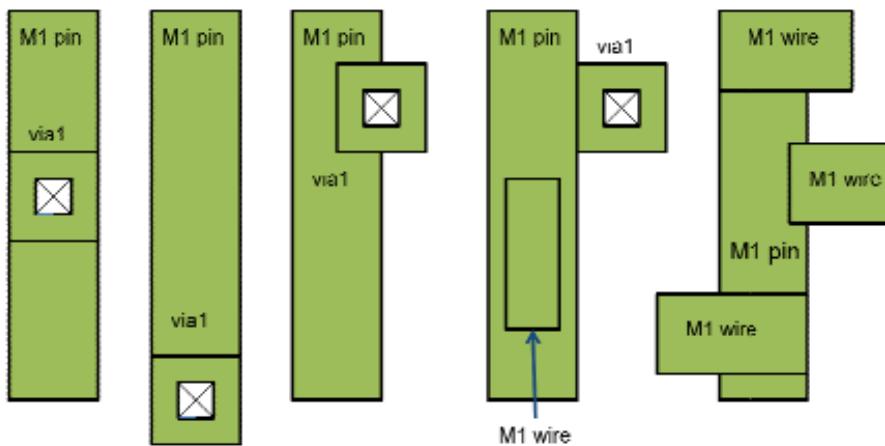
- `via_wire_all_pins`

The connections to any pins by using a via or a wire are restricted. When using a via connection, the via's metal enclosure must be contained within the pin shape. When using a wire, the wire must be contained within the pin shape.

For example, if you use the following command (or use the default settings), all of the connections shown in [Figure 98](#) are valid and no DRC violations are reported:

```
fc_shell> set_app_options \
 -name route.common.connect_within_pins_by_layer_name \
 -value {{M1 off}}
```

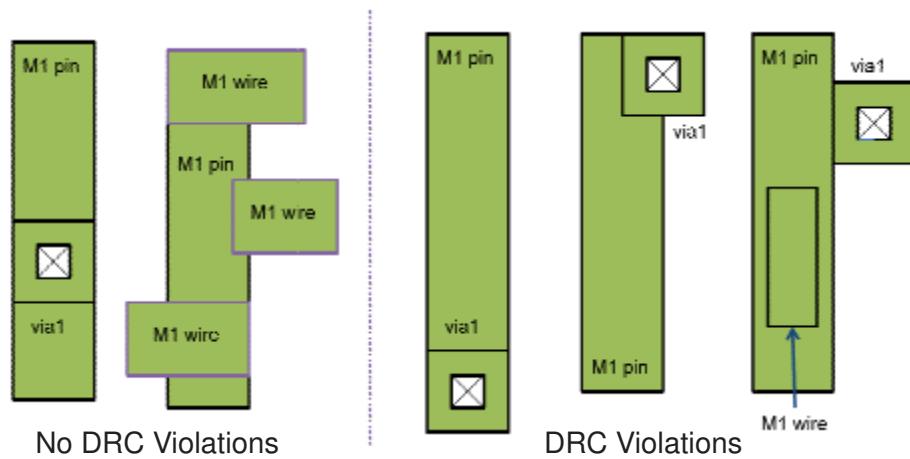
*Figure 98 Unrestricted Pin Connections*



If you set the mode for M1 to `via_all_pins`, as shown in the following example, the via enclosures must be inside the pin shape. The connections shown on the left side of [Figure 99](#) are valid; however, the connections on the right side of the figure cause DRC violations.

```
fc_shell> set_app_options \
 -name route.common.connect_within_pins_by_layer_name \
 -value {{M1 via_all_pins}}
```

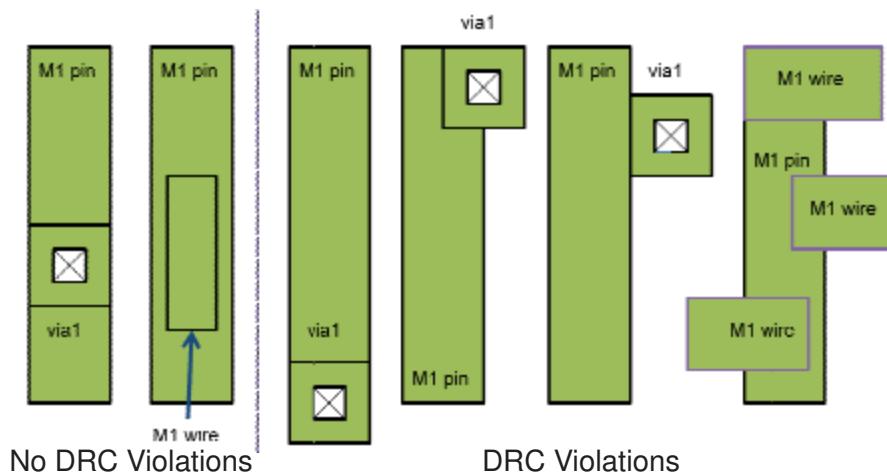
Figure 99 Restricted Via-to-Pin Connections



If you set the mode for M1 to `via_wire_standard_cell_pins`, as shown in the following example, both the via enclosures and wires must be inside the pin shape. The connections shown on the left side of Figure 100 are valid; however the connections on the right side of the figure cause DRC violations.

```
fc_shell> set_app_options \
 -name route.common.connect_within_pins_by_layer_name \
 -value {{M1 via_wire_standard_cell_pins}}
```

Figure 100 Restricted Via-to-Pin and Wire-to-Pin Connections



---

## Controlling Pin Tapering

Pin tapering is the method used to connect wires with nondefault routing rules to pins. Zroute supports pin tapering for both hard and soft nondefault routing rules; the tapering implementation is the same for both types of nondefault routing rules.

**Note:**

By default, if a net has both a nondefault routing rule defined by the `create_routing_rule` command and voltage-based spacing rules defined in the technology file, Zroute performs pin tapering on that net based on the nondefault routing rule. To disable pin tapering on these nets, set the `route.detail.enable_nrd_tapering_on_voltage_rule` application option to false.

You can specify the method used for pin tapering and control the tapering width, as described in the following topics:

- [Specifying the Tapering Method](#)
- [Controlling the Tapering Width](#)

### Specifying the Tapering Method

You specify the tapering method for a nondefault routing rule when you define the rule with the `create_routing_rule` command. By default, Zroute uses distance-based pin tapering; it uses the default routing rule within the tapering distance from the pin and uses the nondefault routing rule beyond the tapering distance. However, advanced process nodes are very sensitive to jogs and fat metal, and sometimes the tapering distance is not sufficient to fix the routing DRC violations on nets with nondefault routing rules. In these cases, you can use layer-based tapering, which targets DRC violations on nets with nondefault routing rules.

**Note:**

Distance-based tapering and layer-based tapering are mutually exclusive. If you define a routing rule that uses both a distance-based tapering option and a layer-based tapering option, the tool uses the layer-based tapering settings and ignores the distance-based tapering settings.

When Zroute performs distance-based pin tapering, it

- Determines the tapering distance, which is about 10 times the mean number of tracks for all routing layers.

To explicitly specify the tapering distance, use the `-taper_distance` option when you create a nondefault routing rule with the `create_routing_rule` command.

- Uses the same tapering distance for all pins.

To specify a different tapering distance for driver pins, use the `-driver_taper_distance` option when you create a nondefault routing rule with the `create_routing_rule` command.

To perform layer-based pin tapering, use the `-taper_over_pin_layers` or `-taper_under_pin_layers` option when you create a nondefault routing rule with the `create_routing_rule` command. To specify a different layer-based tapering distance for driver pins, use the `-driver_taper_over_pin_layers` or `-driver_taper_under_pin_layers` option.

- For pins on the M1 or M2 layers, use the `-taper_over_pin_layers` option (or the `-driver_taper_over_pin_layers` option for driver pins) to specify the number of layers on or above the pin layer available for tapering. A value of 1 enables pin tapering only on the pin layer; a larger value enables pin tapering on additional layers above the pin layer.
- For pins on upper layers, use the `-taper_under_pin_layers` option (or the `-driver_taper_under_pin_layers` option for driver pins) to specify the number of layers on or below the pin layer available for tapering. A value of 1 enables pin tapering only on the pin layer; a larger value enables pin tapering on additional layers below the pin layer.

## Controlling the Tapering Width

By default, the wire is tapered to the default routing width, which is the routing width that is defined for the metal layer in the technology file.

- To taper the wire to the pin width rather than the default routing width, change the `route.detail.pin_taper_mode` application option to `pin_width` from its default of `default_width` before you perform detail routing.
- To enable pin tapering only when required to avoid DRC violations, set the `route.detail.use_wide_wire_effort_level` application option to either `low` or `high`.

This setting improves the nondefault via rate at a cost of longer runtime. You should use this option only when the majority of pins on the nets being routed are accessible with nondefault vias.

- To disable tapering for certain types of pins, set one or more of the following application options to `true`: `route.detail.use_wide_wire_to_input_pin`, `route.detail.use_wide_wire_to_output_pin`, `route.detail.use_wide_wire_to_macro_pin`, `route.detail.use_wide_wire_to_pad_pin`, and `route.detail.use_wide_wire_to_port`.
- To disable tapering for all pins, set the `-taper_distance` option to 0 when you create the nondefault routing rule with the `create_routing_rule` command.

## Controlling Via Ladder Connections

If a pin has a via ladder, by default, Zroute connects to the topmost layer of a via ladder and not directly to the existing pin geometry. To modify the connection constraints, set the following application options:

- `route.detail.via_ladder_upper_layer_via_connection_mode`  
To allow vias to connect to the top level of a via ladder from below, set this application option to `any`.
- `route.detail.allow_default_rule_nets_via_ladder_lower_layer_connection`  
To allow nets with default routing rules to connect directly to the pin or to any layer of the via ladder, set this application option to `true`.

## Setting the Rerouting Mode

By default, Zroute can reroute nets as needed. You can prevent rerouting or limit rerouting to minor changes by setting the `physical_status` attribute on the nets.

- To freeze the net and prevent rerouting, set the attribute to `locked`.
- To limit rerouting to minor changes, set the attribute to `minor_change`.
- To allow Zroute to reroute the nets as needed, set the attribute to `unrestricted`.

For example, to prevent rerouting of the `net1` net, which uses the default routing rule, use the following command:

```
fc_shell> set_attribute -objects [get_nets net1] \
 -name physical_status -value locked
```

## Routing Application Options

The Fusion Compiler tool provides application options that affect the individual routing engines (global routing, track assignment, and detail routing), as well as application options that affect all three routing engines.

- For information about the application options that affect all three routing engines, see the `route.common_options` man page.
- For information about the application options that affect global routing, see the `route.global_options` man page.
- For information about the application options that affect track assignment, see the `route.track_options` man page.
- For information about the application options that affect detail routing, see the `route.detail_options` man page.

Zroute uses these option settings whenever you perform routing functions. When you run a routing command, Zroute writes the settings for any routing options that you have set (or that the tool has set for you) in the routing log. To display the settings for all routing options, not only those that have been set, set the `route.common.verbose_level` application option to 1.

```
fc_shell> set_app_options \
 -name route.common.verbose_level -value 1
```

## Routing Clock Nets

To route clock nets before routing the rest of the nets in the block, use the `route_group` command with the appropriate option to select the nets.

- To route all clock nets, use the `-all_clock_nets` option.
- To route specific clock nets, use the `-nets` option.

For example, to route all clock nets, use the following command:

```
fc_shell> route_group -all_clock_nets
```

The `route_group` command runs global routing, track assignment, and detail routing on the clock nets.

- When routing clock nets, the `route_group` command uses A-tree routing to minimize the distance between each pin and driver.
- When routing clock mesh nets, by default, the `route_group` command uses Steiner routing, which minimizes the total wire length. To use comb routing, set the

`route.common.clock_topology` application option to `comb` before routing the clock mesh nets.

- By default, if the block contains existing global routes, the `route_group` command ignores them during global routing. To perform incremental global routing by reusing existing global routes, use the `-reuse_existing_global_route true` option.
- If the block contains existing detail routes for the clock nets, the `route_group` command performs incremental detail routing.
- By default, the detail router performs a maximum of 40 search and repair iterations. To modify the maximum number of detail routing iterations, use the `-max_detail_route_iterations` option.

**Note:**

Zroute stops before completing the maximum number of iterations if it determines that all violations have been fixed or that it cannot fix the remaining violations.

## Routing Critical Nets

To route a group of critical nets before routing the rest of the nets in the block, use the `route_group` command. To specify the nets to route, use one of the following options:

- `-nets`

Use this option to specify the nets on the command line:

```
fc_shell> route_group -nets collection_of_critical_nets
```

- `-from_file`

Use this option to specify the nets in a file:

```
fc_shell> route_group -from_file file_name
```

If the specified nets are associated with a routing corridor, the nets are routed within the defined region. Note that global routing considers routing corridors as a hard constraint, while track assignment and detail routing consider routing corridors as a soft constraint and might route nets slightly outside of the routing corridor to fix DRC violations.

By default, the `route_group` command ignores existing global routes, reuses dangling wires and existing detail routes on the specified nets, and runs global routing, track assignment, and detail routing on the specified nets.

- To perform incremental global routing, set the `-reuse_existing_global_route` option to `true`.

**Note:**

You cannot use the `-reuse_existing_global_route true` option when routing the nets in a routing corridor. If you use this option, the global router ignores the routing corridors.

- To disable the reuse of dangling wires, set the `-utilize_dangling_wires` option to `false`.
- To stop after global routing, set the `-stop_after_global_route` option to `true`.

By default, the detail router performs a maximum of 40 iterations search and repair iterations. If Zroute determines before then that all violations have been fixed or that it cannot fix the remaining violations, it stops. To change the maximum number of detail routing iterations, use the `-max_detail_route_iterations` option.

By default, the `route_group` command does not fix soft DRC violations, such as bridge rule violations. To enable the fixing of soft DRC violations after the final detail routing iteration, set the `route.common.post_group_route_fix_soft_violations` application option to `true`.

```
fc_shell> set_app_options \
 -name route.common.post_group_route_fix_soft_violations \
 -value true
```

## Routing Secondary Power and Ground Pins

To use Zroute to perform secondary power and ground pin routing,

1. Verify that the secondary power and ground pins have the appropriate attributes in the standard cell frame views.
2. Set the routing constraints for secondary power and ground pin routing.
3. Perform secondary power and ground pin routing by using the `route_group` command.

The following topics describe these steps.

## Verifying the Secondary Power and Ground Pin Attributes

Before you use Zroute to perform secondary power and ground pin routing, you must verify that the cell libraries have the correct attributes on the secondary power and ground pins of the standard cell frame views.

The following attributes are required for secondary power and ground pins:

- `is_secondary_pg`

This attribute must have a setting of `true`.

- `port_type`

This attribute must have a setting of `power` or `ground`.

To verify that these attributes are correctly set on the library pins in the standard cell frame view, use the following command:

```
fc_shell> report_attributes -application \
 [get_lib_pins -of_objects reflib/cell/frame -all \
 -filter "name == attr_name"]
```

where `reflib` is the cell library name, `cell` is the name of the library cell you want to check (or `*` if you want to check all library cells), and `attr_name` is either `is_secondary_pg` or `port_type`.

### See Also

- [Identifying Secondary PG Pins](#)

## Setting the Routing Constraints

You can set routing constraints for secondary power and ground pin routing in the same way as for regular signal routing. For example, you can set constraints by

- Defining the minimum and maximum routing layers by using the `set_routing_rule` command

For more information about using the `set_routing_rule` command, see [Specifying Net-Specific Layer Constraints](#).

- Specifying the preferred pin connections by setting the `route.common.single_connection_to_pins` and `route.common.connect_within_pins_by_layer_name` application options

For example, to require a single connection to the secondary power and ground pins and require that the M1 connections use vias contained within the pin shapes, use the following command:

```
fc_shell> set_app_options \
 -name route.common.single_connection_to_pins \
 -value standard_cell_pins
fc_shell> set_app_options \
 -name route.common.connect_within_pins_by_layer_name \
 -value {{M1 via standard_cell_pins}}
```

- Defining the maximum number of power or ground pins in a cluster by setting the `route.common.number_of_secondary_pg_pin_connections` application option

A cluster is a set of connected secondary power or ground pins that has one connection to a PG strap or ring. By default, the value of this option is 0, which means that there is no limit on the number of secondary power or ground pins in a cluster.

For example, to connect all secondary power and ground pins directly to a PG strap or ring, use the following command:

```
fc_shell> set_app_options \
 -name route.common.number_of_secondary_pg_pin_connections \
 -value 1
```

- Defining a nondefault routing rule for secondary power and ground pin routing

For example, to define a nondefault routing rule named wideSVDD for wide M1 and M2 and set the nondefault routing rule on the VDD2 net, to which the secondary power and ground pins are connected, use the following commands:

```
fc_shell> create_routing_rule wideSVDD -widths { M1 0.3 M2 0.3 }
fc_shell> set_routing_rule -rule wideSVDD {VDD2}
```

For more information about using nondefault routing rules, see [Using Nondefault Routing Rules](#).

By default, the constraints apply to both the secondary power and ground connections and the tie-off connections. To separate these connections so that you can set constraints only for the secondary power and ground connections, set the `route.common.separate_tie_off_from_secondary_pg` application option to `true`.

---

## Routing the Secondary Power and Ground Pins

If the secondary power and ground pins have the appropriate attributes in the frame view, you can use Zroute to route the secondary power and ground pins.

By default, Zroute performs four topology ECO iterations to fix secondary PG cluster fanout violations. The topology ECO iterations can be time consuming, so you can control the effort level, and therefore the runtime, by setting the `route.detail.topology_eco_effort_level` application option to one of the following values:

- `off`, which disables topology ECO
- `low`, which performs one topology ECO iteration
- `medium`, which performs four topology ECO iterations; this is the default
- `high`, which performs eight topology ECO iterations

For example, to connect the secondary power pins to the VDD1 net, run the following command:

```
fc_shell> route_group -nets VDD1
```

The following example adds new cells, which have secondary power and ground pins, to a block; logically connects the power and ground pins; and then connects the secondary power pins to the VDD2 net.

```
fc_shell> add_buffer {TOP/U1001/Z} {libA/BUFHVT} \
 -new_cell_names mynewHVT
fc_shell> add_buffer {TOP/U1002/Z} {libA/BUFHVT} \
 -new_cell_names mynewHVT
fc_shell> legalize_placement
fc_shell> connect_pg_net -automatic
fc_shell> route_group -nets {VDD2}
```

## Routing Signal Nets

Before you route the signal nets, all clock nets must be routed without violations.

You can route the signal nets by using one of the following methods:

- Use the task-specific commands to perform the standalone routing tasks.

- To perform global routing, use the `route_global` command.

For details see, [Global Routing](#).

- To perform track assignment, use the `route_track` command.

For details see, [Track Assignment](#).

- To perform detail routing, use the `route_detail` command.

For details see, [Detail Routing](#).

When you run a standalone routing command, such as `route_global` or `route_detail`, Zroute reads in the block at the beginning of each routing command and updates the block at the end of each command. The router does not check the input data. For example, if the track assignment step is skipped and you run detail routing directly, Zroute might generate bad routing results.

If you need to customize your routing flow or you need to run a large block step-by-step, you might want to use the standalone routing commands instead of automatic routing.

- Use automatic routing (the `route_auto` command).

The `route_auto` basic command performs global routing, track assignment, and detail routing. For details, see [Routing Signal Nets by Using Automatic Routing](#).

When you run `route_auto`, Zroute reads the block before starting routing and updates the block when all routing steps are done. If you stop automatic routing before it performs detail routing, Zroute checks the input data when you restart routing with this command.

Use the `route_auto` command when you run routing to verify convergence, congestion, and design rule quality-of-results (QoR). You also might want to use `route_auto` if congestion QoR is your main goal, rather than timing QoR.

Zroute can insert redundant vias during signal routing. For information about this capability, see [Inserting Redundant Vias on Signal Nets](#).

---

## Global Routing

Before you run global routing,

- Define the common routing application options

For information about the common routing application options, see the `route.common_options` man page.

- Define the global routing application options

For information about the global routing application options, see the `route.global_options` man page.

To perform standalone global routing, use the `route_global` command. By default, global routing is not timing-driven. For information about enabling timing-driven global routing, see [Timing-Driven Global Routing](#).

The global router divides a block into global routing cells. By default, the width of a global routing cell is the same as the height of a standard cell and is aligned with the standard cell rows.

For each global routing cell, the routing capacity is calculated according to the blockages, pins, and routing tracks inside the cell. Although the nets are not assigned to the actual wire tracks during global routing, the number of nets assigned to each global routing cell is noted. The tool calculates the demand for wire tracks in each global routing cell and reports the overflows, which are the number of wire tracks that are still needed after the tool assigns nets to the available wire tracks in a global routing cell.

For advanced node designs, via density can be a concern. To enable via density modeling, set the `route.global.via_cut_modeling` application option to `true`. The tool then calculates the number of vias in each global routing cell and reports via overflows.

Global routing is done in two phases:

- The initial routing phase (phase 0), in which the tool routes the unconnected nets and calculates the overflow for each global routing cell
- The rerouting phases, in which the tool tries to reduce congestion by ripping up and rerouting nets around global routing cells with overflows

The tool might perform several rerouting phases. At the end of each rerouting phase, the tool recalculates the overflows. You should see a reduction in the total number of global routing cells with overflow and in the total overflow numbers. The global router stops and exits from the rerouting phase when the congestion is solved or cannot be solved further or after the maximum number of phases has occurred, as defined by the `-effort_level` option. You can force the global router to perform the maximum number of phases based on the specified effort level by setting the `route.global.force_full_effort` application option to `true`. By default, the tool

uses medium effort and performs a maximum of three rerouting phases. You can perform up to six rerouting phases by specifying ultra effort.

There are five global routing effort levels: minimum, low, medium, high, and ultra.

- Minimum (`-effort_level minimum`)

The minimum effort level uses two times larger global routing cells relative to the other effort levels. It also has a much lower congestion cost and runs only one rerouting phase. It should only be used for prototype routing or for an initial congestion evaluation, not for detail routing.

- Low (`-effort_level low`)

Low effort runs a maximum of two rerouting phases with very similar congestion cost. It is faster in comparison to medium effort and has reasonable QoR. If your block is not very congested, you can use the low effort level.

- Medium (`-effort_level medium`)

Medium effort is the default effort level and runs a maximum of three rerouting phases. Global routing stops after the third phase or when the overflow is resolved, whichever occurs first.

- High (`-effort_level high`)

High effort runs up to four rerouting phases. If your block is congested, use the high effort level.

- Ultra (`-effort_level ultra`)

Ultra effort runs up to six rerouting phases. If your block is very congested, use the ultra effort level.

At the end of global routing, the following information is stored in the design library:

- The g-links and g-vias on each routed net

This information is used for the next routing steps. After Zroute performs track assignment and detail routing, it removes these g-links and g-vias from the design library.

- The congestion data

This information is used to generate a congestion map. By default, only the hard congestion data is saved in the design library. To also save the soft congestion data, set the `route.global.export_soft_congestion_maps` application option to `true` before performing global routing. The soft congestion data includes demand from soft nondefault spacing rules, as well as tool-generated soft rules.

The global router reports block statistics and congestion data after the initial routing phase and after each rerouting phase. When global routing is complete, the global router reports a summary of the wire length and via count.

**Example 18** shows a global routing report. In the congestion report, the Overflow value is the total number of wires in the block that do not have a corresponding track available. The Max value corresponds to the highest number of overutilized wires in a single global routing cell. The GRCs value is the total number of overcongested global routing cells in the block.

#### **Example 18 Global Routing Report**

```

Start Global Route ...
...
Design statistics:
Design Bounding Box (0.00,0.00,3180.00,1154.00)
Number of routing layers = 10
layer M1, dir Hor, min width = 0.05, min space = 0.05 pitch = 0.15
layer M2, dir Ver, min width = 0.06, min space = 0.06 pitch = 0.15
...
Net statistics:
Total number of nets = 255165
Number of nets to route = 248716
Number of single or zero port nets = 1721
4728 nets are fully connected,
of which 4728 are detail routed and 0 are global routed.
1648 nets have non-default rule clock_spacing
...
phase3. Routing result:
phase3. Both Dirs: Overflow = 3320 Max = 3 GRCs = 4405 (0.08%)
phase3. H routing: Overflow = 1759 Max = 2 (GRCs = 1) GRCs = 2756 (0.10%)
phase3. V routing: Overflow = 1560 Max = 3 (GRCs = 20) GRCs = 1649 (0.06%)
phase3. M1 Overflow = 1475 Max = 2 (GRCs = 1) GRCs = 2426 (0.09%)
phase3. M2 Overflow = 1265 Max = 3 (GRCs = 20) GRCs = 1343 (0.05%)
...
Overflow over macro areas

phase3. Both Dirs: Overflow = 293 Max = 2 GRCs = 300 (0.04%)
phase3. H routing: Overflow = 133 Max = 1 (GRCs = 129) GRCs = 139 (0.03%)
phase3. V routing: Overflow = 160 Max = 2 (GRCs = 2) GRCs = 161 (0.04%)
phase3. M1 Overflow = 0 Max = 0 (GRCs = 0) GRCs = 0 (0.00%)
phase3. M2 Overflow = 0 Max = 0 (GRCs = 0) GRCs = 0 (0.00%)
...
Density distribution:
Layer 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 >
1.2
M1 86.6 10.5 0.37 1.46 0.02 0.44 0.18 0.14 0.07 0.03 0.10 0.00 0.00 0.02
M2 68.2 14.4 7.71 4.54 2.07 1.55 0.66 0.28 0.13 0.03 0.33 0.00 0.00 0.02
...
phase3. Total Wire Length = 21154552.81
phase3. Layer M1 wire length = 947324.93
phase3. Layer M2 wire length = 3959478.25
...
phase3. Total Number of Contacts = 2530044
phase3. Via VIA12SQ_C count = 1050582
phase3. Via VIA23SQ_C count = 856311
...
phase3. completed.

```

Before proceeding to detail routing, display the congestion map in the GUI and check the overflow distribution. The congestion report and map help you to identify congested areas. For more information about the congestion report, see [Generating a Congestion Report](#). For more information about the congestion map, see [Generating a Congestion Map](#).

## Global Routing During Design Planning

During design planning you can perform exploration-mode global routing by using the `-floorplan true` option with the `route_global` command.

```
fc_shell> route_global -floorplan true
```

If you are using the hierarchical flow, enable virtual-flat global routing by using the `-virtual_flat` option with the `route_global` command. When you set the `-virtual_flat` option to `all_routing`, Zroute routes all the nets in the block and preserves the hierarchy and pin constraints. You can increase the virtual-flat global routing speed by routing only the top-level nets. To do this, set the `-virtual_flat` option to `top_and_interface_routing_only`. When the `-virtual_flat` option is set to `off`, which is the default, Zroute ignores the physical hierarchy and routes the block as flat.

```
fc_shell> route_global -floorplan true -virtual_flat all_routing
```

## Timing-Driven Global Routing

By default, the `route_global` command is not timing-driven. To enable timing-driven global routing, set the `route.global.timing_driven` application option before you run the `route_global` command. During timing-driven global routing, the global router considers the impact of layer resistance by minimizing the usage of pin access layers, even if this increases wire length.

By default, when you enable timing-driven global routing, the tool

1. Calculates the net delays of the block

If the design library contains global route information, the tool uses the global route information to calculate the net delays; otherwise, it uses virtual routing to calculate the net delays.

The global routing results can vary depending on whether the initial net delays were calculated by using global route information or virtual routing. To remove existing global route information from the signal nets in the block, use the `-global_route` and `-net_types` signal options with the `remove_routes` command, as shown in the following example:

```
fc_shell> remove_routes -global_route -net_types signal
```

2. Performs an initial routing phase (phase 0), in which the tool routes the unconnected nets and calculates the overflow for each global routing cell

3. Performs two rerouting phases, in which the tool tries to reduce congestion by ripping up and rerouting nets around global routing cells with overflows
4. Updates the design timing based on the first three global routing phases
5. Identifies newly critical nets based on the timing update
6. Performs three rerouting phases, in which the tool rips up and reroutes the newly critical nets

To increase the number of rerouting phases after the timing update, set the global routing effort level to `high` or `ultra` by using the `-effort_level` option with the `route_global` command.

To reduce runtime, at a possible cost to the timing QoR, set the `route.global.advance_node_timing_driven_effort` application option.

- When you set this application option to `medium`, the tool rips up and reroutes the newly critical nets only if doing so does not increase congestion. In addition, the number of rerouting phases is reduced for medium-, high-, and ultra-effort global routing.
- When you set this application option to `low`, the tool does not perform the timing update and the number of rerouting phases is reduced for medium, high-, and ultra-effort global routing.

The following table summarizes the timing-driven global routing behavior based on the effort level settings. The shaded cell shows the default behavior.

*Table 30 Number of Rerouting Phases Based on the Effort Level Settings*

<code>route_global -effort_level</code> setting	<code>route.global.advance_node_timing_driven_effort</code> setting		
	<code>low</code>	<code>medium</code>	<code>high (default)</code>
<code>low</code>	2	2	2
<code>medium (default)</code>	3	<sup>3,4</sup> 3	<sup>3,5</sup> 5
<code>high</code>	4	<sup>3,4</sup> 4	<sup>3,4</sup> 6
<code>ultra</code>	6	<sup>3,4</sup> 6	<sup>3,4</sup> 8

To control the tradeoff between timing QoR and DRC convergence, set the `route.global.timing_driven_effort_level` application option. By default, this option has a setting of `high`, which favors timing QoR over DRC convergence.

3. *Timing update performed after the second rerouting phase.*
4. *Newly critical nets are ripped up and rerouted only if it does not increase congestion.*
5. *Newly critical nets are always ripped up and rerouted.*

## Crosstalk-Driven Global Routing

By default, the `route_global` command is not crosstalk-driven. To enable crosstalk-driven global routing, set the `route.global.crosstalk_driven` and `time.si_enable_analysis` application options to `true` before you run the `route_global` command.

When you enable crosstalk-driven global routing, the tool calculates the net delays before invoking the global router. If the design library contains global route information, the tool uses the global route information to calculate the net delays; otherwise, it uses virtual routing to calculate the net delays. The global routing results can vary depending on whether the initial net delays were calculated by using global route information or virtual routing. To remove existing global route information from the signal nets in the block, use the `-global_route` and `-net_types` signal options with the `remove_routes` command, as shown in the following example:

```
fc_shell> remove_routes -global_route -net_types signal
```

## Incremental Global Routing

By default, the global router ignores existing global routes. To perform incremental global routing by reusing the existing global routes, use the `-reuse_existing_global_route` `true` option when you run global routing. Note that this option affects only the global router and not the net delay calculation that occurs before timing-driven or crosstalk-driven global routing.

---

## Track Assignment

Before you run track assignment,

- Define the common routing application options
  - For information about the common routing application options, see the `route.common_options` man page.
- Define the global routing application options
  - For information about the track assignment application options, see the `route.track_options` man page.
- Complete global routing

To perform standalone track assignment, run the `route_track` command.

The main task of track assignment is to assign routing tracks for each global route. During track assignment, Zroute performs the following tasks:

- Assigns tracks in horizontal partitions.
- Assigns tracks in vertical partitions.
- Reroutes overlapping wires.

After track assignment finishes, all nets are routed but not very carefully. There are many violations, particularly where the routing connects to pins. Detail routing works to correct those violations.

**Note:**

Because track assignment replaces the global routes with actual metal shapes, the block no longer contains global routes after track assignment completes.

By default, the `route_track` command is not timing-driven or crosstalk-driven.

- To enable timing-driven mode, set the `route.track.timing_driven` application option.
- To enable crosstalk-driven mode, set the `route.track.crosstalk_driven` and `time.si_enable_analysis` application options to `true`.

At the end of track assignment, Zroute reports a summary of the wire length and via count.

[Example 19](#) shows a track assignment report.

*Example 19 Track Assignment Report*

```
Wire length and via report:

Number of M1 wires: 215327 : 0
Number of M2 wires: 1124740 VIA12SQ_C: 1067462
...
Total number of wires: 2508734 vias: 2769482

Total M1 wire length: 924480.9
Total M2 wire length: 4147032.0
...
Total wire length: 21281278.0

Longest M1 wire length: 1541.7
Longest M2 wire length: 926.0
...
```

---

## Detail Routing

Before you run detail routing,

- Define the common routing application options

For information about the common routing application options, see the `route.common_options` man page.

- Define the detail routing application options

For information about the detail routing application options, see the `route.detail_options` man page.

- Complete global routing and track assignment

The detail router uses the general pathways suggested by global routing and track assignment to route the nets, and then it divides the block into partitions and looks for DRC violations in each partition. When the detail router finds a violation, it rips up the wire and reroutes it to fix the violation. During detail routing, Zroute concurrently addresses routing design rules and antenna rules and optimizes via count and wire length. For more information about antenna rules, see [Finding and Fixing Antenna Violations](#).

To perform standalone detail routing, run the `route_detail` command.

By default, the `route_detail` command

- Performs detail routing on the whole block

You can restrict the routing to a specific area of the block by using the `-coordinates` option (or by specifying or selecting the bounding box in the GUI).

- Uses one uniform partition for the first iteration and adjusts the partitioning for subsequent iterations

Zroute uses the single uniform partition for the first iteration to generate all DRC violations for the chip at the same time. At the beginning of each subsequent iteration, the router checks the distribution of the DRC violations. If the DRC violations are evenly distributed, the detail router uses a uniform partition. If the DRC violations are located in some local areas, the detail router uses nonuniform partitions.

In some cases, such as when a design contains standard cells with long pins, you can improve the detail routing QoR and reduce the runtime by increasing the partition size. To enable this feature, set the `route.detail.optimize_partition_size_for_drc` application option to `true`.

- Performs iterations until one of the following conditions exists:

- All of the violations have been fixed
- The maximum number of iterations has been reached

By default, the maximum number of iterations is 40. You can change this limit by setting the `-max_number_iterations` option.

```
fc_shell> route_detail -max_number_iterations 20
```

- It cannot fix any of the remaining violations

You can change the effort that the detail router uses for fixing the remaining violations before it gives up by setting the `route.detail.drc_convergence_effort_level` application option.

```
fc_shell> set_app_options \
 -name route.detail.drc_convergence_effort_level -value high
```

You can force the detail router to complete the maximum number of iterations, regardless of the DRC convergence status, by setting the `route.detail.force_max_number_iterations` application option to `true`.

```
fc_shell> set_app_options \
 -name route.detail.force_max_number_iterations -value true
```

- Is not timing-driven

To enable timing-driven detail routing, set the `route.detail.timing_driven` application option to `true`.

```
fc_shell> set_app_options \
 -name route.detail.timing_driven -value true
```

By default, when you enable timing-driving detail routing, Zroute uses medium effort to assign timing-critical nets to low-resistance metal layers. To change the extent to which timing-driven detail routing prefers lower-resistance metal layers when routing timing-critical nets, set the `route.common.rc_driven_setup_effort_level` application option. To increase the effort level to use more low-resistance metal layers for routing, set the option to `high`. To reduce the effort level, set the option to `low`. To disable resistance-based routing layer preferences, set the option to `off`.

- Does not fix shorted nets over macro cells

If default detail routing leaves shorted nets over macro cells, analyze the block to determine if the shorts are caused by the availability of only a single layer for routing over the macro cells. If so, use routing guides to encourage river routing over the macros with shorted nets and rerun detail routing. For details, see [Using Routing Guides to Encourage River Routing](#).

If shorted nets remain after using river routing, enable the fixing of shorted nets over macro cells by automatically ripping up and rerouting the shorted nets by setting the `route.detail.repair_shorts_over_macros_effort_level` application option to low, medium, or high and running incremental detail routing. The higher the effort level, the more ECO routing iterations are performed, which can reduce the number of DRC violations at the expense of runtime.

```
fc_shell> set_app_options \
 -name route.detail.repair_shorts_over_macros_effort_level \
 -value high
```

- Does not fix soft DRC violations, such as bridge rule violations

To enable the fixing of soft DRC violations after the final detail routing iteration, set the `route.common.post_detail_route_fix_soft_violations` application option to true.

```
fc_shell> set_app_options \
 -name route.common.post_detail_route_fix_soft_violations \
 -value true
```

You can run additional detail routing iterations on a routed block by running incremental detail routing (the `-incremental` option). Be sure to use the `-incremental` option; otherwise, Zroute restarts at iteration 0 with a fixed-size partition.

```
fc_shell> route_detail -incremental true
```

By default, incremental detail routing does not fix soft DRC violations, such as bridge rule violations. To enable the fixing of soft DRC violations after the final incremental detail routing iteration, set the `route.common.post_incremental_detail_route_fix_soft_violations` application option to true.

```
fc_shell> set_app_options \
 -name route.common.post_incremental_detail_route_fix_soft_violations \
 -value true
```

#### Note:

Incremental detail routing does not fix open nets. To fix open nets, you must run ECO routing. For information about ECO routing, see [Performing ECO Routing](#).

If you want to override certain runtime-intensive application options, you can run detail routing in early mode. To turn on the early mode in detail routing, use the `set_qor_strategy` command with the `-mode` option set as `early`.

```
fc_shell> set_qor_strategy -mode early -stage pnr
```

Early mode detail routing automatically creates a new DRC type called skipped regions that can detect dirty regions with pin overflow. To identify the dirty regions, open the error browser and select **Skipped region**.

If you want to view the DRC violations before postroute optimization, you can save the block after a specified number of iterations by setting the `route.detail.save_after_iterations` application option. The saved block is called `DR_itn`, where `n` is the specified iteration. You can use a string other than DR as the prefix by setting the `route.detail.save_cell_prefix` application option.

Zroute generates a DRC violations summary at the end of each iteration. After completing detail routing, Zroute outputs a final summary report. This report includes all violations detected by Zroute, as well as information about the redundant via conversion rates. If you want an additional report that excludes violations that are not of interest to you, specify the rules to exclude by setting the `route.detail.report_ignore_drc` application option. The syntax to set this option is

```
set_app_options -name route.detail.report_ignore_drc -value list_of_drcs
```

The values used in the `list_of_drcs` argument are the DRC names used in the summary report. If the DRC name includes a space, you must enclose the name in double quotation marks. For a complete list of the supported DRC names, see the man page.

[Example 20](#) shows a detail routing report.

#### *Example 20 Detail Routing Report*

```
Start DR iteration 0: uniform partition
Routed 1/27405 Partitions, Violations = 0
Routed 137/27405 Partitions, Violations = 264
...
DR finished with 7398 violations

DRC-SUMMARY:
 @@@@TOTAL VIOLATIONS = 7398
 Diff net spacing : 194
 Same net spacing : 4
 Diff net via-cut spacing : 2328
 Same net via-cut spacing : 1
 Less than minimum width : 5
 Less than minimum area : 36
 Short : 87
 End of line enclosure : 4742
 Less than NDR width : 1

Total Wire Length = 23928849 micron
Total Number of Contacts = 2932706
Total Number of Wires = 2878293
Total Number of PtConns = 88536
Total Number of Routed Wires = 2878293
Total Routed Wire Length = 23920011 micron
Total Number of Routed Contacts = 2932706
 Layer M1 : 962827 micron
 Layer M2 : 4233755 micron
```

```

...
Via VIA78SQ_C : 1742
Via VIA78SQ_C(rot) : 9
...
Redundant via conversion report:

Total optimized via conversion rate = 98.96% (2902130 / 2932706 vias)
Layer VIA1 = 97.81% (1091973/ 1116367 vias)
Weight 1 = 97.81% (1091973 vias)
Un-optimized = 2.19% (24394 vias)
...
Total double via conversion rate = 98.96% (2902130 / 2932706 vias)
Layer VIA1 = 97.81% (1091973/ 1116367 vias)
Layer VIA2 = 99.97% (1071650/ 1071978 vias)
...
The optimized via conversion rate based on total routed via count =
98.96% (2902130 / 2932706 vias)
Layer VIA1 = 97.81% (1091973/ 1116367 vias)
Weight 1 = 97.81% (1091973 vias)
Un-optimized = 2.19% (24394 vias)
...
Total number of nets = 255165
0 open nets, of which 0 are frozen
Total number of excluded ports = 0 ports of 0 unplaced cells connected to 0
nets
 0 ports without pins of 0 cells connected to
0 nets
 0 ports of 0 cover cells connected to 0
non-pg nets
Total number of DRCs = 7398
Total number of antenna violations = antenna checking not active
Information: Routes in non-preferred voltage areas = 8170 (ZRT-559)

Topology ECO iteration 1 ended with 0 qualifying violations.

```

---

## Routing Signal Nets by Using Automatic Routing

Before you run automatic routing,

- Set the common routing application options

For information about the common routing application options, see the `route.common_options` man page.

- Set the global routing application options

For information about the detail routing application options, see the `route.global_options` man page.

- Set the track assignment application options

For information about the detail routing application options, see the `route.track_options` man page.

- Set the detail routing application options

For information about the detail routing application options, see the `route.detail_options` man page.

To run automatic routing, use the `route_auto` command. By default, the `route_auto` command ignores existing global routes and sequentially invokes global routing, track assignment, and detail routing. The tool does not save the block between routing steps. Use the following options to change the default behavior:

- To perform incremental global routing, use the `-reuse_existing_global_route true` option.
- To stop after track assignment, use the `-stop_after_track_assignment true` option.
- To change the number of search and repair iterations during detail routing from the default of 40, use the `-max_detail_route_iterations` option.
- To save the block after each routing step, set the `-save_after_global_route`, `-save_after_track_assignment`, and `-save_after_detail_route` options to `true`.

The tool uses `auto` as the default prefix for the saved block names. To specify the prefix, use the `-save_cell_prefix` option.

By default, the `route_auto` command is not timing-driven or crosstalk-driven and does not fix soft DRC violations, such as bridge rule violations.

- To enable timing-driven mode, set the `route.global.timing_driven`, `route.track.timing_driven`, and `route.detail.timing_driven` application options to `true`.

By default, when you enable timing-driving routing, Zroute uses medium effort to assign timing-critical nets to low-resistance metal layers. To change the extent to which timing-driven routing prefers lower-resistance metal layers when routing timing-critical nets, set the `route.common.rc_driven_setup_effort_level` application option. To increase the effort level to use more low-resistance metal layers for routing, set the option to `high`. To reduce the effort level, set the option to `low`. To disable resistance-based routing layer preferences, set the option to `off`.

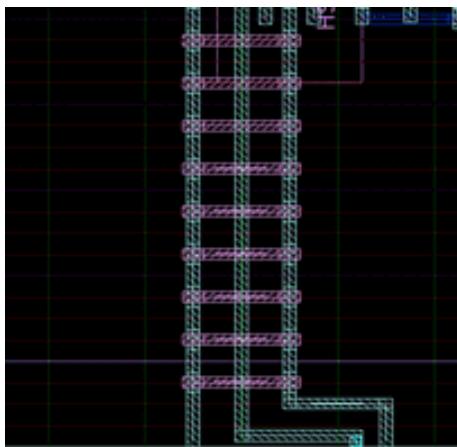
- To enable crosstalk-driven mode, set the `route.global.crosstalk_driven`, `route.track.crosstalk_driven`, and `time.si_enable_analysis` application options to `true`.
- To enable the fixing of soft DRC violations after the final detail routing iteration, set the `route.common.post_detail_route_fix_soft_violations` application option to `true`.

```
fc_shell> set_app_options \
 -name route.common.post_detail_route_fix_soft_violations \
 -value true
```

## Shielding Nets

Zroute shields routed nets by generating shielding wires that are based on the shielding widths and spacing defined in the shielding rules. In addition to shielding nets on the same layer, you also have the option to shield one layer above and one layer below. Shielding above or below the layer is called *coaxial shielding*. [Figure 101](#) shows an example of coaxial shielding. Coaxial shielding provides even better signal isolation than same-layer shielding, but it uses more routing resources.

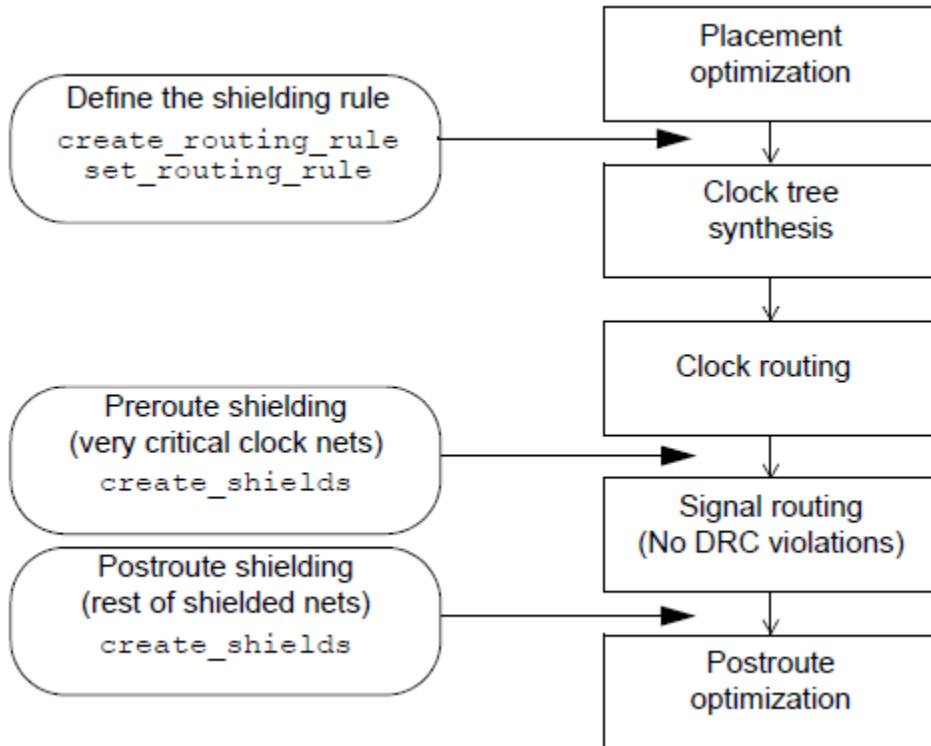
*Figure 101 Coaxial Shielding*



You can perform shielding either before or after signal routing. Shielding before signal routing, which is referred to as preroute shielding, provides better shielding coverage but can result in congestion issues during signal routing. Preroute shielding is typically used to shield critical clock nets. Shielding after signal routing, which is referred to as postroute shielding, has a very minimal impact on routability, but provides less protection to the shielded nets.

Figure 102 shows the Zroute shielding flow, which is described in the topics that follow.

Figure 102 Zroute Shielding Flow



## Defining the Shielding Rules

Before you perform shielding, you must

1. Define the shielding rules.

To define shielding rules, use the `-shield_spacings` and `-shield_widths` options of the `create_routing_rule` command. For example, to specify a shielding rule that uses spacing of 0.1 microns and width of 0.1 microns for metal1 through metal5 and spacing of 0.3 microns and width of 0.3 microns for metal6, use the following command:

```
fc_shell> create_routing_rule shield_rule \
-shield_widths {M1 0.1 M2 0.1 M3 0.1 M4 0.1 M5 0.1 M6 0.3} \
-shield_spacings {M1 0.1 M2 0.1 M3 0.1 M4 0.1 M5 0.1 M6 0.3}
```

For more information about the `create_routing_rule` command, see [Using Nondefault Routing Rules](#).

## 2. Assign shielding rules to the nets to be shielded.

To avoid congestion issues and achieve the best balance of DRC convergence and timing closure, you should apply shielding rules only to high-frequency or critical clock nets and apply double-spacing rules to the lower-frequency clock nets.

- To assign shielding rules to clock nets, use the `set_clock_routing_rules` command.

**Note:**

You can use the `set_clock_routing_rules` command only before clock tree synthesis.

- To assign shielding rules to signal nets, use the `set_routing_rule` command.

For more information about these commands, see [Assigning Nondefault Routing Rules to Nets](#).

## Performing Preroute Shielding

To provide the most protection for critical clock nets, perform shielding on those nets after clock tree routing but before signal net routing.

To add shielding to the routed clock nets based on the assigned shielding rules, use the `create_shields` command.

By default, the `create_shields` command

- Performs shielding on all nets that have predefined shielding rules, except those marked as frozen

To explicitly specify the nets on which to perform shielding, use the `-nets` option.

- Does not perform shielding on wires that are less than four pitches long

To specify the minimum wire length in microns on which to perform shielding for each layer, set the `route.common.min_shield_length_by_layer_name` application option. Use the following syntax to set this option:

```
{ {layer wire_length} ... }
```

Specify the routing layers by using the layer names from the technology file. If you do not specify a value for a layer, Zroute uses the default minimum length of four pitch lengths for that layer.

**Note:**

Wires that are less than the minimum length are not considered when computing the shielding ratio.

- Ties the shielding wires to the ground net

If the block contains multiple ground nets or you want to tie the shielding wires to the power net, use the `-with_ground` option to specify the power or ground net to which to tie the shielding wires.

If the shielding wires can be tied to multiple power or ground nets, specify the nets by setting the `route.common.shielding_nets` application option before running the `create_shields` command.

**Note:**

If you specify both the `create_shields -with_ground` option and the `route.common.shielding_nets` option, the tool issues a warning message and uses the `route.common.shielding_nets` setting.

- Performs same-layer shielding

To perform coaxial shielding, use the `-coaxial_above` and `-coaxial_below` options. By default, the `create_shields` command leaves one routing track open between each used track. To change the default behavior, use one of the following methods:

- Specify the number of open tracks between coaxial shielding segments by using the following options:

- `-coaxial_above_skip_tracks`

This option specifies the number of open tracks between used tracks for coaxial shielding above the shielded net segment layer (`-coaxial_above true`).

- `-coaxial_below_skip_tracks`

This option specifies the number of open tracks between used tracks for coaxial shielding below the shielded net segment layer (`-coaxial_below true`).

For either of these options, you can specify an integer between zero and seven.

- Specify the number of open tracks between coaxial shielding segments on a per-layer basis by using the `-coaxial_skip_tracks_on_layers` option

To disable shielding on a specific layer, set the value to -1; otherwise, specify an integer between 0 and 7 to specify the number of tracks to skip.

- Specify the spacing between coaxial shielding segments by using the following options:

- `-coaxial_above_user_spacing`

This option specifies the spacing in microns between shielding segments for coaxial shielding above the shielded net segment layer (`-coaxial_above true`).

- `-coaxial_below_user_spacing`

This option specifies the spacing in microns between shielding segments for coaxial shielding below the shielded net segment layer (`-coaxial_below true`).

You cannot use this method when performing incremental shielding.

**Note:**

You cannot mix these methods of specifying coaxial shielding.

If the generated coaxial shielding wires violate minimum area or minimum length rules, Zroute automatically patches the wires to satisfy these design rules.

- Connects the shielding wires to the standard-cell power or ground pins and the standard-cell rails

To prevent connections to the standard-cell power and ground pins, set the `-ignore_shielding_net_pins` option to `true`. To prevent connections to the standard-cell rails, set the `-ignore_shielding_net_rails` option to `true`.

- Creates the shielding wires such that they surround the shielded routing shape

To trim the shielding wires so that they align with the shielded routing shape ends, set the `-align_to_shape_end` option to `true`. To force Zroute to create shielding wires only in the preferred direction, set the `-preferred_direction_only` option to `true`. Note that the `-preferred_direction_only` option does not honor route guides to change the preferred routing direction. When you set either of these options to `true`, extra effort is required to connect the shielding wires to the power and ground network and any shielding wires that are not connected to the power and ground network are deleted.

For example, to perform coaxial shielding below the shielded net segment layer on the clock nets, prevent signal routing below the shielded net segment layer, and tie the shielding wires to VSS, use the following command:

```
fc_shell> create_shields -nets $clock_nets \
 -coaxial_below true -coaxial_below_skip_tracks 0 \
 -with_ground VSS
```

When you run the `create_shields` command, it reports both the net-based and length-based average shielding ratios, as shown in the following example:

```
Shielded 82% side-wall of (reset)
Shielded 96% side-wall of (clk)
Shielded 2 nets with average ratio as follows.
 1) 89.00% (total shield ratio/number of shielded nets)
 2) 87.92% (total shield length/total shielded net length)
```

**Note:**

In some cases there is a slight difference in the shielding ratios reported by the `create_shields` and `report_shields` commands. This is due to graph connectivity differences between the two commands. When the reported values differ, use the values reported by the `report_shields` command. The difference in reported values is typically less than three percent, but can be up to five percent.

If you use the `-preferred_direction_only true` option when running the `create_shields` command, but Zroute must use some nonpreferred direction wires for shielding, the shielding ratio report specifies the percentage of nonpreferred direction wires.

By default, the power and ground structure is not included in the shielding ratio calculation. To include the power and ground structure within a threshold distance in the shielding ratio calculation, set the `route.common.pg_shield_distance_threshold` application option, which specifies the distance threshold in microns.

```
fc_shell> set_app_options \
 -name route.common.pg_shield_distance_threshold -value distance
```

Note that this option affects only the shielding ratio calculation and does not change the routing behavior.

## Soft Shielding Rules During Signal Routing

Zroute considers shielding rules as soft rules during signal routing. If a net has a shielding rule and is not shielded before signal routing, by default, Zroute reserves shielding space during the whole routing process: global routing, track assignment, and detail routing. At the end of detail routing, it reports the shielding space violations and the locations where shielding wires cannot be established. The following log file example shows shielding soft spacing violations, which are highlighted in bold text:

```
DRC-SUMMARY:
 @@@@TOTAL VIOLATIONS = 13
 Diff net var rule spacing : 2
 Same net spacing : 2
 Less than minimum area : 4
 Short : 1
 Soft spacing (shielding) : 2
```

Signal routing only reserves space for the shielding; it does not actually insert it. You must run `create_shields` after signal routing to physically place the shielding wires. The reported soft rule violations help you to understand the shielding rate. Note that the router reserves space only for same-layer shielding and not for coaxial shielding; therefore, postroute coaxial shielding can produce a very low shielding rate.

If you want to use power and ground nets for shielding, and do not want Zroute to reserve space for the shielding when power and grounds nets are available, set the `route.common.allow_pg_as_shield` application option to `true` before running signal routing.

## Performing Postroute Shielding

To perform postroute shielding, you use the same command, `create_shields`, that is used for preroute shielding.

If you want to use the default spacings and widths from the technology file for postroute shielding, you do not need to define and assign nondefault routing rules. If you specify the nets to be shielded by using the `-nets` option, the `create_shields` command shields these nets with the default spacing and widths.

Postroute shielding should introduce few to no DRC violations. If DRC violations are created during shielding, the `create_shields` command triggers five detail routing iterations to fix them. If this does not fix the DRC violations, you can fix the remaining violations by running incremental detail routing with the `route_detail -incremental true` command. It is possible that postroute shielding might break some tie-off connections during the shield trimming process. In this case, use the `route_eco` command instead of the `route_detail` command to rebuild the tie-off connections and to fix the DRC violations.

## Shielding Example

[Example 21](#) provides an example of shielding clock nets using preroute shielding and shielding critical nets using postroute shielding.

### Example 21 Shielding Flow Example

```
Define shielding rule
create_routing_rule shield_rule \
 -shield_widths {M1 0.1 M2 0.1 M3 0.1 M4 0.1 M5 0.1 M6 0.3} \
 -shield_spacings {M1 0.1 M2 0.1 M3 0.1 M4 0.1 M5 0.1 M6 0.3}

Assign shielding rule to clock nets
set_clock_routing_rules -clocks CLK \
 -rules shield_rule

Perform clock tree synthesis
synthesize_clock_trees

Route the clock nets, reusing the global routing result
route_group -all_clock_nets -reuse_existing_global_route true

Perform preroute shielding for the clock nets
create_shields -nets $clock_nets -with_ground VSS
```

```
Assign shielding rule to critical nets
set_routing_rule -rule shield_rule $critical_nets

Route signal nets using shielding soft rules
route_auto

Perform postroute shielding for critical nets
create_shields -nets $critical_nets -with_ground VSS
```

---

## Performing Incremental Shielding

By default, Zroute does not perform incremental shielding on nets that are modified after they were shielded. However, you can enable this capability for nets that were initially shielded with the `create_shields` command by changing the `route.common.reshield_modified_nets` application option from its default of `off`.

When you enable incremental shielding, Zroute performs incremental shielding during detail routing by removing the existing shielding from the modified nets and optionally reshielding these nets based on the new topology.

- To remove the existing shielding only, set the `route.common.reshield_modified_nets` option to `unshield`.
- To remove the existing shielding and reshield the modified nets, set the `route.common.reshield_modified_nets` option to `reshield`.

**Note:**

Zroute automatically detects the nets modified within the Fusion Compiler tool; however, nets modified externally and input by reading a DEF file are not supported by incremental shielding.

---

## Reporting Shielding Information

After you run the `create_shields` command, you can

- Query the shield shapes associated with a shielded net by using the `get_shapes -shield_only` command, as described in [Querying Shield Shapes](#)
- Report the shielding statistics by using the `report_shields` command, as described in [Reporting Shielding Statistics](#)

## Querying Shield Shapes

To query the shield shapes associated with a shielded net, use the `get_shapes` command with the `-shield_only` option, as shown in the following example:

```
fc_shell> get_shapes -shield_only -of_objects [get_nets myclk]
{PATH_19_4317 PATH_19_4318 PATH_19_4319 PATH_19_4320 PATH_19_4322
```

```
PATH_17_11590 PATH_17_11591 PATH_17_11592 PATH_17_11593 PATH_17_11594
PATH_17_11595 PATH_17_11596 PATH_17_11597 PATH_17_11599 PATH_17_11600
PATH_17_11601 PATH_17_11602 PATH_17_11604 PATH_17_11605 PATH_17_11606
PATH_17_11607 PATH_17_11608 PATH_17_11609 PATH_17_11610 PATH_17_11611
PATH_17_11612 PATH_17_11613 PATH_17_11614 PATH_17_11615 PATH_17_11616
PATH_17_11617 PATH_17_11618 PATH_17_11619 PATH_17_11620 PATH_17_11621
PATH_17_11622 PATH_17_11623 PATH_17_11624 PATH_17_11625 PATH_17_11626}
```

## Reporting Shielding Statistics

To report the shielding statistics, use the `report_shields` command.

**Note:**

In some cases there is a slight difference in the shielding ratios reported by the `create_shields` and `report_shields` commands. This is due to graph connectivity differences between the two commands. When the reported values differ, use the values reported by the `report_shields` command. The difference in reported values is typically less than three percent, but can be up to five percent.

By default, the power and ground structure is not included in the shielding ratio calculation. To include the power and ground structure within a threshold distance in the shielding ratio calculation, set the `route.common.pg_shield_distance_threshold` application option, which specifies the distance threshold in microns.

```
fc_shell> set_app_options \
 -name route.common.pg_shield_distance_threshold -value distance
```

The default report generated by the `report_shields` command provides overall statistics, as shown in [Reporting Shielding Statistics](#).

*Example 22 Default Shielding Report*

```
fc_shell> report_shields
...
Shielded 82% side-wall of (reset)
Shielded 96% side-wall of (clk)
Shielded 2 nets with average ratio as follows.
 1) 89.00% (total shield ratio/number of shielded nets)
 2) 87.92% (total shield length/total shielded net length)
```

You can output the statistics for each layer by using the `-per_layer true` option with the `report_shields` command, as shown in [Example 23](#).

*Example 23 Layer-Based Shielding Report*

```
fc_shell> report_shields -per_layer true
...
Shielded 82% side-wall of (reset)
Layer: M1 ratio: 0%
Layer: M2 ratio: 40%
```

```

Layer: M3 ratio: 85%
Layer: M4 ratio: 80%
Shielded 96% side-wall of (clk)
Layer: M3 ratio: 0%
Layer: M4 ratio: 96%
Layer: M5 ratio: 100%
Shielded 2 nets with average ratio as follows.
 1) 89.00% (total shield ratio/number of shielded nets)
 2) 87.92% (total shield length/total shielded net length)

```

## Performing Shielding Checks

During routing, Zroute can detect possible issues with shielding by checking for the following conditions:

- Signal net shapes with a `shape_use` attribute of `shield_route`.
- PG net shapes, which might be a PG strap or rail, but have a `shape_use` attribute of `detail_route`.
- Signal, clock, or PG nets that have a shielding nondefault rule but no associated shield shapes, which might be caused by inappropriate `shape_use` attributes.

To enable these checks, set the `route.common.check_shield` application option to `true`.

## Performing Postroute Optimization

The Fusion Compiler tool can perform two types of postroute optimization:

- Logic optimization

This optimization improves the timing, area, and power QoR and fixes logical DRC violations and performs legalization and ECO routing. To perform these optimizations, use the `route_opt` command, as described in [Performing Postroute Logic Optimization](#).

- Routability optimization

This optimization increases the spacing between cells to fix routing DRC violations caused by pin access issues. To perform this optimization, use the `optimize_routability` command, as described in [Fixing DRC Violations Caused by Pin Access Issues](#).

If you run both logic optimization and routability optimization, you should first perform the logic optimization and then the routability optimization.

---

## Performing Postroute Logic Optimization

Before performing postroute optimization, ensure that the block is fully routed and legalized and does not have excessive logical or routing DRC violations.

- To verify the legality of the design, use the `check_legality` command.
- To verify the routing, use the `check_routes` command.

To perform postroute logic optimization,

1. Update the clock latency by using the `compute_clock_latency` command.
2. Run the `route_opt` command two times.

The legalization and ECO routing performed by the first `route_opt` run might impact the block timing, which is then optimized by the second `route_opt` run. As the number of changes during postroute logic optimization decreases, the timing improvement from subsequent `route_opt` runs decreases.

The `route_opt` command performs the following tasks:

1. Performs extraction and updates the timing

By default, the `route_opt` command uses the PrimeTime delay calculation engine, which requires that you use cell libraries generated by O-2018.06 or later versions of the Library Manager tool. Before using PrimeTime delay calculation, use the `check_consistency_settings` command to verify that the Fusion Compiler and PrimeTime settings are consistent, as described in Checking for Consistency in Timing Analysis and Extraction Settings in the *Fusion Compiler Timing Analysis User Guide*.

The `route_opt` command supports both graph-based analysis (GBA), the default, and path-based analysis (PBA). To enable path-based optimization, use the `time.pba_optimization_mode` application option.

- To use path-based timing for the worst path of each endpoint, which is identified by using graph-based analysis, set this application option to `path`.

This is the recommended setting because it provides the best tradeoff between runtime and QoR.

- To use exhaustive path-based search algorithms for each endpoint, set this application option to `exhaustive`.

Using this setting increases the runtime.

## 2. Performs the enabled optimizations

By default, the `route_opt` command optimizes for setup, hold, area, and logical DRC violations for the data paths in the block. To enable additional optimizations, set the application options shown in [Table 31](#) before running the `route_opt` command:

*Table 31 Application Options to Enable route\_opt Optimizations*

Optimization	Application option settings
Concurrent clock and data optimization <sup>6</sup> and timing-driven clock logical DRC fixing	<code>route_opt.flow.enable_ccd = true</code>
Clock logical DRC fixing <sup>7</sup>	<code>route_opt.flow.enable_cto = true</code>
Clock tree area recovery	<code>route_opt.flow.enable_clock_power_recovery = area</code>
Clock tree power recovery	<code>route_opt.flow.enable_clock_power_recovery = power</code>
Power optimization <sup>8</sup>	<code>route_opt.flow.enable_power = true</code>
Path-based optimization with machine learning	<code>route_opt.flow.enable_ml_opto = true</code> <code>time.pba_optimization_mode = path</code>

### Note:

The `route_opt` command honors only the `route_opt` application options; it does not honor the settings of the `opt` application options, except the `opt.common.allow_physical_feedthrough` application option.

## 3. Legalizes the block

## 4. Performs ECO routing

By default, the `route_opt` command performs five detail routing iterations during the ECO routing phase. To change the number of iterations, use the `route.detail.eco_max_number_of_iterations` application option. To perform track assignment instead, set the `route_opt.eco_route.mode` application option to `track`.

6. For information about setting application options to control concurrent clock and data optimization, see [.](#)
7. This optimization is supported only when concurrent clock and data optimization is not enabled.
8. The type of power optimization performed depends on scenario configuration. Total power optimization is performed if there is an active scenario with both leakage power and dynamic power enabled. Leakage power optimization is performed if there is an active scenario with only leakage power enabled.

If the PrimeTime tool reports setup or hold violations during signoff timing analysis, you can use the following process to fix these violations:

1. Specify the target endpoints and their PrimeTime timing information by using the `set_route_opt_target_endpoints` command with the `-setup_timing` and `-hold_timing` options.
2. Optimize the endpoints by using the `route_opt` command.
3. Remove the adjusted timing information by using the `set_route_opt_target_endpoints -reset` command.

You can also use this process to perform incremental optimization on specific endpoints. In this case, use the `-setup_endpoints`, `-hold_endpoints`, and `-ldrc_objects` options to specify the endpoints.

To close the final setup, hold, or logical DRC violations with minimal disturbance to the block, use size-only mode when you run the `route_opt` command. To enable size-only mode, set the `route_opt.flow.size_only_mode` application option to one of the following values:

- `true_footprint`, which allows resizing a cell only to a library cell that is an exact physical match, as determined by the tool after analysis

This ensures that legalization or ECO routing is not required after resizing

- `footprint`, which allows resizing a cell only to a library cell that has the same footprint attribute in the library
- `equal`, which allows resizing a cell only to a library cell that is equal in size
- `equal_or_smaller`, which allows resizing a cell only to a library cell that is equal or smaller in size

To disable size-only mode, set the `route_opt.flow.size_only_mode` application option to `none` or "", which is the default.

## Performing Postroute Optimization Using the `hyper_route_opt` Command

The recommended flow for performing postroute logic optimization consists of multiple iterations of the `route_opt` command with different settings. However, you can reduce the number of postroute optimization iterations by using a single iteration of the `hyper_route_opt` command.

The postroute optimization flow using the `hyper_route_opt` command consists of the following steps:

1. Enable different settings for postroute optimization using the same settings as for the `route_opt` command.

For example, you can

- Specify the path-based analysis (PBA) mode by using the `time.pba_optimization_mode` application option
  - Enable concurrent clock and data optimization by using the `route_opt.flow.enable_ccd` application option
  - Enable power optimization by using the `route_opt.flow.enable_power` application option
2. (Optional) Specify settings for metal fill insertion, PG augmentation, and so on that should to be performed during the different stages of optimization and ECO routing within the `hyper_route_opt` command by using the `smps_hyper_route_opt_post_eco` Tcl-callback procedure.
  3. Run postroute optimization using the `hyper_route_opt` command.
  4. (Optional) Specify endpoints to target by using the `set_route_opt_target_endpoints` command and optimize these endpoints by using the `route_opt` command if there are any violating endpoints remaining after the `hyper_route_opt` command.
  5. (Optional) Fix any remaining route DRC violations by using the `route_detail -incremental` command.

---

## Fixing DRC Violations Caused by Pin Access Issues

In advanced process nodes, the distance between pins decreases, which can result in DRC violations caused by pin access issues. These types of DRC violations can be fixed by using keepout margins to increase the spacing between cells.

If your postroute design has DRC violations, analyze the violations to determine if they are caused by pin access issues. If so, run the `optimize_routability` command to increase the spacing between cells where the DRC violations occur.

The `optimize_routability` command performs the following tasks:

- Analyzes the cells with DRC violations to find violations where cells abut

By default, the command considers all DRC violations. To consider only specific DRC violations, use the `-drc_rules` option. To consider only violations on specific layers, use the `-layer_rules` option.

To preview the number of DRC violations and affected cells, use the `-check_drc_rules` option. When you use this option, the command generates a report but does not move any cells.

- Sets keepout margins on these cells on the side of the cell where the error occurs

By default, the command uses the site width as the keepout width. To specify a keepout width in microns, use the `-keepout_width` option.

**Note:**

These keepout margins are in addition to any existing keepout margins on the cells; the command does not modify the existing keepout margins.

To try to fix the DRC violations by flipping the cells instead of adding keepout margins, use the `-flip` option.

- Legalizes the affected cells

You must use one of the following methods to complete the routes for the cells moved by the optimization:

- Run ECO routing by using the `route_eco` command.
- Run ECO routing by using the `-route` option with the `optimize_routability` command.

When you use this option, the command runs the `route_eco` and `check_routes` commands after completing the optimization.

After moving the cells and performing ECO routing, remove the keepout margins from the cells by using the `optimize_routability -remove_keepouts` command.

## Analyzing and Fixing Signal Electromigration Violations

Signal electromigration problems result from an increase in current density caused by the use of smaller line widths and higher operational speeds in IC designs. Electromigration

can lead to shorts or opens due to metal ion displacement caused by the flow of electrons. The more frequently a net switches, the more susceptible it is to electromigration.

To analyze and fix signal electromigration violations in a detail routed block,

1. Apply the signal electromigration constraints by using the `read_signal_em_constraints` command.

For more information, see [Loading the Signal Electromigration Constraints](#).

2. Apply switching activity by either reading in a SAIF file with the `read_saif` command or annotating the switching activity information on the nets with the `set_switching_activity` command.

For more information, see [Annotating the Switching Activity](#).

3. Report the signal electromigration information by using the `report_signal_em` command.

For more information, see [Analyzing Signal Electromigration](#).

4. Fix any signal electromigration violations by using the `fix_signal_em` command.

For more information, see [Fixing Signal Electromigration Violations](#).

The following example script shows the signal electromigration flow.

```
Open the block and apply the signal electromigration constraints
open_block block1_routed
read_signal_em_constraints em.itf

Load switching information
read_saif block1.saif

Perform signal electromigration analysis
report_signal_em -violated -verbose > block1.signal_em.rpt

Fix signal electromigration violations
fix_signal_em
```

## Loading the Signal Electromigration Constraints

The Fusion Compiler tool supports the following formats for specifying electromigration constraints:

- Interconnect Technology File (ITF)

An ITF file can contain the following types of electromigration constraints: delta temperature, duration and duty ratio, via size, via direction, metal length, and via-to-via spacing relaxation.

The Fusion Compiler tool supports both unencrypted and encrypted ITF files.

- Advanced Library Format (ALF)

An ALF file can contain the following types of electromigration constraints: temperature, metal width, and contact area.

The electromigration constraints are stored in the design library as a lookup table. The tool determines the limit values by looking up the values in the lookup table and then using linear interpolation or extrapolation.

To load the signal electromigration constraints into the design library, use the `read_signal_em_constraints` command. By default, the command reads an unencrypted ITF file. To read an encrypted ITF file, use the `-encrypted` option. To read an ALF file, use the `-format ALF` option. The constraints loaded by the `read_signal_em_constraints` command overwrite any existing signal electromigration constraints in the design library.

For example, to read an unencrypted ITF file, use the following command:

```
fc_shell> read_signal_em_constraints em.itf
```

For example, to read an encrypted ITF file, use the following command:

```
fc_shell> read_signal_em_constraints -encrypted em.itf.enc
```

To read an ALF file, use the following command:

```
fc_shell> read_signal_em_constraints -format ALF em.alf
```

## Analyzing Signal Electromigration

The `report_signal_em` command performs signal electromigration analysis for each net by calculating the current on every edge and comparing this data with the constraints set by the `read_signal_em_constraints` command. For example,

```
fc_shell> report_signal_em -violated -verbose
```

The `report_signal_em` command performs a timing update, if needed, and analyzes all nets for electromigration. The `-verbose` option causes the report to show detailed electromigration analysis information, which usually produces a very large report when used by itself. To get a report with a reasonable size, use the `-violated` option as well, which limits the report to only the nets with electromigration violations. To limit the analysis to specific nets, use the `-nets` option.

You can specify settings for electromigration analysis by setting the application options shown in the following table.

*Table 32 Application Options for Signal Electromigration Analysis*

Application option	Default	Description
<code>em.net_delta_temperature</code>	5.0	Specifies the delta temperature used for RMS limit lookup.
<code>em.net_duration_condition_for_peak</code>	0.0	Specifies the duration value for peak current checking.
<code>em.net_global_rms_relaxation_factor</code>	1.0	Specifies the global relaxation factor for the RMS limit specified in the constraints file.
<code>em.net_metal_line_number</code>	0	Specifies the metal line number used to look up the relaxation factor in the RMS constraint factor table provided in the constraints file.
<code>em.net_min_duty_ratio</code>	0	Specifies the minimum duty ratio for computing the peak current.
<code>em.net_use_waveform_duration</code>	false	When <code>false</code> , the command uses current integral based duration. When <code>true</code> , the command uses current waveform based duration.
<code>em.netViolationRuleTypes</code>	""	Specifies the type of constraint rules to use.

When you perform signal electromigration analysis, the tool computes three current values:

- Average
- Root mean square
- Peak

The default report generated by the `report_signal_em` command lists the number of nets with electromigration violations and the types of constraints violated (mean, absolute

average, RMS, or peak). If you use the `-verbose` option, the report displays detailed information about the violations, as shown in [Example 24](#).

**Example 24 Verbose Electromigration Report**

```
fc_shell> report_signal_em -violated -verbose
...
Net Name: clks/trimch_macroout_cb_group_gpaf0lts1_gpu_c_0_s_0[3]
Violations: RMS PEAK

Flag Segment Layer BBox Coordinates Width/Cut Required
AVERAGE RMS PEAK

v PATH_36_392230 M6D (1450.000, 995.612 -> 1452.715, 995.652) 0.040 0.085
 1.514e-02 (-) 9.650e-01 (5.231e-01) 1.365e+00 (1.349e+00)
VIA_S_11051278 V5D (1449.969, 995.582 -> 1450.071, 995.682) 1 -
 1.465e-02 (-) 7.380e-01 (-) 1.044e+00 (-)
v PATH_35_765189 M5A (1450.001, 994.973 -> 1450.039, 995.651) 0.038 0.047
 1.417e-02 (-) 6.088e-01 (4.981e-01) 8.609e-01 (1.274e+00)
VIA_S_11051279 V5D (1449.969, 994.942 -> 1450.071, 995.042) 1 -
 1.369e-02 (-) 5.249e-01 (-) 7.423e-01 (-)
PATH_36_392231 M6D (1407.370, 994.972 -> 1450.040, 995.012) 0.040 -
 1.311e-02 (-) 2.047e-01 (5.231e-01) 2.894e-01 (9.443e-01)
VIA_S_11051280 V5D (1407.339, 994.942 -> 1407.441, 995.042) 1 -
 1.252e-02 (-) 1.926e-01 (-) 2.723e-01 (-)
...
```

The report shows the segment name (Segment column), layer name (Layer column), location coordinates (Bbox Coordinates column), metal width or via cut number (Width/Cut column), the constraint value (Required column), the average, RMS, and peak current in mA for the wire segment or via at that location. The letter “v” in the Flag column indicates a violation at that location. A hyphen in the Required column means that there is no electromigration constraint for that metal or via layer. In the AVERAGE, RMS, and PEAK columns, the first value is the calculated value and the value in parenthesis is the limit imposed by the electromigration constraints, where a hyphen indicates that there is no electromigration constraint.

---

## Fixing Signal Electromigration Violations

Before fixing signal electromigration violations, make sure that the block is detail routed, the signal electromigration constraints are defined in the design library, the switching activity is defined for all boundary nets, crosstalk analysis is enabled, and there are no timing or DRC violations. You can optionally use the `report_signal_em` command first to report electromigration violations before attempting to fix them.

To fix signal electromigration violations, use the `fix_signal_em` command, which performs the following tasks:

- Runs signal electromigration analysis for nets by calculating the current on every edge of a net and comparing it with the constraints set by the `read_signal_em_constraints` command.
- Fixes violations by applying nondefault routing rules to widen the metal shapes and then performing ECO routing.

The `fix_signal_em` command runs a single fixing iteration; if violations remain after running this command, you must rerun it to address the remaining violations.

By default, the command fixes signal electromigration violations on all nets. To fix the violations only on specific nets, use the `-nets` option.

### See Also

- [Analyzing Signal Electromigration](#)
- [Performing ECO Routing](#)

## Performing ECO Routing

Whenever you modify the nets in your block, you need to run engineering change order (ECO) routing to reconnect the routing.

To run ECO routing, use the `route_eco` command. The `route_eco` command sequentially performs global routing, track assignment, and detail routing to reconnect the routing.

By default, the `route_eco` command

- Considers timing and crosstalk during routing, which means that the tool performs extraction and updates the timing before performing the routing.

The extraction and timing update can be time consuming and might not be necessary for your block. To prevent the extraction and timing update, use the following commands to disable the timing-driven and crosstalk-driven modes:

```
fc_shell> set_app_options \
 -name route.global.crosstalk_driven -value false
fc_shell> set_app_options \
 -name route.global.timing_driven -value false
fc_shell> set_app_options \
 -name route.track.crosstalk_driven -value false
fc_shell> set_app_options \
 -name route.track.timing_driven -value false
fc_shell> set_app_options \
 -name route.detail.timing_driven -value false
```

- Works on all open nets in the block
 

To perform ECO routing only on specific nets, use the `-nets` option to specify the nets.
- Ignores existing global routes
 

To honor the existing global routes and perform incremental global routing, set the `-reuse_existing_global_route` option to `true`.
- Relieves congestion by reusing dangling wires instead of rerouting detail routed wires
 

To disable the reuse of dangling wires, set the `-utilize_dangling_wires` option to `false`.
- Performs a maximum of 40 detail routing iterations
 

To change the maximum number of detail routing iterations, use the `-max_detail_route_iterations` option.
- Fixes hard DRC violations on the entire block by rerouting any wires, whether modified or not
  - To fix DRC violations only in the neighborhood of the open nets, set the `-open_net_driven` option to `true`.

**Note:**

When you use the `-nets` option to perform ECO routing on specific nets, Zroute fixes DRC violations only within the bounding box of the specified nets. In this case, Zroute ignores the setting of the `-open_net_driven` option.

- To change the scope of rerouting performed by the ECO router, use the `-reroute` option.
  - To limit rerouting to modified wires, set this option to `modified_nets_only`.
  - To first attempt to fix DRC violations by rerouting modified nets and then reroute other nets if necessary, set this option to `modified_nets_first_then_others`.
 

A common use for this option is to route the clock nets affected by an ECO in a fully routed block.
- To enable the fixing of soft DRC violations, such as bridge rule violations, after the final detail routing iteration, set the `route.common.post_eco_route_fix_soft_violations` application option to `true`.

Zroute generates a DRC violations summary at the end of each detail routing iteration. Before reporting the final DRC violations, Zroute merges redundant violations. For more

information about the DRC violations reported by Zroute, see [Performing Design Rule Checking Using Zroute](#).

Zroute also reports the nets changed during ECO routing. By default, it reports the first 100 changed nets. You can use the `-max_reported_nets` option to set a different limit on the reported nets. To report all changed nets, set the `-max_reported_nets` option to -1.

### See Also

- [Routing Nets in the GUI](#)

---

## Routing Nets in the GUI

To route nets interactively in the active layout view, draw the routes with the Create Route tool. To activate the Create Route tool, click the  button on the Edit toolbar or choose Create > Route.

To draw route segments, you click points in the layout view. The tool displays flylines and target port and pin locations to guide you in drawing the route segments. It can also check for routing design rule violations as you draw the route segments. You can set options to adjust the routing, control the tool operation, and enable or disable routing aids.

By default, the Create Route tool

- Ignores routing blockages

To force the tool to honor these blockages, change the setting in the Mouse Tool Options panel.

**Note:**

The Create Route tool does not honor routing guides defined by the `create_routing_guide` command.

- Uses the metal width and metal spacing requirements defined in the technology file and ignores nondefault routing rules

To force the tool to honor nondefault routing rules, change the setting in the Mouse Tool Options panel. When you enable this feature, the Create Route tool honors the metal width and metal spacing requirements from the nondefault routing rule and uses these settings to determine the width and pitch of the routes.

**Note:**

The shielding width and shield spacing defined in the nondefault routing rule are not used by the Create Route tool.

If you route on grid and know that there are no obstacles, you can reduce runtime by skipping the spacing checks during automatic welding and automatic alignment. To force the tool to ignore the spacing requirements during these tasks, change the settings in the Mouse Tool Options panel.

You can reverse and reapply Create Route tool operations by using the GUI undo and redo capabilities.

For detailed information about using the Create Route tool, click  on the Mouse Tool Options panel after you activate the Create Route tool. This opens a Help page in the man page viewer.

## Modifying Routed Nets

You can modify routed nets in the GUI by using the following tools:

- Area Push tool

To activate the Area Push tool, click the  button on the Edit toolbar or choose Edit > Area Push. You can use the Area Push tool to move unfixed objects away from a rectangular area on a layer while maintaining their physical connections. You select the layer and control whether the tool complies with nondefault routing rules. The tool supports both interactive and batch push operations.

- Spread Wire tool

To activate the Spread Wires tool, click the  button on the Edit toolbar or choose Edit > Spread Wires. You can use the Spread Wires tool to move selected, unfixed wires evenly between two points on a layer while maintaining their physical connections. You control whether the tool spreads the wires by layer and whether the tool complies with nondefault routing rules.

- Stretch Connected tool

To activate the Stretch Connected tool, click the  button or choose Edit > Stretch Connected. You can use the Stretch Connected tool to move and stretch unfixed wire shapes while optionally maintaining their physical connections.

- Quick Connect tool

To activate the Quick Connect tool, click the  button or choose Edit > Route Utilities > Quick Connect. You can use the Quick Connect tool to quickly connect wires to pin shapes, port shapes, terminals, or other wires.

If you need assistance while using these tools, click  to open a Help page in the man page viewer.

## Cleaning Up Routed Nets

After routing is complete, you can clean up the routed nets by running the `remove_redundant_shapes` command.

```
fc_shell> remove_redundant_shapes
```

By default, this command reads the DRC information stored in the design view of the block and then removes dangling and floating net shapes from all nets in the block based on this information. To run the `check_routes` command to get the DRC information instead of using the information stored in the design view, use the `-initial_drc_from_input false` option.

You can restrict the removal to

- Specific nets by using the `-nets` option
- Specific layers by using the `-layers` option
- Fixed or unfixed route types by using the `-route_types` option

When removing dangling net shapes, the tool does not change topologies or connections and does not touch terminals. In addition, no changes are made to open nets or nets with DRC violations.

You can disable the removal of dangling net shapes by using the `-remove_dangling_shapes false` option. You can disable the removal of floating net shapes by using the `-remove_floating_shapes false` option.

In addition to removing dangling and floating net shapes, this command can also remove loops in the specified nets. To remove loops, use the `-remove_loop_shapes true` option.

By default, the `remove_redundant_shapes` does not report the changes it makes. To report the changes, use the `-report_changed_nets true` option.

For example, to remove dangling net shapes, floating net shapes, and loops from the net named `my_net`, use the following command:

```
fc_shell> remove_redundant_shapes -nets my_net \
 -remove_loop_shapes true
```

After cleaning up the routed nets, reverify the routing, as described in [Performing Design Rule Checking Using Zroute](#).

---

## Analyzing the Routing Results

You can analyze the routing results by reporting on the cell placement and routing statistics. The following topics describe how to perform these tasks:

- [Generating a Congestion Report](#)
  - [Generating a Congestion Map](#)
  - [Performing Design Rule Checking Using Zroute](#)
  - [Performing Signoff Design Rule Checking](#)
  - [Performing Design Rule Checking in an External Tool](#)
  - [Performing Layout-Versus-Schematic Checking](#)
  - [Reporting the Routing Results](#)
  - [Reporting the Wire Length](#)
  - [Using the DRC Query Commands](#)
- 

### Generating a Congestion Report

To generate a congestion report, run the `report_congestion` command.

```
fc_shell> report_congestion
```

By default, the `report_congestion` command uses the congestion map stored with the block to report an overflow summary for the entire block. If a congestion map is not stored with the design, the command generates a congestion map by running global routing in congestion-map-only mode.

**Note:**

By default, only the hard congestion data is saved in the design library. To also save the soft congestion data, set the `route.global.export_soft_congestion_maps` application option to `true` before performing global routing.

The command calculates the overflow as the sum of the overflow for each layer, ignoring any underflow. [Example 25](#) shows the default report, which includes only the hard congestion data.

*Example 25 Default Global Route Congestion Report*

```

Report : congestion
Design :
```

```

Version:
Date :

Layer | overflow | # GRCs has
Name | total | max | overflow (%) | max overflow

Both Dirs | 39 | 8 | 14 (0.26%) | 1
H routing | 6 | 2 | 4 (0.07%) | 2
V routing | 33 | 8 | 10 (0.18%) | 1

```

In the default congestion report,

- “H routing” refers to results for horizontal routes only and “V routing” refers to results for vertical routes only.
- The total overflow value is the total number of wires in the block that do not have a corresponding track available. The max overflow value is the highest number of overutilized wires in a single global routing cell.
- The GRCs overflow value is the total number of overcongested global routing cells in the design. The GRCs max overflow value is the number of global routing cells that have the maximum overflow.

**Note:**

The overflow and global routing cell numbers reported by the `report_congestion` command might look slightly more optimistic than those reported by the `route_global` command because the tool rounds down the congestion information before saving it with the design.

Use the following options to modify the default behavior:

- `-rerun_global_router`

Use this option to rerun the global routing even if the block already has a congestion map.

- `-boundary coordinates`

Use this option to restrict the reporting to a specific region of the block.

- `-layers layers`

Use this option to restrict the reporting to specific layers.

- `-mode global_route_cell_edge_based`

Use this option to report overflow information for each global routing cell.

- `-include_soft_congestion_maps`

Use this option to output soft congestion reports, if they exist. A soft congestion report includes the demand from the soft nondefault spacing rules, as well as tool-generated soft rules. The command outputs a soft congestion report for each spacing weight level of the soft nondefault spacing rules used in the block. Each soft congestion report contains the sum of the hard and soft congestion with a weight level of at least the current soft level.

### See Also

- [Global Routing](#)
- [Defining Minimum Wire Spacing Rules](#)
- [Generating a Congestion Map](#)

---

## Generating a Congestion Map

To display the global route congestion map, choose **View > Map > Global Route Congestion** in the GUI. If the design library contains global route congestion information for the block, the tool generates the congestion map based on this information; otherwise, you must click Reload to generate the congestion map. When you click Reload, the tool opens a dialog box that contains the following command:

```
route_global -congestion_map_only true
```

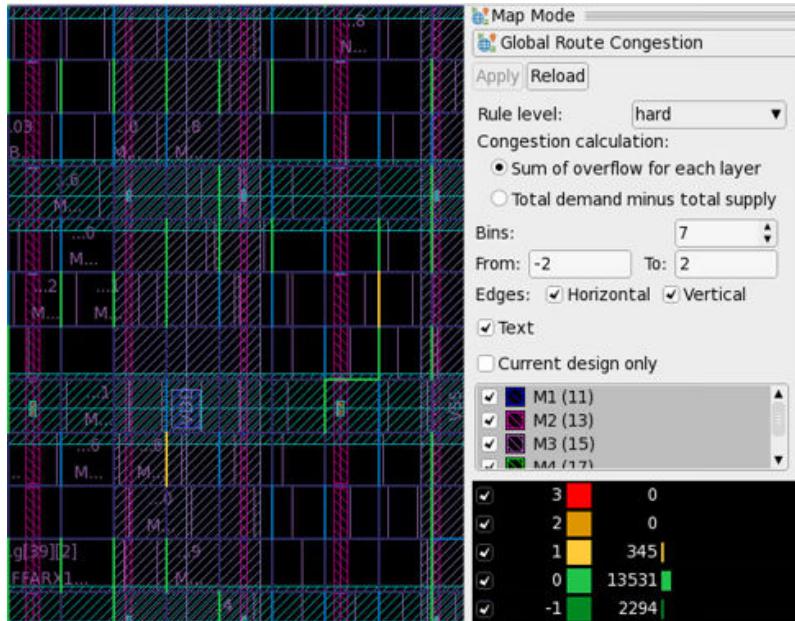
**Note:**

By default, only the hard congestion data is saved in the design library. To also save the soft congestion data, set the `route.global.export_soft_congestion_maps` application option to `true` before performing global routing.

When you click OK in this dialog box, the tool generates a new congestion map. If you want to use different options for the `route_global` command, you can modify this command before clicking OK.

[Figure 103](#) shows an example of a congestion map.

*Figure 103 Global Route Congestion Map*



By default, the congestion map displays only the hard congestion data. To display soft congestion data, select the desired rule level in the Rule level drop-down list. The rule level refers to the spacing weight level of the soft nondefault spacing rules. The congestion map displays the sum of the hard and soft congestion with a weight level of at least the selected rule level.

The congestion map shows the borders between global routing cells highlighted with different colors that represent the congestion values. The congestion map supports two methods for calculating the congestion value:

- Sum of overflow for each layer (the default)

In this mode, the tool calculates the congestion value as the sum of the overflow for all selected layers. Underflow is not considered; if a layer has underflow, it contributes zero overflow to the total overflow calculation.

- Total demand minus total supply

In this mode, the tool calculates the congestion value by subtracting the supply for all selected layers from the demand for all selected layers. Note that because this calculation considers the underflow, it produces a more optimistic congestion result in regions that contain both overflow and underflow.

For example, assume that the METAL2 layer is heavily congested with a demand of 13 routing tracks and a supply of only 7 tracks for an overflow of 6. The METAL4 is

moderately congested with an overflow of 4, and the METAL6 layer is not congested and contains an underflow of 3. Other layers are not used in the congestion calculation in this example.

- The sum of overflow calculation results in a congestion value of  $6+4=10$ . The underflow of 3 for the METAL4 layer is not used in the calculation.
- The total demand calculation results in a congestion value of  $6+4-3=7$ . In this case, the underflow of 3 for the METAL4 layer is used in the calculation.

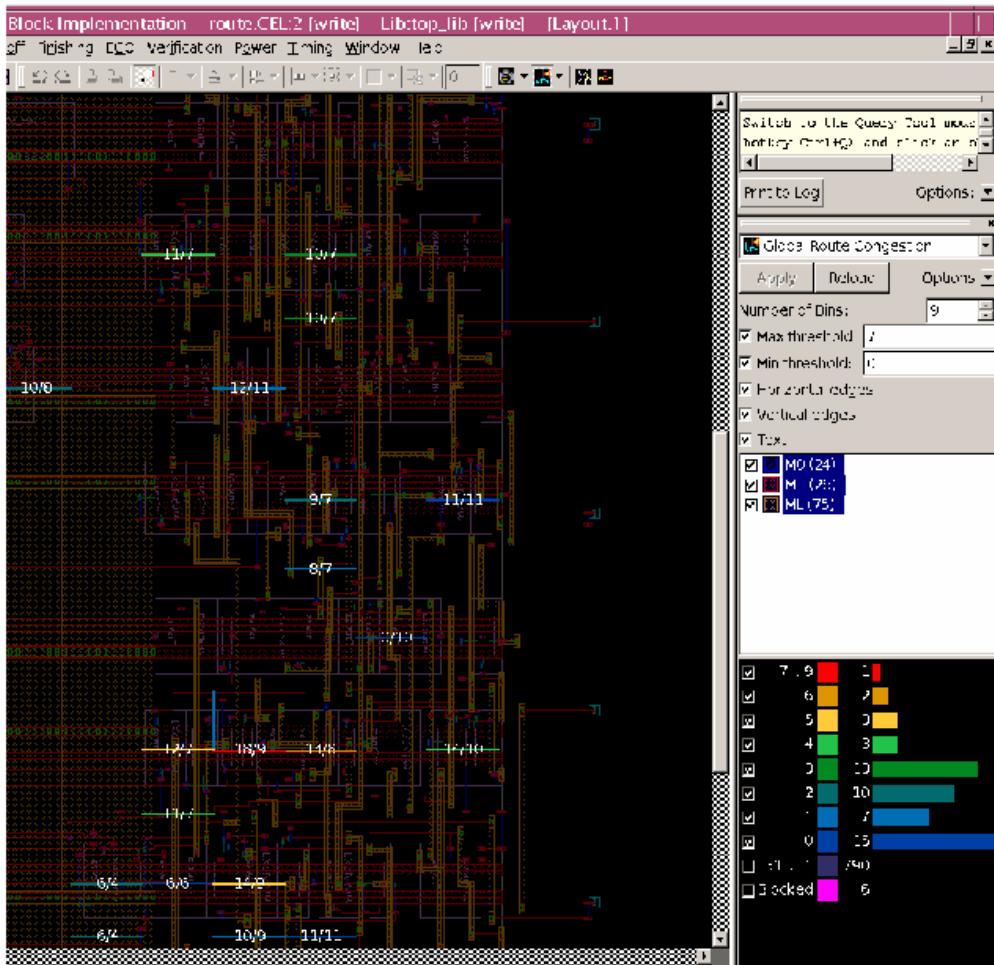
To select the mode, select either “Sum of overflow for each layer” or “Total demand minus total supply” in the “Congestion calculation” section of the Map Mode panel.

By default, all metal layers are selected in the congestion map, except those specified as ignored layers with the `set_ignored_layers` command. To display the congestion map for a subset of layers, select (or deselect) the layers on the Map Mode panel. For example, if the global routing report shows that the maximum overflow occurs on the METAL2 layer, you can deselect all layers, except for METAL2, to display only the METAL2 congestion.

The Map Mode panel also displays a histogram showing the number of global routing cells in different ranges (bins) of congestion values for the selected layers. If your block contains global routing cells that have no available routing resources, an additional bin named Blocked is displayed that shows the number of global routing cells with no routing resources. You can select which bins to display in the congestion map by selecting or deselecting them on the Map Mode panel.

If the block shows congested areas, zoom into the congested area to see the congestion value on the global routing cell. For example, in [Figure 104](#), the red highlight on the edge of the global routing cell shows 18/9. This means there are 9 wire tracks available, but 18 tracks are needed.

*Figure 104 Global Route Overflow on Global Routing Cell*



# Performing Design Rule Checking Using Zroute

To use Zroute to check the routing design rules defined in the technology file, run the `check_routes` command.

By default, the `check_routes` command checks for routing DRC violations, unconnected nets, antenna rule violations, and voltage area violations on all routed signal nets in the block, except those marked as user nets, frozen nets, and PG nets.

- To verify the routing only for specific nets, specify the nets by using the `-nets` option.
  - To check user routes, set the `-check_from_user_shapes` option to true.
  - To check frozen routes, set the `-check_from_frozen_shapes` option to true.

A net is considered frozen when its physical status attribute is set to locked.

To disable checks for routing DRC violations, set the `-drc` option to `false`. To disable checks for unconnected nets, set the `-open_net` option to `false`. To disable checks for antenna rule violations, set the `-antenna` option to `false`. To disable checks for voltage area violations, set the `-voltage_area` option to `false`.

To save time, you can restrict the routing verification to specific regions of the block by using the `-coordinates` option to specify the lower-left and upper-right coordinates for each rectangular region. When you perform area-based DRC, the `check_routes` command checks only for DRC violations and voltage area violations. It does not check for unconnected nets, antenna violations, or tie-to-rail violations, as these are net-based violations.

**Note:**

The `-coordinates` option and the `-nets` option are mutually exclusive; you can use only one of these options.

The `check_routes` command reports the following DRC violations:

- Spacing violations
  - Different-net wire spacing
  - Different-net nondefault wire spacing (note that the DRC report refers to nondefault routing rules as variable rules)
  - Different-net via-cut spacing
  - Different-net nondefault via-cut spacing (note that the DRC report refers to nondefault routing rules as variable rules)
  - Different-net fat extension spacing
  - Dog bone spacing
  - End-of-line spacing
  - Enclosed via spacing
  - Same-net spacing
  - Same-net via-cut spacing
  - Same-net fat extension spacing
  - Special notch spacing
  - U-shape spacing
  - Via-cut to metal spacing
  - Soft spacing

- Area violations
  - Less than minimum area
  - Less than minimum enclosed area
  - Fat wire via keepout area
  - Jog wire via keepout area
- Length and width violations
  - Less than minimum width
  - Less than minimum length
  - Less than minimum edge length
  - Protrusion length
- Contact violations
  - Needs fat contact
  - Needs poly contact
  - Needs fat contact on extension
  - Over maximum stack level
- Enclosure violations
  - End-of-line wire via enclosure
  - Jog wire via enclosure
  - T-shape wire via enclosure
- Others
  - Open nets, except when doing area-based DRC

By default, the `check_routes` command reports a maximum of 200 open nets. To report all open nets, use the `-report_all_open_nets true` option (or select “Report all open nets” in the GUI).

- Antenna violations, except when doing area-based DRC
- Nets crossing the top-cell boundary
- Frozen layers
- Minimum layer

- Maximum layer
- Voltage area violations

**Note:**

These violations are also reported after each detail route iteration.

The `check_routes` command saves the error data to a file named `zroute.err`. You cannot control the naming of the error data file generated by the `check_routes` command, but you can rename the error data file after running the command. To rename the error data file, use the following commands:

```
fc_shell> write_drc_error_data -error_data zroute.err \
 -file_name export.err
fc_shell> open_drc_error_data -file_name export.err
fc_shell> attach_drc_error_data [get_drc_error_data export.err] \
 -name newname.err
```

After you run the `check_routes` command, you can use the DRC query commands to get more information about the violations or use the error browser to examine the violations in the GUI. For information about analyzing the DRC violations, see [Using the DRC Query Commands](#). For information about using the error browser, see the *Fusion Compiler Graphical User Interface User Guide*.

After running `check_routes`, you can use the following command to run incremental detail routing that uses the `check_routes` results as input:

```
fc_shell> route_detail -incremental true \
 -initial_drc_from_input true
```

**Note:**

Incremental detail routing does not fix open nets. To fix open nets, you must run ECO routing. For information about ECO routing, see [Performing ECO Routing](#).

## Performing Signoff Design Rule Checking

Signoff design rule checking runs the IC Validator tool within the Fusion Compiler tool to check the routing design rules defined in the foundry runset. To perform signoff design rule checking, run the `signoff_check_drc` command, as described in [Performing Signoff Design Rule Checking](#). In addition, you can use the `signoff_fix_drc` command to automatically fix the DRC violations detected by the `signoff_check_drc` command. For more information, see [Automatically Fixing Signoff DRC Violations](#).

**Note:**

An IC Validator license is required to run the `signoff_check_drc` and `signoff_fix_drc` commands.

---

## Performing Design Rule Checking in an External Tool

You can perform design rule checking with the Calibre tool, convert the Calibre DRC error file to an Fusion Compiler error data file, and then report or view the errors in the Fusion Compiler tool. To convert the Calibre DRC error file to an Fusion Compiler error data file, use the `read_drc_error_file` command.

For example,

```
fc_shell> read_drc_error_file -file Calibre_error_file
```

By default, the command generates an error data file named `cell_name.err`, where `cell_name` is derived from the Calibre error report. You can specify a name for the error data file by using the `-error_data` option.

**Note:**

The `read_drc_error_file` command supports only flat Calibre DRC error files; it does not support Calibre hierarchy DRC error files.

You can load the error data file created by the `read_drc_error_file` command into the error browser to report or display the DRC violations. For information about using the error browser, see the *Fusion Compiler Graphical User Interface User Guide*.

---

## Performing Layout-Versus-Schematic Checking

To perform layout-versus-schematic (LVS) checking, which checks for inconsistencies in the physical layout, use the `check_lvs` command.

By default, the `check_lvs` command performs the following checks for all signal, clock, and PG nets:

- Shorted nets

A shorted net occurs when a net shapes from different nets touch or intersect.

By default, the command

- Checks for shorts between net shapes in the top-level design, including shapes in top-level blockages

To disable checking for shapes in top-level blockages, use the `-check_top_level_blockages false` option.

- Does not check for shorts between net shapes in the top-level design and net-shapes in child cells

To enable this checking, use the `-check_child_cells true` option. To exclude certain types of child cells from checking, use the `-exclude_child_cell_types`

option to specify one or more of the following cell types: abstract, analog, black\_box, corner, cover, diode, end\_cap, fill, filler, flip\_chip\_driver, flip\_chip\_pad, lib\_cell, macro, module, pad, pad\_spacer, physical\_only, and well\_tap.

- Does not check for shorts with zero-spacing blockages

To enable this checking, use the `-check_zero_spacing_blockages true` option.

**Note:**

The `check_lvs` command supports only default routing blockages that apply to all net types and have a blockage group ID of 0. If the blockage applies only to specific net types or has a nonzero blockage group ID, the command ignores the blockage. In addition, the command ignores corridor routing blockages (which are created when you use the `-reserve_for_top_level_routing` option with the `create_routing_blockage` command) and boundary routing blockages (which are created when you use the `-boundary_internal` or `-boundary_external` options with the `create_routing_blockage` command).

- Open nets

An open net occurs when the pins of a net are not connected by its net shapes.

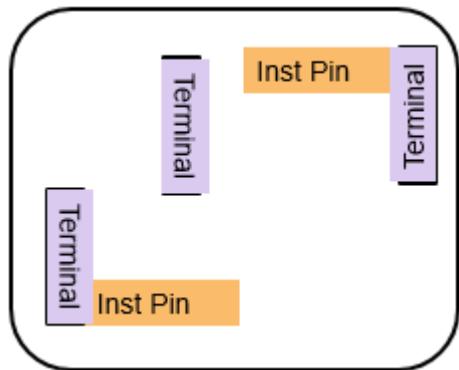
By default, the command

- Treats power and ground terminals as unconnected voltage sources

To treat power and ground terminals as connected, use the `-treat_terminal_as_voltage_source true` option.

For example, assume your block contains the layout shown in [Figure 105](#). By default, the `check_lvs` command reports two opens for this layout. If you use the `-treat_terminal_as_voltage_source true` option, no opens are reported.

Figure 105 Layout With Power and Ground Terminals



- Reports open nets as the bounding box of the open net and does not report floating pins

To report detailed open locations, use the `-open_reporting detailed` option. Note that using this option might increase the runtime. To report the floating pins, use the `-report_floating_pins true` option.

- Floating net shapes

A floating net shape occurs when a net shape is not physically connecting to a pin of its net.

To perform a subset of these checks, use the `-checks` option to specify one or more of the following checks: `short` (shorted nets), `open` (open nets), and `floating_routes` (floating net shapes). To check only specific nets, use the `-nets` option to specify the nets of interest.

By default, the `check_lvs` command reports a maximum of 20 violations for each type of error. To change this limit, use the `-max_errors` option. To report all violations, specify the maximum number of violations as zero (`-max_errors 0`). You can view the violations reported by the `check_lvs` command in the GUI by using the error browser. For information about using the error browser, see the *Fusion Compiler Graphical User Interface User Guide*.

To reduce the runtime required by the `check_lvs` command, enable multithreading by using the `set_host_options` command, as described in [Enabling Multicore Processing](#).

---

## Reporting the Routing Results

To report statistics about the routing results, use the `report_design -routing` command. This command reports the following information:

- Final wiring statistics, including the
  - Number of signal net shapes for each metal layer
  - Signal wire length for each metal layer
  - Number of PG net shapes for each metal layer
  - PG wire length for each metal layer
  - Horizontal and vertical wire distribution for each metal layer
  - Total number of signal net shapes for the block
  - Total signal wire length for the block
- Final via statistics, including the
  - Simple vias used in each via layer, including the number of instances of each via
  - Double via conversion rate for each via layer
  - Double via conversion rate for the block
  - Custom vias used in the block, including the number of instances of each user-defined via

By default, this command reports the information only for the top-level block. To report the information for the entire physical hierarchy, use the `-hierarchical` option.

### See Also

- [Defining Vias](#)
- 

## Reporting the Wire Length

To report the total wire length at any stage of the routing flow, use the consolidated `report_wirerlength` command. This command supports all net types, except power ground nets, by default such as signal nets, clock nets. It also supports non-stripe power ground nets by option control. This command generates a report, which includes the following information in the form of two tables, as shown in the figure:

- Total wire length per layer with separate global and detailed routing lengths
- Total wire length by both horizontal and vertical direction

```

Report : Nets wirelength summary
Design : r4000
Version: S-2021.06-DEV
Date : Mon Apr 19 17:21:33 2021

PER LAYER WIRELENGTH
Layer Global Detailed
M1 0.000 1465.440
M2 0.000 13621.805
M3 0.000 20729.190
M4 0.000 13226.870
M5 0.000 4675.820
M6 0.000 1161.515
M7 0.000 108.640

TOTAL WIRELENGTH Virtual Global Detailed BestAvailable
----- TotalH 26421.001 0.000 27840.660 27840.660
 TotalV 26069.774 0.000 27148.620 27148.620
 Total 52490.775 0.000 54989.280 54989.280

Number of nets = 2741
```

In the generated report,

- Virtual value means the estimated routing wire length for each net. For each net, the engine provides a virtual routing length before real routing. It remains even after `route_auto`.
- BestAvailable means the sum of wire length of all nets irrespective of the stage they are in the current design.

This command also supports slanting wires and counts the real length of an angle wire in total and counts the projection length of the x-axis and y-axis.

## Using the DRC Query Commands

You can get information about DRC violations by using the `get_drc_errors` command to create a collection of DRC violations and then using the `get_attribute` command to query the attributes of the errors. Some attributes that provide information about DRC errors are `type_name`, `bbox`, `objects`, and `shape`. Note that the availability of attribute values depends on the error type and the verification method used. For a list of all attributes associated with DRC errors, use the `list_attributes -application -class drc_error` command.

Before you run the `get_drc_errors` command, you must load the error data files that you want to query. To determine the error data files attached to the current block, use the `get_drc_error_data` command. You must use the `-all` option to include both the opened and unopened error data files in the result.

```
fc_shell> get_drc_error_data -all
```

To load an error data file, use the `open_drc_error_data` command.

```
fc_shell> open_drc_error_data zroute.err
```

To determine the error data types included in the error data file, use the `get_drc_error_types` command.

```
fc_shell> get_drc_error_types -error_data zroute.err
```

By default, the `get_drc_errors` command creates a collection that contains all DRC violations contained in the specified error data file. Use the `-filter` option to restrict the returned errors.

For example, to return only “Diff net spacing” errors, use the following command:

```
fc_shell> get_drc_errors -error_data zroute.err \
 -filter {type_name == "Diff net spacing"}
```

To get the nets associated with an error, use the `get_attribute` command. For example,

```
fc_shell> get_attribute [get_drc_errors -error_data zroute.err 859] \
 objects
{u0_1/n237}
```

## Saving Route Information

To save the route information, use the `write_routes` command. This command generates a Tcl script that reproduces the metal shapes and vias for a block, including their attribute settings.

Note that the `write_routes` command reproduces routes, but not routing blockages. To generate a Tcl script that reproduces routing blockages, use the `write_floorplan` command.

## Deriving Mask Colors

At advanced technology nodes, mask colors must be assigned to wires and vias to ensure correct processing of the net shapes during mask synthesis. The `route_detail` command and other routing commands in the tool usually add the necessary mask colors during routing. For net shapes that do not have a color assignment, use the `derive_mask_constraint` command to derive the color mask for the specified wires and vias from the nearest underlying tracks. In case of wide wires that occupy multiple tracks, the command takes the mask color opposite to the next unoccupied track.

The following example derives the mask color for the net shapes of the `net1` net.

```
fc_shell> derive_mask_constraint \
 [get_shapes -of_objects [get_nets net1]]
```

Use options to the `derive_mask_constraint` command to control how the mask colors are derived.

- Derive cut mask constraints for the specified nets or vias.

```
fc_shell> derive_mask_constraint -derive_cut_mask \
[get_vias -within {{600 600} {700 700}}]
```

- Derive mask constraints from overlapping or touching pins or ports.

```
fc_shell> derive_mask_constraint -follow_pin_mask \
[get_shapes -of_objects [get_nets net1]]
```

- Derive the mask constraint from the wider overlapping or touching object rather than the wire track.

```
fc_shell> derive_mask_constraint -follow_wider_object_mask \
[get_shapes -of_objects [get_nets net1]]
```

## See Also

- [Multiple-Patterning Concepts](#)

## Inserting and Removing Cut Metal Shapes

Cut metal shapes are used in double-patterning designs to reduce the line-end minimum spacing. A cut metal layer and the dimensions of the cut metal shapes inserted on that layer are defined in a `Layer` section in the technology file. The width of the cut metal shapes is defined by the `cutMetalWidth` attribute. The height of the cut metal shapes is defined by either the `cutMetalHeight` or `cutMetalExtension` attribute. A cut metal layer corresponds to the metal layer with the same mask number. For example, `cutMetal1` corresponds to `metal1`. For information about the technology file, see the *Synopsys Technology File and Routing Rules Reference Manual*.

After the routing is finalized, you add cut metal shapes by using the `create_cut_metal` command. This command inserts a cut metal shape between metal shapes when all of the following conditions are met:

- The spacing between the metal shapes in the preferred direction is `cutMetalWidth`.
- There is no existing cut metal shape in the location.
- The metal shapes are on a metal layer that corresponds to a cut metal layer defined in the technology file.

The command inserts cut metal shapes between net shapes, a net shape and a PG shape, a PG shape and an obstruction, and metal shapes within a child cell. After inserting

the cut metal shapes, the command propagates the color of the metal shapes to the cut metal shapes.

When you write a DEF file for a block with cut metal shapes, you must use the `-version 5.8` option with the `write_def` command. When you write a GDSII or OASIS file for a block with cut metal shapes, you must use the `-connect_below_cut_metal` option with the `write_gds` or `write_oasis` command.

If the routing changes after inserting cut metal shapes, you must remove the existing cut metal shapes and reinsert them. To remove all cut metal shapes from a block, use the `remove_cut_metals` command.

# 7

## Chip Finishing and Design for Manufacturing

---

The Fusion Compiler tool provides chip finishing and design for manufacturing and yield capabilities that you can apply throughout the various stages of the design flow to address process design issues encountered during chip manufacturing.

For information about the chip finishing and design for manufacturing features, see the following topics:

- [Inserting Tap Cells](#)
  - [Performing Boundary Cell Insertion](#)
  - [Finding and Fixing Antenna Violations](#)
  - [Inserting Redundant Vias](#)
  - [Optimizing Wire Length and Via Count](#)
  - [Reducing Critical Areas](#)
  - [Inserting Metal-Insulator-Metal Capacitors](#)
  - [Inserting Filler Cells](#)
  - [Inserting Metal Fill](#)
- 

### Inserting Tap Cells

A *tap cell* is a special nonlogic cell with a well tie, substrate tie, or both. These cells are typically used when most or all of the standard cells in the library contain no substrate or well taps. Generally, the design rules specify the maximum distance allowed between every transistor in a standard cell and a well or substrate tap.

Before global placement (during the floorplanning stage), you can insert tap cells in the block to form a two-dimensional array structure to ensure that all standard cells placed subsequently comply with the maximum diffusion-to-tap distance limit. After you insert the tap cells, visually check to ensure that all standard-cell placeable areas are properly protected by tap cells.

The Fusion Compiler tool provides two methods to insert a tap cell array:

- Insert the tap cell array based on the site rows and the maximum distance between tap cells

To use this method, use the `create_tap_cells` command, as described in [Using the `create\_tap\_cells` Command](#).

- Insert the tap cell array using a specified offset and pitch, and optionally insert additional tap cells near the boundary

To use this method, use the `create_cell_array` command, as described in [Using the `create\_cell\_array` Command](#).

Advanced nodes often require the insertion of additional tap cells to manage the substrate and well noise. You can use the following capabilities to insert additional tap cells after standard tap cell insertion:

- Insertion of tap walls

A *tap wall* is a row or column of tap cells placed linearly without any gaps between cells. You can insert a tap wall outside or inside of the boundary of a block, macro, or hard placement blockage. A tap wall that is outside a boundary is called an *exterior tap wall*. A tap wall that is inside a boundary is called an *interior tap wall*.

- To insert an exterior tap wall, use the `create_exterior_tap_walls` command, as described in [Inserting Exterior Tap Walls](#).
- To insert an interior tap wall, use the `create_interior_tap_walls` command, as described in [Inserting Interior Tap Walls](#).

- Insertion of tap meshes

A *tap mesh* is a two-dimensional array of tap cells with a tap cell in each mesh window. To insert a tap mesh, use the `create_tap_meshes` command, as described in [Inserting Tap Meshes](#).

- Insertion of dense tap arrays

A *dense tap array* is a tap cell array whose tap distance is smaller than the tap distance used by the `create_tap_cells` command. To insert a dense tap array, use the `create_dense_tap_cells` command, as described in [Inserting Dense Tap Arrays](#).

---

## Using the `create_tap_cells` Command

To add a tap cell array based on the site rows, use the `create_tap_cells` command. You must specify the name of the library cell to use for tap cell insertion (`-lib_cell` option) and the maximum distance, in microns, between tap cells (`-distance` option).

For example,

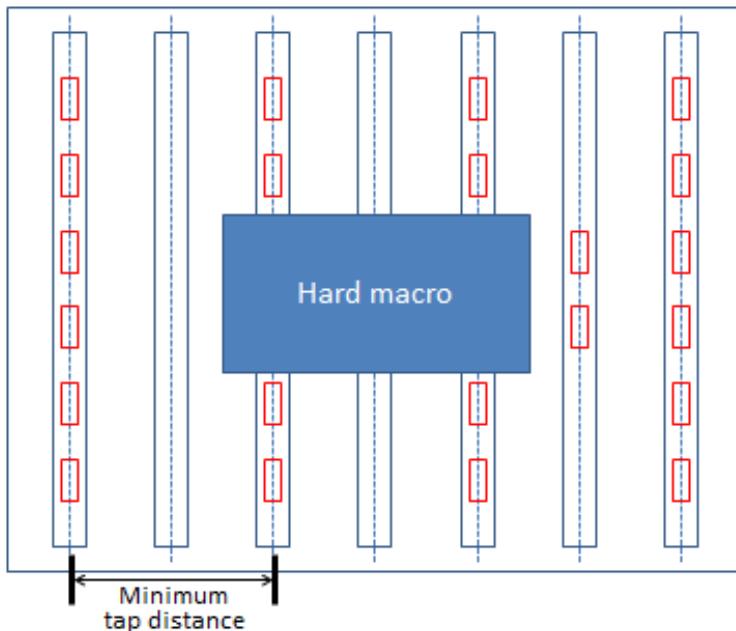
```
fc_shell> create_tap_cells -lib_cell myreflib/mytapcell -distance 30
```

**Note:**

If the `place.legalize.enable_pin_color_alignment_check` application option is `true` (its default is `false`), the command ensures that internal cell pins and metal of the tap cells align with a routing track of the appropriate color when inserting the tap cells and shifts the tap cells to avoid violating color alignment. To avoid signoff DRC errors due to this shifting, reduce the tap distance slightly when this application option is `true`.

By default, the `create_tap_cells` command inserts tap cells in every row for the entire block. The tool starts inserting the tap cells at the left edge of the row and uses the specified tap distance to determine the location of the subsequent tap cells. In addition, if a tap cell does not exist within the minimum tap distance (half the specified tap distance) from each row edge adjacent to the block's boundary, a hard macro, or a hard placement blockage, the tool inserts an additional tap cell. [Figure 106](#) shows the default tap cell placement for a block that uses the every-row insertion pattern. Note that extra tap cells are added to the right of the hard macro to ensure that a tap cell exists within the minimum tap distance from the edge of the hard macro.

*Figure 106 Default Tap Cell Placement*



You can modify the following aspects of the default behavior:

- The library cell used for tap cell insertion on rows with a mirrored (MX) orientation

Use the `-mirrored_row_lib_cell` option to specify the library cell to use for mirrored rows.

- The pattern used to insert the tap cells

Use the `-pattern` option to specify one of the following tap cell insertion patterns:

- `every_row` (the default)

This pattern inserts tap cells in every row. For this pattern, the tap distance specified with the `-distance` option should be approximately twice the maximum diffusion-to-tap value specified in the technology design rules.

- `every_other_row`

This pattern inserts tap cells only in the odd-numbered rows. For this pattern, the tap distance specified with the `-distance` option should be approximately twice the maximum diffusion-to-tap value specified in the technology design rules.

- `stagger`

This pattern inserts tap cells in every row with the tap cells in even rows offset by half the offset value (`-offset` option) relative to the odd rows, which produces a checkerboard-like pattern. For this pattern, the tap distance specified with the `-distance` option should be approximately four times the maximum diffusion-to-tap value specified in the technology design rules.

- The offset from the left edge of the row

Use the `-offset` option to shift the pattern startpoint to the right by the specified distance in microns.

- The handling of fixed cells

By default, tap cells are inserted at the computed tap locations, regardless of whether a fixed cell occupies that location. Use the `-skip_fixed_cells` option to prevent the insertion of tap cells in locations occupied by fixed cells. By default, when you use this option, the command considers the fixed cell as a blockage and breaks the row at the fixed cell, which can result in a tap cell being inserted on each side of the fixed cell. To prevent tap cells from overlapping fixed cells without breaking the row, use the `-preserve_distance_continuity` option with the `-skip_fixed_cells` option. When you use both options, the command shifts the tap cell to avoid overlap with the fixed cell. The `-preserve_distance_continuity` option shifts only those tap cells whose computed tap location is occupied by an instance of one of the specified library cells.

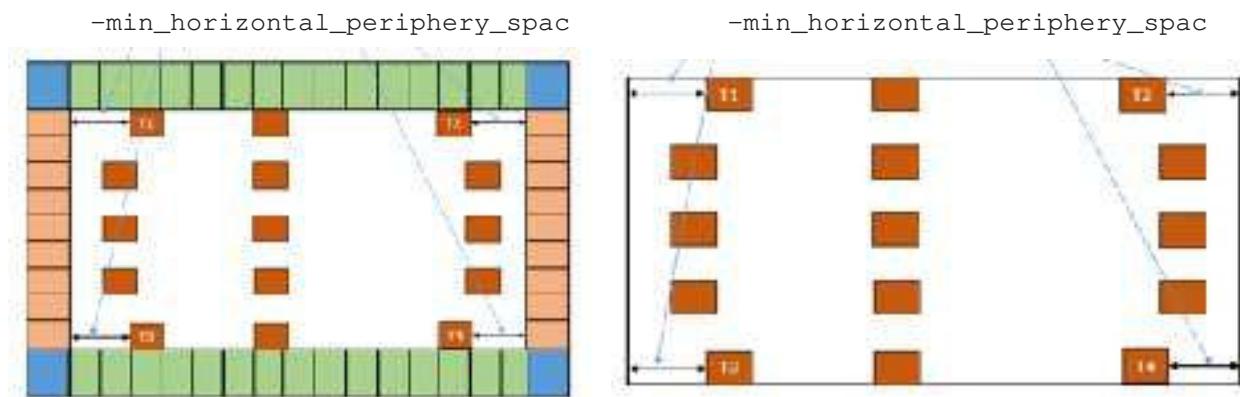
For example, the following command shifts tap cells whose computed tap location is occupied by a fixed instance of the mylib/cell1 library cell:

```
fc_shell> create_tap_cells -lib_cell mylib/tap -distance 30 \
 -skip_fixed_cells -preserve_distance_continuity mylib/cell1
```

When you use the `-skip_fixed_cells` option, you can also specify the minimum horizontal spacing between the boundary and a corner tap cell that is above or below a blocked area by using the `-min_horizontal_periphery_spacing` option. This option specifies the minimum number of unit tiles between the boundary and an affected corner tap cell.

[Figure 107](#) shows how using this option affects the tap cell placement both with and without boundary cells.

*Figure 107 Tap Cell Boundary Row Spacing*



- The addition of extra tap cells

You can use one or both of the following methods to prevent the insertion of extra tap cells:

- Use the `-row_end_tap_bypass` option to specify that the boundary cells contain taps, so additional tap cells are not inserted at the boundary. Note that the command does not verify that the block has boundary cells or that the boundary cells actually contain taps.
- Use the `-at_distance_only` option to allow the insertion of tap cells only at the specified tap distance, half of the tap distance, or one fourth of the tap distance (for the stagger pattern only). Note that using this option can cause DRC violations.

- The regions in which to insert tap cells

Use the `-voltage_area` option to restrict the tap cell insertion to the specified voltage areas.

- The naming convention used to identify the inserted tap cell instances

By default, the tool uses the following naming convention for inserted tap cells:

*tapfiller!library\_cell\_name!number*

Use the `-separator` option to change the separator character from its default of “!”.

To identify the tap cells inserted in a specific run, use the `-prefix` option to specify a prefix string. When you use this option, the tool uses the following naming convention:

*tapfiller!prefix!library\_cell\_name!number*

## Using the `create_cell_array` Command

To add a tap cell array with specific placement, use the `create_cell_array` command. You must specify the name of the library cell to use for tap cell insertion (`-lib_cell` option), and the x- and y-pitches.

For example,

```
fc_shell> create_cell_array -lib_cell myreflib/mytapcell \
 -x_pitch 10 -y_pitch 10
```

By default, the `create_cell_array` command inserts a tap cell in each location in the specified grid for the entire block. The tool starts inserting the tap cells at the bottom-left of the core area and uses the specified x- and y-pitches to determine the location of the subsequent tap cells. The inserted tap cells are snapped to the site rows and placed in the default orientation for the site row. If a location is occupied by a fixed cell, hard macro, soft macro, or power-switch cell, the command does not insert a tap cell at that location.

You can modify the following aspects of the default behavior:

- The region in which to insert tap cells

Use the `-voltage_area` option to restrict the tap cell insertion to the specified voltage areas.

Use the `-boundary` option to restrict the tap cell insertion to the specified region.

If you use both options, the command inserts tap cells only in the overlapping region.

- The offset from the insertion region boundary

Use the `-x_offset` option to shift the pattern startpoint to the right by the specified distance in microns.

Use the `-y_offset` option to shift the pattern startpoint up by the specified distance in microns.

- The insertion pattern

Use the `-checkerboard` option to insert a tap cell in every other location in the specified grid.

- To place a tap cell in the lower-left corner, use the `-checkerboard even` option.
- To keep the lower-left corner empty, use the `-checkerboard odd` option.

In some design methodologies, you must insert a tap cell in empty spaces in the checkerboard pattern that are above or below specific cells, such as critical area breaker cells or boundary cells. To enable the insertion of these extra tap cells, use the `-preserve_boundary_row_lib_cells` option to specify the affected library cells.

- The snapping behavior

To disable snapping of the inserted tap cells to the site rows, set the `-snap_to_site_row false` option. When you set this option to `false`, the tool uses `R0` as the default orientation for the inserted cells.

- The orientation of the inserted tap cell instances

To specify the orientation of the inserted tap cells, use the `-orient` option to specify one of the following orientation values: `R0`, `R90`, `R180`, `R270`, `MX`, `MY`, `MXR90`, or `MYR90`.

- The naming convention used to identify the inserted tap cell instances

By default, the tool uses the following naming convention for inserted tap cells:

`headerfooter_library_cell_name_R#_C#_number`

To identify the tap cells inserted in a specific run, use the `-prefix` option to specify a prefix string. When you use this option, the tool uses the following naming convention:

`prefix_library_cell_name_R#_C#_number`

## Inserting Exterior Tap Walls

An *exterior tap wall* is a row or column of tap cells that is placed outside the boundary of a block, macro, or hard placement blockage. The tap cells in the tap wall are placed linearly without any gaps between cells.

To insert an exterior tap wall, use the `create_exterior_tap_walls` command. You must specify the following information:

- The tap cell used in the tap wall (the `-lib_cell` option)

By default, the command inserts the tap cells with R0 orientation. To place the cells with a different orientation, use the `-orientation` option.

If the design has a FinFET grid, follow these guidelines when selecting the tap cell to ensure proper insertion of the tap wall:

- For horizontal tap walls, ensure that the cell's width in the specified orientation is an integer multiple of the x-pitch of the FinFET grid
- For vertical tap walls, ensure that the cell's height in the specified orientation is an integer multiple of the y-pitch of the FinFET grid

By default, the tool uses the following naming convention for inserted tap cells:

`extTapWall__library_cell_name_R#_C#_number`

To identify the tap cells inserted in a specific run, use the `-prefix` option to specify a prefix string. When you use this option, the tool uses the following naming convention:

`prefix__library_cell_name_R#_C#_number`

- The side along which to insert the tap wall (the `-side` option)

You can specify only a single side, `top`, `bottom`, `right`, or `left`. To insert a tap wall on more than one side, run the command multiple times.

- The tap insertion region (the `-bbox` option)

You specify the tap insertion region by specifying the lower-left and upper-right corners of its bounding box:

```
-bbox {{llx lly} {urx ury}}
```

The bounding box must span a portion of the object's edge on the specified side along which to insert the tap wall. If the bounding box spans multiple edges along the specified side, the command inserts the tap wall along the longest edge. If the bounding box does not span any edges of the specified side, the command issues an error message.

By default, a horizontal tap wall starts at the left edge of the specified bounding box and abuts the top or bottom boundary edge; a vertical tap wall starts at the bottom edge of the specified bounding box and abuts the left or right boundary edge. To specify margins for the tap wall insertion, use the `-x_margin` and `-y_margin` options. If the specified region, including the margins, is too small to fit a tap wall, the command does not insert tap cells.

You should ensure that the insertion region does not contain any placement blockages, fixed cells, or macros. These obstructions can prevent the insertion of the tap cells and result in an incomplete or missing tap wall.

When inserting the tap wall, the command does not honor the standard cell rows or sites and does not cross the object boundary. After insertion, the command fixes the placement of the inserted tap cells.

---

## Inserting Interior Tap Walls

An *interior tap wall* is a row or column of tap cells that is placed inside the boundary of the standard cell placement area of a block, a macro, hard placement blockage, or nondefault voltage area.

**Note:**

You can insert interior tap walls only for designs with a single site definition.

To insert an interior tap wall, use the `create_interior_tap_walls` command. You must specify the following information:

- The tap cell used in the tap wall (the `-lib_cell` option)

The specified tap cell must be a single-height cell that matches the row height and site width.

By default, the command inserts the tap cells with the row orientation. To place the cells with a different orientation, use the `-orientation` option.

By default, the tool uses the following naming convention for inserted tap cells:

`tapfiller__library_cell_name_R#_C#_number`

To identify the tap cells inserted in a specific run, use the `-prefix` option to specify a prefix string. When you use this option, the tool uses the following naming convention:

`prefix__library_cell_name_R#_C#_number`

- The side along which to insert the tap wall (the `-side` option)

You can specify only a single side, `top`, `bottom`, `right`, or `left`. To insert a tap wall on more than one side, run the command multiple times.

By default, the tap cells are placed along all edges on the specified side of the standard cell placement area with no gaps between the tap cells.

- To specify the tap insertion region, use the `-bbox` option to specify the lower-left and upper-right corners of its bounding box:

```
-bbox {{llx lly} {urx ury}}
```

When you use this option, the command inserts a tap wall along the longest edge on the specified side of the placeable area encompassed by the bounding box. The alignment of the tap wall depends on the number of corners of the edge segment encompassed by the bounding box.

- If the edge segment does not contain a corner, the tap wall starts at the left edge for a horizontal tap wall or the bottom edge for a vertical tap wall.
- If the edge segment contains one corner, the tap wall starts at that corner.
- If the edge segment contains two corners along the specified side, the tap wall starts at the left edge for a horizontal tap wall or the bottom edge for a vertical tap wall.

For a horizontal tap wall with two corners, the spacing between the last two cells might be shortened to align with the right corner. If there is not enough space, the last cell might not abut the right corner.

- For horizontal tap walls, you can specify a gap distance between tap cells by using the `-x_spacing` option.

If you specify a value smaller than the tap cell width, the command uses a spacing of zero. If you specify a value that is not a multiple of the site width, the command rounds it down to a multiple of the site width.

When inserting the tap wall, the command honors the standard cell rows and sites; if the rows and sites are not defined, the command does not insert tap cells. Obstructions such as placement blockages, fixed cells, and macros can prevent the insertion of the tap cells and result in an incomplete or missing tap wall.

After insertion, the command fixes the placement of the inserted tap cells.

## Inserting Tap Meshes

A *tap mesh* is a two-dimensional array of tap cells with a tap cell in each mesh window. A tap mesh is typically inserted outside of a place and route block, but you can insert one in any area that is either completely inside or completely outside the core area.

To insert a tap mesh, use the `create_tap_meshes` command. You must specify the following information:

- The tap cell used in the tap mesh (the `-lib_cell` option)

By default, the command inserts the tap cells with R0 orientation. To place the cells with a different orientation, use the `-orientation` option.

By default, the tool uses the following naming convention for inserted tap cells:

`headerfooter__library_cell_name_R#_C#_number`

To identify the tap cells inserted in a specific run, use the `-prefix` option to specify a prefix string. When you use this option, the tool uses the following naming convention:

`prefix__library_cell_name_R#_C#_number`

- The tap insertion region (the `-bbox` option)

You specify the tap insertion region by specifying the lower-left and upper-right corners of its bounding box:

`-bbox {{llx lly} {urx ury}}`

- The mesh window (the `-mesh_window` option)

You specify the size of the mesh window by specifying an integer value for the pitch in the x- and y-directions.

`-mesh_window {x_pitch y_pitch}`

The mesh window is aligned with the lower-left corner of the tap insertion region.

When inserting the tap mesh, the command does not honor the standard cell rows or sites. It places a tap cell in the lower-left corner of each mesh window. If that location is blocked, the command uses the closest available location, which is measured as the Manhattan distance to the lower-left corner of the mesh window. If there is no available location, the command issues a warning and does not place a tap cell in that mesh window.

After insertion, the command fixes the placement of the inserted tap cells.

---

## Inserting Dense Tap Arrays

A *dense tap array* is a tap cell array whose tap distance is smaller than the tap distance used by the `create_tap_cells` command.

To insert a dense tap array, use the `create_dense_tap_cells` command. You must specify the following information:

- The tap cell used in the tap wall (the `-lib_cell` option)

You must specify the same tap cell as when you ran the `create_tap_cells` command.

By default, the tool uses the following naming convention for inserted tap cells:

`tapfiller!library_cell_name!number`

Use the `-separator` option to change the separator character from its default of “!”.

To identify the tap cells inserted in a specific run, use the `-prefix` option to specify a prefix string. When you use this option, the tool uses the following naming convention:

`tapfiller!prefix!library_cell_name!number`

- The tap distance (the `-distance` option)

The specified distance must be smaller than the tap distance used by the `create_tap_cells` command.

- The tap insertion region (the `-bbox` option)

You specify the tap insertion region by specifying the lower-left and upper-right corners of its bounding box:

`-bbox {{llx lly} {urx ury}}`

The `create_dense_tap_cells` command first removes the existing tap cells that are completely within the tap insertion region, and then reinserts the tap cells with the specified tap distance to create a denser array in the specified region. To avoid tap rule violations, specify the same tap cell insertion pattern as was used by the `create_tap_cells` command. The `create_dense_tap_cells` command supports the following options to control the tap cell insertion: `-offset`, `-pattern`, `-skip_fixed_cells`, and `-at_distance_only`. For details about these options, see the man page.

After insertion, the command fixes the placement of the inserted tap cells.

---

## Performing Boundary Cell Insertion

Before placing the standard cells, you can add boundary cells to the block. Boundary cells consist of end-cap cells, which are added to the ends of the cell rows and around the boundaries of objects such as the core area, hard macros, blockages, and voltage areas, and corner cells, which fill the empty space between horizontal and vertical end-cap cells. End-cap cells are typically nonlogic cells such as a decoupling capacitor for the power rail. Because the tool accepts any standard cell as an end-cap cell, ensure that you specify suitable end-cap cells.

To insert boundary cells,

1. Specify the boundary cell insertion requirements by using the `set_boundary_cell_rules` command, as described in [Specifying the Boundary Cell Insertion Requirements](#).
2. Insert the boundary cells based on the specified rules by using the `compile_boundary_cells` or `compile_targeted_boundary_cells` command, as described in [Inserting Boundary Cells](#).
3. Verify the boundary cell placement by using the `check_boundary_cells` or `check_targeted_boundary_cells` command, as described in [Verifying the Boundary Cell Placement](#).

This process supports a single site definition per run. The `compile_boundary_cells` and `compile_targeted_boundary_cells` commands determine the site definition based on the library cells specified with the `set_boundary_cell_rules` command, and insert boundary cells only in the regions that use this site definition. If your design has multiple site definitions, you must run this process one time for each site definition.

---

## Specifying the Boundary Cell Insertion Requirements

To specify the boundary cell insertion requirements, use the `set_boundary_cell_rules` command. You use this command to specify the following requirements:

- The library cells to use for the boundary cells, as described in [Specifying the Library Cells for Boundary Cell Insertion](#)
- The rules used to place the boundary cells, as described in [Specifying Boundary Cell Placement Rules](#)
- The naming convention used for the inserted cells, as described in [Specifying the Naming Convention for Boundary Cells](#)
- The creation of routing guides to honor the metal cut allowed and forbidden preferred grid extension rules, as described in [Creating Routing Guides During Boundary Cell Insertion](#)

**Note:**

This feature is supported only by the `compile_boundary_cells` command; it is not supported by the `compile_targeted_boundary_cells` command.

- The creation of placement blockages to honor the minimum jog and minimum separation rules, as described in [Creating Placement Blockages During Boundary Cell Insertion](#)

**Note:**

This feature is supported only by the `compile_boundary_cells` command; it is not supported by the `compile_targeted_boundary_cells` command.

**Note:**

The settings specified by the `set_boundary_cell_rules` command are saved in the design library.

**See Also**

- [Reporting the Boundary Cell Insertion Requirements](#)

## Specifying the Library Cells for Boundary Cell Insertion

Boundary cells include both end-cap cells placed on the left, right, top, and bottom boundaries, and inside and outside corner cells. You can specify different library cells for each boundary cell type. To specify the library cells, use the following options with the `set_boundary_cell_rules` command:

- `-left_boundary_cell` and `-right_boundary_cell`

These options specify a single library cell that is used for the end-cap cells for the left and right boundaries, respectively.

- `-top_boundary_cells` and `-bottom_boundary_cells`

These options specify a list of library cells that are used for the end-cap cells for the top and bottom boundaries, respectively. The command inserts the cells in the specified order. If the remaining space is smaller than the current cell, the command inserts the next cell in order that fits in the remaining space.

For a vertical-row block, rows start at the bottom and end at the top, so the top boundary is along the left side of the block and the bottom boundary is along the right side of the block.

For the flipped rows in a double-back block, the top boundary cells are used on the bottom boundaries and the bottom boundary cells are used on the top boundaries.

- `-top_tap_cell` and `-bottom_tap_cell`

These options specify a single library cell that is used for the tap cells on the top and bottom boundary rows, respectively. The tool inserts the tap cells to ensure that the end-cap cells inserted on the top and bottom boundary rows comply with the maximum diffusion-to-tap distance limit.

- `-top_left_outside_corner_cell`, `-top_right_outside_corner_cell`,  
`-bottom_left_outside_corner_cell`, and `-bottom_right_outside_corner_cell`

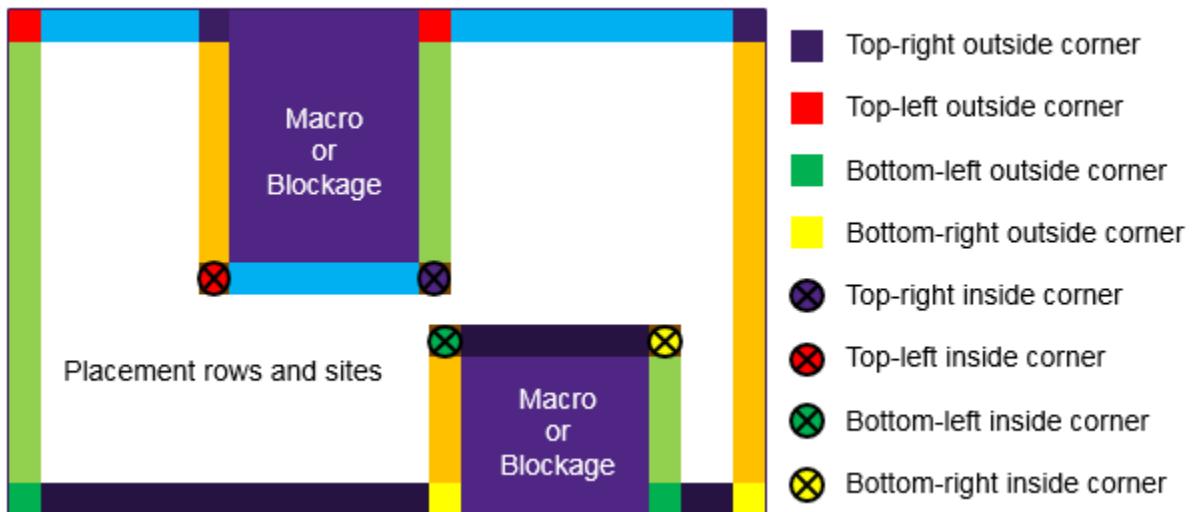
These options specify a single library cell that is used for each outside corner location.

- `-top_left_inside_corner_cells`, `-top_right_inside_corner_cells`,  
`-bottom_left_inside_corner_cells`, and `-bottom_right_inside_corner_cells`

These options specify a list of library cells for each inside corner location. The tool inserts the first corner cell that matches the size of the inside corner. If none matches exactly, it inserts the first cell that can be placed without violating any rules.

[Figure 108](#) shows the boundary and corner cell locations for a horizontal-row block with two hard macros or blockages. Note that the cell locations are flipped when the row is flipped.

*Figure 108    Boundary Cell Locations*



## Specifying Boundary Cell Placement Rules

By default,

- When you run the `compile_boundary_cells` command, the tool places the boundary cells in their default orientation around the core area, hard macros, and hard placement blockages.
- When you run the `compile_targeted_boundary_cells` command, the tool places the boundary cells in their default orientation around the specified objects.

To flip the boundary cell orientations, use one or more of the following options with the `set_boundary_cell_rules` command:

`-mirror_left_boundary_cell`, `-mirror_right_boundary_cell`,  
`-mirror_left_outside_corner_cell`, `-mirror_right_outside_corner_cell`,  
`-mirror_left_inside_corner_cell`, `-mirror_right_inside_corner_cell`,  
`-mirror_left_inside_horizontal_abutment_cell`, and  
`-mirror_right_inside_horizontal_abutment_cell`. You cannot flip the orientation of the top and bottom boundary cells.

In addition, when you use the `compile_boundary_cells` command, you can modify the following aspects of the default placement behavior:

- Swapping of the top and bottom inside corner cells on flipped rows

Use the `-do_not_swap_top_and_bottom_inside_corner_cell` option with the `set_boundary_cell_rules` command to prevent the command from using the bottom inside corner cell on the top inside corner of flipped rows and the top inside corner cell on the bottom inside corner of flipped rows.

- Consideration of voltage areas

Use the `-at_va_boundary` option with the `set_boundary_cell_rules` command to insert horizontal boundary cells on both sides of the voltage area boundaries.

- Existence of one-unit-tile gaps

Use the `-no_1x` option with the `set_boundary_cell_rules` command to prevent boundary cell insertion from creating one-unit-tile gaps. When you use this option, the command does not insert boundary cells on a row when the row length equals two times the corner cell width plus one unit tile width. Note that if the row length equals two times the corner cell width, the command does insert boundary cells.

- Distance between tap cells

Use the `-tap_distance` option with the `set_boundary_cell_rules` command to specify the distance in microns between the tap cells inserted on the top and bottom boundary rows.

- Insertion of boundary cells on short rows

Use the `-min_row_width` option with the `set_boundary_cell_rules` command to prevent insertion of boundary cells on short rows. This option defines the width threshold for inserting boundary cells on a row. If the row width is less than the specified value, the command does not insert boundary cells on the row.

- Insertion of boundary cells in child blocks

Use the `-insert_into_blocks` option with the `set_boundary_cell_rules` command to recursively insert boundary cells in the child blocks. By default, the tool does not insert boundary cells in the child blocks.

## Specifying the Naming Convention for Boundary Cells

By default, the tool uses the following naming convention for inserted boundary cells:

`boundarycell!library_cell_name!number`

To change the separator character from its default of “!,” use the `-separator` option with the `set_boundary_cell_rules` command..

To identify the boundary cells inserted in a specific run, use the `-prefix` option with the `set_boundary_cell_rules` command to specify a prefix string. When you use this option, the tool uses the following naming convention:

`boundarycell!prefix!library_cell_name!number`

## Creating Routing Guides During Boundary Cell Insertion

If your technology file defines metal cut allowed and forbidden preferred grid extension rules, you can create routing guides for these rules during boundary cell insertion by using the `-add_metal_cut_allowed` option with the `set_boundary_cell_rules` command.

When you use this option, the `compile_boundary_cells` command creates the following routing guides:

- Metal cut allowed routing guides, which cover the area taken up by all the placeable site rows reduced by the vertical shrink factor, which is 50 percent of the smallest site row height
- Forbidden preferred grid extension routing guides, which cover the remaining area up to the block boundary

### Note:

This feature is supported only by the `compile_boundary_cells` command; it is not supported by the `compile_targeted_boundary_cells` command.

## Creating Placement Blockages During Boundary Cell Insertion

If your technology has minimum jog or minimum separation requirements, you can create placement blockages for these requirements during boundary cell insertion with the `compile_boundary_cells` command by using the following options with the `set_boundary_cell_rules` command:

- `-min_horizontal_jog`

If a horizontal edge of a macro, including its hard keepout margin, hard placement blockage, or voltage area, including its guard band, is less than the specified value, the `compile_boundary_cells -add_placement_blockage` command creates a placement blockage to prevent a violation.

- `-min_vertical_jog`

If a vertical edge of a macro, including its hard keepout margin, hard placement blockage, or voltage area, including its guard band, is less than the specified value, the `compile_boundary_cells -add_placement_blockage` command creates a placement blockage to prevent a violation.

- `-min_horizontal_separation`

If the continuous horizontal placement area is less than the specified value, the `compile_boundary_cells -add_placement_blockage` command creates a placement blockage to prevent a violation.

- `-min_vertical_separation`

If the continuous vertical placement area is less than the specified value, the `compile_boundary_cells -add_placement_blockage` command creates a placement blockage to prevent a violation.

### Note:

By default, the `compile_boundary_cells` command checks the specified rules, but does not create the placement blockages. To create the placement blockages, you must use the `-add_placement_blockage` option with the `compile_boundary_cells` command.

This feature is supported only by the `compile_boundary_cells` command; it is not supported by the `compile_targeted_boundary_cells` command.

## Reporting the Boundary Cell Insertion Requirements

To report the boundary cell insertion requirements specified by the `set_boundary_cell_rules` command, use the `report_boundary_cell_rules` command. This command reports only the user-specified settings; it does not report any default settings.

---

## Removing Boundary Cell Insertion Requirements

To remove one or more of the boundary cell insertion requirements specified by the `set_boundary_cell_rules` command, use the `remove_boundary_cell_rules` command.

---

## Inserting Boundary Cells

The Fusion Compiler tool provides several methods for inserting boundary cells, depending on where you want to insert the boundary cells.

- To insert boundary cells around the core area, hard macros, and placement blockages, use the `compile_boundary_cells` command.
- To insert boundary cells only around one or more voltage areas, use the `compile_boundary_cells` command with the `-voltage_area` option.
- To insert boundary cells only around specific objects, use the `compile_targeted_boundary_cells` command with the `-target_objects` option. You can use this command to insert boundary cells around specific macros, voltage areas, voltage area shapes, placement blockages, routing blockages, and the core area.

The `set_boundary_cell_rules` command configures boundary cell insertion for both the `compile_boundary_cells` and `compile_targeted_boundary_cells` commands. However, some rules are honored only by the `compile_boundary_cells` command. For details, see [Specifying the Boundary Cell Insertion Requirements](#).

---

## Verifying the Boundary Cell Placement

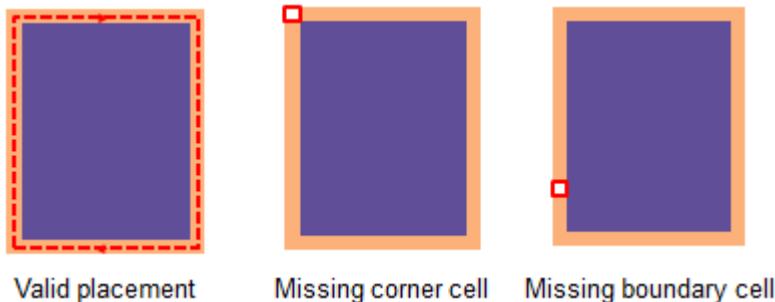
After inserting the boundary cells with the `compile_boundary_cells` command, verify the placement by using the `check_boundary_cells` command. To create an error data file to view the errors in the error browser, use the `-error_view` option. For information about using the error browser, see the *Fusion Compiler Graphical User Interface User Guide*.

This command checks the boundary cell placement for the following issues:

- Missing boundary or corner cells

This check verifies that there are boundary and corner cells around the entire boundary, with no gaps. [Figure 109](#) shows a valid placement, as well as errors caused by missing cells.

**Figure 109 Check for Missing Boundary or Corner Cells**



**Note:**

For established nodes, boundary cells are inserted only on the left and right sides. In this case, the command verifies only that there are no gaps on these sides.

- Incorrect boundary or corner cells

This check verifies that the library cells used for the boundary and corner cells match the cells specified by the `set_boundary_cell_rules` command.

- Incorrect orientation of boundary cells

This check verifies that the orientation of each boundary cell matches the allowed orientations specified by the `set_boundary_cell_rules` command.

- Short rows and edges

This check verifies that each row of boundary cells is wider than the value specified by the `set_boundary_cell_rules -min_row_width` option and that the horizontal edges of each blockage are greater than the value specified by the `set_boundary_cell_rules -min_horizontal_jog` option.

## Finding and Fixing Antenna Violations

In chip manufacturing, gate oxide can be easily damaged by electrostatic discharge. The static charge that is collected on wires during the multilevel metalization process can damage the device or lead to a total chip failure. The phenomenon of an electrostatic charge being discharged into the device is referred to as either antenna or charge-collecting antenna problems.

To prevent antenna problems, the tool verifies that for each input pin the metal antenna area divided by the gate area is less than the maximum antenna ratio given by the foundry:

$$(\text{antenna-area}) / (\text{gate-area}) < (\text{max-antenna-ratio})$$

This check is based on the global and layer-specific antenna rules defined for the block and the antenna properties of the cells in the block.

The antenna flow consists of the following steps:

1. [Define the antenna rules.](#)
  2. [Specify the antenna properties of the pins and ports.](#)
  3. [Analyze and fix the antenna violations.](#)
- 

## Defining Antenna Rules

Antenna rules define how to calculate the maximum antenna ratio for the nets in a block, as well as the antenna ratio for a pin. You must define a global antenna rule by using the `define_antenna_rule` command. You can also specify layer-specific antenna rules by using the `define_antenna_layer_rule` command. The tool uses the global antenna rule whenever a layer-specific antenna rule does not exist.

The following topics describe how to define the antenna rules:

- [Calculating the Maximum Antenna Ratio](#)
- [Calculating the Antenna Ratio for a Pin](#)

## Calculating the Maximum Antenna Ratio

To control the method used to calculate the maximum allowable antenna ratio, you must specify the following information:

- How much protection the diodes provide (the diode protection mode)
 

To specify the diode protection mode, use the `-diode_mode` option with the `define_antenna_rule` command, as described in [Setting the Diode Protection Mode](#). This is a required option of the `define_antenna_rule` command.
- How to perform antenna ratio calculation with diode protection (the diode ratio vector)
 

To specify the diode ratio vector, use the `-diode_ratio` option with the `define_antenna_layer_rule` command, as described in [Specifying the Diode Ratio Vector](#). This is a required option of the `define_antenna_layer_rule` command.

## Setting the Diode Protection Mode

The diode protection mode specifies how much protection the diodes provide.

Advanced technology designs often use cells and macros with different gate oxide thicknesses, which affect the antenna rule calculations. The Fusion Compiler tool uses gate classes to represent the various gate oxide thicknesses. The tool supports up to four gate oxide thicknesses: gate class 0, gate class 1, gate class 2, and gate class 3. If a cell pin does not have gate class data, it is treated as gate class 0.

The Fusion Compiler tool supports 18 diode protection modes, most of which support only a single gate oxide thickness. For advanced technology designs that use cells and macros with different gate oxide thicknesses, you must use one of the diode protection modes that support multiple gate oxide thicknesses: 14, 16, and 18.

To specify the diode protection mode, use the `-diode_mode` option with the `define_antenna_rule` command. This is a required option of the `define_antenna_rule` command.

[Table 33](#) defines how each of the diode protection mode settings affect the maximum antenna ratio for a net. The diode protection mode also affects the computation of the maximum antenna ratio for individual diodes. The antenna ratio calculation formulas for diodes are shown in [Table 35](#); these formulas use the values in the diode ratio vector specified in the `-diode_ratio` option of the `define_antenna_layer_rule` command, as described in [Specifying the Diode Ratio Vector](#).

*Table 33 Diode Protection Mode Settings*

Diode protection mode	Definition
0	The diodes do not provide any protection.
1	The diodes provide unlimited protection. In this mode, the highest metal layer is not checked for antenna violations because the input-to-output connection is completed when the highest metal layer is formed.
2	Diode protection is limited; if more than one diode is connected, the largest value of the maximum antenna ratio for all diodes is used.
3	Diode protection is limited; if more than one diode is connected, the sum of the maximum antenna ratios for all diodes is used.
4	Diode protection is limited; if more than one diode is connected, the sum of the diode-protection values for all diodes is used to compute the maximum antenna ratio.
5	Diode protection is limited; the maximum diode-protection value for all diodes is used to calculate the equivalent gate area.

**Table 33 Diode Protection Mode Settings (Continued)**

Diode protection mode	Definition
6	Diode protection is limited; the sum of the diode-protection values for all diodes is used to calculate the equivalent gate area.
7	Diode protection is limited; the maximum diode-protection value for all diodes is used to calculate the equivalent metal area.
8	Diode protection is limited; the sum of the diode-protection values for all diodes is used to calculate the equivalent metal area.
9	Diode protection is limited; scaling is based on maximum diode protection.
10	Diode protection is limited; scaling is based on total diode protection.
11	Diode protection is limited; scaling is based on maximum diode protection.
12	Diode protection is limited; scaling is based on total diode protection.
13	Obsolete.
14	Diode protection is limited; two separate calculation formulas. Supports gate classes.
15	Internal use only.
16	Diode protection is limited; if more than one diode is connected, the sum of the diode-protection values for all diodes is used to compute the maximum antenna ratio. Supports gate classes.
17	Diode protection is limited; the sum of the diode-protection values for all diodes is used to calculate the equivalent gate area. The gate area calculation is done separately for the input and output pins.
18	Diode protection is limited; if more than one diode is connected, the sum of the diode-protection values for all diodes is used to compute the maximum antenna ratio. Supports gate classes and calculation depends on whether the pin is connected to a diode.

### Specifying the Diode Ratio Vector

The diode ratio vector specifies the values used to calculate the maximum antenna ratio. To specify the diode ratio vector, use the `-diode_ratio` option with the `define_antenna_layer_rule` command. If you do not specify a diode ratio, the tool uses `{0 0 1 0 0}`. The actual usage of the diode ratio vector depends on the diode mode that was specified in the `define_antenna_rule -diode_mode` option.

**Table 34** shows the format of the diode ratio vector for the various diode modes. **Table 35** shows how the vector values are used to calculate the maximum antenna ratio for each diode mode. In this table,

- *dp* represents the diode protection value specified for an output pin. For information about specifying this value, see [Specifying Antenna Properties](#).
- *layerMaxRatio* represents the maximum antenna ratio for the layer, as specified by the `-ratio` option of the `define_antenna_layer_rule` command or one the `-metal_ratio` or `-cut_ratio` option of the `define_antenna_rule` command if the layer-specific value is not defined.

For examples of the maximum antenna ratio calculations, see

- [Example for Diode Modes 2, 3, and 4](#)
- [Example for Diode Modes 5 and 6](#)
- [Example for Diode Modes 7 and 8](#)
- [Example for Diode Mode 14 With Multiple Gate Oxide Thicknesses](#)

**Table 34** Diode Ratio Vector Formats

Diode modes	Format
2-10, 17	<code>{v0 v1 v2 v3 [v4]}</code>
	The <i>v4</i> value represents the upper limit of the diode protection and is optional. If you do not specify the <i>v4</i> value, it is assumed to be 0, which means there is no upper limit.
11, 12	<code>{ {v0 v1 v2 v3 [v4]} {s0 s1 s2 s4 s5} }</code>
	The second vector, <code>{s0 s1 s2 s3 s4 s5}</code> , specifies scaling values.
14	<code>{v0_1 v1_1 v2_1 v3_1 v4_1 v5_1 v6_1 v7_1 v8_1 v9_1}</code> <code>{v0_2 v1_2 v2_2 v3_2 v4_2 v5_2 v6_2 v7_2 v8_2 v9_2}</code> <code>{v0_3 v1_3 v2_3 v3_3 v4_3 v5_3 v6_3 v7_3 v8_3 v9_3}</code> <code>{v0_4 v1_4 v2_4 v3_4 v4_4 v5_4 v6_4 v7_4 v8_4 v9_4}</code>
	You specify one 10-value vector for each of the four gate classes.
16	<code>{v0_1 v1_1 v2_1 v3_1 v4_1 v5_1 v6_1 v7_1}</code> <code>{v0_2 v1_2 v2_2 v3_2 v4_2 v5_2 v6_2 v7_2}</code> <code>{v0_3 v1_3 v2_3 v3_3 v4_3 v5_3 v6_3 v7_3}</code> <code>{v0_4 v1_4 v2_4 v3_4 v4_4 v5_4 v6_4 v7_4}</code>
	You specify one 8-value vector for each of the four gate classes.

**Table 34 Diode Ratio Vector Formats (Continued)**

Diode modes	Format
18	<pre>{v0_1 v1_1 v2_1 v3_1 v4_1 v5_1 v6_1 v7_1 v8_1 v9_1 v10_1 v11_1} {v0_2 v1_2 v2_2 v3_2 v4_2 v5_2 v6_2 v7_2 v8_2 v9_2 v10_2 v11_2} {v0_3 v1_3 v2_3 v3_3 v4_3 v5_3 v6_3 v7_3 v8_3 v9_3 v10_3 v11_3} {v0_4 v1_4 v2_4 v3_4 v4_4 v5_4 v6_4 v7_4 v8_4 v9_4 v10_4 v11_4}</pre> <p>You specify one 12-value vector for each of the four gate classes.</p>

**Table 35 Antenna Ratio Calculation Based on Diode Mode**

Diode mode	Calculation
0, 1	Not used
2, 3, 4	<ul style="list-style-type: none"> <li>If <math>dp &gt; v0</math> and <math>v4 \neq 0</math>,  <math>\text{antenna\_ratio} = \min(((dp + v1) * v2 + v3), v4)</math></li> <li>If <math>dp &gt; v0</math> and <math>v4 = 0</math>,  <math>\text{antenna\_ratio} = (dp + v1) * v2 + v3</math></li> <li>If <math>dp \leq v0</math>,  <math>\text{antenna\_ratio} = \text{layerMaxRatio}</math></li> </ul>
5	$\text{antenna\_ratio} = \text{metal\_area} / (\text{gate\_area} + \text{equi\_gate\_area})$ <ul style="list-style-type: none"> <li>If <math>\text{max\_diode\_protection} &gt; v0</math> and <math>v4 \neq 0</math>,  <math>\text{equi\_gate\_area} = \min((\text{max\_diode\_protection} + v1) * v2 + v3), v4</math></li> <li>If <math>\text{max\_diode\_protection} &gt; v0</math> and <math>v4 = 0</math>,  <math>\text{equi\_gate\_area} = (\text{max\_diode\_protection} + v1) * v2 + v3</math></li> <li>If <math>\text{max\_diode\_protection} \leq v0</math>,  <math>\text{equi\_gate\_area} = 0</math></li> </ul>
6	$\text{antenna\_ratio} = \text{metal\_area} / (\text{gate\_area} + \text{equi\_gate\_area})$ <ul style="list-style-type: none"> <li>If <math>\text{total\_diode\_protection} &gt; v0</math> and <math>v4 \neq 0</math>,  <math>\text{equi\_gate\_area} = \min((\text{total\_diode\_protection} + v1) * v2 + v3), v4</math></li> <li>If <math>\text{total\_diode\_protection} &gt; v0</math> and <math>v4 = 0</math>,  <math>\text{equi\_gate\_area} = (\text{total\_diode\_protection} + v1) * v2 + v3</math></li> <li>If <math>\text{total\_diode\_protection} \leq v0</math>,  <math>\text{equi\_gate\_area} = 0</math></li> </ul>

*Table 35 Antenna Ratio Calculation Based on Diode Mode (Continued)*

Diode mode	Calculation
7	<pre>antenna_ratio = (metal_area - equi_metal_area) / gate_area • If max_diode_protection&gt;v0 and v4&lt;&gt;0,   equi_metal_area = min (((max_diode_protection + v1) * v2 + v3), v4) • If max_diode_protection&gt;v0 and v4=0,   equi_metal_area = (max_diode_protection + v1) * v2 + v3 • If max_diode_protection&lt;=v0,   equi_metal_area = 0 • If equi_metal_area&gt;metal_area,   equi_metal_area = metal_area</pre>
8	<pre>antenna_ratio = (metal_area - equi_metal_area) / gate_area • If total_diode_protection&gt;v0 and v4&lt;&gt;0,   equi_metal_area = min (((total_diode_protection + v1) * v2 + v3), v4) • If total_diode_protection&gt;v0 and v4=0,   equi_metal_area = (total_diode_protection + v1) * v2 + v3 • If total_diode_protection&lt;=v0,   equi_metal_area = 0 • If equi_metal_area &gt; metal_area,   equi_metal_area = metal_area</pre>
9	<pre>antenna_ratio = scale * metal_area / gate_area • If max_diode_protection&gt;v0,   scale = max (1 / ((max_diode_protection + v1) * v2 + v3), v4) • If max_diode_protection&lt;=v0,   scale = 1.0</pre>
10	<pre>antenna_ratio = scale * metal_area / gate_area • If total_diode_protection&gt;v0,   scale = max (1 / ((total_diode_protection + v1) * v2 + v3), v4) • If total_diode_protection&lt;= v0,   scale = 1.0</pre>

*Table 35 Antenna Ratio Calculation Based on Diode Mode (Continued)*

Diode mode	Calculation
11	<pre>antenna_ratio = scale * metal_area / gate_area • If max_diode_protection&lt;v0, scale = s0 • If max_diode_protection&lt;v1, scale = s1 • If max_diode_protection&lt;v2, scale = s2 • If max_diode_protection&lt;v3, scale = s3 • If max_diode_protection&lt;v4, scale = s4 • If max_diode_protection&gt;=v4, scale = s5</pre>
12	<pre>antenna_ratio = scale * metal_area / gate_area • If total_diode_protection&lt;v0, scale = s0 • If total_diode_protection&lt;v1, scale = s1 • If total_diode_protection&lt;v2, scale = s2 • If total_diode_protection&lt;v3, scale = s3 • If total_diode_protection&lt;v4, scale = s4 • If total_diode_protection&gt;=v4, scale = s5</pre>
13	N/A
14	<p>If connected to a diode,</p> <ul style="list-style-type: none"> <li>(metal or via area) / (gate-area) &lt;= (dp*v1 + v2) // formula1 and (metal or via area) / (v5*gate-area + v6* dp) &lt;= v7// formula2</li> </ul> <p>If not connected to a diode,</p> <ul style="list-style-type: none"> <li>(metal or via area) / (gate-area) &lt;= v3 // formula1 and (metal or via area) / (v8 * gate-area) &lt;= v9 // formula2</li> </ul>
15	N/A

Table 35 Antenna Ratio Calculation Based on Diode Mode (Continued)

Diode mode	Calculation
16	<ul style="list-style-type: none"> <li>If <math>dp &gt; v0</math> and <math>v4 &lt;&gt; 0</math>, allowable max-antenna-ratio = <math>\min(((dp + v1) * v2 + v3), v4)</math></li> <li>If <math>dp &gt; v0</math> and <math>v4 = 0</math>, allowable max-antenna-ratio = <math>(dp + v1) * v2 + v3</math></li> <li>If <math>dp \leq v0</math>, allowable max-antenna-ratio = <i>layerMaxRatio</i></li> <li>If (gate-area <math>\leq v5</math>), antenna-ratio = antenna-area/(gate-area + <math>v6 * dp</math>) else antenna-ratio = antenna-area/ (gate-area + <math>v6 * dp + v7</math>)</li> </ul>
17	<pre>antenna_ratio = metal_area / (gate_area + equi_gate_area) • If total_diode_protection &gt; v0 and v4 &lt;&gt; 0,   equi_gate_area = min (((total_diode_protection + v1) * v2 + v3), v4) • If total_diode_protection &gt; v0 and v4 = 0,   equi_gate_area = (total_diode_protection + v1) * v2 + v3 • If total_diode_protection &lt;= v0,   equi_gate_area = 0</pre>
18	<p>If connected to a diode,</p> <ul style="list-style-type: none"> <li>If <math>dp &gt; v0</math> and <math>v4 &lt;&gt; 0</math>, allowable max-antenna-ratio = <math>\min(((dp + v1) * v2 + v3), v4)</math></li> <li>If <math>dp &gt; v0</math> and <math>v4 = 0</math>, allowable max-antenna-ratio = <math>(dp + v1) * v2 + v3</math></li> <li>If <math>dp \leq v0</math>, allowable max-antenna-ratio = <i>layerMaxRatio</i></li> <li>If (gate-area <math>\leq v5</math>), antenna-ratio = antenna-area/(gate-area + <math>v6 * dp</math>) else antenna-ratio = antenna-area/ (gate-area + <math>v6 * dp + v7</math>)</li> </ul> <p>If not connected to a diode,</p> <ul style="list-style-type: none"> <li>If (gate-area <math>&lt; v8</math>), allowable max-antenna-ratio = <math>v9</math> else allowable max-antenna-ratio = <math>v10 * (\text{gate-area})^{v11}</math></li> <li>antenna-ratio = (antenna-area)/(gate-area)</li> </ul>

### Example for Diode Modes 2, 3, and 4

Assume that the antenna area is calculated using surface area in single-layer mode (antenna mode 1), the diode ratio vector for the M1 layer is {0.7 0.0 200 2000}, the *layerMaxRatio* value for the M1 layer is 400, and the following diodes are connected to a single M1 net: diode A with a diode-protection value of 0.5, diode B with a diode-protection

value of 1.0, and diode C with a diode-protection value of 1.5. In this example, the *v4* value is not specified so a value of 0 is used.

Use the following commands to define the antenna rules for this example:

```
fc_shell> define_antenna_rule -mode 1 -diode_mode diode_mode \
 -metal_ratio 400 -cut_ratio 20
fc_shell> define_antenna_layer_rule -mode 1 -layer "M1" \
 -ratio 400 -diode_ratio {0.7 0.0 200 2000}
```

The maximum antenna ratio for each diode is computed by using the formula for the diode mode from [Table 35](#). In this case,

- If  $dp > v0$  and  $v4 \neq 0$ ,

$$\text{allowable max-antenna-ratio} = \min((dp + v1) * v2 + v3), v4)$$

- If  $dp > v0$  and  $v4 = 0$ ,

$$\text{allowable max-antenna-ratio} = (dp + v1) * v2 + v3$$

- If  $dp \leq v0$ ,

$$\text{allowable max-antenna-ratio} = layerMaxRatio$$

[Table 36](#) shows the maximum antenna ratio for each diode calculated using this formula.

*Table 36 Calculation of Maximum Antenna Ratio for Each Diode*

Diode	Protection value	Maximum Antenna Ratio
A	0.5	400 Protection value is less than the <i>v0</i> value of 0.7, so use maximum antenna ratio of 400.
B	1.0	2200 Protection value is greater than the <i>v0</i> value of 0.7, so maximum antenna ratio is $(1.0+0.0) * 200 + 2000 = 2200$ .
C	1.5	2300 Protection value is greater than the <i>v0</i> value of 0.7, so maximum antenna ratio is $(1.5+0.0) * 200 + 2000 = 2300$ .

The maximum antenna ratio for the net is computed by using the formula for the diode mode from [Table 34](#):

- For diode mode 2, the maximum antenna ratio for the net is the largest of the maximum antenna ratio values for the diodes, 2300.
- For diode mode 3, the maximum antenna ratio for the net is the sum of the maximum antenna ratios for the diodes,  $400 + 2200 + 2300 = 4900$ .
- For diode mode 4, the maximum antenna ratio for the net is computed by using the formula from [Table 35](#) using the sum of the diode-protection values of the diodes,  $(0.5+1.0+1.5) * 200 + 2000 = 2600$ .

### Example for Diode Modes 5 and 6

Assume that the antenna area is calculated using surface area in single-layer mode (antenna mode 1), the diode ratio vector for the M1 layer is {0 0 1 0} (the default diode ratio vector), the *layerMaxRatio* value for the M1 layer is 400, and the following diodes are connected to a single M1 net: diode A with a diode-protection value of 0.5, diode B with a diode-protection value of 1.0, and diode C with a diode-protection value of 1.5. In this example, the *v4* value is not specified so a value of 0 is used.

Use the following command to define the antenna rules for this example:

```
fc_shell> define_antenna_rule -mode 1 -diode_mode diode_mode \
 -metal_ratio 400 -cut_ratio 20
```

Because this example uses the global *layerMaxRatio* value and the default diode ratio vector, you do not need to define a layer-specific antenna rule.

As shown in [Table 35](#), for diode modes 5 and 6, the maximum antenna ratio is computed as

*metal\_area / (gate\_area + equi\_gate\_area)*

where the equivalent gate area, *equi\_gate\_area*, is computed by using the diode ratio vector:  $(diode\_protection + v1) * (v2 + v3)$ .

- For diode mode 5, the equivalent gate area is computed using the maximum diode protection, which is 1.5, so the maximum antenna ratio for the net is

$$metal\_area / (gate\_area + ((1.5 + 0) * (1 + 0))) = metal\_area / (gate\_area + 1.5)$$

- For diode mode 6, the equivalent gate area is computed using the total diode protection, which is  $0.5+1.0+1.5=3.0$ , so the maximum antenna ratio for the net is

$$metal\_area / (gate\_area + ((3.0 + 0) * (1 + 0))) = metal\_area / (gate\_area + 3.0)$$

### Example for Diode Modes 7 and 8

Assume that the antenna area is calculated using surface area in single-layer mode (antenna mode 1), the diode ratio vector for the M1 layer is {0.7 0.0 150 800}, the *layerMaxRatio* value for the M1 layer is 400, and the following diodes are connected to a single M1 net: diode A with a diode-protection value of 0.5, diode B with a diode-protection value of 1.0, and diode C with a diode-protection value of 1.5. In this example, the *v4* value is not specified so a value of 0 is used.

Use the following commands to define the antenna rules for this example:

```
fc_shell> define_antenna_rule -mode 1 -diode_mode diode_mode \
 -metal_ratio 400 -cut_ratio 20
fc_shell> define_antenna_layer_rule -mode 1 -layer "M1" \
 -ratio 400 -diode_ratio {0.7 0.0 150 800}
```

As shown in [Table 35](#), for diode modes 7 and 8, the maximum antenna ratio is computed as

$$(metal\_area - equi\_gate\_area) / gate\_area$$

where the equivalent gate area, *equi\_gate\_area*, is computed by using the diode ratio vector: (*diode\_protection* + *v1*) \* (*v2* + *v3*).

- For diode mode 7, the equivalent metal area is computed using the maximum diode protection, which is 1.5, so the maximum antenna ratio for the net is

$$(metal\_area - ((1.5 + 0) * (150 + 800))) / gate\_area = (metal\_area - 1025) / gate\_area$$

- For diode mode 8, the equivalent metal area is computed using the total diode protection, which is 0.5+1.0+1.5=3.0, so the maximum antenna ratio for the net is

$$(metal\_area - ((3.0 + 0) * 150 + 800)) / gate\_area = (metal\_area - 1250) / gate\_area$$

### Example for Diode Mode 14 With Multiple Gate Oxide Thicknesses

Assume that the antenna area is calculated using surface area in single-layer mode (antenna mode 1), the design contains cells with two different gate oxide thicknesses, the diode ratio vector for the first oxide thickness (gate class 0) for the M1 layer is {1 0 1e9 1e9 0 1 2 285 1 285}, the diode ratio vector for the second oxide thickness (gate class 1) for the M1 layer is {1 0 1e9 1e9 1 0 2 165 1 28}, and the *layerMaxRatio* value for the M1 layer is 285.

For diode mode 14, the diode ratio vector contains 10 values for each gate class. Because the design uses only two gate oxide thicknesses, only the vectors for gate class 0 and gate class 1 are used. However, you must specify all 40 values when defining the antenna

rule. For unused gate classes (gate class 2 and gate class 3 in this example), specify very large ratios to disable the checks.

Use the following commands to define the antenna rules for this example:

```
fc_shell> define_antenna_rule -mode 1 -diode_mode 14 \
 -metal_ratio 285 -cut_ratio 10
fc_shell> define_antenna_layer_rule -mode 1 -layer "M1" \
 -ratio 285 -diode_ratio {1 0 1e9 1e9 0 1 2 285 1 285 \
 1 0 1e9 1e9 1 0 2 165 1 28 \
 1 0 1e9 1e9 0 1 2 1e9 1 1e9 \
 1 0 1e9 1e9 0 1 2 1e9 1 1e9 }
```

## Calculating the Antenna Ratio for a Pin

The tool supports several ways to calculate the antenna ratio (antenna-area/gate-area). You control this calculation by specifying the following information:

- How the antenna area is calculated (the antenna area mode)
- Which metal segments are considered for the calculation (the antenna recognition mode)

To specify these modes, use the `-mode` option with the `define_antenna_rule` and `define_antenna_layer_rule` commands, as described in [Setting the Antenna Mode](#). This is a required option for both of these commands.

### Setting the Antenna Mode

The antenna mode controls the antenna ratio calculation by determining

- How the antenna area is calculated (the antenna area mode)

The tool supports the following area calculation modes:

- Surface area, which is calculated as  $W \times L$
- Sidewall area, which is calculated as  $(W + L) \times 2 \times \text{thickness}$

#### Note:

If you use sidewall area calculation, you must define the metal thickness by specifying the `unitMinThickness`, `unitNomThickness`, and `unitMaxThickness` attributes in each `Layer` section of the technology file.

- Which metal segments are considered for the calculation (the antenna recognition mode)

To tool supports the following antenna recognition modes:

- Single-layer mode

In single-layer mode, the tool considers only the metal segments on the current layer; the metal segments on all lower layers are ignored. This mode allows the best routability. In this mode, the antenna ratio is calculated as

$$\text{antenna\_ratio} = \text{connected metal area of the layer / total gate area}$$

- Accumulated-ratio mode

In accumulated-ratio mode, the tool considers the metal segments on the current layer and the lower-layer segments connected to the input pins. In this mode, the antenna ratio is calculated as

$$\text{antenna\_ratio} = \text{accumulation of the single-layer mode ratios of the current layer and layers below}$$

- Accumulated-area mode

In accumulated-area mode, the tool considers the metal segments on the current and lower layers. In this mode, the antenna ratio is calculated as

$$\text{antenna\_ratio} = \text{connected metal area of the current and lower layers/ total gate area}$$

For a detailed example of the antenna recognition modes, see [Antenna Recognition Mode Example](#).

The Fusion Compiler tool supports six antenna modes, which are described in [Table 37](#). To specify the antenna mode, use the `-mode` option with the `define_antenna_rule` and `define_antenna_layer_rule` commands. This is a required option for both commands.

**Table 37     Antenna Mode Settings**

Antenna mode value	Area calculation mode	Antenna recognition mode
1	Surface	Single-layer
2	Surface	Accumulated-ratio
3	Surface	Accumulated-area
4	Sidewall	Single-layer
5	Sidewall	Accumulated-ratio

Table 37 Antenna Mode Settings (Continued)

Antenna mode value	Area calculation mode	Antenna recognition mode
6	Sidewall	Accumulated-area

### Antenna Recognition Mode Example

Figure 110 shows a layout example with a lateral view, which is used to explain the antenna recognition modes. Table 38 shows the antenna ratios for each antenna recognition mode for this layout example.

Figure 110 Layout Example for Antenna Recognition Modes

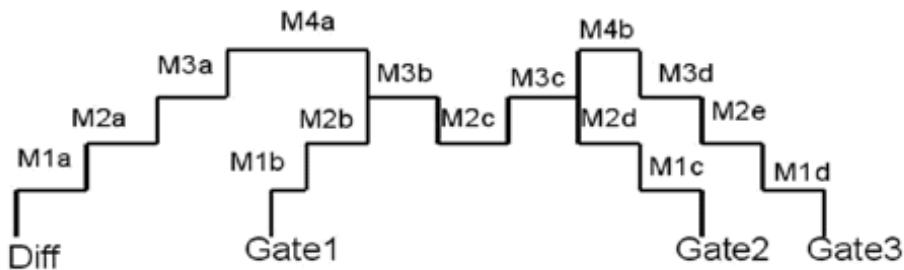


Table 38 Antenna Recognition Modes and Ratios

Considered segments	Antenna ratio
Single-layer mode	M1 ratios <ul style="list-style-type: none"> <li>Gate1: M1b / Gate1</li> <li>Gate2: M1c / Gate2</li> <li>Gate3: M1d / Gate3</li> </ul> M2 ratios <ul style="list-style-type: none"> <li>Gate1: M2b / Gate1</li> <li>Gate2: M2d / Gate2</li> <li>Gate3: M2e / Gate3</li> </ul> M3 ratios <ul style="list-style-type: none"> <li>Gate1, 2: <math>(M3b + M3c) / (Gate1 + Gate2)</math></li> <li>Gate3: M3d / Gate3</li> </ul> M4 ratios <ul style="list-style-type: none"> <li>Gate1, 2, 3: <math>(M4a + M4b) / (Gate1 + Gate2 + Gate3)</math></li> </ul>

*Table 38 Antenna Recognition Modes and Ratios (Continued)*

Considered segments	Antenna ratio
Accumulated-ratio mode	<p>M1 ratios</p> <ul style="list-style-type: none"> <li>• Gate1: M1b / Gate1</li> <li>• Gate2: M1c / Gate2</li> <li>• Gate3: M1d / Gate3</li> </ul> <p>M2 ratios</p> <ul style="list-style-type: none"> <li>• Gate1: M1b / Gate1 + M2b / Gate1</li> <li>• Gate2: M1c / Gate2 + M2d / Gate2</li> <li>• Gate3: M1d / Gate3 + M2e / Gate3</li> </ul> <p>M3 ratios</p> <ul style="list-style-type: none"> <li>• Gate1: M1b / Gate1 + M2b / Gate1 + (M3b + M3c) / (Gate1 + Gate2)</li> <li>• Gate2: M1c / Gate2 + M2d / Gate2 + (M3b + M3c) / (Gate1 + Gate2)</li> <li>• Gate3: M1d / Gate3 + M2e / Gate3 + M3d / Gate3</li> </ul> <p>M4 ratios</p> <ul style="list-style-type: none"> <li>• Gate1: M1b / Gate1 + M2b / Gate1 + (M3b + M3c) / (Gate1 + Gate2) + (M4a + M4b) / (Gate1 + Gate2 + Gate3)</li> <li>• Gate2: M1c / Gate2 + M2d / Gate2 + (M3b + M3c) / (Gate1 + Gate2) + (M4a + M4b) / (Gate1 + Gate2 + Gate3)</li> <li>• Gate3: M1d / Gate3 + M2e / Gate3 + M3d / Gate3 + (M4a + M4b) / (Gate1 + Gate2 + Gate3)</li> </ul>
Accumulated-area mode	<p>M1 ratios</p> <ul style="list-style-type: none"> <li>• Gate1: M1b / Gate1</li> <li>• Gate2: M1c / Gate2</li> <li>• Gate3: M1d / Gate3</li> </ul> <p>M2 ratios</p> <ul style="list-style-type: none"> <li>• Gate1: (M1b + M2b) / Gate1</li> <li>• Gate2: (M1c + M2d) / Gate2</li> <li>• Gate3: (M1d + M2e) / Gate3</li> </ul> <p>M3 ratios</p> <ul style="list-style-type: none"> <li>• Gate1, 2: (M1b + M1c + M2b + M2c + M2d + M3b + M3c) / (Gate1 + Gate2)</li> <li>• Gate3: (M1d + M2e + M3d) / Gate3</li> </ul> <p>M4 ratios</p> <ul style="list-style-type: none"> <li>• Gate1, 2, 3: all metal areas / (Gate1 + Gate2 + Gate3)</li> </ul>

## Specifying Antenna Properties

In general, the antenna properties for standard cells and hard macros are defined in their frame views in the reference libraries. You can set default values for the antenna

properties, which apply to cells that do not have antenna properties defined in the reference libraries.

- `route.detail.default_diode_protection`

Specifies the diode protection value used for standard cell output pins during antenna analysis if the diode protection value is not specified in the e view of the cell.

- `route.detail.default_gate_size`

Specifies the gate size used for standard cell input pins during antenna analysis if the gate size is not specified in the e view of the cell.

- `route.detail.default_port_external_antenna_area`

Specifies the antenna area used for ports (top-level pins) during antenna analysis if the antenna area is not specified in the e view of the cell.

- `route.detail.default_port_external_gate_size`

Specifies the gate size used for ports (top-level pins) during antenna analysis if the gate size is not specified in the e view of the cell.

- `route.detail.macro_pin_antenna_mode`

Specifies how macro cell pins are treated for antenna considerations.

- `route.detail.port_antenna_mode`

Specifies how the ports (top-level pins) are treated for antenna considerations.

If you are using a hierarchical flow and create a block abstraction for a block, you must use the `derive_hier_antenna_property` command to extract the antenna information from the block to use at the next level of hierarchy.

**Note:**

If the hierarchical antenna properties are not defined for all layers for a macro, Zroute treats the data as incomplete, skips antenna analysis, and issues a ZRT-311 warning message. If you get this error message, see [SolvNet article 027178, “Debugging the ZRT-311 Message.”](#)

**See Also**

- [Annotating Antenna Properties on Hard Macro Cells](#)

## Analyzing and Fixing Antenna Violations

If the design library contains antenna rules, Zroute automatically analyzes and fixes antenna violations. To disable the analysis and correction of antenna rules during detail routing, set the `route.detail.antenna` application option to `false`.

Just like other design rules, antenna rules are checked and corrected during detail routing. This concurrent antenna rule correction architecture reduces total runtime by minimizing the iterations.

By default, Zroute

- Checks antenna rules and corrects violations for all clock and signal nets

To disable fixing of antenna violations on specific nets, set the `route.detail.skip_antenna_fixing_for_nets` application option. Note that Zroute analyzes the antenna rules and reports the antenna violations on these nets, but does not fix the violations.

- Does not check or correct antenna rules for power and ground nets

To check and correct antenna rules for power and ground nets, set the `route.detail.check_antenna_on_pg` application option to `true`.

- Starts fixing antenna violations in the second iteration, after initial routing is complete and the basic DRC violations have been fixed

To change the iteration in which Zroute starts fixing antenna violations, set the `route.detail.antenna_on_iteration` application option.

- Performs layer hopping to fix antenna violations

Layer hopping decreases the antenna ratio by splitting a large metal polygon into several upper-level polygons. Zroute performs the following types of layer hopping:

- Breaking the antenna with a higher-level metal segment

Zroute uses this technique to fix most antenna violations. For antenna violations that happen at metal-N, inserting a small segment of metal-(N+1) close to the gate reduces the ratio between the remaining metal-N, making the ratio much lower. This approach is not suitable for fixing top-metal layer antenna violations when the output pin can provide only limited protection because there is no way for the router to break antenna violations at the topmost metal layer.

- Moving down to a lower-level metal

Zroute uses this technique to fix only topmost layer antenna violations when output pins provide only limited protection. For antenna violations that happen at metal-N,

replace part of the metal-N with metal-(N-1 or lower) to reduce the ratio. However, splitting the metal layer into many pieces might have a negative impact on RC and timing delay.

Zroute can also insert diodes to fix antenna violations. To enable the insertion of diodes to fix antenna violations, set the `route.detail.insert_diodes_during_routing` application option to `true`. To force Zroute to fix antenna violations by inserting diodes, disable layer hopping by setting the `route.detail.hop_layers_to_fix_antenna` application option to `false`. For information about inserting diodes to fix antenna violations, see [Inserting Diodes During Detail Routing](#).

If both layer hopping and diode insertion are enabled, by default, Zroute first tries to use layer hopping to fix the antenna violation. To change the preference to diode insertion, set the `route.detail.antenna_fixing_preference` application option to `use_diodes`.

As with other design rule violations, antenna violations are reported at the end of each detail routing iteration. For example,

```
DRC-SUMMARY:
@0000000 TOTAL VIOLATIONS = 506
@0000 Total number of instance ports with antenna violations = 1107
```

To check for antenna violations, use the `check_routes -antenna true` command.

## Inserting Diodes During Detail Routing

One way to protect gates from antenna effects is to provide a discharge path for the accumulated charge to leave the net. However, the discharge path should not allow current to flow during normal chip operation. Discharging can be accomplished by inserting a reverse-biased diode on the net close to the gate that is being protected.

To enable diode insertion during detail routing, set the `route.detail.insert_diodes_during_routing` application option to `true`.

To control diode insertion, set the following application options:

- To specify a preference for fixing antenna violations by using diode insertion rather than layer hopping, set the `route.detail.antenna_fixing_preference` application option to `use_diodes`.
- To require fixing of antenna violations by using diode insertion, disable layer hopping by setting the `route.detail.hop_layers_to_fix_antenna` application option to `false`.
- By default, when you enable diode insertion, Zroute can fix an antenna violation either by adding a new diode or by using an existing spare diode. Zroute determines which

method to use, based on which is closest to the required location: an empty location for a new diode or an existing spare diode.

- To specify a preference for using a new diode or using an existing spare diode, set the `route.detail.diode_preference` application option to `new` or `spare`, respectively. To reset the diode preference to the default behavior, set the `route.detail.diode_preference` application option to `none`.
- If you want Zroute to use only one of these methods, set the `route.detail.diode_insertion_mode` application option to `new` to force the insertion of new diodes or to `spare` to force the use of existing spare diodes.

To reset the diode insertion method to the default behavior, set the `route.detail.diode_insertion_mode` application option to `new_and_spare`.

**Note:**

To take advantage of spare diodes for antenna violation fixing, you need to add the spare diodes either before or after standard-cell placement and before routing the areas where antenna violations might occur.

- When inserting new diodes, Zroute selects the diodes from the reference libraries and inserts them into existing open spaces. To control which diodes are used, set the `route.detail.diode_libcell_names` application option.

When you specify the diode library cells, use only the cell names. If you include the library name, the tool does not recognize the diode cells.

- By default, Zroute reuses existing filler cell locations for diode insertion.

To prevent Zroute from reusing these locations, set the `route.detail.reuse_filler_locations_for_diodes` application option to `false`.

Zroute considers voltage areas when inserting diode cells and also observes the logic hierarchy assignments for diode cells.

- If a pin has an antenna violation, the diode cells are inserted at the same level of logic hierarchy as the violating pin.
- If a top-level port has an antenna violation, by default, the diode cells are inserted at the top level. However, if the port belongs to a voltage area, you can insert the diode cells in the logic hierarchy associated with the voltage area by setting the `route.detail.use_lower_hierarchy_for_port_diodes` application option to `true`.

## Inserting Diodes After Detail Routing

You can fix antenna violations after detail routing by explicitly specifying which violations to fix and providing constraints for fixing them. Based on the specified constraints and the

setting of the `route.detail.diode_insertion_mode` application option, Zroute either inserts new diodes or reuses existing spare diodes to fix the specified violations.

To insert diodes after detail routing, use the `create_diodes` command. When you use this command, you must use the `-options` option to specify the location of each antenna violation to fix by specifying the port and cell instance, the reference cell for the diode, the number of diodes to insert, the highest allowed routing layer used for connecting the diode, and the maximum distance from the specified pin that the diode can be inserted. If a diode cannot be inserted or reused within the specified distance, the tool does not insert a diode for that violation.

Use the following format to specify these values:

```
{port_name instance_name diode_reference number_of_diodes
max_routing_layer max_routing_distance}
```

**Note:**

You can use this command to insert diodes for top-level ports by specifying the name of the top-level block for the cell instance.

The `create_diodes` command uses the following application options to control diode insertion:

- To control whether Zroute inserts new diodes or reuses spare diodes, set the `route.detail.diode_insertion_mode` application option.
- To control whether Zroute can reuse filler cell locations, set the `route.detail.reuse_filler_locations_for_diodes` application option.
- To specify the logic hierarchy in which to insert the diodes for top-level ports, set the `route.detail.use_lower_hierarchy_for_port_diodes` application option.

For example, to insert 2 Adiode diode cells to fix an antenna violation on the A port of the CI cell instance, where the diodes must be inserted using no higher layer than M5 and within 2.5 microns of the port, enter the following command:

```
fc_shell> create_diodes {{A CI Adiode 2 M5 2.5}}
```

## Inserting Redundant Vias

Redundant via insertion is an important design-for-manufacturing (DFM) feature that is supported by Zroute throughout the routing flow. In each routing stage, Zroute concurrently optimizes via count as well as wire length. The redundant via result is measured by the redundant via conversion rate, which is defined as the percentage of single vias converted into redundant vias. You should also pay attention to the number of unoptimized

single vias. If a block has fewer unoptimized single vias, it is usually better for DFM. The following topics describe how to insert redundant vias:

- [Inserting Redundant Vias on Clock Nets](#)
- [Inserting Redundant Vias on Signal Nets](#)

---

## Inserting Redundant Vias on Clock Nets

Zroute can insert redundant vias on clock nets either during or after routing. This topic describes how to insert redundant vias during clock routing. For information about postroute redundant via insertion, see [Postroute Redundant Via Insertion](#).

To insert redundant vias on clock nets during clock routing,

1. Specify the redundant vias in a nondefault routing rule by using the `create_routing_rule` command, as described in [Specifying Nondefault Vias](#).

Be sure to consider both DFM and routing when you select the redundant vias; otherwise, if you select the redundant vias based only on DFM considerations, you could negatively impact the routability.

In addition to using nondefault routing rules to define the redundant vias for clock nets, you can also use them to define stricter wire width and spacing rules and to define the tapering distance. For more information about nondefault routing rules, see [Using Nondefault Routing Rules](#).

2. Assign the nondefault routing rule to the clock nets by using the `set_clock_routing_rules` command.

```
fc_shell> set_clock_routing_rules -rules clock_via_rule
```

3. Run clock tree synthesis.

```
fc_shell> synthesize_clock_trees
```

4. Route the clock nets.

```
fc_shell> route_group -all_clock_nets \
 -reuse_existing_global_route true
```

Zroute reserves space for the redundant vias during global routing and inserts the redundant vias during detail routing.

---

## Inserting Redundant Vias on Signal Nets

You can perform redundant via insertion in the following ways:

- Postroute redundant via insertion
- Concurrent soft-rule-based redundant via insertion
- Near 100 percent redundant via insertion

In general, you should start with postroute redundant via insertion. If postroute redundant via insertion results in a redundant via rate of at least 80 percent, you can try to improve the redundant via rate by using concurrent soft-rule-based redundant via insertion. If postroute redundant via insertion results in a redundant via rate of at least 90 percent, you can try to improve the redundant via rate by using near 100 percent redundant via insertion.

**Note:**

As the redundant via rate increases, it becomes more difficult to converge on the routing design rules and you might see a reduction in signal integrity; therefore, you should use near 100 percent redundant via insertion only for those blocks that truly require such a high redundant via rate. In addition, achieving very high redundant via rates might require you to modify the floorplan utilization to allow enough space for the redundant vias.

The following topics describe the default via mapping table, how to define a customized via mapping table, how to insert redundant vias by using various methods, and how to report the redundant via rate.

- [Viewing the Default Via Mapping Table](#)
- [Defining a Customized Via Mapping Table](#)
- [Postroute Redundant Via Insertion](#)
- [Concurrent Soft-Rule-Based Redundant Via Insertion](#)
- [Near 100 Percent Redundant Via Insertion](#)
- [Preserving Timing During Redundant Via Insertion](#)
- [Reporting Redundant Via Rates](#)

## Viewing the Default Via Mapping Table

By default, Zroute reads the default contact codes from the technology file and generates an optimized via mapping table. To see the default mapping table, use the `add_redundant_vias -list_only true` command.

In most cases you achieve better results if you use a customized mapping table rather than the default mapping table. For information about defining a customized mapping table, see [Defining a Customized Via Mapping Table](#).

If you have not previously defined a customized mapping table for the block, you can see the default mapping table by using the `add_redundant_vias -list_only true` command.

**Note:**

After you have created a customized mapping table by using the method described in [Defining a Customized Via Mapping Table](#), this command shows the customized mapping table.

[Example 26](#) shows an example of a default via mapping table.

*Example 26 Default Via Mapping Table*

```
fc_shell> add_redundant_vias -list_only
...
Redundant via optimization will attempt to replace the following vias:
 VIA12SQ_C -> VIA12SQ_C_2x1 VIA12SQ_C_2x1(r) VIA12SQ_C_1x2 VIA12SQ_
C_1x2(r)
 VIA12SQ_C(r) -> VIA12SQ_C_2x1 VIA12SQ_C_2x1(r) VIA12SQ_C_1x2 VIA12SQ_
C_1x2(r)
...
 VIA89(r) -> VIA89_C_1x2(r) VIA89_1x2(r) VIA89_1x2 VIA89_C_1x2
 VIA89_C_2x1(r) -> VIA89_C_2x1(r) VIA89_2x1(r) VIA89_2x1 VIA89_C_2x1
 VIA9RDL -> VIA9RDL_2x1 VIA9RDL_1x2
```

## Defining a Customized Via Mapping Table

To define a customized via mapping, use the `add_via_mapping` command. At a minimum, you must specify the source via and its replacement vias by using the `-from` and `-to` options. The vias listed in these options must be either vias defined in the technology file or design-specific vias created by the `create_via_def` command. The vias listed in the `-from` option must be simple vias or via arrays. The vias listed in the `-to` option can be simple vias, simple via arrays, or custom vias. For information about creating design-specific vias, see [Defining Vias](#).

Use the following options to refine the via mapping:

- `-weight`

By default, all mappings have the same priority, and Zroute selects the redundant vias to use based on routability. To set the priority for a mapping, use the `-weight` option to assign an integer weight value between 1 and 10 to the mapping. If you do not assign a weight, the tool assigns a weight of 1. During redundant via insertion, Zroute uses the higher weighted redundant vias first.

- `-transform`

By default, Zroute can rotate or flip the via arrays during redundant via insertion (the `-transform all` option).

- To allow only rotation of the via arrays during redundant via insertion, use the `-transform rotate` option.
- To allow only flipping of the via arrays during redundant via insertion, use the `-transform flip` option.
- To allow only the specified orientation of the via array during redundant via insertion, use the `-transform none` option.

The tool saves the mappings defined by the `add_via_mapping` command in a via mapping table in the design library. By default, if you try to add a via mapping that already exists in the table, the command fails. To overwrite an existing mapping definition, use the `-force` option.

[Example 27](#) shows an example of using the `add_via_mapping` command to define a customized via mapping table.

### Example 27 Customized Via Mapping Table

```
fc_shell> add_via_mapping \
 -from {VIA12 1x1} -to {VIA12T 2x1} -weight 5
fc_shell> add_via_mapping \
 -from {VIA12 1x1} -to {VIA12 2x1} -weight 1
```

To see the customized via mappings associated with a block, use the `report_via_mapping` command.

- By default, this command shows all user-defined via mappings.
- To show the via mappings for specific source vias, use the `-from` option.
- To show the via mappings for specific replacement vias, use the `-to` option.

To remove mappings from the via mapping table, use the `remove_via_mappings` command.

- To remove all via mappings, use the `-all` option.
- To remove the via mappings for specific source vias, use the `-from` option.
- To remove the via mappings for specific replacement vias, use the `-to` option.

### **Using a Subset of the Via Mapping Table for Redundant Via Insertion**

To use a subset of the via mapping table for redundant via insertion, specify which weight groups from the via mapping table to use by setting one or both of the `route.common.redundant_via_include_weight_group_by_layer_name` and `route.common.redundant_via_exclude_weight_group_by_layer_name` application options.

- If you set only the `route.common.redundant_via_include_weight_group_by_layer_name` application option, Zroute selects redundant vias only from the specified weight groups. If you do not specify an entry for a layer, redundant via insertion is not performed on that layer.
- If you set only the `route.common.redundant_via_exclude_weight_group_by_layer_name` application option, Zroute selects redundant vias from all weight groups in the original via mapping table, except the specified weight groups. If you do not specify an entry for a layer, all weight groups are used for that layer.
- If you set both application options, Zroute selects redundant vias from the weight groups specified in the `route.common.redundant_via_include_weight_group_by_layer_name` application, excluding the weight groups specified in the `route.common.redundant_via_exclude_weight_group_by_layer_name` application option.

### **Postroute Redundant Via Insertion**

To perform postroute redundant via insertion, use the `add_redundant_vias` command.

This command can replace single-cut vias with multiple-cut via arrays, single-cut vias with other single-cut vias that have a different contact code, and multiple-cut via arrays

with different multiple-cut via arrays. During redundant via insertion, the detail router also checks the design rules within the neighboring partition to minimize DRC violations.

By default, the `add_redundant_vias` command inserts redundant vias on all nets. To insert redundant vias only on specific nets, use the `-nets` option to specify the nets. Using net-specific redundant via insertion allows you to further improve the optimized via rate without causing large-scale routing and timing changes.

After the vias are checked and replaced, the detail router rechecks for DRC violations and corrects any violations.

If the percentage of redundant vias is not high enough, you can increase the effort level by using the `-effort` option to get a better redundant via rate. Increasing the effort level to high can increase the redundant via rate by about 3 to 5 percent by shifting the vias to make room for additional vias. However, because high-effort redundant via insertion moves the vias more, it can result in a less lithography-friendly pattern at the 45-nm technology node and below. In this case, you should use concurrent soft-rule-based redundant via insertion to improve the redundant via rate.

You can also try to increase the postroute redundant via rate by setting the `route.detail.optimize_wire_via_effort_level` application option to `high`, which reduces the number of single vias and makes more room for redundant vias by reducing wire length.

After you perform the initial postroute redundant via insertion, set the `route.common.post_detail_route_redundant_via_insertion` application option to enable automatic insertion of redundant vias after subsequent detail routing or ECO routing. This helps to maintain the redundant via rate in your block.

## Concurrent Soft-Rule-Based Redundant Via Insertion

Soft-rule-based redundant via insertion can improve the redundant via rate by reserving space for the redundant vias during routing. You can use concurrent soft-rule-based redundant via insertion during both initial routing and ECO routing. The actual via insertion is not done during routing; you must still perform postroute redundant via insertion by using the `add_redundant_vias` command.

### Note:

Reserving space during routing increases the routing runtime. You should use this method only when needed to improve the redundant via rate beyond that provided by postroute redundant via insertion and the postroute approach resulted in a redundant via rate of at least 80 percent.

To perform concurrent soft-rule-based redundant via insertion,

1. (Optional) Define the via mapping table as described in [Defining a Customized Via Mapping Table](#).
2. Enable concurrent soft-rule-based redundant via insertion.

By default, concurrent redundant via insertion is disabled.

- To enable concurrent soft-rule-based redundant via insertion during initial routing, set the `route.common.concurrent_redundant_via_mode` application option to `reserve_space`.

```
fc_shell> set_app_options \
 -name route.common.concurrent_redundant_via_mode \
 -value reserve_space
```

To control the effort used to reserve space for the redundant vias during initial routing, set the `route.common.concurrent_redundant_via_effort_level` application option. By default, Zroute uses low effort. The higher effort levels result in a better redundant via conversion rate at the expense of runtime. The low and medium efforts affect only global routing and track assignment, while high effort also affects detail routing, which can impact design rule convergence.

**Note:**

If you enable the `route.common.concurrent_redundant_via_mode` option before running the `place_opt` command, the redundant vias are considered during congestion estimation.

- To enable concurrent soft-rule-based redundant via insertion during ECO routing, set the `route.common.eco_route_concurrent_redundant_via_mode` application option to `reserve_space`.

```
fc_shell> set_app_options \
 -name route.common.eco_route_concurrent_redundant_via_mode \
 -value reserve_space
```

To control the effort used to reserve space for the redundant vias during ECO routing, set the `route.common.eco_route_concurrent_redundant_via_effort_level` application option.

**Note:**

Using concurrent soft-rule-based redundant via insertion during ECO routing can impact timing and design rule convergence. In general, you should use this method only when you used near 100 percent redundant via insertion during initial routing.

3. Route the block.

During routing, Zroute reserves space for the redundant vias and fixes hard design rule violations.

4. Perform postroute redundant via insertion.

During postroute redundant via insertion, Zroute inserts the redundant vias in the reserved locations.

## Near 100 Percent Redundant Via Insertion

You can achieve a redundant via rate near 100 percent by using hard-rule-based redundant via insertion. Hard-rule-based redundant via insertion can improve the redundant via rate by treating redundant vias as hard design rules during routing. You can use nearly 100 percent redundant via insertion only during initial routing; this method is not supported during ECO routing. When you use near 100 percent redundant via insertion during initial routing, you should use soft-rule-based redundant via insertion during ECO routing to preserve the redundant via rate achieved during initial routing.

**Note:**

This method can result in a very large runtime increase for congested blocks. You should use this method only when needed to improve the redundant via rate beyond that provided by concurrent soft-rule-based redundant via insertion and the soft-rule-based approach resulted in a redundant via rate of at least 90 percent.

To perform concurrent hard-rule-based redundant via insertion,

1. Enable nearly 100 percent via insertion by setting the `route.common.concurrent_redundant_via_mode` application option to `insert_at_high_cost`. (By default, concurrent redundant via insertion is disabled.)

```
fc_shell> set_app_options \
 -name route.common.concurrent_redundant_via_mode \
 -value insert_at_high_cost
```

To control the effort used to reserve space for the redundant vias, set the `route.common.concurrent_redundant_via_effort_level` application option.

**Note:**

If you enable the `route.common.concurrent_redundant_via_mode` option before running the `place_opt` command, the redundant vias are considered during congestion estimation.

## 2. Route the block.

During routing, Zroute inserts the redundant vias and fixes hard design rule violations. In general, redundant via insertion has the same priority as other hard design rules; however, if design rule checking does not converge during detail routing, Zroute automatically relaxes the redundant via constraints to improve DRC convergence.

## Preserving Timing During Redundant Via Insertion

When you insert redundant vias, it changes the timing of your block. Short nets tend to slow down due to increased capacitance, whereas long nets tend to speed up due to decreased resistance. Zroute redundant via insertion has a timing-preservation mode that allows you to perform redundant via insertion without affecting the block timing by preventing insertion of redundant vias on critical nets.

To enable timing-preservation mode for redundant via insertion, define the timing preservation constraints by using the following options of the `add_redundant_vias` command:

- `-timing_preserve_setup_slack_threshold`
- `-timing_preserve_hold_slack_threshold`
- `-timing_preserve_nets`

You should timing-preservation mode only at the end of the flow, after using the normal redundant via insertion flows, which converge both the redundant via rate and the timing QoR. Timing-preservation mode can slightly increase the redundant via rate while maintaining timing. However, if you use timing-preservation mode earlier in the flow, before timing is met, it might severely reduce the redundant via rate due to critical nets.

## Reporting Redundant Via Rates

After redundant via insertion, whether concurrent or postroute, Zroute generates a redundant via report that provides the following information:

- The via conversion rate for nondefault vias

The via conversion rate for nondefault vias is listed at the top of the report as the total optimized via conversion rate.

- The optimized via conversion rate for each layer

The optimized via conversion rate includes both double vias and DFM-friendly bar vias, which have a single cut but a larger metal enclosure. The tool reports two values for the via conversion rate:

- The total optimized via conversion rate

This value is computed based on the total via count, which includes both fixed vias and routed vias. Fixed vias are vias that cannot be optimized by the router, such as unrouted vias and user-defined vias.

- The optimized via conversion rate based on the total routed via count

This value is computed based only on the routed via count, which includes only those vias that can be optimized by the router.

**Note:**

The optimized via conversion rate is not useful if you are using bar vias.

- The distribution of optimized vias by weight for each layer

To determine the via conversion rate for conversions above a certain weight, you must add the reported conversion rates for those weights. For example, in [Example 28](#), the via conversion rate for weight 5 and above for layer V03 is  $10.75+64.50=75.25\%$ .

**Note:**

The conversion rate for unweighted vias is reported as “Un-optimized.”

- The total double via conversion rate for the block

**Note:**

The redundant via rate reported by the `report_design` command differs from the redundant via rate reported by Zroute during redundant via insertion or the `check_routes` command. This difference occurs because the `report_design` command reports the double via rate for both PG vias and signal vias, while Zroute reports the double via rate only for signal vias. In addition, Zroute bases the conversion rate only on the redundant via mapping.

[Example 28](#) shows an example of the redundant via report.

**Example 28 Redundant Via Report**

```
Total optimized via conversion rate = 96.94% (1401030 / 1445268 vias)
Layer V01 = 41.89% (490617 / 1171301 vias)
Weight 10 = 9.64% (112869 vias)
Weight 5 = 32.25% (377689 vias)
Weight 1 = 0.01% (59 vias)
Un-optimized = 58.11% (680684 vias)
```

```

Layer V02 = 76.20% (1567822 / 2057614 vias)
 Weight 10 = 43.51% (895270 vias)
 Weight 5 = 28.62% (588805 vias)
 Weight 1 = 4.07% (83747 vias)
 Un-optimized = 23.80% (489792 vias)
Layer V03 = 81.87% (687115 / 839297 vias)
 Weight 10 = 64.50% (541369 vias)
 Weight 5 = 10.75% (90224 vias)
 Weight 1 = 6.62% (55522 vias)
 Un-optimized = 18.13% (152182 vias)
Layer V04 = 81.60% (226833 / 277977 vias)
 Weight 10 = 81.45% (226418 vias)
 Weight 1 = 0.15% (415 vias)
 Un-optimized = 18.40% (51144 vias)
...
Layer V09 = 85.47% (1329 / 1555 vias)
 Weight 10 = 85.47% (1329 vias)
 Un-optimized = 14.53% (226 vias)

Total double via conversion rate = 46.69% (2158006 / 4622189 vias)

```

## Optimizing Wire Length and Via Count

During detail routing, Zroute optimizes wire length and via count in the areas where DRC violations occur; however, it does not optimize the layout in areas where no DRC violations occur.

To improve the manufacturing yield, use the `optimize_routes` command to perform standalone optimization of wire length and via count after performing detail routing and redundant via insertion.

By default, Zroute selects the nets to reroute based on the overall cost. For each selected net, Zroute determines whether to reroute all the shapes in the net or just a portion of them. To select the nets to reroute, use the `-nets` option to specify the nets. When you specify the nets to optimize, you can also use the `-reroute_all_shapes_in_nets` option to control whether Zroute must reroute all the associated net shapes.

By default, Zroute performs a maximum of 40 detail routing iterations to fix DRC violations that exist after the optimization. You can use the `-max_detail_route_iterations` option to control the maximum number of detail routing iterations.

## Reducing Critical Areas

A critical area is a region of the block where, if the center of a random particle defect falls there, the defect causes circuit failure, thereby reducing yield. A conductive defect causes a short fault, and a nonconductive defect causes an open fault.

The following topics describe how to

- Reduce critical area short faults by performing wire spreading
- Reduce critical area open faults by performing wire widening

## Performing Wire Spreading

After you have performed detail routing and redundant via insertion, you can perform wire spreading to increase the average spacing between wires, which reduces the critical area short faults and therefore improves yield.

To perform wire spreading, use the `spread_wires` command. By default, the `spread_wires` command uses the following settings to spread the signal wires on the same layer:

To perform wire spreading, use the `spread_wires` command. By default, the `spread_wires` command uses the following settings to spread the signal wires on the same layer:

- Spreads the wires by half a pitch in the preferred direction

To modify the spread distance, use the `-pitch` option. You specify the spread distance as a multiplier for the layer pitch. For example, to specify a spread distance of 1.5 times the layer pitch, use the following command:

```
fc_shell> spread_wires -pitch 1.5
```

- Uses twice the layer pitch as the minimum jog length

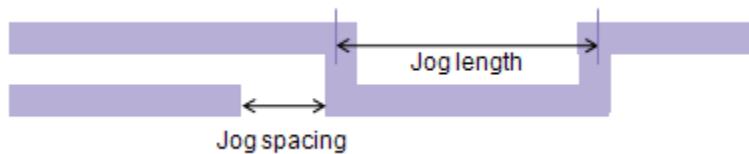
To modify the minimum jog length, use the `-min_jog_length` option. You specify the minimum jog length as an integer multiple of the layer pitch.

- Uses the minimum layer spacing plus one half the layer pitch as the minimum jog length

To modify the minimum jog spacing, use the `-min_jog_spacing_by_layer_name` option. You specify the minimum jog spacing in microns for each layer. The tool uses the default jog spacing for any unspecified layers.

[Figure 111](#) shows how the jog length and jog spacing values are used in wire spreading.

**Figure 111** *Wire Spreading Results*



In the following example, the minimum jog length is set to three times the layer pitch, the minimum jog spacing for the M1 layer is set to 0.07 microns, and the minimum jog spacing for the M2 layer is 0.08 microns. All other metal layers use the default minimum jog spacing.

```
fc_shell> spread_wires -min_jog_length 3 \
 -min_jog_spacing_by_layer_name {{M1 0.07} {M2 0.08}}
```

After spreading, the `spread_wires` command performs detail routing iterations to fix any DRC violations caused as a result of spreading.

When you change the layout, it can change the timing of your block. Wire spreading has a timing-preservation mode that allows you to perform wire spreading without affecting the block timing.

To enable timing-preservation mode for wire spreading, define the timing preservation constraints by using the following options with the `spread_wires` command:

- `-timing_preserve_setup_slack_threshold threshold`
- `-timing_preserve_hold_slack_threshold threshold`
- `-timing_preserve_nets nets`

The threshold values are floating-point numbers in library units. Wire spreading is performed only on nets with slack greater than or equal to the specified values or the nets specified in the `-timing_preserve_nets` option, as well as adjacent nets on the same layer within two routing pitches.

## Performing Wire Widening

After you have performed detail routing, redundant via insertion, and wire spreading, you can perform wire widening to increase the average width of the wires, which reduces the critical area open faults and therefore improves yield.

To perform wire widening, use the `widen_wires` command. By default, the `widen_wires` command widens all wires in the block to 1.5 times their original width. For more flexibility, you can use the `-widen_widths_by_layer_name` option to define up to five possible wire widths to use for each layer. For example, to define possible wire widths of 0.07 and 0.06 microns for the M1 layer; wire widths of 0.08 and 0.07 microns for the M2 layer; and 1.5 times the existing wire width for all other layers, enter the following command:

```
fc_shell> widen_wires \
 -widen_widths_by_layer_name {{M1 0.07 0.06} {M2 0.08 0.07}}
```

When you perform wire widening, the spacing between neighboring wires is decreased, which can reduce the improvement in critical area shorts gained from wire spreading. You can control the tradeoff between wire spreading and wire widening by using the

`-spreading_widening_relative_weight` option. By default, wire spreading and wire widening are given equal priority. To weight the priority toward wire widening and reduced critical area open faults, set this option to a value between 0.0 and 0.5. To weight the priority toward wire spreading and reduced critical area short faults, set this option to a value between 0.5 and 1.0.

After widening, the `widen_wires` command performs detail routing iterations to fix any DRC violations caused as a result of widening. Note that the widened wires do not trigger fat wire spacing rules.

When you widen the wires, it changes the timing of your block. Wire widening has a timing-preservation mode that allows you to perform wire widening without affecting the block timing.

To enable timing-preservation mode for wire widening, define the timing preservation constraints by using the following options with the `widen_wires` command:

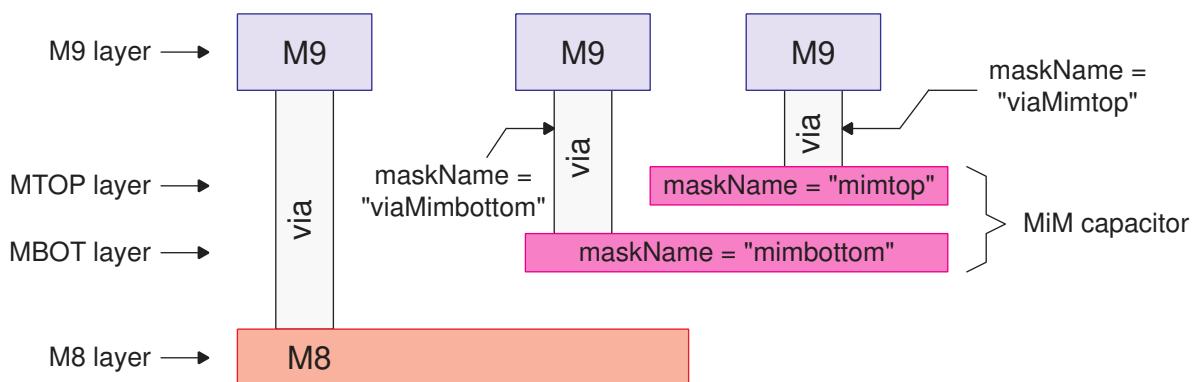
- `-timing_preserve_setup_slack_threshold threshold`
- `-timing_preserve_hold_slack_threshold threshold`
- `-timing_preserve_nets nets`

The threshold values are floating-point numbers in library units. Wire widening is not performed on nets with slack less than the specified values or the nets specified in the `-timing_preserve_nets` option.

## Inserting Metal-Insulator-Metal Capacitors

A metal-insulator-metal (MiM) capacitor is made of two special-purpose conducting layers separated by an insulator. This capacitor is inserted between two regular metal layers, as shown in [Figure 112](#).

*Figure 112 Cross-Section View of MiM Capacitor Layers*



MiM capacitors are typically connected between power and ground to help maintain a constant supply voltage in the presence of electrical noise. The dimensions of a MiM capacitor are usually customized to fit the power strap geometry of a specific power plan.

To insert an array of MiM capacitors into a block and connect them to the power rails, use the `create_mim_capacitor_array` command. At a minimum, you must specify the MiM capacitor library cell, as well as the x- and y-increments of the array.

For example,

```
fc_shell> create_mim_capacitor_array \
 -lib_cell my_lib/mim_ref_cell
 -x_increment 20 -y_increment 20
```

By default, the `create_mim_capacitor_array` command

- Inserts MiM capacitors for the entire block

To restrict the insertion to a specific region, use the `-boundary` option to specify the rectangular or rectilinear region. You can specify only a single region.

The command does not consider standard cells, macros, placement blockages, or voltage areas when inserting the MiM capacitors.

- Inserts the MiM capacitors in the R0 orientation

To change the orientation, use the `-orientation` option. When you change the orientation, it affects all MiM capacitor cells in the inserted array.

- Uses the following naming convention for the inserted MiM capacitor cells:

*mimcap!library\_cell\_name!number*

To identify the MiM capacitor cells inserted in a specific run, use the `-prefix` option to specify a prefix string. When you use this option, the tool uses the following naming convention:

*mimcap!prefix!library\_cell\_name!number*

## Inserting Filler Cells

To ensure that all power nets are connected, you can fill empty space in the standard-cell rows with filler cells. The Fusion Compiler tool supports filler cells with and without metal

and supports both single-height and multiheight filler cells. Filler cell insertion is often used to add decoupling capacitors to improve the stability of the power supply.

**Note:**

Before inserting filler cells, ensure that the block is legalized by using the `check_legality` command.

The Fusion Compiler tool can select the filler cells to use based on

- An ordered list of filler cell references

In the standard filler cell insertion flow, you specify an ordered list of filler library cells and the tool inserts the first cell that fits in each gap. If your technology requires specific filler cells abutting the standard cells, you can insert these required filler cells first, and then use standard filler cell insertion to fill the remaining space.

For details about this method, see [Standard Filler Cell Insertion](#).

- Threshold-voltage rules

In the threshold-voltage-based flow, the tool selects the filler cells by using user-defined insertion rules, which are based the threshold-voltage types of the cells that border the gap. This flow is typically used only for established foundry nodes.

For details about this method, see [Threshold-Voltage-Based Filler Cell Insertion](#).

## Standard Filler Cell Insertion

The standard filler cell insertion flow uses the `create_stdcell_fillers` command to insert filler cells in the block and the `remove_stdcell_fillers_with_violation` command to perform design rule checking on the filler cells and remove filler cells with violations.

During filler cell insertion, the `create_stdcell_fillers` command uses the placement legalizer to ensure that the inserted filler cells honor advanced node placement rules and physical constraints such as placement blockages and keepout margins. When the `place.legalize.enable_advanced_legalizer` application option is set to `true`, the command uses the advanced legalization algorithms for 2D rule checking and cell interaction, which can reduce filler cell insertion runtime.

To insert filler cells in your block,

1. Ensure that the block is legalized by using the `check_legality` command.

When verifying that a block is fillable, the `check_legality` command assumes that all standard cells that have a `design_type` attribute of `filler` can be used as filler cells. However, the `create_stdcell_fillers` command inserts only the cells specified in

the `-lib_cells` option. This difference might result in unfillable gaps after filler cell insertion even though the `check_legality` command did not report any issues.

2. (Optional) Insert required filler cells to abut specific standard cells by using the `create_left_right_filler_cells` command, as described in [Abutting Standard Cells With Specific Filler Cells](#).
3. (Optional) Enable multithreaded filler cell insertion by using the `set_host_options` command to specify the multicore configuration.

```
fc_shell> set_host_options -max_cores n
```

The `create_stdcell_fillers` command uses the multicore configuration only when the technology file contains minimum threshold voltage, oxide-diffusion (OD), or trimpoly (TPO) rules.

**Note:**

The multicore configuration specified with the `set_host_options` command applies to all Fusion Compiler commands that support multicore processing.

4. Insert metal filler cells by using the `create_stdcell_fillers` command.

Use the `-lib_cells` option to specify the metal filler library cells to insert. The command tries to insert the filler cells in the order that you specify; for the best results, specify them from the largest to the smallest.

To determine the filler cells available in a cell library, use the following command:

```
fc_shell> get_lib_cells ref_lib/* -filter "design_type==filler"
```

To sort the filler cells by decreasing size, use the `sort_collection` command, as shown in the following example:

```
fc_shell> set FILLER_CELLS \
 [get_object_name [sort_collection -descending \
 [get_lib_cells ref_lib/* -filter "design_type==filler"] area]]
```

For more information, see [Controlling Standard Filler Cell Insertion](#).

5. Connect the inserted filler cells to the power and ground (PG) network by using the `connect_pg_net -automatic` command.
6. Remove the metal filler cells with DRC violations by using the `remove_stdcell_fillers_withViolation` command.

Removing the filler cells can expose new violations; therefore, you must sometimes run this command multiple times to remove all violating filler cells.

For more information, see [Checking for Filler Cell DRC Violations](#).

7. Insert nonmetal filler cells by using the `create_stdcell_fillers` command.

Use the `-lib_cells` option to specify the nonmetal filler library cells to insert. The tool tries to insert the filler cells in the order that you specify; for the best results, specify them from the largest to the smallest.

For more information, see [Controlling Standard Filler Cell Insertion](#).

8. Connect the inserted filler cells to the PG network by using the `connect_pg_net -automatic` command.

### See Also

- [Generic ECO Flow for Timing or Functional Changes](#)
- [Removing Filler Cells](#)

## Controlling Standard Filler Cell Insertion

By default, the `create_stdcell_fillers` command fills all empty space in the horizontal standard-cell rows of the entire block by inserting instances of the filler library cells specified by the `-lib_cells` option.

You can control the following aspects of the filler cell insertion:

- The selection of filler cells based on lowest leakage current

By default, the command does not consider leakage current when inserting filler cells. To reduce the leakage current, use the `-leakage_vt_order` option to specify the threshold voltage layers in order of decreasing leakage current. The command uses this information to select the filler cells with the lowest leakage current that meet the legalization requirements.

To check whether the filler cells inserted in the current block result in the lowest possible leakage power, use the `-leakage_vt_check` option with the `create_stdcell_fillers` command. When you use this option, the command only checks the existing filler cells; it does not insert filler cells. You must also use the `-lib_cells` and `-leakage_vt_order` options to specify the available filler cells. The check reports the filler cells that could be replaced with a specified filler cell with lower leakage current without causing legalization errors. By default, the command reports a maximum of 100 violations. To change the maximum number of reported violations, set the `chf.create_stdcell_fillers.max_leakage_vt_orderViolations` application option.

**Note:**

This feature has the following requirements:

- You must enable the advanced legalizer by setting the `place.legalize.enable_advanced_legalizer` application option to `true`. For more information about the advanced legalizer, see [Enabling Advanced Legalization Algorithms](#).
- The threshold voltage layers must be defined in the technology file. A threshold voltage layer is identified by a `maskType` attribute of `implant` in the `Layer` section of the technology file.
- The percentage of empty space to fill

By default, the command fills all the empty space. To leave some empty space, use the `-utilization` option to specify the percentage of empty space to fill.

To control the relative amount of various types of filler cells, such as ULVT and LVT decoupling capacitor cells, use the `-type_utilization` option to specify the insertion percentage for each type of cell. The sum of all the percentages specified in this option must be less than or equal to 100. When you use this option, the `create_stdcell_fillers` command might leave some empty spaces. To fill these empty spaces, use the `-fill_remaining` option. When you use this option, the command uses the cells specified in the `-lib_cells` option but not the `-type_utilization` option to fill the empty spaces. For example,

```
insert 70 percent ULVT cells, 30 percent LVT cells
fc_shell> create_stdcell_fillers -lib_cells $FILLER_CELLS \
 -type_utilization { /*DCAP16*ULVT */DCAP8*ULVT */DCAP4*ULVT
 */DCAP2*ULVT */DCAP1*ULVT} 70
 { /*DCAP16*LVT */DCAP8*LVT */DCAP4*LVT
 */DCAP2*LVT */DCAP1*LVT} 30 } \
 -fill_remaining
```

By default, the tool randomly selects the filler cells from the specified library cells. To control the cell selection priority, use the `-prefer_type_ordering` option with the `-type_utilization` option. When you use the `-prefer_type_ordering` option, the tool selects the library cells in each group specified in the `-type_utilization` option in priority order, from left to right. If some of the cells are difficult to insert due to legalization rules or library cell size, specify them before cells that are easier to insert to

ensure that they are selected. For example, when the `-prefer_type_ordering` option is used with the previous example, the tool

- First tries to insert the ULVT decoupling capacitors to a utilization of 70 percent, first trying to insert `*/DCAP16*ULVT` cells, then `*/DCAP8*ULVT` cells, and so on
- Then tries to insert the LVT decoupling capacitors to a utilization of 30 percent, first trying to insert `*/DCAP16*LVT` cells, then `*/DCAP8*LVT` cells, and so on
- Then filling the remaining gaps using the remaining cells specified in the `-lib_cells` option

```
insert 70 percent ULVT cells, 30 percent LVT cells
fc_shell> create_stdcell_fillers -lib_cells $FILLER_CELLS \
 -type_utilization { {*/DCAP16*ULVT */DCAP8*ULVT */DCAP4*ULVT
 */DCAP2*ULVT */DCAP1*ULVT} 70
 {*/DCAP16*LVT */DCAP8*LVT */DCAP4*LVT
 */DCAP2*LVT */DCAP1*LVT} 30 } \
 -prefer_type_ordering \
 -fill_remaining
```

- The region in which to insert the filler cells
  - Use the `-bboxes` option to restrict filler cell insertion to the specified bounding boxes.
  - Use the `-voltage_area` option to restrict filler cell insertion to the specified voltage areas.
- Whether to check for power net violations

By default, the command does not check for power net violations, as this check increases runtime and is required only for metal filler cell insertion. However, this might result in the `remove_stdcell_fillers_withViolation` command removing many decoupling capacitors.

To prevent power net violations caused by filler cell insertion, use the `-rules check_pnet` option.

- Gap prevention

By default, the command assumes that the smallest cell size is one unit site. If the smallest cell in your design is larger than one unit site, this could cause gaps during filler cell insertion. To prevent the command from inserting filler cells that leave a gap, specify the smallest cell size as a multiple of the unit site by using the `-smallest_cell_size` option. You can specify a value of 1, 2, or 3.

**Note:**

The advanced legalizer automatically prevents gaps; therefore, this option is ignored when `place.legalize.enable_advanced_legalizer` application option is set to `true`.

- The handling of hard placement blockages

By default, the command honors hard placement blockages and does not insert filler cells in regions affected by these blockages. To ignore the hard placement blockages and insert filler cells in those regions, use the `-ignore_hard_blockages` option.

- The naming convention used to identify the inserted filler cell instances

By default, the command uses the following naming convention for inserted filler cells:

`xofiller!filler_library_cell_name!number`

To identify the filler cells inserted in a specific run, use the `-prefix` option to specify a prefix string. When you use this option, the command uses the following naming convention:

`xofiller!prefix!filler_library_cell_name!number`

- The behavior when the legalizer detects errors

By default, the command fails if the legalizer detects an error. To force the command to complete filler cell insertion even when errors occur, use the `-continue_on_error` option. When you use this option, the resulting block might have legalization errors that must be fixed manually.

## Checking for Filler Cell DRC Violations

By default, the `remove_stdcell_fillers_withViolation` command checks for DRC violations between all cell instances with `xofiller` in their name and top-level signal routing objects and removes the violating filler cells.

You can control the following aspects of the filler cell checking:

- The string used to identify the filler cells

To specify a different string to identify the filler cells, use the `-name` option.

- The region in which to check for violations

To restrict filler cell check to specific regions, use the `-boundary` option.

- The checked objects

To check for violations between filler cells and all neighboring objects, such as signal net shapes, other filler cells, standard cells, PG rails, and terminals, use the `-check_between_fixed_objects true` option.

- Restrict DRC to shorts-checking only

By default, the command removes all filler cells that cause a routing DRC violation. To remove only those filler cells that cause a short by overlapping a routing shape, use the `-shorts_only true` option.

- Whether double-patterning rules are checked

The command checks for double-patterning violations only when the following criteria are met:

- The `route.common.color_based_dpt_flow` application option is `true`.
- The technology file defines double-patterning rules

To check for DRC violations on filler cells before removing them, use the `-check_only true` option. When you use this option, this command writes a report to a file named `block_fillers_withViolation.rpt` in the current working directory. The report lists the filler cells with violations and reports the first violation for each filler cell.

## Fixing Remaining Mask Design Rule Violations

Some mask design rules are not handled during filler cell insertion because they have a low probability of occurrence and they require extensive computation time if handled during insertion.

After filler cell insertion, use the `replace_fillers_by_rules` command to fix these design rule violations. You must specify the rule to fix by using the `-replacement_rule` option. [Table 39](#) lists the design rules supported by this command and the keywords used to specify these rules.

*Table 39 Design Rules Supported by the replace\_fillers\_by\_rules Command*

Design rule	Description	-replacement_rule keyword
End orientation	This rule allows the leftmost or rightmost filler cell to be flipped to a different orientation. For more information, see <a href="#">End Orientation Rule</a> .	<code>end_orientation</code>
Half row adjacency for half-height filler cells	This rule restricts the placement of half-height filler cells. For more information, see <a href="#">Half Row Adjacency Rule</a> .	<code>half_row_adjacency</code>

**Table 39     Design Rules Supported by the `replace_fillers_by_rules` Command (Continued)**

Design rule	Description	-replacement_rule keyword
Illegal abutment	This rule prevents a filler cell from abutting certain other filler cells. For more information, see <a href="#">Illegal Abutment Rule</a> .	<code>illegal_abutment</code>
Maximum horizontal length	This rule restricts the horizontal length of a continuous row of filler cells. For more information, see <a href="#">Maximum Horizontal Length Rule</a> .	<code>od_horizontal_distance</code>
Maximum horizontal edge length	This rule restricts the horizontal length of a sequence of abutted objects on a specific layer. For more information, see <a href="#">Maximum Horizontal Edge Length Rule</a> .	<code>horizontal_edges_distance</code>
Maximum stacking for small filler cells	This rule restricts the horizontal length of a sequence of abutted objects on a specific layer. For more information, see <a href="#">Maximum Stacking for Small Filler Cells Rule</a> .	<code>small_filler_stacking</code>
Maximum vertical edge length	This rule restricts the vertical length of a column of stacked filler cells. For more information, see <a href="#">Maximum Vertical Edge Length Rule</a> .	<code>max_vertical_constraint</code>
Tap cell spacing	This rule restricts the distance between a tap cell's oxide diffusion (OD) layer and the neighboring OD layers. For more information, see <a href="#">Tap Cell Spacing Rule</a> .	<code>od_tap_distance</code>
N/A	Randomly replace filler cells with custom filler cells. For more information, see <a href="#">Random Filler Cell Replacement</a> .	<code>random</code>

## End Orientation Rule

The end orientation rule searches for a continuous horizontal sequence of cells and modifies the orientation of the leftmost or rightmost filler cells if they are in the specified list of target filler cells.

To select this rule, use the `-replacement_rule end_orientation` option with the `replace_fillers_by_rules` command. When you select this rule, you must specify

- The target filler cells

To specify the target filler cells, use the `-target_fillers` option.

- The orientation modification
  - To change the orientation for the leftmost filler cells, use the `-left_end` option.
  - To change the orientation for the rightmost filler cells, use the `-right_end` option.
  - To change the orientation for both the leftmost and rightmost filler cells, use both options.

The valid values for these options are

- `inverse`, which flips the cell from its existing orientation
- `R0_or_MX`, which changes an `MY` orientation to `R0` or an `R180` orientation to `MX`
- `MY_or_R180`, which changes an `R0` orientation to `MY` or an `MX` orientation to `R180`

To report the cells whose orientation would be changed, without actually changing the orientations, use the `-check_only` option. This option is valid only for the end orientation rule.

### Half Row Adjacency Rule

The half row adjacency rule searches for half-height filler cells and replaces them based on the types of their neighboring standard cells. The standard cells are classified as one of the following types:

- Inbound

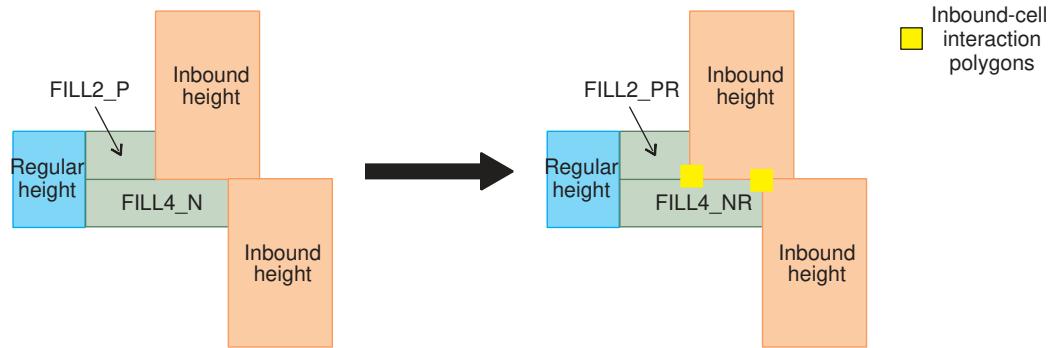
The top or bottom boundary of an inbound standard cell is at the half-height location of the cell rows. When a half-height filler cell abuts an inbound cell, the abutting side of the half-height filler cell must have inbound-cell interaction polygons (ICIP).

- Regular

The top and bottom boundaries of a regular standard cell are at the full-height location of the cell rows. When a half-height filler cell abuts a regular cell, the abutting side of the half-height filler cell must not have inbound-cell interaction polygons (ICIP).

The replacement filler cells have the same height, width, and threshold voltage as the original filler cells. The cell libraries include both p-type and n-type replacement filler cells with inbound-cell interaction polygons (ICIP). For p-type half-height filler cells, the inbound-cell interaction polygons (ICIP) are at the bottom corners. For n-type half-height filler cells, the inbound-cell interaction polygons (ICIP) are at the top corners. [Figure 113](#) shows the replacement of the half-height filler cells using the half row adjacency rule.

Figure 113 Half-Height Filler Cell Replacement With Half Row Adjacency Rule



To select this rule, use the `-replacement_rule half_row_adjacency` option with the `replace_fillers_by_rules` command. The following table lists the options used to specify the replacement half-height filler cells. Each list can contain only one cell of each size.

Table 40 Half-Height Replacement Filler Cells

Abutting inbound standard cells	Option to specify replacement filler cells
None	<code>-p_none, -n_none</code>
Left-only	<code>-p_left, -n_left</code>
Right-only	<code>-p_right, -n_right</code>
Left and right	<code>-p_left_right, -n_left_right</code>
Top or bottom (applies only to 2X filler cells)	<code>-p_left_center_right, -n_left_center_right</code>

### Illegal Abutment Rule

The illegal abutment rule restricts the filler cells that can abut certain other filler cells.

To select this rule, use the `-replacement_rule illegal_abutment` option with the `replace_fillers_by_rules` command. When you select this rule, you must specify

- The filler cells considered for the check and their replacements

The filler cells considered for the check are called *target cells*. To specify the target filler cells and their replacements, use the following syntax with the `-replace_abutment` option:

```
{ {target_cell1 refill_cell1} ... }
```

To ignore filler cells that have a `physical_status` attribute of `fixed` or `locked`, use the `-skip_fixed_cells` option with the `replace_fillers_by_rules` command.

- The filler cells that cannot abut the target filler cells

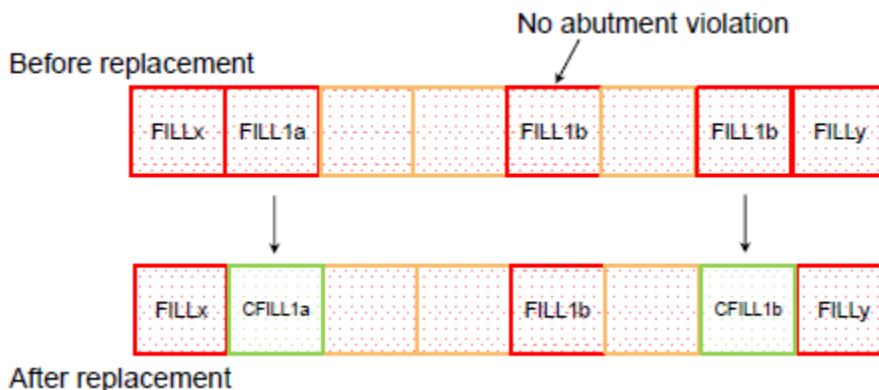
These filler cells are called *illegal cells*. To specify the illegal cells, use the `-illegal_abutment` option. Note that the target filler cells specified in the `-replace_abutment` option are also considered illegal cells.

For example, the following command fixes illegal abutment violations where the `FILL1a` and `FILL1b` filler cells cannot abut the `FILLx` and `FILLY` cells:

```
fc_shell> replace_fillers_by_rules -replacement_rule illegal_abutment \
 -replace_abutment { {FILL1a CFILL1a} {FILL1b CFILL1b} } \
 -illegal_abutment {FILLx FILLY}
```

**Figure 114** shows the original row, which violates this rule, and the resulting row, which uses the replacement filler cells to fix the violations. The target filler cells are shown in red, while the replacement filler cells are shown in green.

Figure 114 Fixing an Illegal Abutment Violation



## Maximum Horizontal Edge Length Rule

The maximum horizontal edge length rule restricts the horizontal length of a sequence of abutted objects on a specific layer.

To select this rule, use the `-replacement_rule horizontal_edges_distance` option with the `replace_fillers_by_rules` command. When you select this rule, you must specify

- The maximum horizontal length
  - To specify the maximum length, use the `-max_constraint_length` option.
- The layer to check
  - To specify the layer, use the `-layer` option.
- The cells considered for the check and the filler cells that can be used to replace the violating filler cells

The filler cells considered for the check are called *constrained cells*. The filler cells considered to replace violating cells are called *unconstrained cells*.

- To specify multiple constrained cell groups and their replacement cells, use the following syntax with the `-refill_table` option:

```
{ {{refill_cell1} {constrained_cells1}} ... }
```

You can use this syntax to fix violations for threshold-voltage-based filler cells.

- To specify a single constrained cell group, use the `-constraint_fillers` option. To specify the replacement cells, use the `-non_constraint_fillers` option.

### Note:

If you use both methods, the command uses only the information specified in the `-refill_table` option.

For example, the following command identifies and fixes maximum horizontal edge length violations where the horizontal length on the M2 layer exceeds 30 um:

```
fc_shell> replace_fillers_by_rules \
 -replacement_rule horizontal_edges_distance \
 -max_constraint_length 30 -layer M2 \
 -refill_table { {{VT1_FILL1} {VT1_DECAP4 VT1_DECAP2}} \
 {{VT2_FILL1} {VT2_DECAP4 VT2_DECAP2}} }
```

## Maximum Horizontal Length Rule

The maximum horizontal length rule restricts the horizontal length of a sequence of abutted filler cells.

To select this rule, use the `-replacement_rule od_horizontal_distance` option with the `replace_fillers_by_rules` command. When you select this rule, you must specify

- The maximum horizontal length
 

To specify the maximum length, use the `-max_constraint_length` option.
- The cells considered for the check and the filler cells that can be used to replace the violating filler cells

The filler cells considered for the check are called *constrained cells*. The filler cells considered to replace violating cells are called *unconstrained cells*.

- To specify multiple constrained cell groups and their replacement cells, use the following syntax with the `-refill_table` option:

```
{ {{refill_cell1} {constrained_cells1}} ... }
```

You can use this syntax to fix violations for threshold-voltage-based filler cells.

- To specify a single constrained cell group, use the `-constraint_fillers` option. To specify the replacement cells, use the `-non_constraint_fillers` option.

### Note:

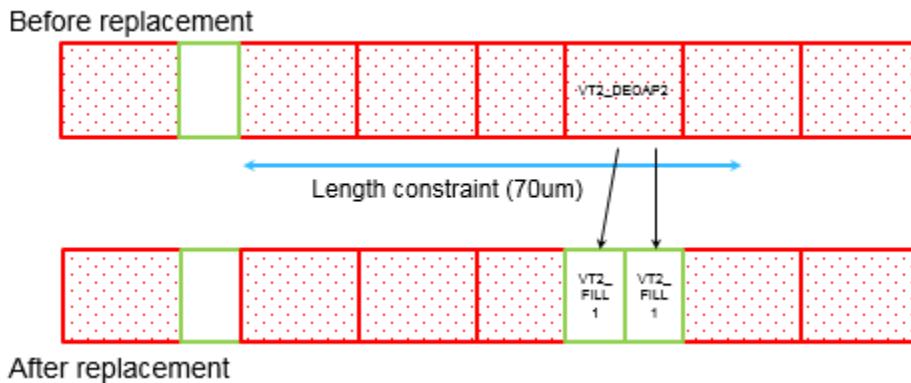
If you use both methods, the command uses only the information specified in the `-refill_table` option.

For example, the following command identifies and fixes maximum horizontal length violations where the horizontal length exceeds 70 um:

```
fc_shell> replace_fillers_by_rules \
 -replacement_rule od_horizontal_distance -max_constraint_length 70 \
 -refill_table { {{VT1_FILL1} {VT1_DECAP4 VT1_DECAP2}} \
 {{VT2_FILL1} {VT2_DECAP4 VT2_DECAP2}} }
```

[Figure 115](#) shows the original row, which violates this rule, and the resulting row, which uses unconstrained filler cells to fix the violation. The constrained cells are shown in red, while the unconstrained cells are shown in green.

Figure 115 Fixing a Maximum Vertical Edge Length Violation



### Maximum Stacking for Small Filler Cells Rule

The maximum stacking for small filler cells rule restricts the stacking height of small filler cells.

To select this rule, use the `-replacement_rule small_filler_stacking` option with the `replace_fillers_by_rules` command. When you select this rule, you must specify

- The maximum stacking height
 

To specify the maximum stacking height, use the `-max_constraint_length` option.
- The cells considered for the check
 

To specify the small filler cells to consider for the check, use the `-small_fillers` option.
- The filler cells that can be used to replace the violating filler cells
 

To specify the filler cells that can be used to replace the violating filler cells, use the `-replacement_fillers` option.

If the vertical stacking of the small filler cells specified in the `-small_fillers` option exceeds the height specified in the `-max_constraint_length` option, the command replaces two or three consecutive filler cells adjacent to the violating cell with one of the large filler cells specified in the `-replacement_fillers` option. If it is not possible to replace the filler cells, the command reorders the filler cells to fix the violation. The command maintains the threshold voltage at the location of the violating filler cell; however, the threshold voltage at the location of non-violating filler cells might change. The command does not check for threshold voltage violations caused by these changes.

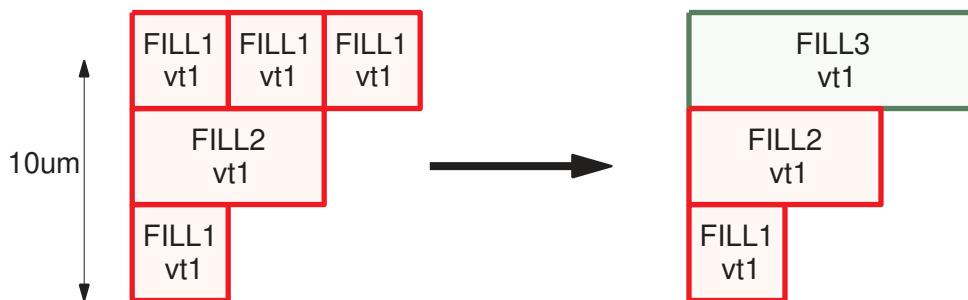
For example, the following command identifies and fixes maximum stacking violations where the stacking height exceeds 10 um:

```
fc_shell> replace_fillers_by_rules \
 -replacement_rule small_filler_stacking -max_constraint_length 10 \
 -small_fillers {FILL1 FILL2} -replacement_fillers {FILL3 FILL4} \
```

The following figures illustrate the replacement and reordering techniques used to fix maximum stacking violations. In these figures, the small filler cells are shown in red, while the large filler cells are shown in green.

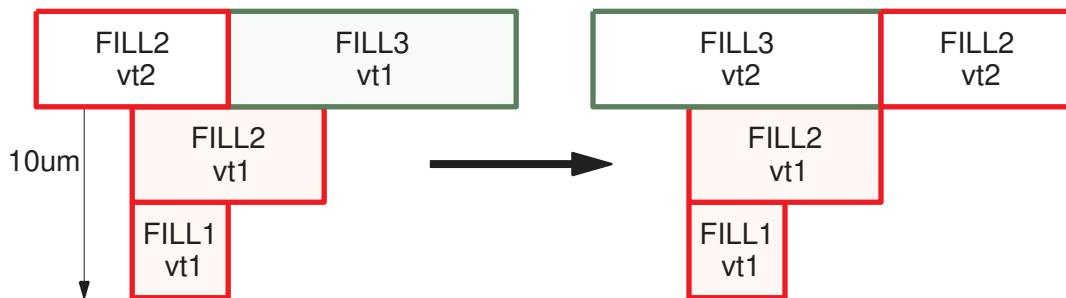
[Figure 116](#) shows the original stack, which violates this rule, and the resulting stack, which uses a replacement filler cell to fix the violation.

*Figure 116 Fixing Maximum Stacking Violation by Replacement*



[Figure 117](#) shows the original stack, which violates this rule, and the resulting stack, which uses reordering to fix the violation. Note that the threshold voltage of the FILL3 cell is changed so that the vt2 threshold voltage is maintained in the original location.

*Figure 117 Fixing Maximum Stacking Violation by Reordering*



## Maximum Vertical Edge Length Rule

The maximum vertical edge length rule restricts the vertical length of a stack of certain filler cells and standard cells.

To select this rule, use the `-replacement_rule max_vertical_constraint` option with the `replace_fillers_by_rules` command. When you select this rule, you must specify

- The maximum vertical length

To specify the maximum vertical length, use the `-max_constraint_length` option.

- The cells considered for the check

These cells are called *constrained cells*. To specify the constrained filler cells, use the `-constraint_fillers` option. By default, all standard cells are constrained. To exclude specific standard cells from consideration, use the `-exception_cells` option. To ignore the length violation when a constrained cell abuts an exception cell, use the `-noViolationAlongExceptionCells` option.

- The filler cells that can be used to replace the violating filler cells

These filler cells are called *unconstrained cells*; they are not considered when measuring the stack height.

- To specify the filler cells that can be used to break a continuous edge of constrained cells, use the `-non_constraint_fillers` option.
- To specify the filler cells whose left side can be used to break a continuous edge of constrained cells, use the `-non_constraint_left_fillers` option.
- To specify the filler cells whose right side can be used to break a continuous edge of constrained cells, use the `-non_constraint_right_fillers` option.

### Note:

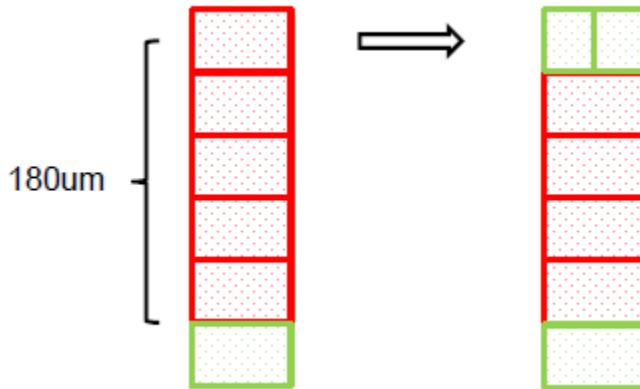
Any filler cells that are not identified as constrained, unconstrained, or exception cells are considered constrained cells.

For example, the following command identifies and fixes maximum vertical edge length violations where the vertical length exceeds 180 um:

```
fc_shell> replace_fillers_by_rules \
 -replacement_rule max_vertical_constraint -max_constraint_length 180 \
 -constraint_fillers {DECAP8 DECAP4} -non_constraint_fillers {FILL1} \
 -exception_cells {BUF1 TAP1}
```

**Figure 118** shows the original stack, which violates this rule, and the resulting stack, which uses unconstrained filler cells to fix the violation. The constrained cells are shown in red, while the unconstrained cells are shown in green.

Figure 118 Fixing a Maximum Vertical Edge Length Violation



### Tap Cell Spacing Rule

The tap cell spacing rule restricts the distance between a target tap cell's oxide diffusion (OD) layer and the OD layers of its neighboring cells on the left and right sides.

To select this rule, use the `-replacement_rule od_tap_distance` option with the `replace_fillers_by_rules` command. When you select this rule, you must specify

- The disallowed spacing range

The spacing distance is specified as the number of sites. To specify the disallowed spacing range, use the `-tap_distance_range` option.

- The tap cells considered for the check

These cells are called *target cells*. To specify the target tap cells, use the `-tap_cells` option.

- The tap cells used to replace the violating tap cells

These cells are called *replacement cells*.

- To specify the replacement tap cell for left-side violations, use the `-leftViolation_tap` option.
- To specify the replacement tap cell for right-side violations, use the `-rightViolation_tap` option.
- To specify the replacement tap cell for both-side violations, use the `-bothViolation_tap` option.

#### Note:

The target and replacement tap cells must all have the same size.

By default, the command measures the distance to the cells that abut the target tap cell. If the tap cells abut library cells that do not have an OD layer, use the `-adjacent_non_od_cells` option to specify these library cells. The command skips the specified non-OD cells and measures to distance from the target tap cell to the first neighbor with an OD layer.

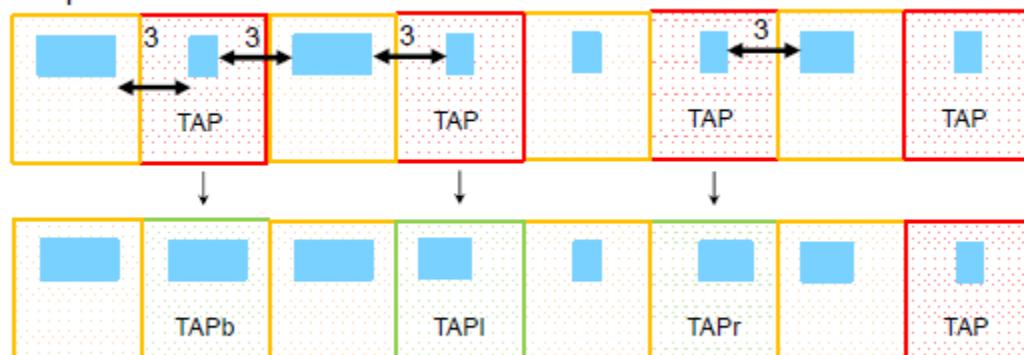
For example, the following command replaces tap cells named TAP that have a separation distance of three sites:

```
fc_shell> replace_fillers_by_rules -replacement_rule od_tap_distance \
 -tap_cells {TAP} -tap_distance_range {3 3} \
 -leftViolation_tap {TAPl} -rightViolation_tap {TAPr} \
 -bothViolation_tap {TAPb}
```

Figure 119 shows the original row, which violates this rule, and the resulting row, which uses the replacement filler cells to fix the violations. The target filler cells are shown in red, while the replacement filler cells are shown in green.

Figure 119 Fixing a Tap Cell Spacing Violation

#### Before replacement



#### After replacement

### Random Filler Cell Replacement

To randomly replace existing filler cells, use the `-replacement_rule random` option with the `replace_fillers_by_rules` command. When you select this rule, you must specify

- The filler cells to be replaced and their replacements

The filler cells to be replaced are called *target cells*. To specify the target filler cells and their replacements, use the following syntax with the `-random_replace` option:

```
{ {target_cell1 {refill_cell1a refill_cell1b ...}} ... }
```

To ignore filler cells that have a `physical_status` attribute of `fixed` or `locked`, use the `-skip_fixed_cells` option with the `replace_fillers_by_rules` command.

When the command finds one of the target cells, it replaces it by randomly selecting one of the replacement cells specified for that target cell. For example, the following command replaces FILL1 cells with one of FILL1a, FILL1b, or FILL1c and replaces FILL2 cells with one of FILL2a or FILL2b:

```
fc_shell> replace_fillers_by_rules -replacement_rule random \
 -random_replace { {FILL1 {FILL1a FILL1b FILL1c}} \
 {FILL2 {FILL2a FILL2b}} }
```

## Abutting Standard Cells With Specific Filler Cells

If your technology requires specific filler cells on the left or right sides of certain standard cells, use the `create_left_right_filler_cells` command to insert these filler cells. After you run the `create_left_right_filler_cells` command, perform standard filler cell insertion to fill the remaining gaps in the design.

You must use the `-lib_cells` option to specify the rules for filler cell insertion. Each rule uses the following format:

```
{standard_cells} {left_filler_cells} {right_filler_cells}
```

- For the `standard_cells` argument,
  - You must specify a list of one or more standard cells
  - You can use wildcards to specify the cell names
  - If you specify multiple rules for the same standard cell, the command uses only the last rule specified for that cell.
- For the `left_filler_cells` and `right_filler_cells` arguments,
  - At least one of these arguments must contain one or more filler library cells

If only one of the arguments contains filler library cells, the tool inserts filler cells only on that side of the standard cells.

  - The specified filler cells must meet the following requirements:
    - They must have a `design_type` attribute of `lib_cell` or `filler`.
    - They must use the same site definition as the standard cells.
    - The height of the standard cells must be an integer multiple of the filler cell heights.
  - The command tries to insert the filler cells in the order that you specify; for the best results, specify them from the largest to the smallest.

- You can use wildcards for the filler cell names.

When you use wildcards, the order of the cells is the same as that returned by the `get_lib_cells` command.

The following example inserts filler cells on the left and right sides of the SC1 standard cells and only on the left side of the SC2 standard cells. The left and right filler cells are specified in decreasing size order, so that the command inserts the largest possible filler cell.

```
fc_shell> create_left_right_filler_cells \
 -lib_cells {
 { {mylib/SC1} {mylib/LF4X mylib/LF2X} {mylib/RF2X mylib/RF1X} }
 { {mylib/SC2} {mylib/LF2X mylib/LF1X} {} } }
```

When the `create_left_right_filler_cells` command inserts the filler cells,

- It checks for legal locations for the filler cells, but does not check advanced rules such as minimum threshold voltage, oxide-diffusion (OD), or trimpoly (TPO) rules.
- It inserts the filler cells immediately next to the standard cells; it does not consider the intercell spacing rules.

## See Also

- [Standard Filler Cell Insertion](#)

## Controlling Cell-Based Filler Cell Insertion

You can control the following aspects of the filler cell insertion with the `create_left_right_filler_cells` command:

- The region in which to insert the filler cells

By default, the command inserts filler cells next to the specified standard cells in the entire block.

- Use the `-boundaries` option to restrict filler cell insertion to the specified regions. You can specify one or more rectangular or rectilinear regions.
- Use the `-voltage_areas` option to restrict filler cell insertion to the specified voltage areas.

- The orientation of the filler cells

By default, the command uses the allowable orientations of the row to place the filler cells. To use the same orientation as the standard cell, use the `-follow_stdcell_orientation` option. When you use this option, the tool swaps the right and left filler cells when the cell is flipped.

- The handling of one-unit-tile gaps

By default, the command does not consider the one-unit-tile rule. To prevent the tool from inserting filler cells that would leave a one-unit-tile gap, use the `-rules no_1x` option. When you enable this rule, do not include filler library cells with a width of one-unit tile in the `-lib_cells` option; otherwise, the tool ignores this rule.

- The naming convention used to identify the inserted filler cell instances

By default, the command uses the following naming convention for inserted filler cells:

`xofiller!filler_library_cell_name!number`

To identify the filler cells inserted in a specific run, use the `-prefix` option to specify a prefix string. When you use this option, the command uses the following naming convention:

`xofiller!prefix!filler_library_cell_name!number`

## Threshold-Voltage-Based Filler Cell Insertion

In the threshold-voltage-based flow, the tool selects the filler cells by using user-defined insertion rules, which are based the threshold-voltage types of the cells that border the gap. This flow is typically used only for established foundry nodes.

To perform threshold-voltage-based filler cell insertion,

1. Ensure that the block is legalized by using the `check_legality` command.
2. Annotate the threshold-voltage types on the reference cells by using the `set_cell_vt_type` command.

For example, to specify that all cells whose names end with `_vtA` have a threshold voltage type of `vtA` and the `invx1` cell has a threshold voltage type of `default`, use the following commands:

```
fc_shell> set_cell_vt_type -lib_cells "*/*_vtA" -vt_type vtA
fc_shell> set_cell_vt_type -lib_cells "mylib/invx1" -vt_type default
```

3. Define the rules for inserting the filler cells by using the `set_vt_filler_rule` command.

You define the rules by specifying the filler cells to insert based on the threshold-voltage types of the cells on the left and right of the gap being filled.

For example, to specify that `filler2x` or `filler1x` cells can be inserted in a gap that has `default` threshold-voltage cells on the left and right sides and `fillerA2x` and `fillerA1x` cells

can be inserted in a gap that has vtA threshold-voltage cells on the left and right sides, use the following commands:

```
fc_shell> set_vt_filler_rule -vt_type {default default} \
 -filler_cells {myLib/filler2x myLib/filler1x}
fc_shell> set_vt_filler_rule -vt_type {vtA vtA} \
 -filler_cells {myLib/fillerA2x myLib/fillerA1x}
```

4. Insert the filler cells by using the `create_vtcell_fillers` command.

## Controlling Threshold-Voltage-Based Filler Cell Insertion

By default, the `create_vtcell_fillers` command fills all empty space in the horizontal standard-cell rows of the entire block.

You can control the following aspects of the filler cell insertion:

- The region in which to insert the filler cells
  - Use the `-region` option to restrict filler cell insertion to the specified regions.
  - Use the `-voltage_area` option to restrict filler cell insertion to the specified voltage areas.
- The naming convention used to identify the inserted filler cell instances

By default, the command uses the following naming convention for inserted filler cells:

`xofiller!filler_library_cell_name!number`

To identify the filler cells inserted in a specific run, use the `-prefix` option to specify a prefix string. When you use this option, the command uses the following naming convention:

`xofiller!prefix!filler_library_cell_name!number`

To change the separator character from the default exclamation mark (!), use the `-separator` option.

## Removing the Threshold-Voltage Filler Cell Information

To remove the filler cell insertion rules and the threshold-voltage type annotations, use the `-clear_vt_information` option with the `create_vtcell_fillers` command.

---

## Removing Filler Cells

Filler cells inserted by the Fusion Compiler tool have instance names that start with the string `xofiller`. To remove all the filler cells from a block, use the `remove_cells` command, as shown in the following example:

```
fc_shell> remove_cells [get_cells xofiller*]
```

If you used the standard filler cell insertion flow and want to remove only those filler cells with DRC violations, use the `remove_stdcell_fillers_withViolation` command, as described in [Checking for Filler Cell DRC Violations](#).

---

## Inserting Metal Fill

After routing, you can fill the empty spaces in the block with metal wires to meet the metal density rules required by most fabrication processes. Before inserting metal fill, the block should be close to meeting timing and have very few or no DRC violations.

To insert metal fill, run the `signoff_create_metal_fill` command, as described in [Inserting Metal Fill With IC Validator In-Design](#).

**Note:**

An IC Validator license is required to run the `signoff_create_metal_fill` command.

# 8

## IC Validator In-Design

---

The IC Validator In-Design feature provides the ability to use the IC Validator tool to perform physical implementation and verification tasks within the Fusion Compiler tool. You can use IC Validator In-Design to perform the following tasks:

- Signoff design rule checking (the `signoff_check_drc` command)
- Fixing the violations detected during signoff design rule checking (the `signoff_fix_drc` command)
- Augmenting the power grid based on voltage drop information (the `signoff_create_pg_augmentation` command)
- Inserting metal fill (the `signoff_create_metal_fill` command)
- Fixing isolated vias (the `signoff_fix_isolated_via` command)

**Note:**

An IC Validator license is required to run the IC Validator In-Design functions.

To learn about using these IC Validator In-Design functions, see the following topics:

- [Preparing to Run IC Validator In-Design Commands](#)
- [Performing Signoff Design Rule Checking](#)
- [Automatically Fixing Signoff DRC Violations](#)
- [Improving Instance Voltage Drop by Augmenting the Power Grid](#)
- [Inserting Metal Fill With IC Validator In-Design](#)
- [Automatically Fixing Isolated Vias](#)

---

## Preparing to Run IC Validator In-Design Commands

The following topics describe the tasks you must perform before you run the IC Validator In-Design commands:

- [Setting Up the IC Validator Environment](#)
  - [Enabling IC Validator Multicore Processing](#)
  - [Defining the Layer Mapping for IC Validator In-Design Commands](#)
- 

### Setting Up the IC Validator Environment

To run an IC Validator In-Design command, you must specify the location of the IC Validator executable by setting the `ICV_HOME_DIR` environment variable. You can set this variable in your `.cshrc` file. To specify the location of the IC Validator executable, use commands similar to those shown in the following example:

```
setenv ICV_HOME_DIR /root_dir/icv
set path = ($path $ICV_HOME_DIR/bin/LINUX.64)
```

You must ensure that the version of the IC Validator executable that you specify is compatible with the Fusion Compiler version that you are using. To report the version compatibility, use the `report_versions` command.

For more information about the IC Validator tool, see the IC Validator documentation, which is available on SolvNet.

---

### Enabling IC Validator Multicore Processing

By default, the IC Validator In-Design commands use a single process to perform design rule checking. To reduce the turnaround time for design rule checking, use multicore processing, which includes multithreading, distributing processing, and parallel command execution. After you specify how many cores to use on different hosts, the IC Validator tool determines exactly how to take advantage of multithreading and parallel command execution to achieve the fastest runtimes.

To enable multicore processing, you must define the configuration by using the `set_host_options` command. By default, the configuration applies to all Fusion Compiler commands that support multicore processing. To define a configuration that applies only to the IC Validator In-Design commands, use the `-target ICV` option.

You can enable multicore processing on specific hosts, by using a job scheduler such as the Load Sharing Facility (LSF), or by using a combination of these methods. The following topics describe these methods:

- [Running IC Validator on Specific Hosts](#)
- [Running IC Validator Using a Job Scheduler](#)
- [Running IC Validator Using Hybrid Multicore Processing](#)

For general information about the `set_host_options` command, see [Enabling Multicore Processing](#).

## Running IC Validator on Specific Hosts

To enable IC Validator multicore processing on one or more specific hosts, use the `set_host_options` command to specify the following information:

- The host names

If you do not specify a host name or submit protocol, the IC Validator tool runs on the host on which the Fusion Compiler tool is currently running.

- The number of cores

Use the `-num_processes` or `-max_cores` options to specify the number of cores. If you use both options, the IC Validator tool uses the product of the settings as the number of cores.

The specified number of cores applies to each specified host.

For example, to enable the IC Validator tool to use 4 cores on the current host, use the following command:

```
fc_shell> set_host_options -target ICV -num_processes 4
```

To enable the IC Validator tool to use 16 cores (8 cores each on machineA and machineB), use the following command:

```
fc_shell> set_host_options -target ICV \
 -num_processes 2 -max_cores 4 {machineA machineB}
```

## Running IC Validator Using a Job Scheduler

To enable IC Validator multicore processing using a job scheduler, use the `set_host_options` command to specify the following information:

- The submit protocol
  - To use LSF, use the `-submit_protocol lsf` option.  
 The LSF `bsub` command must be in your Linux path.
  - To use the Univa Grid Engine (UGE) job scheduler, which was previously known as SGE (Sun Grid Engine) or GRD (Global Resource Directory), use the `-submit_protocol grid` option.  
 The Grid `qsub` command must be in your Linux path.

If you do not specify a host name or submit protocol, the IC Validator tool runs on the host on which the Fusion Compiler tool is currently running.

- The number of hosts to use
 

Use the `-num_processes` option to specify the number of hosts.
- The number of cores to use on each host
 

Use the `-max_cores` option to specify the number of cores per host.
- (Optional) Extra arguments needed for the submit command
 

Use the `-submit_command` option to specify the submit command with the extra arguments.

For example, to enable 4 hosts to be acquired by LSF, each of which can use 8 cores (for a total of 32 cores), use the following command:

```
fc_shell> set_host_options -submit_protocol lsf -target ICV \
 -num_processes 4 -max_cores 8
```

To set the memory requirement of the jobs in the previous example to 1000M, use the following command:

```
fc_shell> set_host_options -submit_protocol lsf -target ICV \
 -num_processes 4 -max_cores 8 \
 -submit_command {bsub -R rusage[mem=1000]}
```

To enable 4 hosts to be acquired by the Grid Engine, each of which can use 8 cores (for a total of 32 cores), use the following command:

```
fc_shell> set_host_options -submit_protocol grid -target ICV \
 -num_processes 4 -max_cores 8
```

To set the process name of the jobs in the previous example to bnormal, use the following command:

```
fc_shell> set_host_options -submit_protocol grid -target ICV \
 -num_processes 4 -max_cores 8 -submit_command "qsub -P bnormal"
```

## Running IC Validator Using Hybrid Multicore Processing

If you are using LSF, you can specify both specific hosts and an LSF configuration for adding additional hosts as they become available. To use this method,

- Specify the configuration for the specific hosts as described in [Running IC Validator on Specific Hosts](#).
- Specify the LSF configuration as described in [Running IC Validator Using a Job Scheduler](#), but use the `-add_hosts` option in addition to the other command options.

This configuration represents both optional resources and maximum resources – the resources can be given to the IC Validator tool at the beginning, middle, or end of the run, or not given at all.

For example, to use four cores on the current host and potentially acquire up to four more hosts from LSF, use the following commands:

```
fc_shell> set_host_options -target ICV -max_cores 4
fc_shell> set_host_options -target ICV -add_hosts \
 -submit_protocol lsf -num_processes 4
```

## Defining the Layer Mapping for IC Validator In-Design Commands

A layer mapping file maps the technology layers to the layers used in the runset file. You specify the location of the layer mapping file by setting the `signoff.physical.layer_map_file` application option.

Each line in the layer mapping file specifies a layer in the technology file and the corresponding layer in the runset file using the following syntax:

```
tf_layer[:tf_purpose][:use_type][:mask_type]
 runset_layer[:runset_data_type]
```

Each line can contain the following items:

- `tf_layer` – Layer number in the technology file, a required item
- `tf_purpose` – Purpose number in the technology file
- `use_type` – Fusion Compiler usage of the geometries in the design

**Valid values are** power, ground, signal, clock, boundary, hard\_placement\_blockage, soft\_placement\_blockage, routing\_blockage, area\_fill, and track.

- *mask\_type* – Multiple-patterning mask constraint of the geometries

Valid values for metal layers are `mask_one`, `mask_two`, `mask_three`, and `mask_same`. Via layers support the additional values `MASK_FOUR` through `MASK_FIFTEEN`.

- *runset\_layer* – Layer number in the runset file, a required item
- *runset\_data\_type* – Layer data type number in the runset file

Any text in a line that comes after a semicolon character (;) is considered a comment and has no effect on layer mapping.

## Performing Signoff Design Rule Checking

IC Validator In-Design signoff design rule checking (DRC) runs the IC Validator tool within the Fusion Compiler tool to check the routing design rules defined in the foundry signoff runset.

You can perform signoff design rule checking by

- [Running the `signoff\_check\_drc` command](#)
- [Using the Live DRC feature in the Fusion Compiler GUI](#)

After place and route, use the `signoff_check_drc` command to perform signoff design rule checking on the entire block. After engineering change orders (ECO), use either method to perform incremental signoff design rule checking on the modified portions of the block.

### See Also

- [Automatically Fixing Signoff DRC Violations](#)

## Running the `signoff_check_drc` Command

To perform signoff design rule checking by running the `signoff_check_drc` command,

1. Set up the IC Validator environment as described in [Setting Up the IC Validator Environment](#).
2. (Optional) Enable distributed processing by using the `set_host_options` command, as described in [Enabling IC Validator Multicore Processing](#).

3. Set the application options for signoff design rule checking.

At a minimum, you must specify the foundry runset to use for design rule checking by setting the `signoff.check_drc.runset` application option.

For information about the options for command-based signoff design rule checking, see [Setting Options for Signoff Design Rule Checking](#).

4. Save the block to disk.

When you run signoff design rule checking, the IC Validator tool uses the on-disk information for the block, not the information in memory. To ensure accurate information, use the `save_block` command to save the current state of the block before running signoff design rule checking.

5. Run signoff design rule checking by using the `signoff_check_drc` command.

By default, the `signoff_check_drc` command performs the following tasks:

1. Loads the block into the IC Validator tool, as described in [Reading Blocks for Signoff Design Rule Checking](#)
2. Performs signoff design rule checking, as described in [Signoff Design Rule Checking](#)
3. Generates an error data file and IC Validator results files, as described in [Signoff DRC Results Files](#)

You can use the results files to view the violations in the error browser, as described in [Using the Error Browser](#), or to perform automatic design rule fixing, as described in [Automatically Fixing Signoff DRC Violations](#).

You can use the following methods to analyze the DRC violations reported by the `signoff_check_drc` command:

- View the violations in the error browser, as described in [Using the Error Browser](#).
- View the density of violations in a heat map, as described in [Viewing the Violations in an ICV Heat Map](#).

## See Also

- [Checking Signoff Design Rules Interactively in the GUI](#)

## Setting Options for Signoff Design Rule Checking

Before you run the `signoff_check_drc` command, configure the run by setting the application options shown in [Table 41](#). To set the application options, use the `set_app_options` command. To see the current settings, use the `report_app_options` command.

*Table 41 Application Options for Signoff Design Rule Checking*

Application option	Default	Description
<code>signoff.check_drc.runset</code> (required)	N/A	Specifies the foundry runset to use for design rule checking.
<code>signoff.check_drc.auto_eco_threshold_value</code>	20	Specifies the maximum percentage of change to the block to perform incremental design rule checking using the <code>-auto_eco true</code> option.
<code>signoff.check_drc.excluded_cell_types</code>	{}	Specifies the cell types to exclude from checking. You can specify one or more of the following values: <code>lib_cell</code> (standard cells), <code>macro</code> (macro cells), <code>pad</code> (I/O pad cells), and <code>filler</code> (filler cells).
<code>signoff.check_drc.fill_view_data</code>	<code>read_if_upToDate</code>	Controls whether the IC Validator tool reads the fill data. By default, it reads the fill data only if its timestamp is newer than the timestamp of the design view. To read the fill data regardless of its timestamp, set this application option to <code>(read)</code> . To never read the fill data, set this application option to <code>discard</code> .
<code>signoff.check_drc.ignore_blockages_in_cells</code>	<code>true</code>	Controls whether the IC Validator tool reads only the pin information from the frame view ( <code>true</code> ) or both the blockages and the pin information ( <code>false</code> ).
<code>signoff.check_drc.ignore_child_cell_errors</code>	<code>false</code>	Controls whether the IC Validator tool reports both top-level and child-level errors to the error data file ( <code>false</code> ) or only top-level errors ( <code>true</code> ).
<code>signoff.check_drc.max_errors_per_rule</code>	1000	Specifies the maximum number of errors to report per rule.
<code>signoff.check_drc.read_design_views</code>	{}	Specifies the reference cells for which the IC Validator tool reads the design view instead of the frame view. Using the design view can expose problems that are masked by the frame view abstraction.

**Table 41 Application Options for Signoff Design Rule Checking (Continued)**

Application option	Default	Description
signoff.check_drc.read_layout_views	{}	Specifies the reference cells for which the IC Validator tool reads the layout view instead of the frame view. Using the layout view can expose problems that are masked by the frame view abstraction.
signoff.check_drc.run_dir	signoff_check_drc_run	Specifies the run directory, which contains the files generated by the signoff_check_drc command. You can specify either a relative path, in which case the directory is created under the current working directory, or an absolute path.
signoff.check_drc.user_defined_options	(none)	Specifies additional options for the IC Validator command line. The string that you specify in this option is added to the command line used to invoke the IC Validator tool. The Fusion Compiler tool does not perform any checking on the specified string.
signoff.physical.layer_map_file	(none)	Specifies the name of the layer mapping file. If the technology file and the foundry runset file used by the IC Validator tool do not use the same layer numbers, you must supply a layer mapping file to map the technology layers to the layers used in the runset file. For details about the format of the layer mapping file, see <a href="#">Defining the Layer Mapping for IC Validator In-Design Commands</a> .
signoff.physical.merge_exclude_libraries	(none)	Specifies the cell libraries whose cells are excluded from replacement with the cell data in the stream files specified in the signoff.physical.merge_stream_files application option.
signoff.physical.merge_stream_files	(none)	Specifies the stream (GDSII or OASIS) files to merge into the current block for signoff design rule checking. When you use this option, the GDSII or OASIS data replaces the cell library data for the cells defined in the specified stream files.

## Reading Blocks for Signoff Design Rule Checking

By default, the IC Validator tool reads the design view for the top-level block and library cell instances, and the pin information from the frame view for the macro cell and I/O pad cell instances.

- To read both the pin information and the routing blockages from the frame view for macro cells and I/O pad cells, set the `signoff.check_drc.ignore_blockages_in_cells` application option to `false`.
- To read the GDSII or OASIS data for specific reference cells, specify the stream files by setting the `signoff.physical.merge_stream_files` application option.

When you use this option, the GDSII or OASIS data replaces the cell library view for the cells defined in the specified stream files, except for cells in the cell libraries specified in the `signoff.physical.merge_exclude_libraries` application option.

- To read the layout view for specific reference cells, specify the cells by setting the `signoff.check_drc.read_layout_views` application option.

The layout view is identical to the GDSII or OASIS data that was used to create it, but reduces the runtime for signoff design rule checking.

**Note:**

By default, layout views are not included in the cell libraries. To save the layout views, you must set the `lib.workspace.save_layout_views` application option to `true` during library preparation, as described in the *Library Manager User Guide*.

- To read the design view for specific reference cells, specify the cells by setting the `signoff.check_drc.read_design_views` application option.

**Note:**

By default, design views are not included in the cell libraries. To save the design views, you must set the `lib.workspace.save_design_views` application option to `true` during library preparation, as described in the *Library Manager User Guide*.

The order of precedence for child cell data is

1. GDSII or OASIS data specified with the `signoff.physical.merge_stream_files` application option
2. Layout views specified with the `signoff.check_drc.read_layout_views` application option

If the IC Validator tool cannot find a layout view and the cell is specified in the `signoff.check_drc.read_design_views` application option, it reads the design view.

Otherwise, it reads the frame view. If it cannot find the frame view, the cell is ignored during design rule checking.

3. Design views specified with the `signoff.check_drc.read_design_views` application option

If the IC Validator tool cannot find a design view, it reads the frame view instead. If it cannot find the frame view, the cell is ignored during design rule checking.

#### 4. Frame views

For example, to read the design view for all blocks, use the following commands:

```
fc_shell> set_app_options \
 -name signoff.check_drc.read_design_views -value {*}
fc_shell> signoff_check_drc
```

To read the design view for the top-level block and all instances of reference cells whose name start with mc, and the frame view for all other child blocks, use the following commands:

```
fc_shell> set_app_options \
 -name signoff.check_drc.read_design_views -value {mc*}
fc_shell> signoff_check_drc
```

## Signoff Design Rule Checking

By default, signoff design rule checking has the following default behavior:

- Performs checking on all cell types (standard cells, filler cells, macro cells, and I/O pad cells)
  - To exclude certain cell types from design rule checking, set the `signoff.check_drc.excluded_cell_types` application option. Specify one or more of the following values: `lib_cell` (standard cells), `filler` (filler cells), `macro` (macro cells), and `pad` (I/O pad cells).
- Performs design rule checking on the entire block
  - To perform design rule checking only on those areas of the block that have been modified since the last run, use the `-auto_eco` option.

#### Note:

You can use this option only if you have previously run the `signoff_check_drc` command and the percentage of change to the block since that run is less than the change threshold. The default change threshold is 20 percent; to modify the change threshold, set the `signoff.check_drc.auto_eco_threshold_value` application option.

By default, the tool compares the current block to the version used for the previous `signoff_check_drc` run. To compare the current block to a different block, use the `-pre_eco_design` option to specify the comparison block. The tool gets the information required for incremental change detection from the error data file named `block_sdrc.err`, where `block` is either the current block name or the name specified by the `-pre_eco_design` option.

By default, the DRC violations reported after performing incremental signoff design rule checking represent only the DRC violations detected in the changed areas of the block. To merge the initial DRC violations with the DRC violations detected during the incremental run, set the `signoff.check_drc.merge_incremental_error_data` application option to `true` before running the `signoff_check_drc -auto_eco` command. By default, the command reads the initial DRC violations from the `signoff_check_drc.err` file. To read the initial DRC violations from a different error data file, specify the file name with the `signoff.check_drc.merge_base_error_name` application option.

- To perform design rule checking only in specific regions, use the `-coordinates` option, the `-excluded_coordinates` option, or both. You can specify one or more regions for both options. If you specify both options, the command does not check design rules in the overlapping regions.

To specify the coordinates, use the following format:

```
 {{x1 y1} {x2 y2} ... {xn yn}}
```

Note that there must be a space between each set of coordinates.

- Performs design rule checking for all rules specified in the foundry runset
  - To check only specific rules, use the `-select_rules` option. Specify the rules by specifying a matching pattern for the rule names. The rule names are specified in the COMMENT section in the runset file.
  - To prevent checking of specific rules, use the `-unselect_rules` option. Specify the rules by specifying a matching pattern for the rule names. The rule names are specified in the COMMENT section in the runset file.

#### **Note:**

You can use this option with the `-select_rules` option to customize the set of rules checked by the `signoff_check_drc` command.

For example, to restrict the `signoff_check_drc` command to route validation, select the metal layer rules and exclude the metal density rules.

- Performs design rule checking for all routing layers
  - To perform design rule checking only on specific routing layers, use the `-select_layers` option. Specify the routing layers by using the layer names from the technology file.
  - To perform design rule checking on all runset layers, including nonrouting layers, use the `-check_all_runset_layers true` option.  
You would typically use this setting to perform a quick design rule check on the entire block after you complete design rule checking on the routing layers. When you use this option, the block information comes from the design view and the validation tool checks all layers, including the nonrouting layers.
- Reports a maximum of 1000 errors for each rule, including both top-level and child-level violations
  - To override the maximum error count, set the `signoff.check_drc.max_errors_per_rule` application option.
  - To ignore child-level violations, set the `signoff.check_drc.ignore_child_cell_errors` application option to `true`.

The `signoff_check_drc` command uses the options that you specify to generate the command used to invoke signoff design rule checking in the IC Validator tool. You can specify additional options for the IC Validator command line by setting the `signoff.check_drc.user_defined_options` application option. The string that you specify in this option is added to the command line used to invoke the IC Validator tool. The Fusion Compiler tool does not perform any checking on the specified string.

### **Generating Input for the Automatic Fixing Flow**

If you plan to use the signoff DRC results as input to the automatic fixing flow, set the `signoff.check_drc.ignore_child_cell_errors` application option to `true` before you run the `signoff_check_drc` command. Zroute can fix only top-level violations; using this option ensures that signoff design rule checking reports only fixable violations.

### **Signoff DRC Results Files**

The `signoff_check_drc` command writes its output files to the run directory, which is specified by the `signoff.check_drc.run_dir` application option (or the `signoff_check_drc_run` directory if you do not use this option). The following files are written to the run directory:

- The error data file

By default, the error data generated by the `signoff_check_drc` command is saved in a file named `signoff_check_drc.err`, which is stored in the design library. To specify the name for the error data file, use the `-error_data` option. The error data shows child-

level errors on only one of the cell instances, which makes it easier to identify lower-level errors.

You can use the error data file to report or display the DRC violations in the error browser. For information about using the error browser, see Using the Error Browser.

- The IC Validator results files

You can use the following IC Validator results files for debugging the signoff DRC results:

- *block.LAYOUT\_ERRORS*

This file contains details about the errors detected during the `signoff_check_drc` run.

- *block.RESULTS*

This file contains a summary of the `signoff_check_drc` run results.

- *signoff\_check\_drc.log*

This file contains a summary of the `signoff_check_drc` run environment.

- *icv\_config\_out*

This file contains the paths to the top-level block and the cell libraries.

- *icv\_sdrc.conclude*

This file contains an error summary report.

- *layer.map*

This file contains the layer mapping file generated by the `signoff_check_drc` command.

- *signoff\_check\_drc.rc*

This file contains the IC Validator runset environment variables.

- *./run\_details* directory

This directory contains all the data generated by the IC Validator tool for the signoff DRC run.

For more information about these files, see the IC Validator documentation.

---

## Viewing the Violations in an ICV Heat Map

An ICV heat map displays the density of signoff DRC violations in a block. You can use the heat map to locate problematic areas in your design.

**Note:**

This feature requires version P-2019.06 or later of the IC Validator tool and an IC Validator NXT license.

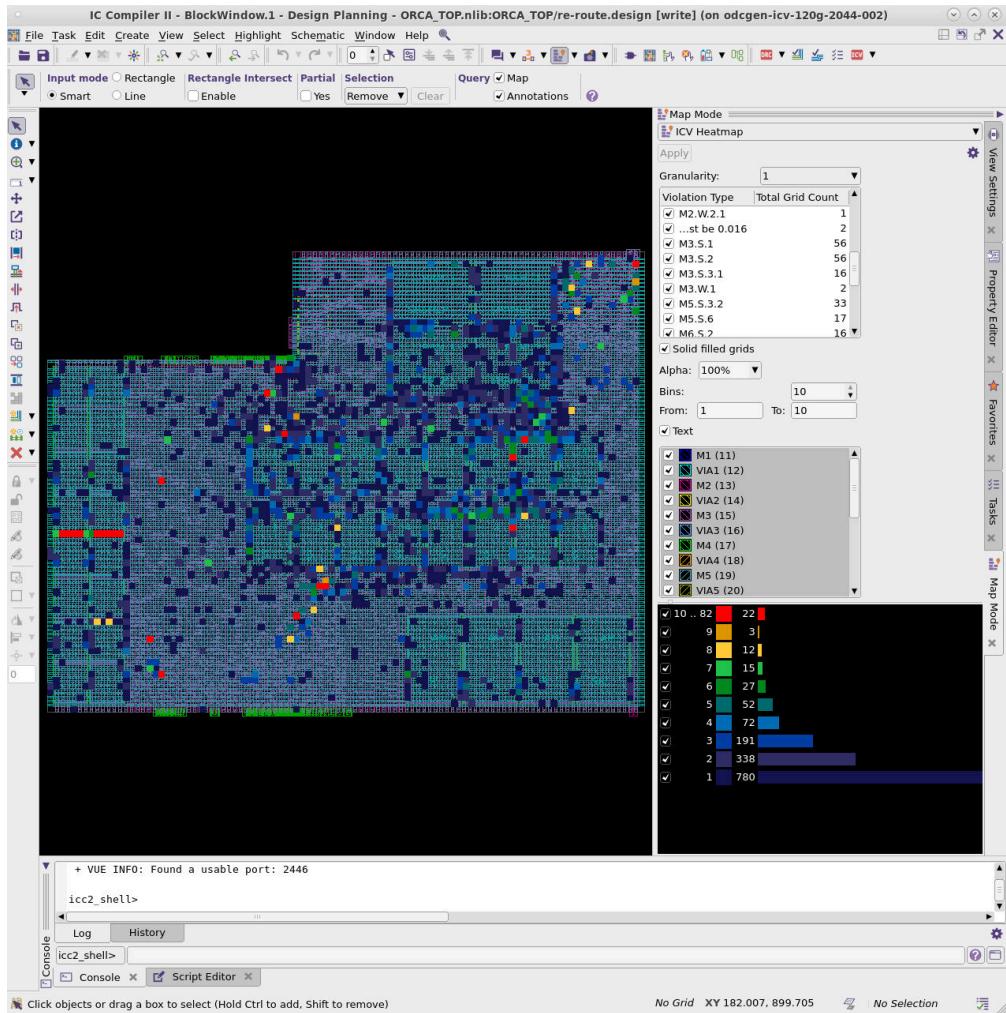
To view the ICV heat map,

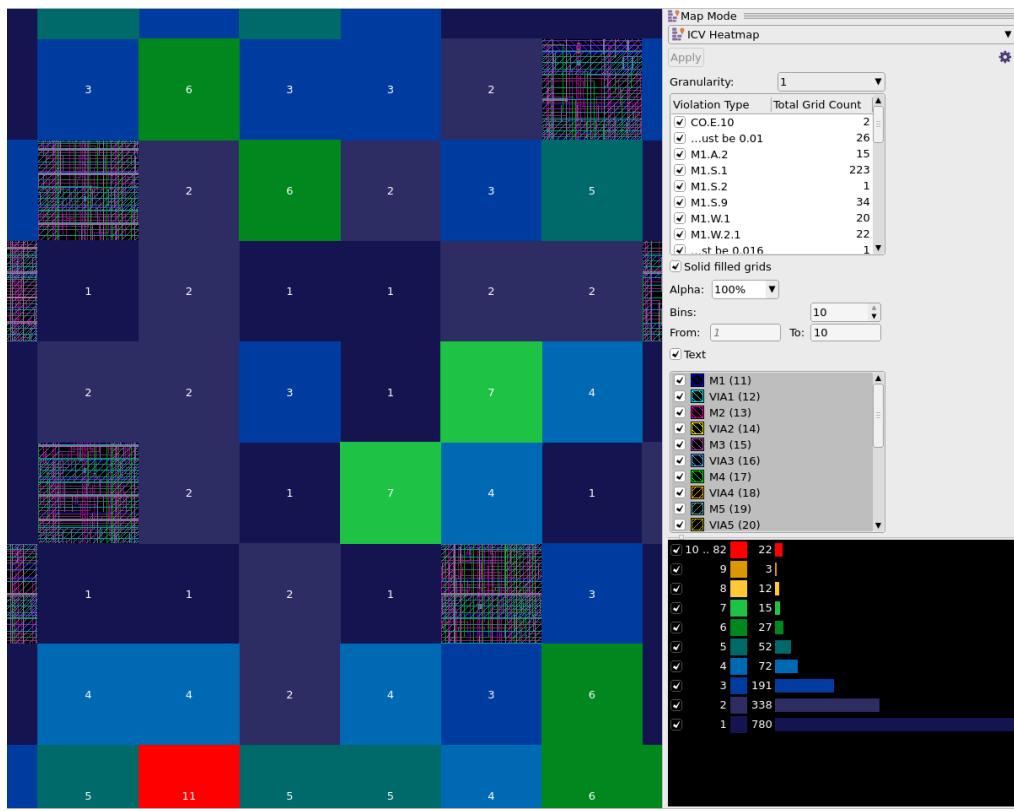
1. Enable the ICV heat map by setting the `signoff.check_drc.enable_icv_explorer_mode` application option to `true`.  
`fc_shell> set_app_options -name  
signoff.check_drc.enable_icv_explorer_mode \  
-value true`
2. Perform signoff design rule checking as described in [Running the signoff\\_check\\_drc Command](#).
3. In the GUI, choose View > Map > ICV Heatmap.

[Figure 120](#) shows an ICV heat map.

Chapter 8: IC Validator In-Design  
Performing Signoff Design Rule Checking

Figure 120 ICV Heat Map





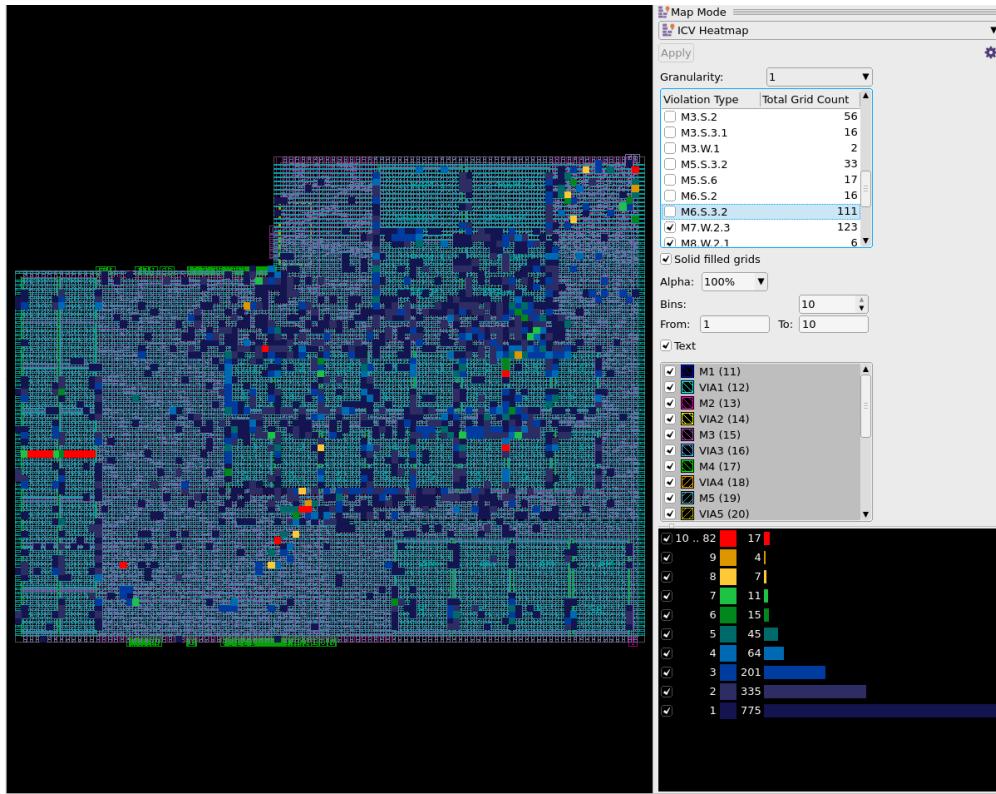
## Configuring an ICV Heat Map

You can select or unselect violation types from the heat map, as shown in figure [Figure 121](#) and [Figure 122](#).

*Figure 121 Violation Type*

Violation Type	Total Grid Count
<input checked="" type="checkbox"/> M3.S.2	56
<input checked="" type="checkbox"/> M3.S.3.1	16
<input checked="" type="checkbox"/> M3.W.1	2
<input checked="" type="checkbox"/> M5.S.3.2	33
<input checked="" type="checkbox"/> M5.S.6	17
<input checked="" type="checkbox"/> M6.S.2	16
<input checked="" type="checkbox"/> M6.S.3.2	111
<input checked="" type="checkbox"/> M7.W.2.3	123
<input checked="" type="checkbox"/> M8.W.2.1	6

Figure 122 Violation Type in ICV Heat Map



To control the opacity of the heat map squares, adjust the Alpha percentage value. Set this percentage to 100% to create a completely filled block. The lower you set the percentage, the more transparent the block becomes, as shown in [Figure 123](#) and [Figure 124](#).

Figure 123 Completely Filled Block

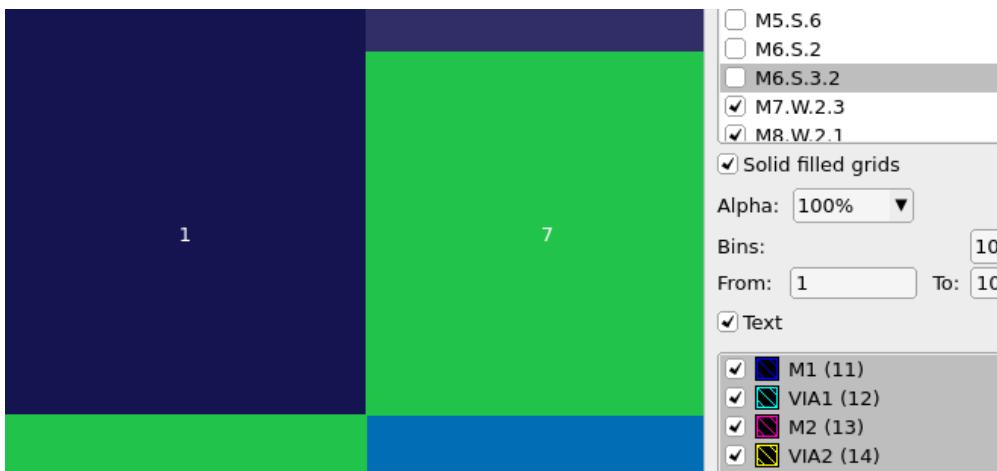
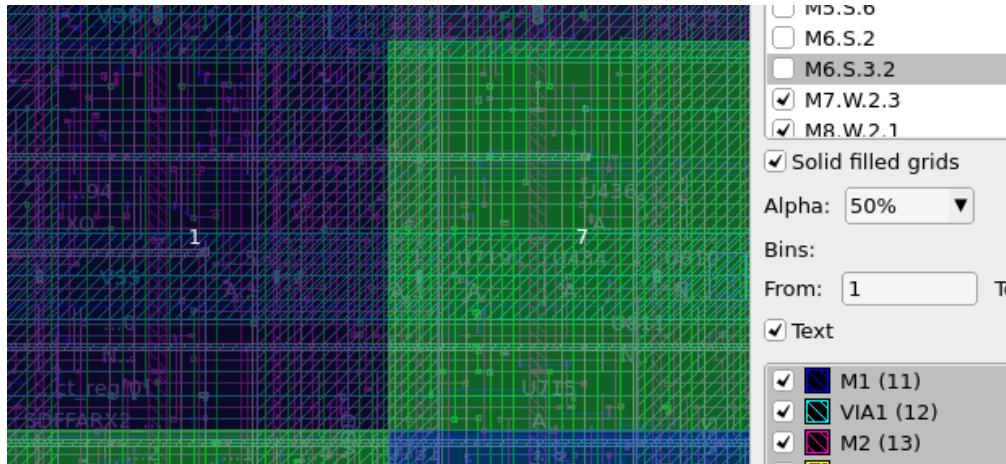


Figure 124



You can adjust the range of colored squares by modifying the Bins value, as shown in Figure 125 and Figure 126.

Figure 125 Bins Value Set to 5

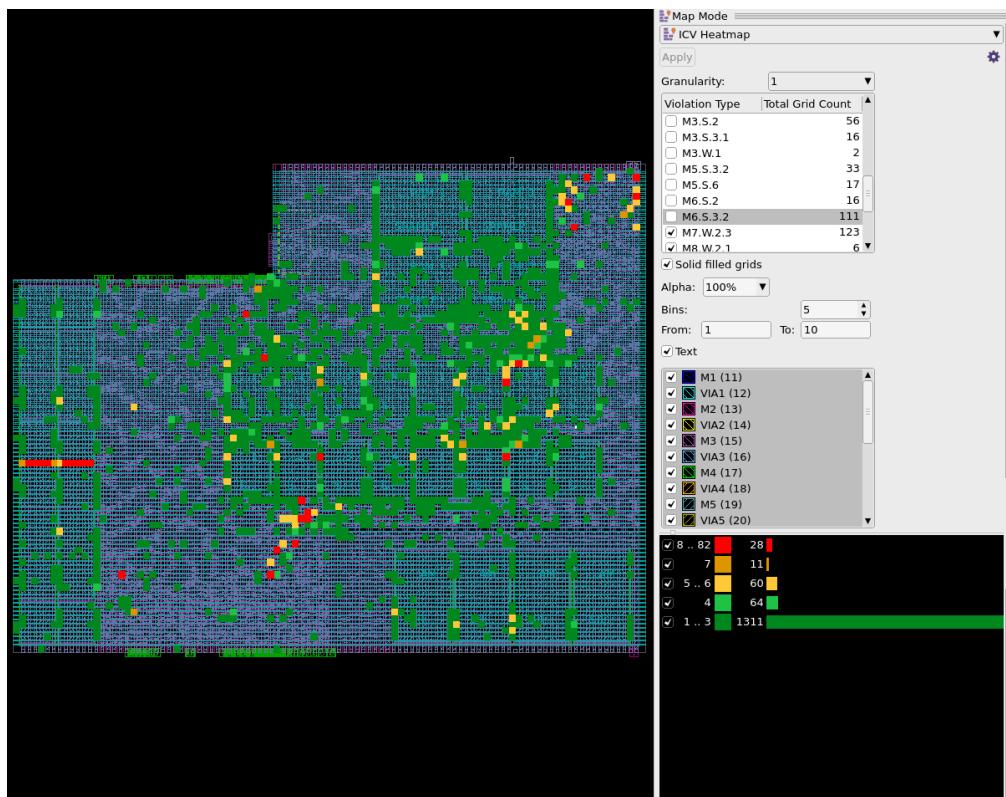
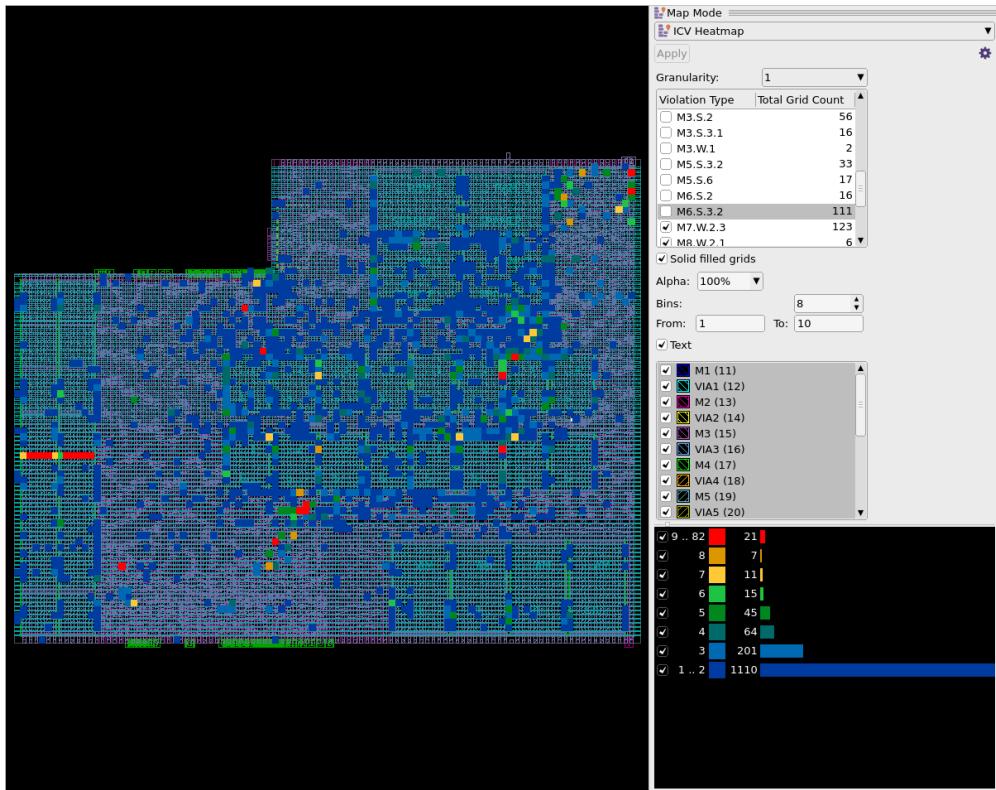


Figure 126 Bins Value Set to 8

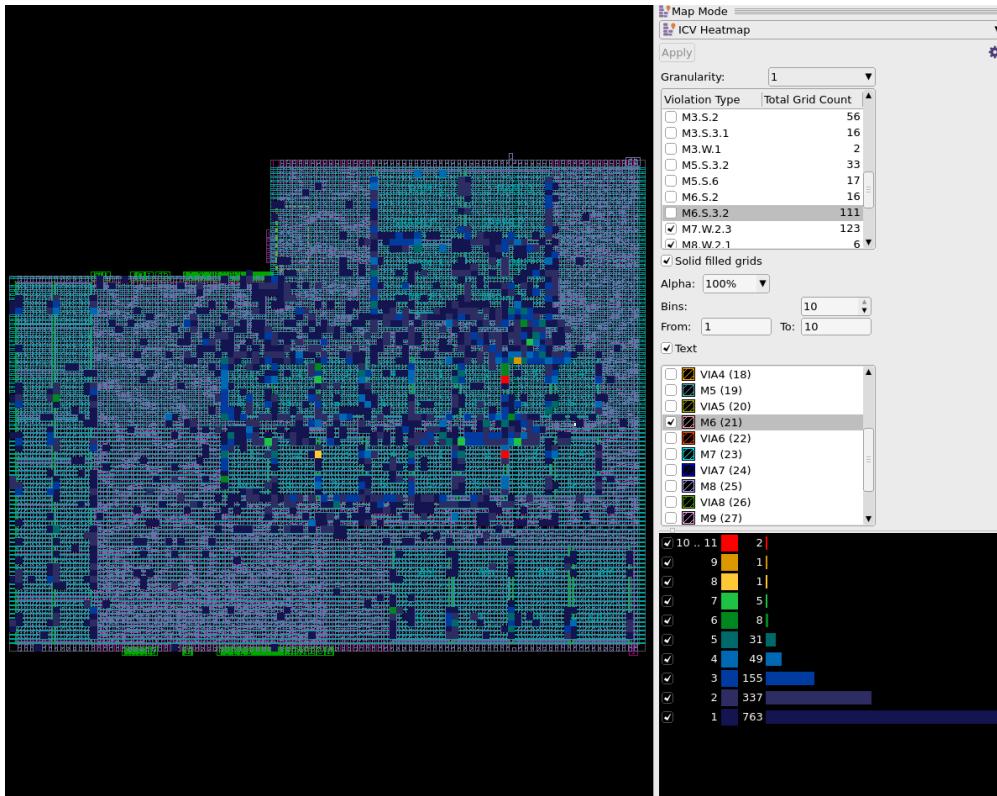


You can select or unselect layers from the heat map, as shown in [Figure 127](#) and [Figure 128](#).

Figure 127 Selecting Layers



Figure 128 Selecting Layers in the ICV Heatmap



You can modify the squares of the heat map by right-clicking the colored squares and selecting Set Style. See [Figure 129](#) and [Figure 130](#).

Figure 129 Set Style

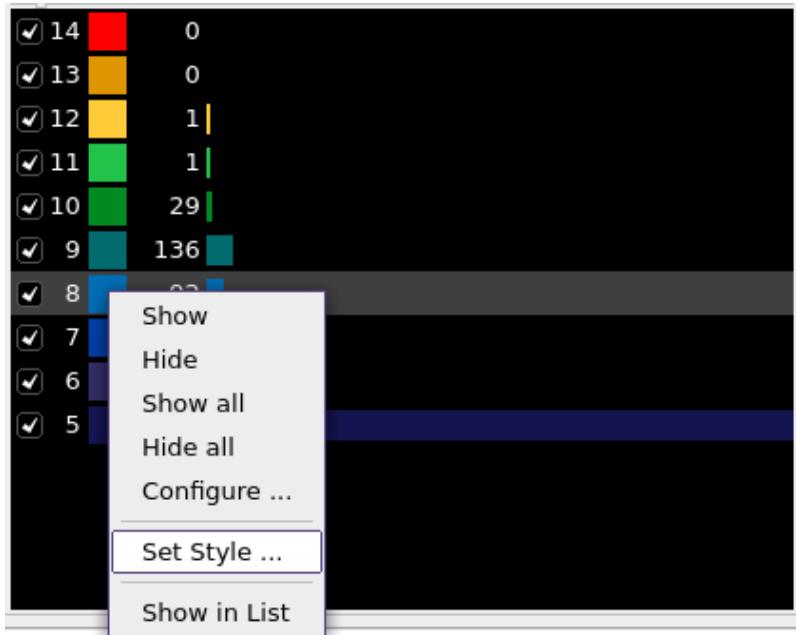
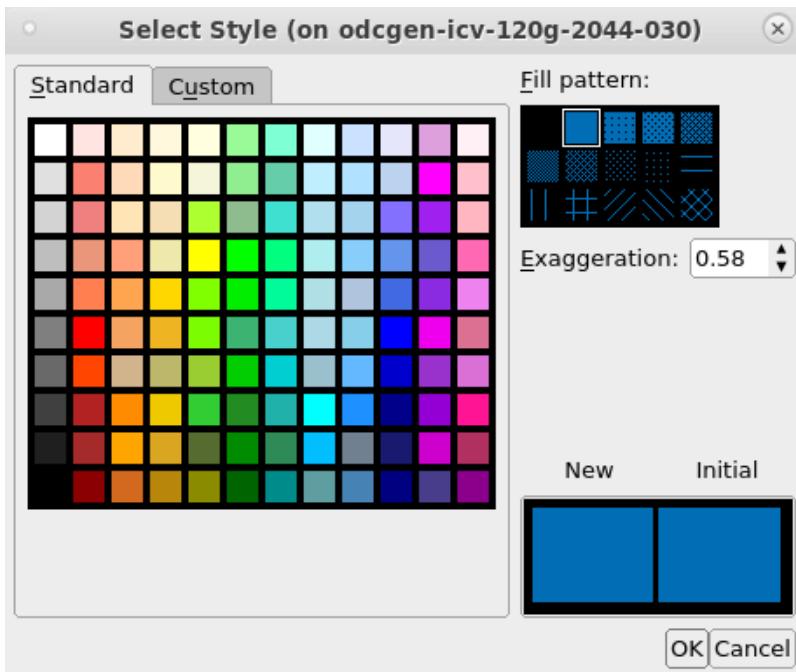


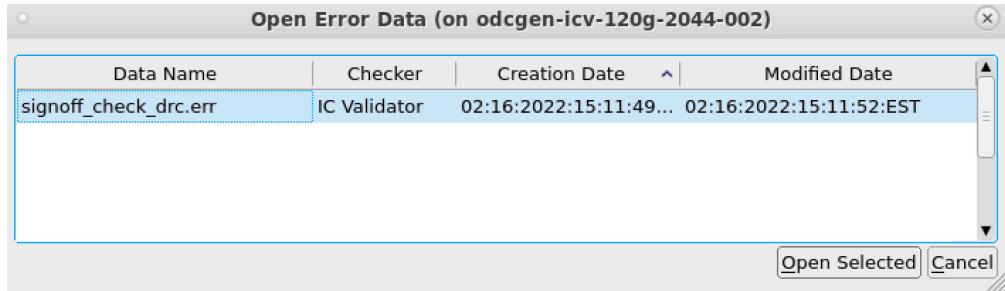
Figure 130 Select Style



## Highlighting Violations From the Error Browser Onto an ICV Heat Map

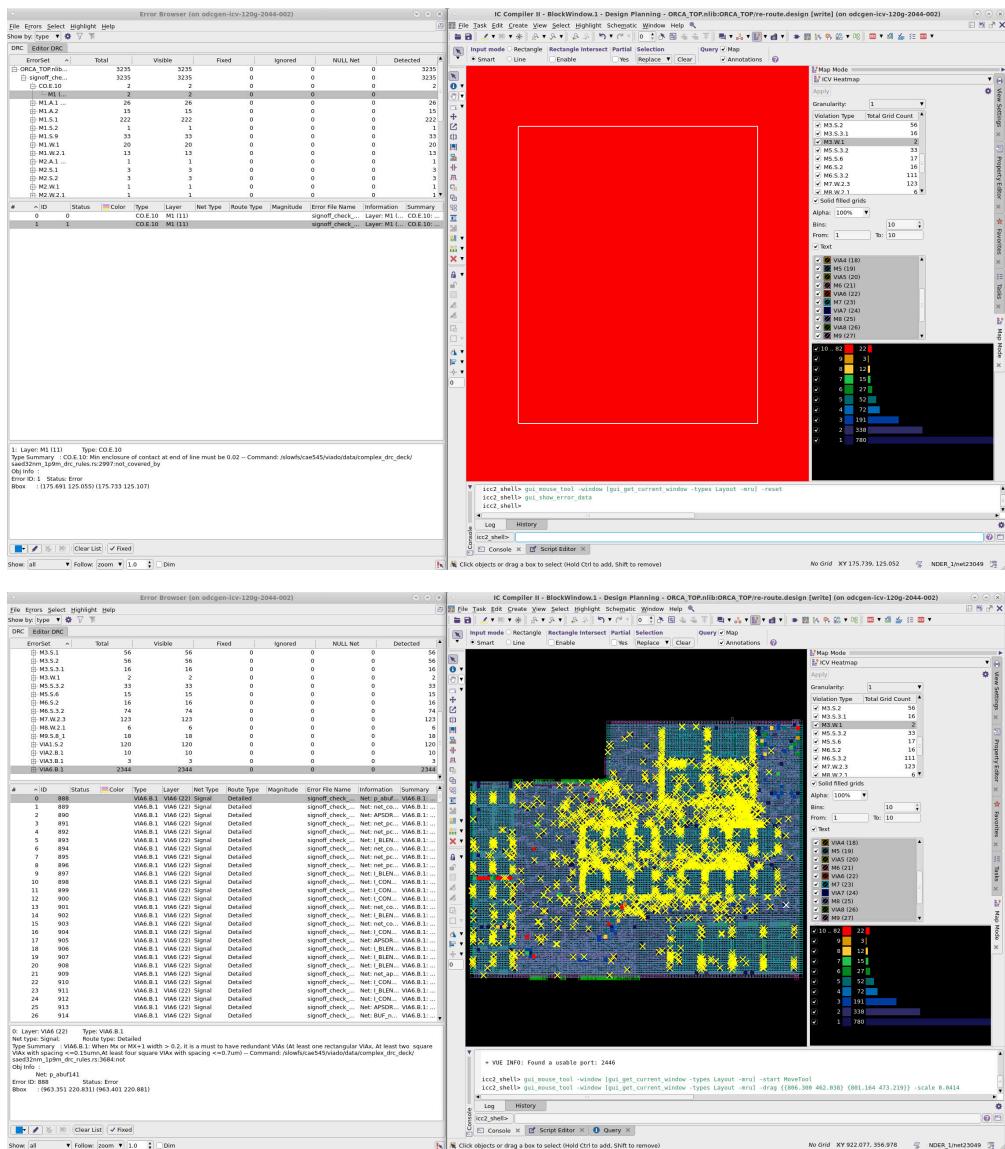
To highlight violations from the Error Browser:

1. In the GUI, choose View > Error Browser.
2. Select the error data generated from the DRC run.



3. Select violations from the Error Browser to automatically highlight them on the heat map.

## Chapter 8: IC Validator In-Design Performing Signoff Design Rule Checking



## Automatically Fixing Signoff DRC Violations

Zroute can use the signoff DRC results generated by the `signoff_check_drc` command to automatically fix the detected design rule violations.

To perform automatic fixing of the signoff DRC violations,

1. Set up the IC Validator environment as described in [Setting Up the IC Validator Environment](#).
2. Set up the physical signoff options as described in [Setting Options for Signoff Design Rule Checking](#).
3. (Optional) Generate a configuration file that aids in the fixing process.

The configuration file defines the layer-to-DRC-rule mapping. The `signoff_fix_drc` command processes only those rules that are specified in the configuration file.

By default, the `signoff_fix_drc` command automatically generates the configuration. You can manually generate a configuration file, as described in [Creating an Autofix Configuration File](#).

4. Set the application options for signoff DRC fixing.

For information about the options for signoff DRC fixing, see [Setting Options for Signoff Design Rule Checking](#).

5. Save the block to disk.

When you run the `signoff_fix_drc` command, the IC Validator tool uses the on-disk information for the block, not the information in memory. To ensure accurate information, use the `save_block` command to save the current state of the block before running the `signoff_fix_drc` command.

6. Run signoff DRC fixing by using the `signoff_fix_drc` command as described in [Running the signoff\\_fix\\_drc Command](#).

If your block uses double-patterning technology, first perform signoff DRC fixing for all other routing design rules, and then perform signoff DRC fixing for only the double-patterning rules, as described in [Automatically Fixing Double-Patterning Odd-Cycle Violations](#).

## Creating an Autofix Configuration File

The autofix configuration file has the following format:

```
"mask_layer_name" "full_IC_Validator_DRC_rule_comment"
```

You can put comments in the configuration file by starting the line with the pound sign (#).

If you have already run the `signoff_check_drc` command, you can generate a configuration file by running the `$ICV_HOME_DIR/contrib/generate_layer_rule_map.pl` script. This script has the following syntax:

```
generate_layer_rule_map.pl
 -dplog IC_Validator_log_file
 -tech_file technology_file
 -o config_file_name
```

The IC Validator log file that you specify in the `-dplog` option is the log file that was generated by the previous `signoff_check_drc` run. This file is called `runset.dp.log` and is located in the `run_details` subdirectory of the `signoff_drc_run` directory (or the directory specified by the `signoff.check_drc.run_dir` application option).

To specify the location of the configuration file you created, use the `signoff.fix_drc.config_file` application option.

## Setting Options for Signoff DRC Fixing

Before you run the `signoff_fix_drc` command, configure the run by setting the application options shown in [Table 42](#). To set the application options, use the `set_app_options` command. To see the current settings, use the `report_app_options` command.

*Table 42 Application Options for Signoff DRC Fixing*

Application option	Default	Description
<code>signoff.fix_drc.advanced_guidance_for_rules</code>	<code>all</code>	Controls whether signoff DRC fixing uses advanced routing guidance. By default, the <code>signoff_fix_drc</code> command uses advanced routing guidance ( <code>all</code> ). To disable this feature, set the application option to <code>off</code> .
<code>signoff.fix_drc.check_drc</code>	<code>local</code>	Controls whether signoff DRC checking is performed only on those portions of the block affected by signoff DRC fixing ( <code>local</code> , which is the default) or on the entire block ( <code>global</code> ).
<code>signoff.fix_drc.custom_guidance</code>	(none)	Enables a signoff DRC fixing run that targets double-patterning odd-cycle violations when it is set to <code>dpt</code> . For information about fixing double-patterning odd-cycle violations, see <a href="#">Automatically Fixing Double-Patterning Odd-Cycle Violations</a> .

**Table 42 Application Options for Signoff DRC Fixing (Continued)**

Application option	Default	Description
signoff.fix_drc. fix_detail_route_drc	global	Controls whether the <code>signoff_fix_drc</code> command fixes routing DRC violations for the entire block ( <code>global</code> ) or only in the areas near the signoff DRC violations ( <code>local</code> ).
signoff.fix_drc. init_drc_error_db	(none)	Specifies the location of the signoff DRC results from the previous run. By default, the <code>signoff_fix_drc</code> command runs the <code>signoff_check_drc</code> command to generate the initial signoff DRC results.
signoff.fix_drc. last_run_full_chip	false	Controls whether the signoff DRC checking in the final repair loop runs only on those portions of the block affected by signoff DRC fixing ( <code>false</code> ) or on the whole block ( <code>true</code> ).
signoff.fix_drc. max_detail_route_iterations	5	Specifies the maximum number of search and repair loops to run after signoff DRC fixing.
signoff.fix_drc. max_errors_per_rule	1000	Specifies the maximum number of DRC violations to fix per rule.
signoff.fix_drc.run_dir	signoff_fix_drc_run	Specifies the run directory, which contains the files generated by the <code>signoff_fix_drc</code> command. You can specify either a relative path, in which case the directory is created under the current working directory, or an absolute path.
signoff.fix_drc. target_clock_nets	false	Controls whether signoff DRC fixing targets data nets ( <code>false</code> ) or clock nets ( <code>true</code> ).
signoff.fix_drc. user_defined_options	(none)	Specifies additional options for the IC Validator command line. The string that you specify in this option is added to the command line used to invoke the IC Validator tool. The Fusion Compiler tool does not perform any checking on the specified string.

## Running the signoff\_fix\_drc Command

By default, the `signoff_fix_drc` command performs the following tasks:

1. Runs the `signoff_check_drc` command to generate the initial signoff DRC results, as described in [Checking for DRC Violations](#)

If you have already run the `signoff_check_drc` command, you can use the existing signoff DRC results instead of rerunning signoff design rule checking. To reuse the existing results, specify the directory that contains the results by setting the `signoff.fix_drc.init_drc_error_db` application option. You can specify either a relative path or an absolute path. If you specify a relative path, it is relative to the current working directory.

For example, to use the results from the default `signoff_check_drc` run directory, use the following commands:

```
fc_shell> set_app_options \
 -name signoff.fix_drc.init_drc_error_db \
 -value signoff_check_drc_run
fc_shell> signoff_fix_drc
```

**Note:**

To use existing signoff DRC results as input to the `signoff_fix_drc` command, you must use the signoff DRC application options described in [Generating Input for the Automatic Fixing Flow](#).

2. Runs two repair loops to fix the DRC violations

In each repair loop, the `signoff_fix_drc` command performs the following tasks:

- a. Tries to fix the DRC violations detected in the previous IC Validator signoff DRC run, as described in [Fixing DRC Violations](#)
- b. Runs IC Validator signoff design rule checking on the modified block, as described in [Checking for DRC Violations](#)

To change the number of repair loops, use the `-max_number_repair_loop` option; you can specify an integer between 1 and 10. If no signoff DRC violations remain after a loop, the tool does not perform additional repair loops.

As the repair loop number increases, so does the physical scope of the rerouting performed by that repair loop. To change the scope, increase the number of the initial repair loop by using the `-start_repair_loop` option. Note that increasing the initial loop number might cause a greater disturbance to the block, which would require additional design rule checking and fixing by Zroute.

3. Writes a summary report of the results

For information about the summary report, see [Summary Report for Automatic Design Rule Fixing](#).

### Fixing DRC Violations

In each repair loop, the `signoff_fix_drc` command tries to fix the DRC violations detected in the previous signoff DRC run.

- For the first loop, the tool uses the IC Validator data in the directory specified by the `signoff.fix_drc.init_drc_error_db` application option (or the `signoff_drc_run_init` directory if you do not use this option).
- For successive loops, the tool uses the IC Validator DRC data from the previous loop, which is stored in the directory specified by the `signoff.fix_drc.run_dir` application option (or the `signoff_fix_drc_run` directory if you do not use this option).

When fixing DRC violations, the `signoff_fix_drc` command has the following default behavior:

- Tries to fix all design rule violations detected in the previous signoff DRC run except those design rules that have more than 1000 violations
- Performs DRC fixing on data nets for the whole block

To modify the regions for design rule fixing, use one or both of the following options:

- `-coordinates`

This option restricts design rule fixing to the specified regions.

- `-excluded_coordinates`

This option prevents design rule fixing in the specified regions. If you specify this option with the `-coordinates` option, the command does not fix design rules in the overlapping regions.

To specify the coordinates, use the following format:

```
{ {x1 y1} {x2 y2} ... {xn yn} }
```

Note that there must be a space between each set of coordinates.

To prevent fixes that might impact timing-critical nets, use one or both of the following options:

- -nets

This option explicitly specifies the critical nets.

- -timing\_preserve\_setup\_slack\_threshold

This option enables the tool to automatically determine the critical nets based on a setup slack threshold. The unit for the slack threshold setting is the library time unit.

- Performs five iterations of detail routing after DRC fixing to fix any DRC violations introduced during DRC fixing
- Saves the modified block in a design view named *block\_ADR\_#*

To change the default behavior, use the following options:

- signoff.fix\_drc.max\_errors\_per\_rule

This application option specifies the error threshold for ignoring a design rule.

- signoff.fix\_drc.fix\_detail\_route\_drc

This application option controls the scope of DRC fixing. To reduce runtime by performing DRC fixing only in areas near the signoff DRC violations, set this application option to `local`.

- signoff.fix\_drc.target\_clock\_nets

This application option controls whether DRC fixing targets data nets or clock nets. To target clock nets, set this application option to `true`.

- signoff.fix\_drc.max\_detail\_route\_iterations

This application option specifies the number of detail routing iterations; you can specify an integer between 0 and 1000.

## Checking for DRC Violations

When performing signoff design rule checking, the `signoff_fix_drc` command has the following default behavior:

- Reads the design view for the top-level block and library cell instances, and the pin information from the frame view for the macro cell and I/O pad cell instances

For information about changing the view used for specific cells, see [Reading Blocks for Signoff Design Rule Checking](#).

- After the initial run, performs design rule checking only on those portions of the design affected by design rule fixing

To perform signoff design rule checking on the entire block for the last repair loop, set the `signoff.fix_drc.last_run_full_chip` application option to `true`. Setting this option to `true` increases accuracy but also increases runtime.

To perform signoff design rule checking on the entire block for all repair loops, set the `signoff.fix_drc.check_drc` application option to `global`. Setting this option to `global` increases accuracy but can greatly increase runtime.

- Does not check for violations in child blocks
- Excludes rules that are not suitable for automatic design rule fixing

To explicitly specify the design rules checked by the IC Validator tool, set the `-select_rules rules` and `-unselect_rules rules` application options.

- Stores the IC Validator results in the `signoff_fix_drc_run` directory

To use a different run directory, set the `signoff.fix_drc.run_dir` application option.

**Note:**

The IC Validator results for the initial signoff DRC run are stored in the `signoff_drc_run_init` directory.

In addition, you can specify options to add to the IC Validator command line by setting the `signoff.fix_drc.user_defined_options` application option.

**See Also**

- [Performing Signoff Design Rule Checking](#)

## Automatically Fixing Double-Patterning Odd-Cycle Violations

If your block uses double-patterning technology, you must perform separate signoff DRC fixing runs for the non-double-patterning routing rules and the double-patterning routing rules.

To perform automatic fixing of the signoff DRC violations for a block that uses double-patterning technology,

1. Perform automatic fixing for the non-double-patterning signoff DRC violations by using the process described in [Automatically Fixing Signoff DRC Violations](#).

When you run the `signoff_check_drc` and `signoff_fix_drc` commands, use the `-unselect_rules` option to ignore the double-patterning rules during signoff design rule checking.

```
fc_shell> signoff_check_drc -unselect_rules {list_of_dpt_rules}
```

**Note:**

To determine the double-patterning rules for your technology, see the design rule manual (DRM) provided by your vendor.

2. Perform automatic fixing for the double-patterning odd-cycle violations by using the process described in [Automatically Fixing Signoff DRC Violations](#).

Before you run the `signoff_fix_drc` command, set the `signoff.fix_drc.custom_guidance` application option to `dpt`.

```
fc_shell> set_app_options \
 -name signoff.fix_drc.custom_guidance -value dpt
```

When you run the `signoff_check_drc` and `signoff_fix_drc` commands, use the `-select_rules` option to consider only the double-patterning rules during signoff design rule checking.

```
fc_shell> signoff_check_drc -select_rules {list_of_dpt_rules}
```

3. Perform signoff design rule checking to verify the results by using the process described in [Performing Signoff Design Rule Checking](#).

## Summary Report for Automatic Design Rule Fixing

The `signoff_fix_drc` command writes a summary report file, `result_summary.rpt`, to the current working directory. [Example 29](#) shows an example of this report.

**Example 29 result\_summary.rpt Example**

```
Input DRC Error Database..... ./inputs/signoff_check_drc_run
Limit to Cell..... my_design
Maximum Errors/Command..... 1000
```

## Chapter 8: IC Validator In-Design Performing Signoff Design Rule Checking

```
SIGNOFF AUTOFIX DRC SUMMARY:
 : : : : REMAINING :
 : TOTAL : PROCESSED : TARGETED : TARGETED : FIX :
DRC NAME : DRC : DRC : DRC : DRC : RATE : COMMENT
M0.W.3 :
Maximum width : 1 : 1 : 0 : 0 : -- : 1
 ignored: <#1>
M4.L.3 :
Edge length wi : 4 : 4 : 0 : 0 : -- : 4 ignor
 ed: <#2>
M4.L.5:2 :
Edge length : 2 : 2 : 0 : 0 : -- : 2
 ignored: <#2>
M4.W.3 :
Maximum width : 1 : 1 : 0 : 0 : -- : 1
 ignored: <#1>
VIA1.S.10.3 :
Space of : 10 : 10 : 2 : 0 : 100.0% : 8
 ignored: <#2>
TOTAL : : : 2 : 0 : 100.0% :
```

**NOTE:**

Reasons : Detail Info

#1 : size of error shapes greater than threshold

#2 : shapes associated with error are not Zroute modifiable

The following table describes each of the columns in the summary report:

**Table 43      Definitions of Columns in result\_summary.rpt**

Column heading	Description
DRC NAME	Name of the design rule from the runset
TOTAL DRC	Total number of DRC violations detected by the signoff_check_drc run
PROCESSED DRC	Total number of the detected DRC violations that went through the filtering process to determine whether they can be targeted
TARGETED DRC	Number of DRC violations that automatic DRC fixing will attempt to fix after the filtering process
REMAINING TARGETED DRC	Number of DRC violations that remain after automatic DRC fixing
FIX RATE	The ratio of remaining targeted DRC violations to targeted DRC violations
COMMENT	Explanation for ignored DRC violations; the detailed reason codes are provided in the NOTES section of the report

## Checking Signoff Design Rules Interactively in the GUI

The Live DRC feature performs interactive signoff design rule checking in the Fusion Compiler GUI.

**Note:**

This feature requires version P-2019.06 or later of the IC Validator tool.

To perform interactive signoff design rule checking using Live DRC,

1. Set up the IC Validator environment as described in [Setting Up the IC Validator Environment](#).
2. Set the application options for interactive signoff design rule checking.

At a minimum, you must specify the following information:

- The foundry runset to use for design rule checking by setting the `signoff.check_drc_live.runset` application option
- The layer mapping file that maps the technology file layers to the runset layers by setting the `signoff.physical.layer_map_file` application option

For information about the options for interactive signoff design rule checking, see [Setting Options for Interactive Design Rule Checking](#).

3. Ensure that the layout window displays the objects on which you want to perform design rule checking.

By default, interactive design rule checking uses the frame view of each cell. If you want more detail than what is provided in the frame view, use the GDSII or OASIS view for specific cells by specifying the files containing the cells in the `signoff.physical.merge_stream_files` application option. For example,

```
fc_shell> set_app_options \
 -name signoff.physical.merge_stream_files \
 -value {stdcell.gds macro.oas}
```

For information about controlling the objects displayed for design rule checking, see [Displaying Objects for Design Rule Checking](#).

4. Display the DRC toolbar by right-clicking in the GUI menu bar and selecting DRC Tools.

For details about the icons in the DRC toolbar, see [DRC Toolbar](#).

5. Configure the rules to use for design rule checking by choosing Edit > Options > ICV-Live or by clicking the gear icon ( ) in the DRC toolbar.

To exclude all density or connectivity rules from design rule checking, set the `signoff.check_drc_live.exclude_command_class` application option.

6. Run design rule checking by choosing Edit > ICV > Run ICV-Live on Current View or by clicking the ICV Run button () in the DRC toolbar.

The first time that you run Live DRC, the tool performs the following initialization steps, which can take a couple of minutes:

- Processes the IC Validator runset to get the list of available rules and caches the runset
- Processes the GDSII or OASIS files specified in the `signoff.physical.merge_stream_files` application option

Unless the layers, rules, or stream files change, subsequent runs do not perform this caching and therefore are much quicker. To reduce the initial runtime, you can perform this initialization before running Live DRC by using the `signoff_init_live_drc` command.

The tool displays the design rule checking results in the error browser. You can select errors in the error browser and then fix them interactively. For information about using the error browser, see [Using the Error Browser](#)

### See Also

- [Running the signoff\\_check\\_drc Command](#)

## Displaying Objects for Design Rule Checking

When you run interactive design rule checking, the IC Validator tool performs design rule checking on what is displayed in the layout window, plus an extension of 1 micron. You can modify the size of the extension by setting the `signoff.check_drc_live.halo` application option.

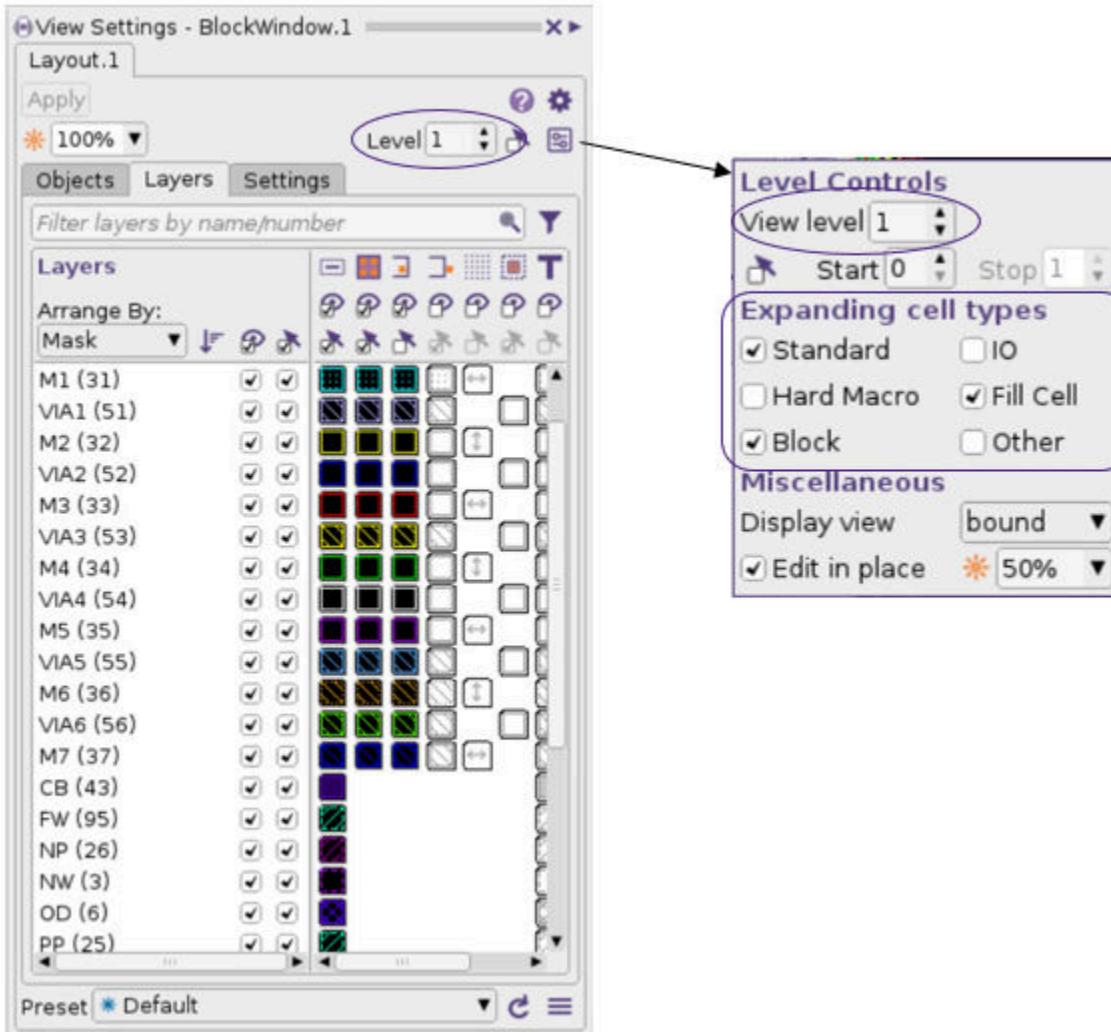
Use the View Settings and Hierarchy Settings panels to control the layers, levels, and cell types displayed in the layout window.

- Display the View Settings panel by clicking the View Settings icon () in the panel area.
  - Specify the number of levels to display in the Level field.
  - Select the Layers tab and enable visibility for the desired layers.
- Display the Hierarchy Settings panel by clicking the Hierarchy Settings icon () in the View Settings panel or the menu bar.

- Specify the number of view levels in the “View level” field.
- Select Standard, “Fill Cell,” and Block in the “Expanding cell types” section.

[Figure 131](#) shows the View Settings and Hierarchy Settings panels used to control the layout window display for design rule checking.

*Figure 131 Settings Used to Display Objects for Design Rule Checking*



## DRC Toolbar

The DRC toolbar provides buttons that you can use to configure and run interactive design rule checking with Live DRC, and to view the reported DRC violations in the layout view.

[Figure 132](#) shows the DRC toolbar.

Figure 132 DRC Toolbar



Table 44 DRC Toolbar Buttons

Button	Description
	Runs design rule checking using Live DRC.
	Opens the dialog box to view and sets the Live DRC application options.
	Opens the dialog box to configure the rules to use for design rule checking .
	Selects the region in which to perform design rule checking.
	Resets the design rule checking region to that used in the previous Live DRC run.
	Enables or disables the error environment overlay of GDSII or OASIS shapes from lower level cells that contribute to DRC errors. The error environment overlay is available only when the <code>signoff.physical.merge_stream_files</code> application option is set before doing DRC checking.
	Clears or highlights all DRC violations detected by Live DRC.
	Iterates through the DRC violations for the current rule from Live DRC.
	Controls zoom or pan when iterating through the DRC violations.
19 of 27 GRDFIRST.S.MD.1 ▾	Shows the design rule being highlighted.

## Setting Options for Interactive Design Rule Checking

Before you run interactive design rule checking, configure the run by setting the application options shown in [Table 45](#). To set the application options, use the `set_app_options` command. To see the current settings, use the `report_app_options` command.

*Table 45 Application Options for Interactive Design Rule Checking*

Application option	Default	Description
<code>signoff.check_drc_live.check_only_visible_layers</code>	<code>all_layers</code>	Controls whether Live DRC includes the rules defined on non-displayed layers. By default, Live DRC includes the rules for all layers. Valid values are <code>all_layers</code> , <code>visible_involved</code> , and <code>visible_only</code> .
<code>signoff.check_drc_live.default_layers</code>	<code>(none)</code>	Specifies the layers with color masks to write into the default layers for Live DRC.
<code>signoff.check_drc_live.exclude_command_class</code>	<code>{ {density true} {connectivity true}}</code>	Specifies the types of rules to exclude from DRC checking by Live DRC. By default, the density and connectivity rules are excluded.
<code>signoff.check_drc_live.halo</code>		Specifies the distance in microns by which to expand the region in which Live DRC performs checking.
<code>signoff.check_drc_live.hierarchy_level</code>	<code>visible</code>	Specifies the maximum hierarchical depth processed by Live DRC..
<code>signoff.check_drc_live.keep_enclosing_results</code>	<code>false</code>	Specifies whether to keep DRC violations that are outside of the checking region. By default, Live DRC keeps only those violations that are fully within the checking region.  If you set this application option to <code>true</code> , Live DRC keeps the DRC violations that are fully inside the trim region. The trim region is the checking region plus half of the distance specified by the <code>signoff.check_drc_live.halo</code> application option.  Violations that are fully outside the trim region are always discarded.

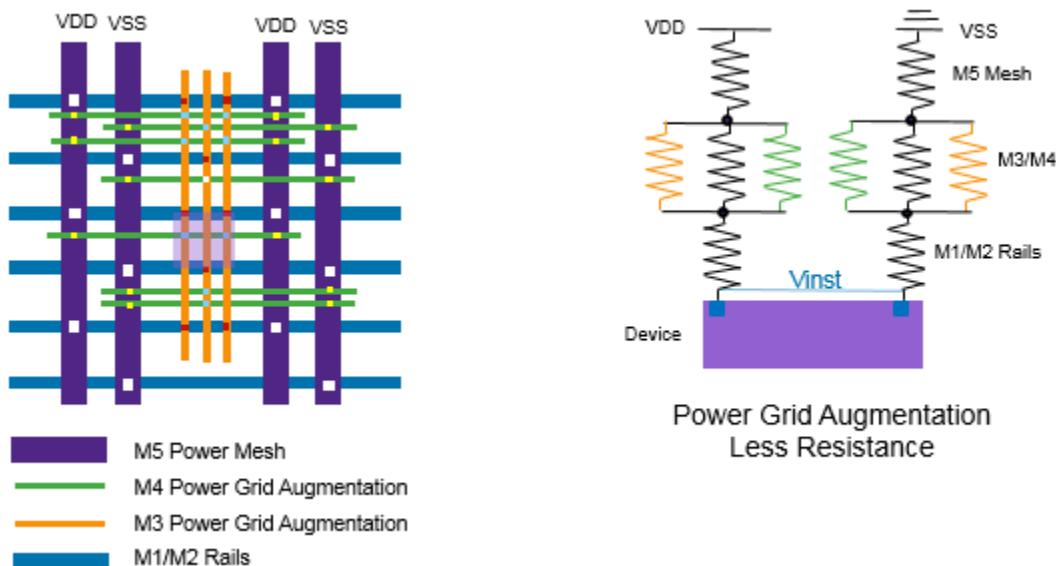
**Table 45 Application Options for Interactive Design Rule Checking (Continued)**

Application option	Default	Description
signoff.check_drc_live. run_dir	signoff_check_drc_live_run	Specifies the run directory, which contains the files generated by the interactive signoff design rule checking. You can specify either a relative path, in which case the directory is created under the current working directory, or an absolute path.
signoff.check_drc_live. runset <b>(required)</b>	(none)	Specifies the foundry runset to use for design rule checking.
signoff.check_drc_live. user_defined_options	(none)	Specifies additional options for the IC Validator command line. The string that you specify in this option is added to the command line used to invoke the IC Validator tool. The Fusion Compiler tool does not perform any checking on the specified string.
signoff.physical. layer_map_file <b>(required)</b>	(none)	Specifies the name of the layer mapping file that maps the layer names in the technology file to those in the foundry runset. For details about the format of the layer mapping file, see <a href="#">Defining the Layer Mapping for IC Validator In-Design Commands</a> .
signoff.physical. merge_stream_files	(none)	Specifies the stream (GDSII or OASIS) files to merge into the current block for interactive design rule checking. When you use this option, the GDSII or OASIS data replaces the cell library data for the cells defined in the specified stream files.

## Improving Instance Voltage Drop by Augmenting the Power Grid

Power grid augmentation is a technique used to improve the instance voltage drop. When you augment the power grid, the added metal shapes act as parallel resistors, which reduces the resistance of the power grid as seen by the standard cells. [Figure 133](#) shows an example of power grid augmentation and how it lowers the power grid resistance.

*Figure 133 Power Grid Augmentation*



The Fusion Compiler tool supports the following methods of power grid augmentation:

- [Standard power grid augmentation](#), which inserts as many PG augmentation shapes as possible to lower the instance voltage drop
- [Timing-driven power grid augmentation](#), which inserts PG augmentation shapes in the specified regions of the block, except around timing-critical nets
- [Guided power grid augmentation](#), which inserts PG augmentation shapes only from cells with IR violations to their tap with the minimum path resistance

Each method uses RedHawk Fusion analysis to drive the power grid augmentation and uses the IC Validator tool to insert the PG augmentation shapes. For detailed information about performing power grid augmentation, see the topic associated with the method you want to use.

### See Also

- [RedHawk and RedHawk-SC Fusion](#)

## Standard Power Grid Augmentation

Standard power grid augmentation inserts as many PG augmentation shapes as possible to lower the instance voltage drop.

To perform standard power grid augmentation,

1. Ensure that the block is fully routed with very few or no DRC violations.
2. Run voltage drop analysis using RedHawk Fusion, as described in [Performing Voltage Drop Analysis](#).

The IC Validator tool uses the results of this voltage drop analysis to drive the power grid augmentation.

3. Set up the IC Validator environment as described in [Setting Up the IC Validator Environment](#).
4. (Optional) Enable distributed processing by using the `set_host_options` command, as described in [Enabling IC Validator Multicore Processing](#).
5. Set the application options for performing power grid augmentation, as described in [Setting Options for Power Grid Augmentation](#).

At a minimum, you must define the power and ground nets on which to perform PG augmentation.

- To specify the power net, set the `signoff.create_pg_augmentation.power_net_name` application option.
- To specify the ground net, set the `signoff.create_pg_augmentation.ground_net_name` application option.

If your design contains more than one power net, you must perform PG augmentation once for each power net.

6. Save the block to disk.

When you run the `signoff_create_pg_augmentation` command, the IC Validator tool uses the on-disk information for the block, not the information in memory. To ensure accurate information, use the `save_block` command to save the current state of the block before running the `signoff_create_pg_augmentation` command.

7. Perform power grid augmentation by using the `signoff_create_pg_augmentation` command.

When you run this command, you must specify the technology node for which to perform the augmentation by using the `-node` option.

If you must perform incremental PG augmentation because your design contains more than one power net, use the `-mode add` command on subsequent runs to append the PG augmentation shapes to the existing shapes.

8. Run voltage drop analysis using RedHawk Fusion, as described in [Performing Voltage Drop Analysis](#).

This second voltage drop analysis run measures the voltage drop after power grid augmentation.

During standard PG augmentation, the `signoff_create_pg_augmentation` command performs the following tasks:

1. Inserts PG augmentation shapes for the specified regions

When the IC Validator tool inserts the PG augmentation shapes, it considers the routing design rules to ensure that it does not create design rule violations.

By default, the command performs PG augmentation for the entire block. To specify the regions for PG augmentation, use one or more of the following options:

- `-coordinates_ground`

This option restricts PG augmentation for the ground net to the specified regions.

- `-excluded_coordinates_ground`

This option prevents PG augmentation for the ground net in the specified regions. If you specify this option with the `-coordinates_ground` option, the command does not perform PG augmentation in the overlapping regions.

- `-coordinates_power`

This option restricts PG augmentation for the power net to the specified regions.

- `-excluded_coordinates_power`

This option prevents PG augmentation for the power net in the specified regions. If you specify this option with the `-coordinates_power` option, the command does not perform PG augmentation in the overlapping regions.

The shapes added during PG augmentation have a `shape_use` attribute of `pg_augmentation`. You can use this attribute to query the shapes added during PG augmentation. For example,

```
fc_shell> get_shapes -filter {shape_use=~pg_augmentation}
```

2. Generates a summary report that shows the following information for each PG net:
  - The distribution of power grid augmentation shapes added on each layer
  - The total number of metal shapes added, the total number of via shapes added, and the total number of all shapes added
3. Saves the updated block to disk

### See Also

- [Timing-Driven Power Grid Augmentation](#)
- [Guided Power Grid Augmentation](#)
- [Removing PG Augmentation Shapes](#)

## Setting Options for Power Grid Augmentation

Before you run the `signoff_create_pg_augmentation` command, configure the run by setting the application options shown in [Table 46](#). To set the application options, use the `set_app_options` command. To see the current settings, use the `report_app_options` command.

*Table 46 Application Options for Power Grid Augmentation*

Application option	Default	Description
<code>signoff.create_pg_augmentation.ground_net_name</code> (required)	(none)	Specifies the name of the ground net.
<code>signoff.create_pg_augmentation.power_net_name</code> (required)	(none)	Specifies the name of the power net.
<code>signoff.create_pg_augmentation.run_dir</code>	<code>signoff_create_pg_augmentation_run</code>	Specifies the run directory, which contains the files generated by the <code>signoff_create_pg_augmentation</code> command.  You can specify either a relative path, in which case the directory is created under the current working directory, or an absolute path.

*Table 46 Application Options for Power Grid Augmentation (Continued)*

Application option	Default	Description
<code>signoff.create_pg_augmentation.user_defined_options</code>	(none)	Specifies additional options for the IC Validator command line. The string that you specify in this option is added to the command line used to invoke the IC Validator tool. The Fusion Compiler tool does not perform any checking on the specified string.

## Timing-Driven Power Grid Augmentation

Timing-driven power grid augmentation inserts PG augmentation shapes in the specified regions of the block, except around timing-critical nets. By default, the tool does not perform timing-driven PG augmentation.

To perform timing-driven power grid augmentation,

1. Ensure that the block is fully routed with very few or no DRC violations.
2. Run voltage drop analysis using RedHawk Fusion, as described in [Performing Voltage Drop Analysis](#). The IC Validator tool uses the results of this voltage drop analysis to drive the power grid augmentation.
3. Set up the IC Validator environment as described in [Setting Up the IC Validator Environment](#).
4. (Optional) Enable distributed processing by using the `set_host_options` command, as described in [Enabling IC Validator Multicore Processing](#).
5. Set the application options for performing power grid augmentation, as described in [Setting Options for Power Grid Augmentation](#).

At a minimum, you must define the power and ground nets on which to perform PG augmentation.

- To specify the power net, set the `signoff.create_pg_augmentation.power_net_name` application option.
- To specify the ground net, set the `signoff.create_pg_augmentation.ground_net_name` application option.

If your design contains more than one power net, you must perform PG augmentation once for each power net.

6. Save the block to disk.

When you run the `signoff_create_pg_augmentation` command, the IC Validator tool uses the on-disk information for the block, not the information in memory. To ensure accurate information, use the `save_block` command to save the current state of the block before running the `signoff_create_pg_augmentation` command.

7. Perform timing-driven power grid augmentation by using the `signoff_create_pg_augmentation` command with one or both of the following options:

- `-nets`

This option explicitly specifies the critical nets.

- `-timing_preserve_setup_slack_threshold`

This option enables the tool to automatically determine the critical nets based on a setup slack threshold. The unit for the slack threshold setting is the library time unit.

You must also specify the technology node for which to perform the augmentation by using the `-node` option.

If you must perform incremental PG augmentation because your design contains more than one power net, use the `-mode add` command on subsequent runs to append the PG augmentation shapes to the existing shapes.

8. Run voltage drop analysis using RedHawk Fusion, as described in [Performing Voltage Drop Analysis](#).

This second voltage drop analysis run measures the voltage drop after power grid augmentation.

During timing-driven PG augmentation, the `signoff_create_pg_augmentation` command

1. Performs timing analysis, including multicorner-multimode analysis, to minimize timing impact
2. Identifies timing-critical nets based on the options you specify
3. Removes existing PG augmentation shapes

By default, the command removes the PG augmentation shapes from the entire block. To keep the existing PG augmentation shapes, use the `-mode add` option.

4. Inserts PG augmentation shapes for the specified regions, except in the areas around the critical nets

5. Saves the updated block to disk
6. Generates a summary report

#### See Also

- [Standard Power Grid Augmentation](#)
- [Guided Power Grid Augmentation](#)
- [Removing PG Augmentation Shapes](#)

---

## Guided Power Grid Augmentation

Guided power grid augmentation inserts PG augmentation shapes only near cells with IR violations. For each candidate cell, guided PG augmentation inserts PG augmentation shapes in the region from the cell to its tap with the minimum path resistance. Because the shapes are added only in specific regions, guided PG augmentation inserts fewer metal shapes and uses less routing area, which decreases the coupling capacitance impacts.

To perform guided power grid augmentation,

1. Ensure that the block is fully routed with very few or no DRC violations.
2. Run voltage drop and minimum path resistance analysis using RedHawk Fusion, as described in [Performing Voltage Drop Analysis](#) and [Performing Minimum Path Resistance Analysis](#).
3. Set up the IC Validator environment as described in [Setting Up the IC Validator Environment](#).
4. (Optional) Enable distributed processing by using the `set_host_options` command, as described in [Enabling IC Validator Multicore Processing](#).
5. Save the block to disk.

When you run the `fix_pg_wire` command, the IC Validator tool uses the on-disk information for the block, not the information in memory. To ensure accurate information, use the `save_block` command to save the current state of the block before running the `fix_pg_wire` command.

6. Perform guided power grid augmentation by using the `fix_pg_wire` command with one or both of the following options:

- `-number_pins`

This option specifies the number of worst IR-drop violators for which to perform PG augmentation.

- `-voltage_drop_threshold`

This option enables the tool to automatically determine the cells to process based on a voltage drop threshold.

In addition, you must specify

- The power and ground nets on which to perform PG augmentation by using the `-supply_nets` option
- The technology node for which to perform the augmentation by using the `-node` option

For information about the `fix_pg_wire` command, see the man page.

7. Run voltage drop analysis using RedHawk Fusion, as described in [Performing Voltage Drop Analysis](#).

This second voltage drop analysis run measures the voltage drop after power grid augmentation.

## See Also

- [Standard Power Grid Augmentation](#)
- [Timing-Driven Power Grid Augmentation](#)
- [Removing PG Augmentation Shapes](#)

---

## Removing PG Augmentation Shapes

To remove PG augmentation shapes, use the `-mode remove` option with the `signoff_create_pg_augmentation` command. When you use this command, the IC Validator tool removes all PG augmentation shapes from the current block.

## Inserting Metal Fill With IC Validator In-Design

After routing, you can fill the empty spaces in the block with fill shapes to meet the metal density rules required by most fabrication processes. Before inserting metal and via fill, the block should be close to meeting timing and have only a very few or no DRC violations.

To insert metal fill,

1. Set up the IC Validator environment as described in [Setting Up the IC Validator Environment](#).
2. (Optional) Enable distributed processing by using the `set_host_options` command, as described in [Enabling IC Validator Multicore Processing](#).
3. Set the application options for metal fill insertion.

For information about the metal fill insertion options, see [Setting Options for Signoff Metal Fill Insertion](#).

4. Save the block to disk.

When you run the `signoff_create_metal_fill` command, the IC Validator tool uses the on-disk information for the block, not the information in memory. To ensure accurate information, use the `save_block` command to save the current state of the block before running the `signoff_create_metal_fill` command.

5. Perform metal fill insertion by using the `signoff_create_metal_fill` command as described in [Performing Metal Fill Insertion](#).

The `signoff_create_metal_fill` command respects routing blockages defined by the `create_routing_blockage` command and does not insert metal fill in the blockages; however, it does not respect routing guides created by the `create_routing_guide` command.

### Note:

If you are performing pattern-based metal fill insertion and the blockage layer numbers differ between the technology file and the foundry runset file, you must provide a layer mapping file, as described in [Defining the Layer Mapping for IC Validator In-Design Commands](#).

When the IC Validator tool performs metal fill insertion, it creates an internal subdesign named *block.FILL* in the design view of the block and inserts the fill cells in this internal subdesign. The fill cells are named *FILL\_INST\_#*. To query the fill cells, use the `get_fill_cells` command.

For information about the result files generated by the `signoff_create_metal_fill` command, see [Signoff Metal Fill Result Files](#).

After you insert metal fill, you can

- Display the added metal fill in the layout view in the GUI, as described in [Viewing Metal Fill in the GUI](#)
- Modify the metal fill, as described in [Modifying Metal Fill](#)
- Perform extraction for timing analysis using the real metal fill, as described in [Performing Real Metal Fill Extraction](#)

## Setting Options for Signoff Metal Fill Insertion

Before you run the `signoff_create_metal_fill` command, configure the run by setting the application options shown in [Table 47](#). To set the application options, use the `set_app_options` command. To see the current settings, use the `report_app_options` command.

*Table 47 Application Options for Signoff Metal Fill Insertion*

Application option	Default	Description
<i>Options that apply to all flows</i>		
<code>signoff.create_metal_fill.apply_nondefault_rules</code>	<code>false</code>	Controls whether metal fill insertion honors nondefault routing rules.
<code>signoff.create_metal_fill.auto_eco_threshold_value</code>	20	Specifies the maximum percentage of change to the block to perform incremental metal fill insertion using the <code>-auto_eco true</code> option.
<code>signoff.create_metal_fill.flat</code>	<code>false</code>	Controls whether the IC Validator tool uses the metal fill mode specified in the runset file ( <code>false</code> ) or uses the flat metal fill mode ( <code>true</code> ).
<code>signoff.create_metal_fill.read_design_views</code>	{}	Specifies the reference cells for which the IC Validator tool reads the design view instead of the frame view. Using the design view can expose problems that are masked by the frame view abstraction.
<code>signoff.create_metal_fill.read_layout_views</code>	{}	Specifies the reference cells for which the IC Validator tool reads the layout view instead of the frame view. Using the layout view can expose problems that are masked by the frame view abstraction.

**Table 47 Application Options for Signoff Metal Fill Insertion (Continued)**

Application option	Default	Description
signoff.create_metal_fill.run_dir	signoff_fill_run	Specifies the run directory, which contains the files generated by the signoff_create_metal_fill command. You can specify either a relative path, in which case the directory is created under the current working directory, or an absolute path.
signoff.create_metal_fill.user_defined_options	(none)	Specifies additional options for the IC Validator command line. The string that you specify in this option is added to the command line used to invoke the IC Validator tool. The Fusion Compiler tool does not perform any checking on the specified string.
signoff.physical.merge_exclude_libraries	(none)	Specifies the cell libraries whose cells are excluded from replacement with the cell data in the stream files specified in the signoff.physical.merge_stream_files application option.
signoff.physical.merge_stream_files	(none)	Specifies the stream (GDSII or OASIS) files to merge into the current block for metal fill insertion. When you use this option, the GDSII or OASIS data replaces the cell library data for the cells defined in the specified stream files.

*Options that apply only to pattern-based metal fill insertion*

signoff.create_metal_fill.runset	N/A	Specifies the foundry runset to use for pattern-based metal fill insertion. This application option is not used for track-based metal fill insertion
signoff.physical.layer_map_file	(none)	Specifies the name of the layer mapping file. In general, the technology file and the foundry runset file used by the IC Validator tool use the same layer numbers. If they do not, you must supply a layer mapping file to map the technology layers to the layers used in the runset file (for details about the format of the layer mapping file, see <a href="#">Defining the Layer Mapping for IC Validator In-Design Commands</a> ).

*Options that apply only to track-based metal fill insertion*

**Table 47 Application Options for Signoff Metal Fill Insertion (Continued)**

Application option	Default	Description
<code>signoff.create_metal_fill.max_density_threshold</code>	(none)	<p>Specifies the maximum density threshold value for a track fill layer. Use the following format to specify this information for each layer:</p> $\{layer\_name\} \{max\_density\}$ <p>This application option applies only to track-based metal fill insertion and does not apply when you use the <code>-mode add</code> option.</p>
<i>Options that apply only to timing-driven metal fill insertion (pattern-based or track-based)</i>		
<code>signoff.create_metal_fill.fill_over_net_on_adjacent_layer</code>	false	Controls whether the IC Validator tool inserts metal fill within the minimum spacing in the vertical extension of the net on the adjacent layers when performing timing-driven metal fill insertion.
<code>signoff.create_metal_fill.fill_shielded_clock</code>	false	Controls whether the IC Validator tool inserts metal fill for shielded clock nets when performing timing-driven metal fill insertion.
<code>signoff.create_metal_fill.fix_density_errors</code>	false	Controls whether the IC Validator tool performs density error fixing during timing-driven metal fill insertion.
<code>signoff.create_metal_fill.space_to_clock_nets</code>	Two times the minimum spacing specified in the technology file	<p>Specifies the minimum spacing between metal fill and a clock net on the same layer. Note that you must specify clock nets by using the <code>-nets</code> option.</p> <p>Any unspecified layers use the default spacing.</p>
<code>signoff.create_metal_fill.space_to_nets</code>	Two times the minimum spacing specified in the technology file	<p>Specifies the minimum spacing between metal fill and a timing-critical net on the same layer.</p> <p>Any unspecified layers use the default spacing.</p>
<code>signoff.create_metal_fill.space_to_nets_on_adjacent_layer</code>	Two times the minimum spacing specified in the technology file	<p>Specifies the minimum spacing between metal fill and a timing-critical net on an adjacent layer.</p> <p>Any unspecified layers use the default spacing.</p>

---

## Performing Metal Fill Insertion

You can use the `signoff_create_metal_fill` command to perform the following tasks:

- [Pattern-Based Metal Fill Insertion](#)
- [Track-Based Metal Fill Insertion](#)
- [Using an IC Validator Parameter File](#)
- [Typical Critical Dimension Metal Fill Insertion](#)
- [Timing-Driven Metal Fill Insertion](#)
- [Incremental Metal Fill Insertion](#)

### Pattern-Based Metal Fill Insertion

Pattern-based metal fill insertion is the default mode for the `signoff_create_metal_fill` command. It inserts metal and via fill by using a foundry runset. Before running the `signoff_create_metal_fill` command, you must specify the runset by setting the `signoff.create_metal_fill.runset` application option.

During pattern-based metal fill insertion, the `signoff_create_metal_fill` command performs the following tasks:

1. Loads the block into the IC Validator tool, as described in [Reading Blocks for Signoff Metal Fill Insertion](#)
2. Removes existing metal and via fill  
By default, the command removes the metal and via fill from the entire block. For information about performing incremental metal fill insertion, see [Incremental Metal Fill Insertion](#).
3. Inserts metal and via fill in the empty regions  
By default, the command uses the spacing rules defined in the technology file to perform metal fill insertion on all metal and via layers for the entire block. The

command uses the metal fill mode specified in the runset file, which is either hierarchical or flat.

To modify the default behavior,

- Set one or more of the following application options before running the `signoff_create_metal_fill` command:

- `signoff.create_metal_fill.apply_nondefault_rules`

To enable the use of nondefault spacing rules in addition to the spacing rules defined in the technology file, set this application option to `true`.

- `signoff.create_metal_fill.flat`

To force the use of the flat metal fill mode, set this application option to `true`.

- Use one or more of the following command options with the `signoff_create_metal_fill` command:

- `-select_layers` or `-all_runset_layers`

To restrict the layers on which to insert metal fill, use these options, as described in [Specifying the Layers for Metal Fill Insertion](#).

- `-coordinates` and `-excluded_coordinates`

To restrict the regions on which to insert metal fill, use these options, as described in [Specifying the Regions for Metal Fill Insertion](#).

4. Removes floating via fill from the specified via layers and from the via layers associated with the specified metal layers

Floating via fill is via fill that does not have both an upper and lower metal enclosure.

5. Saves the fill data in the design view

6. Writes the result files to the run directory

For information about the generated result files, see [Signoff Metal Fill Result Files](#).

## See Also

- [Timing-Driven Metal Fill Insertion](#)

## Reading Blocks for Signoff Metal Fill Insertion

By default, the IC Validator tool reads the design view for the top-level block and the library cell instances, and the pin information from the frame view for the macro cell and I/O pad cell instances.

- To read the GDSII or OASIS data for specific reference cells, specify the stream files by setting the `signoff.physical.merge_stream_files` application option.

When you use this option, the GDSII or OASIS data replaces the cell library view for the cells defined in the specified stream files, except for cells in the cell libraries specified with the `signoff.physical.merge_exclude_libraries` application option.

- To read the layout view for specific reference cells, specify the cells by setting the `signoff.create_metal_fill.read_layout_views` application option.

The layout view is identical to the GDSII or OASIS data that was used to create them, but reduces the runtime for signoff design rule checking.

**Note:**

By default, layout views are not included in the cell libraries To save the layout views, you must set the `lib.workspace.save_layout_views` application option to `true` during library preparation, as described in the *Library Manager User Guide*.

- To read the design view for specific reference cells, specify the cells by setting the `signoff.create_metal_fill.read_design_views` application option.

**Note:**

By default, design views are not included in the cell libraries To save the design views, you must set the `lib.workspace.save_design_views` application option to `true` during library preparation, as described in the *Library Manager User Guide*.

The order of precedence for child cell data is

1. GDSII or OASIS data specified with the `signoff.physical.merge_stream_files` application option
2. Layout views specified with the `signoff.create_metal_fill.read_layout_views` application option

If the IC Validator tool cannot find a layout view and the cell is specified in the `signoff.check_drc.read_design_views` application option, it reads the design view. Otherwise, it reads the frame view. If it cannot find the frame view, the cell is ignored during design rule checking.

3. Design views specified with the `signoff.create_metal_fill.read_design_views` application option

If the IC Validator tool cannot find a design view, it reads the frame view instead. If it cannot find the frame view, the cell is ignored during design rule checking.

4. Frame views

### Specifying the Layers for Metal Fill Insertion

By default, the `signoff_create_metal_fill` command inserts metal fill on all the metal routing layers and via fill on all the via layers. To modify the layers for metal fill insertion, use one of the following options:

- `-select_layers metal_fill_layers`

This option restricts metal fill insertion to the specified set of metal and via layers.

- `-all_runset_layers true`

This option enables metal fill insertion on all the fill layers specified in the runset. This option applies only to pattern-based metal fill insertion.

When you specify the layers for metal fill insertion, the `signoff_create_metal_fill` command

1. Removes the existing metal and via fill from the block

By default, the command removes all existing metal and via fill. For incremental metal fill insertion, the command removes metal and via fill only from the specified layers.

2. Inserts metal and via fill only on the specified layers

3. Removes floating via fill from the specified via layers and from the via layers associated with the specified metal layers

For example, if you specify `-select_layers {M2}`, the tool removes floating via fill from the V1 and V2 layers.

## Specifying the Regions for Metal Fill Insertion

By default, the `signoff_create_metal_fill` command inserts metal and via fill for the whole chip. To modify the regions for metal fill insertion, use one or both of the following options:

- `-coordinates`

This option restricts metal fill insertion to the specified regions.

**Note:**

The bounding box coordinates passed to the IC Validator tool in the `METAL_FILL_SELECT_WINDOW` parameter are enlarged by 1 um to avoid DRC violations on the boundary of the specified regions during metal fill insertion. The actual metal fill insertion occurs within the regions specified by the `-coordinates` option.

In addition, when you use the `-coordinates` option, the `signoff_create_metal_fill` command always uses the flat metal fill mode.

- `-excluded_coordinates`

This option prevents metal fill insertion in the specified regions. If you specify this option with the `-coordinates` option, the command does not perform metal fill insertion in the overlapping regions.

To specify the coordinates, use the following format:

`\{{x1 y1} {x2 y2} ... {xn yn}\}`

Note that there must be a space between each set of coordinates.

When you specify the regions for metal fill insertion, the `signoff_create_metal_fill` command

1. Removes all existing metal and via fill from the block

For standard metal fill insertion, the command removes all existing metal and via fill. For incremental metal fill insertion, the command removes metal and via fill only from the specified regions.

2. Inserts metal and via fill only on the specified regions

3. Removes floating via fill from the specified regions

For example, to remove all metal fill from the block and then fill all empty regions outside the bounding box with corners at (100,150) and (300,200), use the following command:

```
fc_shell> signoff_create_metal_fill \
 -excluded_coordinates \{{100 150} {300 200}\}
```

## Track-Based Metal Fill Insertion

Track-based metal fill insertion inserts metal and via fill by using a runset derived from the attributes and rules in the technology file to create fill shapes aligned to tracks. It does not use the runset specified by the `signoff.create_metal_fill.runset` application option. Track-based metal fill insertion offers the following benefits as compared to pattern-based metal fill insertion:

- Higher density
- Better control of density
- Well-balanced mask distribution for double-patterning layers

To perform track-based metal fill insertion, set the `-track_fill` option of the `signoff_create_metal_fill` command to a value other than `off`. Track-based metal fill insertion supports three modes:

- Sparse

This is the default mode. In this mode, track-based metal fill insertion skips one track between signal shapes and fill shapes.

- Dense

To enable this mode, use the `-fill_all_tracks true` option. In this mode, track-based metal fill insertion does not skip tracks between signal shapes and fill shapes.

**Note:**

If your block uses double-patterning technology, using this option increases runtime if the block is not precolored.

- Mixed

To use sparse mode for some layers and dense mode for other layers, you must set the appropriate parameters in a parameter file, as described in [Using an IC Validator Parameter File](#).

During track-based metal fill insertion, the `signoff_create_metal_fill` command performs the following tasks:

1. Loads the block into the IC Validator tool, as described in [Reading Blocks for Signoff Metal Fill Insertion](#)
2. Removes existing metal and via fill

By default, the command removes the metal and via fill from the entire block. For information about performing incremental metal fill insertion, see [Incremental Metal Fill Insertion](#).

### 3. Inserts metal and via fill in the empty regions

By default, the command uses the design rules defined in the technology file to insert fill shapes on-track on all metal and via layers for the entire block. The fill shapes have a length of 5 microns and a width equal to the `defaultWidth` attribute defined for the layer in the technology file.

You can modify the default behavior by using

- Application options set before running the `signoff_create_metal_fill` command:
  - `signoff.create_metal_fill.apply_nondefault_rules`  
 To enable the use of nondefault spacing rules in addition to the spacing rules defined in the technology file, set this application option to `true`.
- Command options used with the `signoff_create_metal_fill` command:
  - `-track_fill foundry_node`  
 To enable foundry-specific design rules, use the appropriate foundry keyword with the `-track_fill` option. To see the list of supported keywords, use the following command:  
`fc_shell> signoff_create_metal_fill -track_fill list`
  - `-select_layers`  
 To restrict the layers on which to insert metal fill, use this option, as described in [Specifying the Layers for Metal Fill Insertion](#).
  - `-coordinates` and `-excluded_coordinates`  
 To restrict the regions on which to insert metal fill, use these options, as described in [Specifying the Regions for Metal Fill Insertion](#).
- Parameters set in the IC Validator parameter file:
  - `mx_fill_width`, `mx_min_fill_length`, and `mx_max_fill_length`  
 To change the size of the fill shapes, set these parameters, as described in [Using an IC Validator Parameter File](#).

### 4. Removes floating via fill from the specified via layers and from the via layers associated with the specified metal layers

Floating via fill is via fill that does not have both an upper and lower metal enclosure.

5. (Optional) Trims the metal fill for each layer to try to meet the maximum density threshold defined by the `signoff.create_metal_fill.max_density_threshold` application option

The metal fill is trimmed only for those layers specified in the application option. If you do not set this option, the metal fill is not trimmed.

6. Saves the fill data in the design view

By default,

- The IC Validator tool assigns a data type of 0 to the fill shapes

To use different data type values, set the `mx_fill_datatype` and `viax_fill_datatype` parameters in the IC Validator parameter file.

- The IC Validator tool does not set mask constraints on the fill shapes

If your block uses double-patterning technology, use the `-output_colored_fill true` option to set mask constraints on the fill shapes.

If your block uses double-patterning technology, use the `-output_colored_fill` option to set mask constraints on the fill shapes. When you use this option, the IC Validator tool assigns a data type of 235 for fill shapes with a `mask_one` mask constraint and 236 for fill shapes with a `mask_two` mask constraint. To assign different data types for the colored fill, set the following parameters in the IC Validator parameter file: `mx_fill_datatype_color1`, `viax_fill_datatype_color1`, `mx_fill_datatype_color2`, and `viax_fill_datatype_color2`.

- The IC Validator tool does not write exclude layers

If your foundry uses exclude layers, define the data types for the exclude layers by setting the `mx_exclude_layer_datatype` parameters in the IC Validator parameter file. The IC Validator tool outputs the exclude layers that have defined data types.

For information about using an IC Validator parameter file, see [Using an IC Validator Parameter File](#).

7. Writes the result files to the run directory

In addition to the standard output files generated by signoff metal fill insertion, when you perform track-based metal fill insertion, you can output detailed density and density gradient reports by using the `-report_density` option. When you enable this option, the `signoff_create_metal_fill` command writes the following report files:

- `prefix_color_balance_and_density_report.txt`
- `prefix_fill_density_gradient_report.txt`

You can specify either `on` (in which case the tool uses “track\_fill” as the prefix) or the prefix string. For example, to perform track-based metal fill insertion and use a prefix of `my_prefix` for the detailed density and density gradient reports, use the following command:

```
fc_shell> signoff_create_metal_fill -track_fill true \
 -report_density my_prefix
```

For information about the generated result files, see [Signoff Metal Fill Result Files](#).

## See Also

- [Timing-Driven Metal Fill Insertion](#)

## Using an IC Validator Parameter File

You can customize the metal fill insertion by using an IC Validator parameter file.

- For pattern-based metal fill insertion, you pass the parameter file to the IC Validator tool by setting the `signoff.create_metal_fill.user_defined_options` application option.

```
fc_shell> set_app_options \
 -name signoff.create_metal_fill.user_defined_options \
 -value {-D INDESIGN_USER_DEFINED_PARAM_FILE=file_name}
```

- For track-based metal fill insertion, you pass the parameter file to the IC Validator tool by using the `-track_fill_parameter_file` option with the `signoff_create_metal_fill` command.

```
fc_shell> signoff_create_metal_fill -track_fill true \
 -track_fill_parameter_file file_name
```

When you run track-based metal fill insertion, the IC Validator tool writes a parameter file named `track_fill_params.rh` into the run directory. This file contains the default settings for the supported parameters. You can modify this file as necessary and use it on subsequent `signoff_create_metal_fill` runs.

### Note:

In some cases, the IC Validator behavior can be controlled by either an IC Validator parameter or an application option. In these cases, if both are specified, the IC Validator parameter setting overrides the application option setting.

[Table 48](#) shows the commonly used parameters supported in the parameter file. Unless otherwise specified, these parameters apply only to track-based metal fill insertion.

**Table 48 IC Validator Metal Fill Insertion Parameters**

Parameter	Default	Description
<i>Metal fill insertion mode parameters</i>		
mx_sparse_fill	1	<p>Controls whether metal fill insertion uses sparse mode (1) or dense mode (0) for the specified metal layer.</p> <p><b>Note:</b> If you use the <code>-fill_all_tracks true</code> option, the default changes to 0.</p>
<i>Fill size and spacing parameters</i>		
mx_fill_staggering	0.075 microns	Controls staggering for fill patterns.
mx_ignore_route_guide	0	Controls whether the IC Validator tool honors (0) or ignores (1) routing guides on the specified layer during metal fill insertion.
mx_ignore_system_blockage	0	Controls whether the IC Validator tool honors (0) or ignores (1) system blockages on the specified layer during metal fill insertion.
mx_via_enclosure	Derived from ContactCode section	Specifies the minimum metal enclosure values for vias.
mx_fill_width	defaultWidth <sup>9</sup>	Specifies the width of the metal fill shapes.
mx_min_fill_length	Derived from minArea <sup>9</sup>	Specifies the minimum length of the metal fill shapes.
mx_max_fill_length	5 microns	<p>Specifies the maximum length of the metal fill shapes.</p> <p>Decreasing this value can increase the via density at the expense of decreasing the metal density.</p>
mx_fill2fill_side_spacing		Specifies the minimum spacing between fill shapes.
mx_fill2route_side_spacing		Specifies the minimum spacing between metal fill and net shapes.
mx_fill2fill_end_spacing		Specifies the minimum end-to-end spacing between fill shapes.
mx_fill2route_end_spacing		Specifies the minimum end-to-end spacing between metal fill and net shapes.

9. Attribute setting from the Layer section of the technology file

**Table 48 IC Validator Metal Fill Insertion Parameters (Continued)**

Parameter	Default	Description
<code>mx_fill2Blockage_x</code> <code>mx_fill2Blockage_y</code>		Specifies the minimum spacing between fill shapes and blockages.
<code>mx_fill2routeGuide_x</code> <code>mx_fill2routeGuide_y</code>		Specifies the minimum spacing between fill shapes and routing guides.
<code>mx_fill2chipBoundary_x</code> <code>mx_fill2chipBoundary_y</code>		Specifies the minimum spacing between fill shapes and the chip boundary.
<code>mx_min_area_with_via</code>		Specifies the minimum fill area on which vias can be created.
<code>mx_EXCLUDED_CELLS_OVERSIZE_VALUE</code>	0.2 microns	Specifies the minimum spacing between the macros specified in the <code>mx_EXCLUDED_CELLS</code> parameter and fill shapes.
<code>mx_EXCLUDED_CELLS</code>	{}	Specifies the macros that use the spacing defined in the <code>mx_EXCLUDED_CELLS_OVERSIZE_VALUE</code> parameter.
<code>vx_iso_via_distance</code>	3.0 microns	Specifies the maximum distance within which a neighboring via must exist.
<code>vx_min_spacing</code>	<code>minSpacing<sup>9</sup></code>	Specifies the minimum spacing between vias. Be careful when changing the value of these parameters. A value smaller than the default can cause DRC violations, while a value larger than the default negatively impacts the via density.
<code>vx_per_metal</code>	1	Specifies the maximum number of via fill shapes contained in a metal fill shape. You can use this parameter to increase the via density. However, changing the default can cause many metal fill shapes to be connected to each other, which can increase the overall capacitance for the nearby signal nets.
<i>Exclude layer parameters</i>		
<code>mx_exclude_layer_datatype</code>	-1	Creates an exclude layer with the specified data type. The default setting of -1 prevents the creation of an exclude layer.
<i>Fill output mode parameters</i>		

**Table 48 IC Validator Metal Fill Insertion Parameters (Continued)**

Parameter	Default	Description
mx_compress_fill vx_compress_fill	0	Controls whether the fill is compressed (1) or uncompressed (0) for the specified layer.  <b>Note:</b> If you use the <code>-mode add</code> option, compression is not performed regardless of the setting of this parameter.
mx_fill_datatype vx_fill_datatype	0	Specifies the data type of the inserted fill shapes.
output_colored_fill	0	Controls whether fill shapes are assigned mask constraints in blocks that use double-patterning technology.  <b>Note:</b> If you use the <code>-output_colored_fill true</code> option, the default changes to 1.
mx_fill_datatype_color1 vx_fill_datatype_color1 mx_fill_datatype_color2 vx_fill_datatype_color2	0	Specifies the data type of the inserted colored fill shapes.  <b>Note:</b> This parameter is used only colored fill is enabled.

#### Density calculation parameters

exclude_bounding_box_blockage_for_density_computation	1	Controls whether the regions specified by the <code>-excluded_coordinates</code> option are excluded from the density computation.
exclude_route_guides_for_density_computation	0	Controls whether routing guides are excluded from the density computation.
exclude_system_metal_blockages_for_density_computation	0	Controls whether system metal blockages are excluded from the density computation.

**Table 48 IC Validator Metal Fill Insertion Parameters (Continued)**

Parameter	Default	Description
mx_density_gradient_window <sup>10</sup>	windowSize <sup>9</sup>	<p>Specifies the window size used for density gradient checking.</p> <p>You can also control the window size by setting the signoff.report_metal_density.gradient_window_size application option.</p> <p><b>Note:</b></p> <p>If the <code>windowSize</code> attribute is not specified in the technology file and the <code>signoff.report_metal_density.gradient_window_size</code> application option is not set, the tool uses a default of 50 microns.</p>
mx_max_density_window	50 microns	Specifies the window size used for maximum density checking.
mx_max_open_area_rule <sup>10</sup>		Specifies the maximum size of a square area that contains no polygons and does not interact with any polygons.
mx_min_density <sup>10</sup>	minDensity <sup>9</sup>	<p>Specifies the minimum percentage of metal allowed in the window.</p> <p>You can also control the window size by setting the signoff.report_metal_density.min_density application option.</p> <p><b>Note:</b></p> <p>If the <code>minDensity</code> attribute is not specified in the technology file and the <code>signoff.report_metal_density.min_density</code> application option is not set, the tool uses a default of 10 percent.</p>
mx_min_density_window <sup>10</sup>	windowSize <sup>9</sup>	<p>Specifies the window size used for minimum density checking.</p> <p>You can also control the window size by setting the signoff.report_metal_density.density_window application option.</p> <p><b>Note:</b></p> <p>If the <code>windowSize</code> attribute is not specified in the technology file and the <code>signoff.report_metal_density.density_window</code> application option is not set, the tool uses a default of 50 microns.</p>

10. This parameter applies to timing-driven metal fill insertion, whether pattern-based or track-based.

**Table 48 IC Validator Metal Fill Insertion Parameters (Continued)**

Parameter	Default	Description
mx_window_step_size	Half the window size	<p>Specifies the step size in the x- and y-directions used for density checking. The first value is the x-direction step size. The second value is the y-direction step size. These can be the same or different values.</p> <p>You can also control the step size by setting the <code>signoff.report_metal_density.density_window_step</code> application option. When you use this application option, it uses the specified value for the step size in both the x- and y-directions.</p>

## Typical Critical Dimension Metal Fill Insertion

Typical critical dimension (TCD) structures can improve yield for designs. These structures are made up of marker and fill layers, both of which are added to the block during TCD metal fill insertion.

To enable TCD metal fill insertion, set the `signoff.create_metal_fill.tcd_fill` application option to `true` before running the `signoff_create_metal_fill` command.

By default, the command inserts TCD structures on all metal and via layers for the entire block. To modify the default behavior, use the following options:

- `-select_layers`

To restrict the layers on which to insert metal fill, use this option, as described in [Specifying the Layers for Metal Fill Insertion](#).

- `-coordinates` and `-excluded_coordinates`

To restrict the regions on which to insert metal fill, use these options, as described in [Specifying the Regions for Metal Fill Insertion](#).

### Note:

You cannot use timing-driven metal fill insertion with TCD metal fill insertion.

### See Also

- [Removing Metal Fill](#)

## Timing-Driven Metal Fill Insertion

Timing-driven metal fill insertion inserts metal and via fill in the specified regions of the block, except around timing-critical nets. Timing-driven metal fill insertion is supported

for both pattern-based and track-based metal fill insertion. By default, the tool does not perform timing-driven metal fill insertion.

To perform timing-driven metal fill insertion, use one or both of the following options with the `signoff_create_metal_fill` command:

- `-nets`

This option explicitly specifies the critical nets.

- `-timing_preserve_setup_slack_threshold`

This option enables the tool to automatically determine the critical nets based on a setup slack threshold. The unit for the slack threshold setting is the library time unit.

Be careful when choosing the threshold value; using a large threshold value can result in too many critical nets, which could reduce the metal density and create large empty areas.

During timing-driven metal fill insertion, the `signoff_create_metal_fill` command

1. Performs timing analysis, including multicorner-multimode analysis, to minimize timing impact
2. Identifies timing-critical nets based on the options you specify
3. Removes existing metal and via fill

By default, the command removes the metal and via fill from the entire block. For information about performing incremental metal fill insertion, see [Incremental Metal Fill Insertion](#).

4. Inserts metal and via fill in the empty regions for the specified regions, except in the areas around the critical nets

To enable fill insertion for shielded timing-critical clock nets, set the `signoff.create_metal_fill.fill_shielded_clock` application option to `true`.

5. Invokes the IC Validator tool to perform metal fill insertion

By default, the minimum spacing between the metal fill and the net shapes of the critical nets is two times the minimum spacing specified in the technology file. This spacing requirement also applies to the vertical extension of the critical net on the adjacent layers. To modify these requirements, set the appropriate application options, as described in [Specifying the Spacing Requirements for Timing-Driven Metal Fill Insertion](#).

**Note:**

You can restrict the regions in which to perform timing-driven metal fill insertion, as described in [Specifying the Regions for Metal Fill Insertion](#)

and [Performing Metal Fill Insertion Only in Modified Regions](#); however, you cannot specify the layers. When performing timing-driven metal fill insertion, the `signoff_create_metal_fill` command inserts metal fill on all routing layers (or all changed layers, if you perform metal fill insertion only in the changed regions).

#### 6. (Optional) Fixes density errors

In some cases, the spacing requirements for timing-driven metal fill insertion might cause density errors. To fix these errors automatically during timing-driven metal fill insertion, set the `signoff.create_metal_fill.fix_density_errors` application option to `true` (the default is `false`). For information about the density design rules, see [Defining the Density Design Rules](#).

#### 7. Stores the metal fill data as fill cell objects in the design view

#### 8. Writes the result files to the run directory

For information about the generated result files, see [Signoff Metal Fill Result Files](#).

### See Also

- [Pattern-Based Metal Fill Insertion](#)
- [Track-Based Metal Fill Insertion](#)

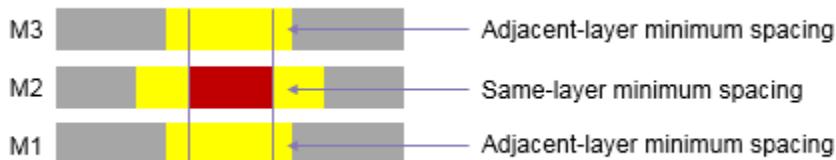
### Specifying the Spacing Requirements for Timing-Driven Metal Fill Insertion

By default, the minimum spacing between the metal fill and the net shapes of the critical nets is based on the minimum spacing specified in the technology file.

- For metal fill on the same layer as the net shape, the default minimum spacing is two times the minimum spacing value specified for the layer in the technology file.
- For metal fill on layers adjacent to the net shape, the default minimum spacing is the minimum spacing value specified for the layer in the technology file.

For example, assume that the red rectangle in [Figure 134](#) is a critical net shape. The yellow regions indicate where fill cannot be inserted and the gray regions indicate the inserted fill.

*Figure 134 Minimum Spacing From Metal Fill to Critical Net Shape*



To override the default spacing requirements for timing-driven metal fill insertion, set the following application options:

- `signoff.create_metal_fill.space_to_nets`

This application option defines the same-layer minimum spacing requirements between metal fill and a net shape of a timing-critical net.

- `signoff.create_metal_fill.space_to_clock_nets`

This application option defines the same-layer minimum spacing requirements between metal fill and a net shape of a user-defined clock net that is specified in the `-nets` option.

- `signoff.create_metal_fill.space_to_nets_on_adjacent_layer`

This application option defines the adjacent-layer minimum spacing requirements between metal fill and a net shape of a timing-critical net.

- `signoff.create_metal_fill.space_to_clock_nets_on_adjacent_layer`

This application option defines the adjacent-layer minimum spacing requirements between metal fill and a net shape of a user-defined clock net that is specified in the `-nets` option.

For each of these application options, use the following syntax to specify the spacing to use for each layer:

```
{ {layer1 value1} ... {layern valuen} }
```

You can specify the spacing value either as a multiple of the minimum spacing (*nx*) or a distance in microns.

For example, to set the minimum spacing between metal fill and a timing-critical net on the same layer to four times the minimum spacing for the M2 and M3 layers, use the following command:

```
fc_shell> set_app_options \
 -name signoff.create_metal_fill.space_to_nets \
 -value {{M2 4x} {M3 4x}}
```

To set the minimum spacing between metal fill and a timing-critical net on the same layer to 0.125 microns for the M2 layer and 0.133 microns for the M3 layer, use the following command:

```
fc_shell> set_app_options \
 -name signoff.create_metal_fill.space_to_nets \
 -value {{M2 0.125} {M3 0.133}}
```

## Defining the Density Design Rules

The IC Validator tool can check and fix the following density design rules:

- Minimum density, which specifies the minimum percentage of metal allowed in the density checking window

This rule is checked when you enable density error fixing by setting the `signoff.create_metal_fill.fix_density_errors` application option to `true`.

By default, the IC Validator tool uses the setting defined for the `minDensity` attribute in the `DensityRule` sections of the technology file. If this attribute is not defined for a layer, the IC Validator tool uses a default of 10 percent. To override the default, define the `mx_min_density` parameter in the IC Validator parameter file.

- Density gradient, which specifies the maximum percentage difference between the fill density of adjacent density checking windows

This rule is checked if it is defined in the technology file and you enable density error fixing by setting the `signoff.create_metal_fill.fix_density_errors` application option to `true`.

The IC Validator tool uses the setting defined for the `maxGradientDensity` attribute in the `DensityRule` sections of the technology file. If this attribute is not defined for a layer, the IC Validator tool does not check this rule.

- Maximum open area, which specifies the maximum size of a square area that contains no polygons and does not interact with any polygons

This rule is checked if it is defined.

The maximum open area rule is defined in an IC Validator parameter file. Use the following syntax to define the maximum open area rule for each metal layer:

```
mn_max_open_area_rule = value;
```

where `n` is the metal layer number and `value` is the side length of the square in microns.

For more information about the IC Validator parameter file, see [Using an IC Validator Parameter File](#).

For more information about the technology file, see the *Synopsys Technology File and Routing Rules Reference Manual*.

## Incremental Metal Fill Insertion

If you have already used the `signoff_create_metal_fill` command to performed metal fill insertion on a block, you can use incremental metal fill insertion to perform the following tasks:

- Insert additional metal and via fill to specific layers or regions, as described in [Adding to Existing Metal and Via Fill](#).
- Replacing existing metal and via fill in specific locations, as described in [Replacing Existing Metal and Via Fill](#).
- Replacing existing metal and via fill in modified regions, as described in [Performing Metal Fill Insertion Only in Modified Regions](#).

### Adding to Existing Metal and Via Fill

To insert metal and via fill without first removing the existing metal fill, use the `-mode add` option with the `signoff_create_metal_fill` command. When you use this mode, you can control where the metal fill insertion occurs by using one or more of the following options:

- `-select_layers`, which restricts the layers on which to perform metal fill insertion, as described in [Specifying the Layers for Metal Fill Insertion](#)
- `-coordinates` or `-excluded_coordinates`, which restrict the regions on which to perform metal fill insertion, as described in [Specifying the Regions for Metal Fill Insertion](#).

When you use these options with the `-mode add` option, the tool does not fix density errors regardless of the setting of the `signoff.create_metal_fill.fix_density_errors` application option.

### See Also

- [Pattern-Based Metal Fill Insertion](#)
- [Track-Based Metal Fill Insertion](#)

### Replacing Existing Metal and Via Fill

To remove and insert metal and via fill only for the specified locations, use the `-mode replace` option with the `signoff_create_metal_fill` command. You would typically

use this mode after ECO changes. When you use this mode, you must also specify one or more of the following options:

- `-select_layers`, which restricts the layers on which to perform metal fill insertion

For example, to remove all metal fill on the M1 and M3 layers and refill those two layers without affecting the metal fill on other layers, use the following command:

```
fc_shell> signoff_create_metal_fill -mode replace \
 -select_layers {M1 M3}
```

For more information about this option, see [Specifying the Layers for Metal Fill Insertion](#).

- `-coordinates` or `-excluded_coordinates`, which restrict the regions on which to perform metal fill insertion

For more information about these options, see [Specifying the Regions for Metal Fill Insertion](#).

- `-nets` or `-timing_preserve_setup_slack_threshold`, which restricts the nets on which to perform metal fill insertion

For more information about these options, see [Timing-Driven Metal Fill Insertion](#).

## See Also

- [Pattern-Based Metal Fill Insertion](#)
- [Track-Based Metal Fill Insertion](#)
- [Timing-Driven Metal Fill Insertion](#)

## Performing Metal Fill Insertion Only in Modified Regions

If you have previously run the `signoff_create_metal_fill` command on a block, you can use the `-auto_eco true` option to restrict the metal fill insertion to those regions of the block that have been modified since the last run.

### Note:

You can use this option only if the percentage of change to the block since the previous `signoff_create_metal_fill` run is less than the change threshold. The default change threshold is 20 percent; to modify the change threshold, set the `signoff.create_metal_fill.auto_eco_threshold_value` application option.

When you use the `-auto_eco true` option, the tool automatically determines the modified regions and layers, and then removes the existing metal fill and redoes metal fill insertion only in those locations; all other existing metal fill is retained.

By default, the tool compares the current block to the version used for the previous `signoff_create_metal_fill` run. To compare the current block to a different block, use the `-pre_eco_design` option to specify the comparison block.

**Note:**

A change on a metal layer triggers metal fill removal and insertion on the adjacent via layers to ensure that no floating or hanging vias remain when the metal shapes are removed from the ECO area. If a cell instance changes, the tool takes a conservative approach and considers changes on all layers.

For example, to remove the metal fill from the regions modified since the last time you ran the `signoff_create_metal_fill` command and then fill only those regions using the pattern-based mode, use the following command:

```
fc_shell> signoff_create_metal_fill -auto_eco true
```

You can use the `-select_layers` option to specify the layers on which to perform metal fill insertion; however, the tool performs metal fill insertion on a specified layer only if it is one of the automatically detected changed layers. You cannot use the `-coordinates` or `-excluded_coordinates` options to restrict the metal fill insertion regions.

When you use the `-auto_eco true` option,

- You cannot use the `-mode`, `-all_runset_layers`, and `-report_density` options
- The tool does not insert dense fill on the double-patterning layers regardless of the setting of the `-fill_all_tracks` option
- The tool does not fix density errors regardless of the setting of the `signoff.create_metal_fill.fix_density_errors` application option

**See Also**

- [Pattern-Based Metal Fill Insertion](#)
- [Track-Based Metal Fill Insertion](#)
- [Timing-Driven Metal Fill Insertion](#)

---

## Signoff Metal Fill Result Files

The `signoff_create_metal_fill` command stores the output files in the run directory specified by the `signoff.create_metal_fill.run_dir` application option (or the

`signoff_fill_run` directory if you do not use this option). For each fill run, the command creates a subdirectory named `icv_run_#` that contains the files generated for that run.

When you perform metal fill insertion, the tool writes the following files to the output directory:

- `block.LAYOUT_ERRORS`, which contains details about the detected errors
- `block.RESULTS`, which contains a summary of the run results
- `signoff_create_metal_fill.log`, which contains a summary of the run environment
- `icv_config_out`, which contains the paths to the top-level block and the cell libraries
- `layer.map`, which contains the generated layer mapping file
- `metal_fill_params.rh`, which contains the metal fill parameters and options that you specified
- `metal_fill_compress_params.rh`, which contains the storage commands and indicates whether the compression mode was flat (NONE) or hierarchical (AUTO)
- `./run_details` directory, which contains all the data generated by the IC Validator tool for the metal fill insertion run

---

## Querying Metal Fill

Metal fill shapes are stored in fill cells, which use the following naming convention:  
`FILL_INST_#`.

- To query the fill cells in the block, use the `get_fill_cells` command.

By default, this command reports only the fill cells in the top-level design. To report all of the fill cells in the hierarchy, use the `-hierarchical` option.

- To query the fill shapes, use the `get_shapes -include_fill` command.

To return only the fill shapes associated with specific fill cells, use the `-of_objects` option to specify the fill cells.

---

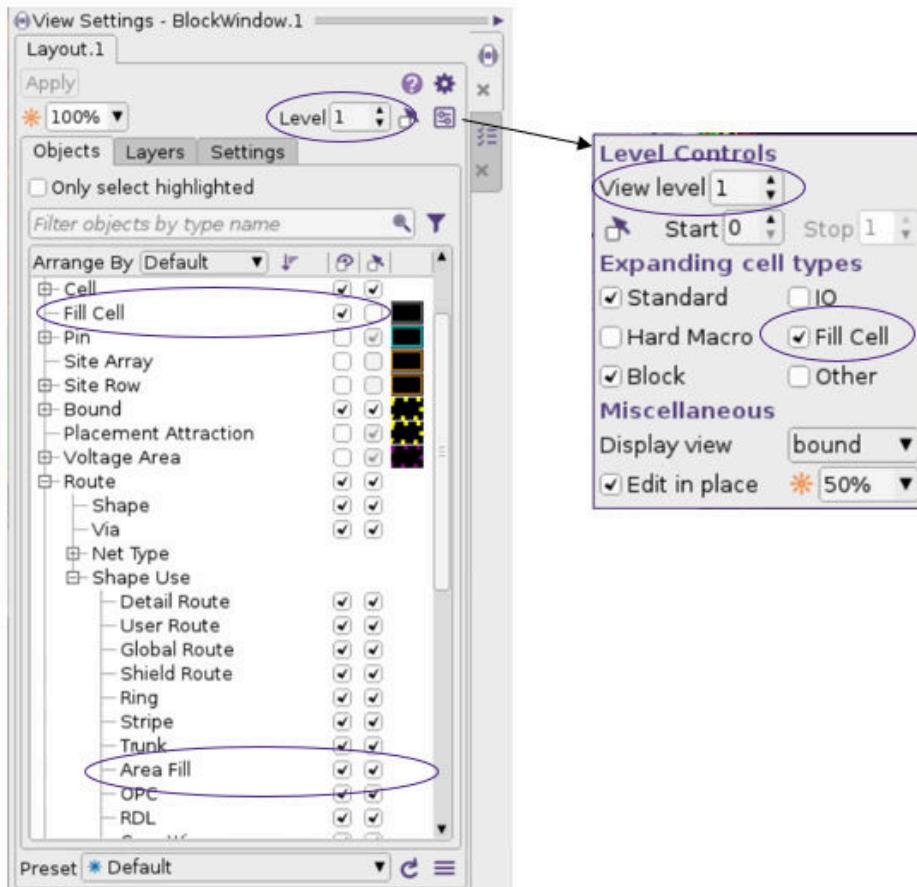
## Viewing Metal Fill in the GUI

To view the metal fill shapes in the layout view of the GUI,

1. Display the View Settings panel by clicking the View Settings icon () in the panel area.
  - Set the Level to 1.
  - Select the Objects tab and enable visibility for Fill Cell objects and Area Fill metal shapes (if you do not see Area Fill in the object list, expand the Route object and Shape Use attribute).
2. Display the Hierarchy Settings panel by clicking the Hierarchy Settings icon () in the View Settings panel or the menu bar.
  - Set the “View level” to 1.
  - Select “Fill Cell” in the “Expanding cell types” section.

[Figure 135](#) shows the View Settings and Hierarchy Settings panels with the required settings for displaying fill shapes.

Figure 135 Settings Used to Display Fill Shapes in the GUI



## Reporting the Metal Density

To report the metal density information, run the `signoff_report_metal_density` command.

To calculate the metal density information for a metal layer, the tool requires the following information:

- The density rule for the layer

The tool determines the density rule for each layer by using the following settings, in order of priority:

1. The setting of the `mx_min_density` parameter in the IC Validator parameter file specified by the `signoff.report_metal_density.user_defined_options` application option
2. The setting of the `signoff.report_metal_density.min_density` application option
3. The setting of the `minDensity` attribute in the `DensityRule` section of the technology file
4. 10 percent

- The grid sizes for the layer

The tool determines the grid sizes for each layer by using the following settings, in order of priority:

1. The setting of the `mx_min_density_window` or `mx_density_gradient_window` parameter in the IC Validator parameter file specified by the `signoff.report_metal_density.user_defined_options` application option
2. The setting of the `signoff.report_metal_density.density_window` or `signoff.report_metal_density.gradient_window_size` application option
3. The setting of the `windowSize` attribute in the `DensityRule` section of the technology file
4. 50 microns

For information about defining the metal density and density gradient rules in the technology file, including the `windowSize` attribute, see the *Synopsys Technology File and Routing Rules Reference Manual*.

By default, the `signoff_report_metal_density` command

- Uses the fill data for the metal density calculation
  - To use the design view if its timestamp is newer than the fill data, set the `signoff.report_metal_density.fill_view_data` application option to `read_if_upToDate`.
  - To use the design view regardless of its timestamp, set the `signoff.report_metal_density.fill_view_data` application option to `discard`.

- Creates the density and gradient windows starting from the lower-left corner of the chip boundary using a step size of half the window size
  - To change the starting point for creating the windows, use the `-starting_point` option with the `signoff_report_metal_density` command.
  - To change the step size, set the `signoff.report_metal_density.density_window_step` application option or the `mx_window_step_size` parameter in the IC Validator parameter file specified by the `signoff.report_metal_density.user_defined_options` application option. If you specify both, the IC Validator parameter overrides the application option.
- Computes the metal density for the entire block

To compute the metal density for specific regions, use the `-coordinates` option with the `signoff_report_metal_density` command. When you use this option, the command combines the specified regions when reporting the metal density; it does not report the metal density individually for each specified region.

To specify the coordinates, use the following format:

```
{ {x1 y1} {x2 y2} ... {xn yn} }
```

Note that there must be a space between each set of coordinates.

- Computes the metal density for each metal layer from the minimum routing layer to the maximum routing layer

To compute the metal density for specific layers, use the `-select_layers` option with the `signoff_report_metal_density` command.

- Does not generate heat maps

To generate heat maps to enable viewing of the metal density information in the GUI, set the `signoff.report_metal_density.create_heat_maps` application option to true. For information about viewing the heat maps, see [Viewing Density Heat Maps in the GUI](#).

- Writes the report files to a directory named `signoff_report_metal_density_run` under the current working directory

To change the name of the IC Validator working directory, set the `signoff.report_metal_density.run_dir` application option.

By default, the report files are named `report_metal_density.txt` and `gradient_density_report.txt`. To change the file names, use the `-output` option with the `signoff_report_metal_density` command. When you use this option, the density report has the specified file name, while the gradient density report adds a `.gradient` file extension. For example, if you specify `-output density.rpt`, the density report is named `density.rpt` and the gradient density report is named `density.rpt.gradient`.

The `signoff_report_metal_density` command uses the options that you specify to generate the command line to invoke the IC Validator tool. You can specify additional options for the IC Validator command line by setting the `signoff.report_metal_density.user_defined_options` application option. The string that you specify in this option is added to the command line used to invoke metal density reporting in the IC Validator tool. The Fusion Compiler tool does not perform any checking on the specified string.

### See Also

- [Using an IC Validator Parameter File](#)

## Viewing Density Heat Maps in the GUI

After running the `signoff_report_metal_density` command, you can view heat maps in the GUI for both the minimum density and gradient density reports.

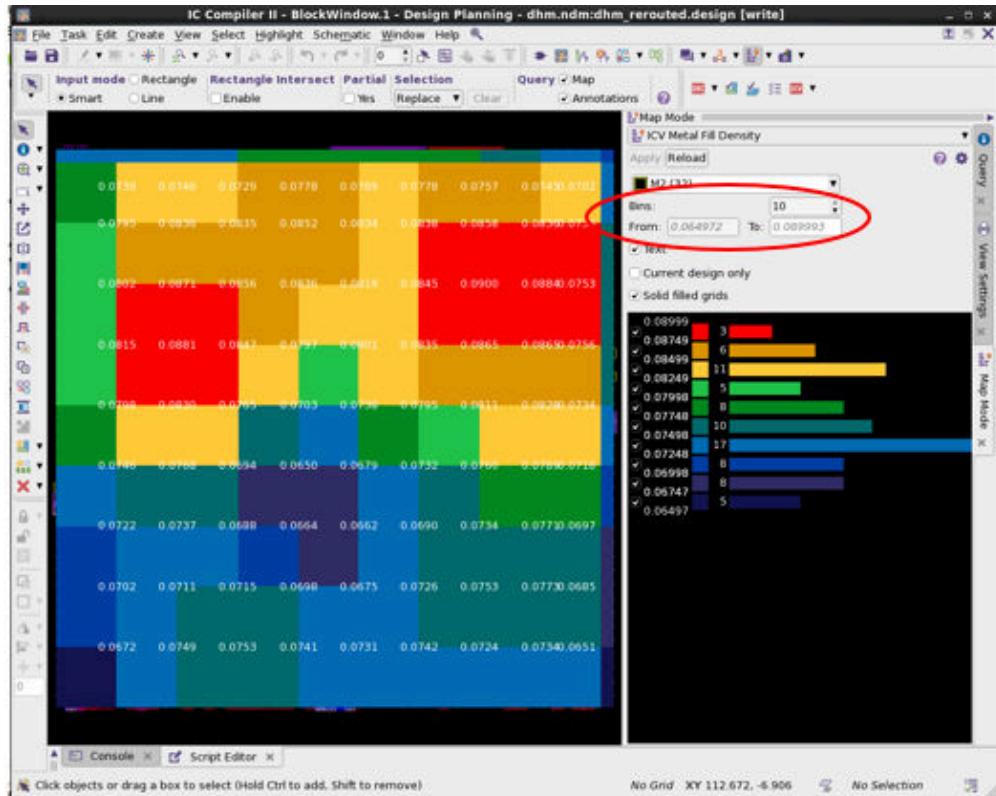
### Note:

The `signoff_report_metal_density` command generates the heat maps only when the `signoff.report_metal_density.create_heat_maps` application option is set to true.

- To display the minimum density heat map, choose **View > Map > ICV Metal Fill Density**.

[Figure 136](#) shows a minimum density heat map. In this heat map, each colored tile corresponds in size to the density window step. To customize or standardize the density values in the heat map, set the bin range, which is indicated by a red oval in the figure.

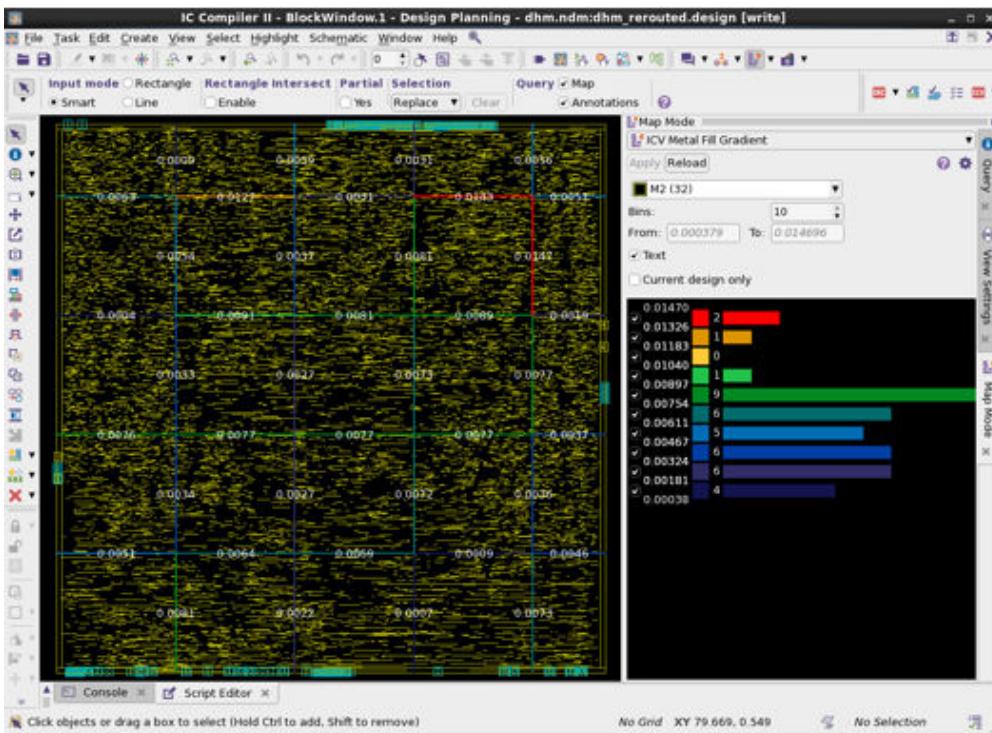
Figure 136 Minimum Density Heat Map



- To display the density gradient heat map, choose **View > Map > ICV Metal Fill Gradient**.

[Figure 137](#) shows a density gradient heat map. In this heat map, each colored bar corresponds in size to the gradient window size.

*Figure 137 Density Gradient Heat Map*



## Removing Metal Fill

The method you use to remove metal fill depends on the extent of the metal fill removal:

- To remove all metal fill from a block, specific layers of a block, or specific regions of a block, use the `signoff_create_metal_fill` command with the `-mode remove` option.

This command uses the IC Validator tool to remove metal fill from the current block, as described in [Removing Metal Fill With the IC Validator Tool](#).

- To remove specific fill shapes, use the `remove_shapes` or `remove_fill_cells` commands.

You would typically use this method when manually fixing DRC violations related to the metal fill. You might also be able to fix the DRC violation by modifying the metal fill shapes, as described in [Modifying Metal Fill](#).

## Removing Metal Fill With the IC Validator Tool

By default, when you use the `signoff_create_metal_fill -mode remove` command, the IC Validator tool removes all metal and via fill from the current block, including the typical critical dimension (TCD) structures.

- To remove the metal fill only from specific regions, use the `-coordinates` option, as described in [Specifying the Regions for Metal Fill Insertion](#).
- To remove the metal fill only from specific layers, use the `-select_layers` option, as described in [Specifying the Layers for Metal Fill Insertion](#).

By default, when you use this option, the IC Validator tool does not remove the TCD structures. To remove the TCD structures in addition to the metal and via fill, set the `signoff.create_metal_fill.tcd_fill` application option to `true` before running the `signoff_create_metal_fill` command.

- To remove the metal fill only over critical nets, use one or both of the `-nets` and `-timing_preserve_setup_slack_threshold` options to identify the critical nets.

### Note:

The existing fill is not considered when determining the critical nets for pattern-based metal fill, but not for track-based metal fill.

- To honor certain rules when removing the metal fill, use the `-remove_by_rule` option. You can specify one or more of the following rules:
  - Nondefault routing rules (`ndr`)

This rule applies to both pattern-based and track-based metal fill.

- Maximum density rules (`max_density_threshold`)

This rule applies only to track-based metal fill. When you enable this rule, metal fill removal honors the maximum density threshold set by the `signoff.create_metal_fill.max_density_threshold` application option.

For example, to honor nondefault routing rules during fill removal, use the following command:

```
fc_shell> signoff_create_metal_fill -mode remove \
 -remove_by_rule {ndr}
```

### See Also

- [Specifying the Layers for Metal Fill Insertion](#)
- [Specifying the Regions for Metal Fill Insertion](#)

---

## Modifying Metal Fill

In some cases, you might find that you can manually fix a DRC violation by changing the boundary of metal fill shapes, adding metal fill shapes, or removing metal fill shapes.

- To change the boundary of a metal fill shape, use the `set_attribute` command to modify its `bbox` attribute.
- To add a metal fill shape in the top-level fill cell, use the `create_shape -shape_use area_fill` command.
- To add a metal fill shape in a specific fill cell, use the `create_shape -fill_cell` command.
- To remove a metal fill shape, use the `remove_shapes` command.

You can also modify metal fill shapes in the GUI. To select or modify fill shapes in the GUI,

1. Select “Multiple Levels Active” () in the View Settings panel.
  2. In the “Hierarchy Settings” panel, set “View level” to a minimum of 2 and select “Fill Cell” in “Expanding cell types.”
- 

## Performing Real Metal Fill Extraction

To enable real metal fill extraction,

1. Associate non-emulation TLUPlus files with the timing corners by using the `set_parasitic_parameters` command, as described in the *Fusion Compiler Timing Analysis User Guide*.
2. Enable real metal fill extraction by using the following command:

```
fc_shell> set_extraction_options \
 -real_metalfill_extraction floating
```

For more information about performing extraction in the Fusion Compiler tool, see the *Fusion Compiler Timing Analysis User Guide*.

---

## Automatically Fixing Isolated Vias

An isolated via is a via that does not have neighboring vias close enough to meet the requirements of the technology.

To check for and fix isolated vias,

1. Set up the IC Validator environment as described in [Setting Up the IC Validator Environment](#).
2. (Optional) Enable distributed processing by using the `set_host_options` command, as described in [Enabling IC Validator Multicore Processing](#).
3. Set the application options for fixing isolated vias.

At a minimum, you must define the maximum distance within which a neighboring via must exist so that a via is not considered an isolated via. To define this information for each via layer, set the `signoff.fix_isolated_via.isolated_via_max_range` application option, which has the following syntax:

```
{ {via_layer1 distance1} ... {via_layern distancen} }
```

where the `via_layer` argument uses the mask names, such as `via1`, and the `distance` argument is in microns.

For information about the options available for fixing isolated vias, see [Setting Options for Fixing Isolated Vias](#).

4. Run isolated via checking and fixing by using the `signoff_fix_isolated_via` command as described in [Running the signoff\\_fix\\_isolated\\_via Command](#).

---

## Setting Options for Fixing Isolated Vias

Before you run the `signoff_fix_isolated_via` command, configure the run by setting the application options shown in [Table 49](#). To set the application options, use the `set_app_options` command. To see the current settings, use the `report_app_options` command.

**Table 49 Application Options for Signoff Isolated Via Fixing**

Application option	Default	Description
signoff. fix_isolated_via. isolated_via_max_range (required)	N/A	Specifies the distance in microns within which a neighboring via must exist on each via layer. If a neighboring via is not found within this distance, a via is considered an isolated via and the command tries to add a fixing via within the specified distance.
signoff. fix_isolated_via. avoid_net_types	{clock pg}	Specifies the types of nets to avoid when fixing isolated vias. Specify a list that contains one or more of the following values: clock, pg, and none.
signoff. fix_isolated_via. run_dir	signoff_fix_isolated_via_run	Specifies the run directory, which contains the files generated by the signoff_fix_isolated_via command. You can specify either a relative path, in which case the directory is created under the current working directory, or an absolute path.
signoff. fix_isolated_via. user_defined_options	(none)	Specifies additional options for the IC Validator command line. The string that you specify in this option is added to the command line used to invoke the IC Validator tool. The Fusion Compiler tool does not perform any checking on the specified string.

## Running the signoff\_fix\_isolated\_via Command

You can use the `signoff_fix_isolated_via` command either to check for isolated vias only, or to check for and fix the isolated vias.

### Checking for Isolated Vias

To check for isolated vias without fixing them, run the `signoff_fix_isolated_via` command with the `-check_only true` option.

```
fc_shell> signoff_fix_isolated_via -check_only true
```

When you run the `signoff_fix_isolated_via` command in check-only mode, it generates a summary report that specifies the number of isolated vias detected on each via layer.

## Checking and Fixing Isolated Vias

By default, the `signoff_fix_isolated_via` command both checks for and fixes isolated vias.

The command performs the following tasks:

1. Checks for isolated vias by using the ranges defined in the `signoff.fix_isolated_via.isolated_via_max_range` application option
2. (Optional) Performs track-based metal fill insertion to insert dummy fill shapes within the specified range around each detected isolated via
  - By default, the `signoff_fix_isolated_via` command ignores the existing fill data and performs track-based metal fill insertion to insert the fill shapes to use for fixing the isolated vias.

The fill shapes are inserted on-track following the design rules defined in the technology file. Some foundries have additional design rules, which you enable by setting the appropriate `TRACK_FILL_FOUNDARY_name` IC Validator variable. To set this variable, set the `signoff.create_metal_fill.user_defined_options` application option, as shown in the following example:

```
fc_shell> set_app_options \
 -name signoff.create_metal_fill.user_defined_options \
 -value {-D TRACK_FILL_FOUNDARY_name}
```

- If you have already performed track-based metal fill insertion, you can use the existing fill shapes to fix isolated vias by using the `-update_track_fill true` option.

When you use this option, you must use the `-track_fill_runset_include_file` option to specify the parameter file used for the initial track-based metal fill insertion. This parameter file is named `track_fill_params.rh` and is located in the run directory for the initial run. For example, if the run directory for the initial track-based metal fill insertion run is named `init_fill`, use the following command to fix isolated vias using the existing fill shapes:

```
fc_shell> signoff_fix_isolated_via -update_track_fill true \
 -track_fill_runset_include_file init_fill/track_fill_params.rh
```

For information about performing track-based metal fill insertion, see [Track-Based Metal Fill Insertion](#).

3. Inserts fixing vias within the specified range by using one of the following methods, in order of priority:
  - a. Inserting a via between an existing non-wide net shape and a fill shape
  - b. Extending the line end of an existing non-wide net shape and inserting a via between the extension and a fill shape
  - c. Inserting a via between a wide metal shape and a fill shape

When inserting the fixing vias, the tool considers the routing rules, including double-patterning rules and nondefault spacing rules, to prevent the introduction of DRC violations.

**Note:**

If the block is either very congested or very sparse, the `signoff_fix_isolated_via` command might not be able to fix all isolated vias.

4. Removes the unused dummy fill shapes
5. (Optional) Saves the updated block to disk
  - By default, the `signoff_fix_isolated_via` command does not save the updated block to disk. After running the command, you must use the `save_block` command to save the updated block to disk.
  - To save the block at the end of the `signoff_fix_isolated_via` run, use the `-save_design true` option with the `signoff_fix_isolated_via` command.
6. Saves the results to disk

The `signoff_fix_isolated_via` command generates the following results files:

- A summary report

This report specifies the number of isolated vias for each via layer before and after fixing.

- An error data file

By default, the error data generated by the `signoff_fix_isolated_via` command is saved in a file named `signoff_fix_isolated_via.err`. To specify the name for the error data file, use the `-error_data` option. To report or display the information in the error data file, use the error browser, as described in the *Fusion Compiler Graphical User Interface User Guide*.

# 9

## Routing Using Custom Router

---

Custom Router is a shape-based router that supports gridded or gridless routing for custom digital, mixed signal, and analog routing needs. In the Fusion Compiler environment, you can use Custom Router to create interconnects (or routes) for critical signals between blocks and continue with the Fusion Compiler tool to complete the physical implementation. You can also preroute critical signals or clock nets by using a set of custom routing constraints.

In the Fusion Compiler environment, Custom Router provides the following features:

- Prerouting in the batch mode
- Tcl-based routing constraint management
- Automatic routing with custom routing constraints
- Hybrid flow for prerouting
- Double Data Rate (DDR) net routing flow

Custom Router is fully integrated with the Fusion Compiler tool, and supports advanced design rules for 20 nm and below technologies.

To learn about using Custom Router in the Fusion Compiler environment, see the following topics:

- [Using Custom Router in the Fusion Compiler Tool](#)
- [Before Using Custom Router](#)
- [Defining Routing Constraints](#)
- [Managing Constraint Groups](#)
- [Using Custom Routing Application Options](#)
- [Routing With the Custom Router](#)
- [Shielding the Nets](#)
- [Checking the Routing Results](#)

- [Using a Hybrid Routing Flow](#)
- [Using a DDR Routing Flow](#)

These topics describe how to use Custom Router to create custom routes in the Fusion Compiler environment. For detailed information about Custom Router, see the *Custom Compiler Custom Router User Guide*.

---

## Using Custom Router in the Fusion Compiler Tool

The Fusion Compiler tool provides a set of Tcl commands to run Custom Router in the Fusion Compiler environment. These commands define routing constraints for adding wires on one or more nets at the block level or adding differential pair routing, bus routing, shielding, and other routing features.

A set of application options is also available for specifying the parameters for performing custom routing. For more information about how to define routing constraints and application options, see [Defining Routing Constraints](#) and [Using Custom Routing Application Options](#).

The `route_custom` command uses the specified routing constraints, as well as information from the technology file and the design, to perform automatic routing using the Custom Router. For more information about using the `route_custom` command, see [Routing With the Custom Router](#).

Custom Router supports the following constraints:

- Net shielding and differential pairs
- Matched length
- Variable width and space
- Pin width matching and tapering

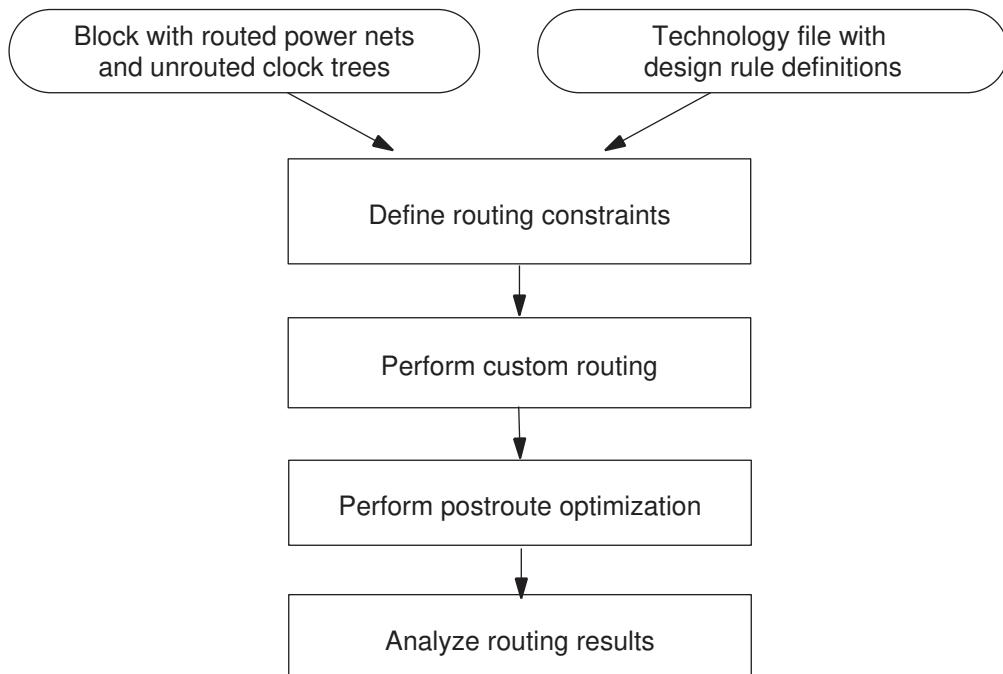
Custom Router supports the following Fusion Compiler nondefault routing rules:

- Routing layers and vias
- Routing width and spacing
- Routing grids
- Inter- and intra-group spacing
- Taper halo
- Routing blockage

- Routing corridors
- Shielding

[Figure 138](#) shows the basic Custom Router flow.

*Figure 138 Basic Custom Router Flow*



## Before Using Custom Router

Before you run Custom Router, you must

- Ensure the necessary design rules are correctly defined in the technology file, such as layer and via definitions.

For more information, see the *Synopsys Technology File and Routing Rules Reference Manual*.

- Check the connectivity information and look for pin blockages in the design.

For more information, see the "Reviewing the Design" topic in the *Custom Compiler Custom Router User Guide*.

- Ensure sure that the power and ground nets in the block have been routed.

For more information, see the *Fusion Compiler Design Planning User Guide*.

- Check the routability of the placement as explained in [Checking Routability](#).

## Defining Routing Constraints

The Fusion Compiler tool provides a set of application options to control custom routing results. The application options are applied globally. See [Using Custom Routing Application Options](#).

You can also define net-specific routing constraints that override the routing application options. Routing constraints provide guidance during routing. In the Fusion Compiler environment, Custom Router honors both the Custom Router constraints and the Fusion Compiler nondefault routing rules.

**Note:**

You must apply net-specific routing constraints to physical nets. When specifying nets that will be constrained, use the `get_nets -physical_context` command to ensure the physical net is returned. For example:

```
fc_shell> create_net_shielding \
 -for [get_nets -physical_context pr*] \
 -disabled_layers {M5 M6} -sharing true \
 -layer_gaps {M2 3 M3 5}
{shielding_4}
```

[Table 50](#) lists the commands to define the Custom Router constraints by creating constraint groups. For information about the supported Fusion Compiler nondefault routing rules, see [Using Nondefault Routing Rules](#).

To check or remove the created constraint groups, use the `report_constraint_groups` or `remove_constraint_groups` command. For more information about checking or removing constraint groups, see [Managing Constraint Groups](#).

*Table 50 Commands to Define Custom Router Routing Constraints*

Command	Description
<code>create_bus_routing_style</code>	Specifies the routing style for group of nets that are routed as a single bus structure (or trunk). For more information, see <a href="#">Defining the Bus Routing Style</a> .
<code>create_differential_group</code>	Defines a differential group for routing. For more information, see <a href="#">Creating Differential Groups</a> .

**Table 50 Commands to Define Custom Router Routing Constraints (Continued)**

Command	Description
<code>create_wire_matching</code>	Defines matched wire length for a group of nets or pin-to-pin connections. For more information, see <a href="#">Defining Matching Wire Lengths</a> .
<code>create_length_limit</code>	Defines the minimum wire length for a group of nets or pin-to-pin connections. For more information, see <a href="#">Defining Minimum Wire Lengths</a> .
<code>create_net_shielding</code>	Defines the shielding style for a group of nets or bundles. For more information, see <a href="#">Inserting Shields on the Nets</a>
<code>create_net_priority</code>	Determines the order in which the nets are routed. Nets with a higher priority are routed first. For more information, see <a href="#">Defining the Net Priority</a> .

## Defining the Bus Routing Style

To route a collection of nets or bundles as a single bus structure (trunk), create a constraint group that defines the bus routing style by using the `create_bus_routing_style` command.

You need to specify the following information:

- The name of the constraint group

If you do not specify a name, the command names the constraint group `bus_style_n`, where *n* is a unique integer.

By default, the command returns an error if the assigned name already exists. To delete the existing constraint group, use the `-force` option.

- The nets, bundles, and topology edges to which the constraint group applies

To specify the objects that constitute the constraint group, use the `-for` option.

- The constraints

To specify the constraints that are applied per layer to the trunk and override any settings on the individual bits, use the options listed in [Table 51](#).

**Table 51 Bus Routing Style Constraints**

Option	Description
-valid_layers	Specifies a list of the valid layers for trunk routing. When specified, the tool derives the minimum and maximum layer information from the list.
-layer_spacings	Specifies the minimum spacing for each routing layer. For example, {M1 5.0 M2 6.0}
-layer_widths	Specifies the minimum width for each routing layer. For example, {M1 5.0 M2 6.0}
-shield_placement	<p>Specifies the trunk shielding placement. Valid values are:</p> <ul style="list-style-type: none"> <li>◦ default (default): Uses the value from custom.route.bus_intra_shield_placement.</li> <li>◦ double_interleave: Shields each wire individually.</li> <li>◦ half_interleave: Inserts shared shielding every two wires.</li> <li>◦ interleave: Inserts shared shielding between adjacent wires.</li> <li>◦ outside: Places shields on the outside of the trunk or group.</li> </ul> <p>See also <a href="#">Intra-shield Placement</a>.</p>
-corner_type	<p>Specifies how the tool routes the bus trunks at corners. Valid values are:</p> <ul style="list-style-type: none"> <li>◦ auto (default): Automatically determines the optimal corner type for the bit or pin alignment to reduce routing congestion.</li> <li>◦ cross: Routes the bit so that they overlap at the corner.</li> <li>◦ river: Routes the bit so that they do not overlap at the corner.</li> </ul> <p>See also <a href="#">Bus Routing Options</a>.</p>
-gap	Specifies the spacing between the outermost bus bits and other shapes in the block. The default is 0.

The following example creates a bus routing constraint group for the nets that match the pattern pr\*. The command names the constraint group bus\_style\_1, because no name is assigned to the constraint group. The shield placement is interleave. The minimum spacing is 3 microns for the M2 layer and 5 microns for the M3 layer.

```
fc_shell> create_bus_routing_style \
 -for [get_nets -physical_context pr*] \
 -shield_placement interleave \
 -layer_spacings {M2 3 M3 5}
{bus_style_1}
```

## Creating Differential Groups

To create a differential group or a differential pair for the nets, bundles, or topology edges, use the `create_differential_group` command. Custom Router pairs the closest pins, one from each net, and routes from a common gather point between the pins. The tool routes the net pair the closest to each other with the most similar routing patterns.

You need to specify the following information:

- The name of the constraint group

If you do not specify a name, the command names the constraint group `differential_pair_n` or `differential_group_n`, where *n* is a unique integer.

By default, the command returns an error if the assigned name already exists. To delete the existing constraint group, use the `-force` option.

- The nets in the differential group

To specify the nets included in the created group, use the `-for` option. If you specify two nets, the nets form a differential pair. If you specify three nets, the nets form a differential group.

- The constraints for the differential group

To define the constraints that are applied per layer to the trunks and override any settings on the individual bits, use the options listed in [Table 52](#).

*Table 52 Differential Group Constraints*

Option	Constraint
<code>-twist_style</code>	Specifies the twist style. Valid values are <ul style="list-style-type: none"> <li>◦ <code>diagonal</code>: Applies 45-degree twists to the wires.</li> <li>◦ <code>none</code> (default): Does not twist the wires.</li> <li>◦ <code>orthogonal</code>: Applies 90-degree twists to the wires.</li> </ul>
<code>-twist_interval</code>	Specifies the distance between each twist.
<code>-twist_offset</code>	Specifies the distance between the first twist and the connected pins. The default is 0.
<code>-valid_layers</code>	Specifies the layers on which to route the differential group.
<code>-layer_spacings</code>	Specifies the minimum spacing for each routing layer.
<code>-layer_widths</code>	Specifies the minimum width for each routing layer.

Option	Constraint
-shield_placement	<p>Specifies how to shield the main bus trunk if one or more nets of the constraint group are shielded.</p> <p>Valid values are</p> <ul style="list-style-type: none"> <li>◦ double_interleave: Shields each wire individually.</li> <li>◦ half_interleave: Inserts shared shielding every two wires.</li> <li>◦ interleave: Inserts shared shielding between adjacent wires.</li> <li>◦ outside (default) : Places shields on the outside of the trunk or group.</li> </ul> <p>See also <a href="#">Intra-shield Placement</a>.</p>
-gap	Specifies the spacing in microns between the outermost bus bits and other shapes in the block. The default is 0.

The following example creates a differential pair for the prn and prp nets. The command names the differential pair differential\_pair\_0, because there is no name assigned to the different pair. No twist style is specified and the shield placement is set to default.

```
fc_shell> create_differential_group \
 -for [get_nets -physical_context {prn prp}] \
 -shield_placement default
{differential_pair_0}
```

## Inserting Shields on the Nets

To add shielding on the selected nets to reduce crosstalk or parasitic effects, run the `create_net_shielding` command.

You need to specify the following information:

- The name of the constraint group

If you do not specify a name, the command names the constraint group `shielding_n`, where *n* is a unique integer.

By default, the command returns an error if the assigned name already exists. To delete the existing constraint group, use the `-force` option.

- The nets, bundles, and topology edges to which the constraint group applies
- Use the `-for` option.
- The constraints for shielding styles and layer-specific requirements

To define the net shielding constraints that are applied per layer to the trunk and override any settings on the individual bits, use the options listed in [Table 53](#).

**Table 53 Net Shielding Constraints**

Option	Description
<code>-shield_net</code>	Specifies the net you want to use as the shield. When routing a single signal net, the tool uses this net for the left and bottom shields.
<code>-shield_net_2</code>	Specifies the secondary net you want to use as the shield. When routing a single signal net, the tool uses this net for the right and top shields.
<code>-via_defs</code>	Specifies the vias that can be used to connect to the shield. By default, all via definitions can be used.
<code>-gap</code>	Specifies the global minimum spacing between route objects and shield objects. The default is 0.
<code>-max_gap</code>	Specifies the global maximum spacing between route objects and shield objects. The default is 0.
<code>-width</code>	Specifies the global shield width. The default is 0.
<code>-layer_gaps</code>	Specifies the layer-specific minimum spacing between route objects and shield objects. The default is 0.
<code>-layer_max_gaps</code>	Specifies the layer-specific maximum spacing between route objects and shield objects. The default is 0.
<code>-layer_widths</code>	Specifies the layer-specific shield width. The default is 0.
<code>-min_segment</code>	Specifies the minimum length of wiring to shield. The default is 0.
<code>-sharing</code>	Specifies whether the shielding can be shared among the nets in the constraint group. Valid values are <code>unset</code> , <code>true</code> and <code>false</code> .
<code>-group_shield</code>	Uses a single set of shield shapes to shield all nets in the constraint group. This option is mutually exclusive with the <code>-enclose_pins</code> option.
<code>-enclose_pins</code>	Specifies whether pins are enclosed by the shield. This option is mutually exclusive with the <code>-group_shield</code> option. Valid values are <code>unset</code> , <code>true</code> , and <code>false</code> .

Option	Description
<code>-enclose_vias</code>	Specifies whether vias are enclosed by the shield. Valid values are <code>unset</code> , <code>true</code> , and <code>false</code> .
<code>-disabled_layers</code>	Specifies the layers that cannot be used for shielding.

The following example creates a constraint group and specifies constraints for adding shields on the nets whose names start with pr. The minimum spacing is 3 microns for the M2 layer and 5 microns for the M3 layer. All constituent objects can share the shield; the M5 and M6 layers cannot be used for shielding.

```
fc_shell> create_net_shielding \
 -for [get_nets -physical_context pr*] \
 -disabled_layers {M5 M6} -sharing true -layer_gaps {M2 3 M3 5}
{shielding_4}
```

## Defining the Net Priority

Use the net priority constraint to define the routing order for specific nets. Nets with a higher priority are routed first. To define the priority for a collection of nets or bundles, create a constraint group by using the `create_net_priority` command. You need to specify the following information:

- The name of the constraint group

If you do not specify a name, the command assigns a name.

By default, the command returns an error if the assigned name already exists. To delete the existing constraint group, use the `-force` option.

- The nets, bundles, and topology edges to which the constraint group applies

Use the `-for` option.

- The net priority

To specify the priority, use the `-priority` option. You can specify an integer between -128 and 128, inclusive.

The following example creates a constraint group named abc for the nets and bundles that match the pattern pr\*. The net priority is set to 15.

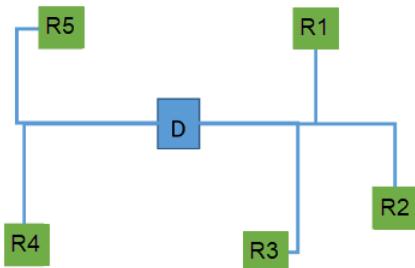
```
fc_shell> create_net_priority abc \
 -for [get_nets -physical_context pr*] \
 -priority 15
{abc}
```

## Defining Minimum Wire Lengths

To define different minimum wire lengths for each object in a group of nets or pin-to-pin connections, use the `create_length_limit` command. Use this feature to perform length-limit routing on the nets with a single driver and multiple receivers, and run point-to-point routing between the drivers, steiners, and receivers. The Custom Router generates the steiner nodes in the background when the `create_length_limit` command is executed. Each of the receivers has either a **matched length constraint** or an independent length constraint from the driver.

[Figure 139](#) shows an example of a single driver with multiple receivers, where each receiver has an individual length constraint.

*Figure 139 Receivers With Individual Wire Length Constraints From Driver D*



### Syntax:

```

create_length_limit
 -for objects
 [-min_value float]
 [-exclude pins or ports]
 [-driver pin or port]
 [-force]
 [intent_name]

```

Command or Options	Description
<code>create_length_limit</code>	Creates a length-limit constraint.
<code>-for</code>	Name of the driver pin or port, or net group that must comply with the constraint.
<code>-min_value</code>	(Optional) Minimum wire length. Default: 0
<code>-exclude</code>	(Optional) Excludes the specified pins or ports from the length matching constraint.
<code>-driver</code>	(Optional) Name of the driver pin or port.

Command or Options	Description
<code>-force</code>	(Optional) Overwrites the existing constraint. Use the option to delete existing constraint groups.
<code>intent_name</code>	(Optional) Name of the constraint. If you do not specify a name, the command assigns a name. By default, the command returns an error if the assigned name already exists. To delete an existing constraint, use the <code>-force</code> option.

In [Example 30](#), the commands create five wire-length constraints, one for each of the receivers shown in [Figure 139](#).

*Example 30 Five Wire-Length Constraints for the Five Receivers Connected to Driver D*

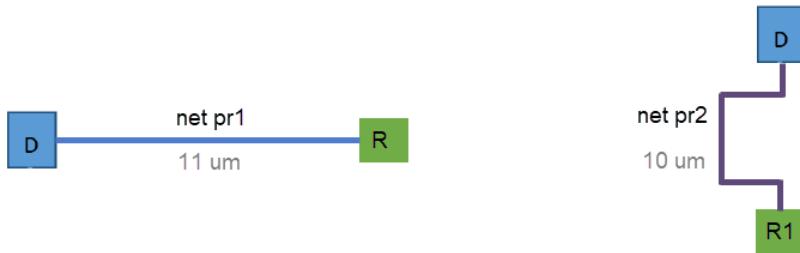
```
fc_shell> create_length_limit -for [get_nets -physical_context $net] \
 -driver [get_pins -physical_context D/o1]
fc_shell> create_length_limit -for [get_pins -physical_context R1/a] \
 -min_value 38.0
fc_shell> create_length_limit -for [get_pins -physical_context R2/a] \
 -min_value 40.0
fc_shell> create_length_limit -for [get_pins -physical_context R3/a] \
 -min_value 45.0
fc_shell> create_length_limit -for [get_pins -physical_context R4/a] \
 -min_value 42.0
fc_shell> create_length_limit -for [get_pins -physical_context R5/a] \
 -min_value 41.0
```

In [Example 31](#), the command creates a constraint group named abc for the nets and bundles that match the pattern pr\*. The minimum wire length is set to 10, so the length for each of the wires in the group is at least 10 microns. See [Figure 140](#).

*Example 31 Minimum Wire Length Constraint for pr\* Nets*

```
fc_shell> create_length_limit abc -for [get_nets -physical_context pr*] \
 -min_value 10
{abc}
```

*Figure 140 Minimum Wire Length Constraint for pr\* Nets*

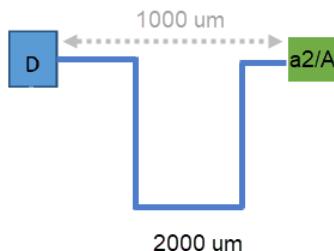


In [Example 32](#), the command creates a constraint group named `gcr_length_limit_a2` for the `a2/A` pin. The minimum wire length is set to 2000. See [Figure 141](#).

*Example 32 Minimum Wire Length Constraint for the a2/A Pin*

```
fc_shell> create_length_limit -for [get_pins -physical_context a2/A] \
 -min_value 2000.0 -force gcr_length_limit_a2
```

*Figure 141 Minimum Wire Length Constraint for the a2/A Pin*



## Defining Matching Wire Lengths

To define a group of nets or pin-to-pin connections that must have the same wire length, use the `create_wire_matching` command. By matching wire lengths, you can minimize clock skews and timing violations.

Two types of length-matching methods are available:

- Pin-based: Wire lengths between pins must match.

[Figure 142](#) and [Figure 143](#) show examples of pin-based wire length matching. In both examples, all the receivers have a matched length constraint from driver D.

- Net-based: Total length of wires of specified nets must match.

[Figure 144](#) shows an example of net-based wire length matching. In the example, the total wire length of nets `pr1` and `pr2` are within the specified 10-um tolerance.

Figure 142 Receivers With Matched Wire Length From Driver D

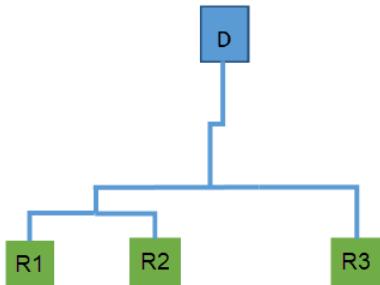


Figure 143 Receivers With Matched Wire Length From Driver D'

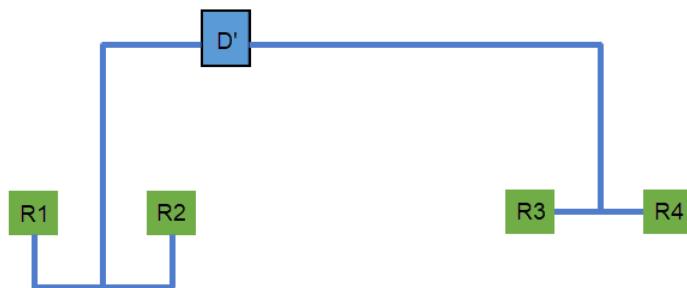
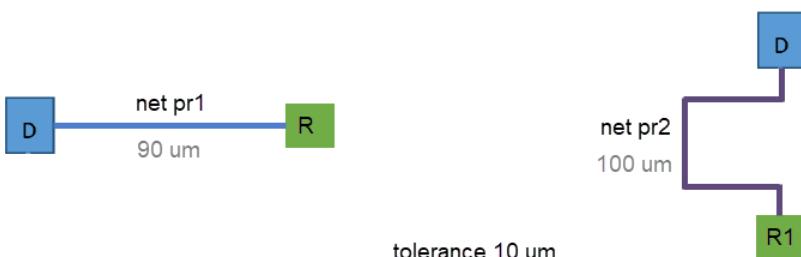


Figure 144 Net-Based Wire Length Matching



### Syntax:

```

create_wire_matching
 -for objects
 -tolerance float
 [-match_type type]
 [-exclude objects]
 [-relative]
 [-driver pin or port]
 [-force]
 [intent_name]

```

Command or Options	Description
<code>create_wire_matching</code>	Creates a length-matching constraint.
<code>-for</code>	Name of the driver pin or port ( <a href="#">Figure 142</a> and <a href="#">Figure 143</a> ), or net group ( <a href="#">Figure 144</a> ) that must comply with the constraint.
<code>-tolerance</code>	Length tolerance. The float value must be $\geq 0$ . To specify the tolerance as a percentage, use the <code>-relative</code> option.
<code>-relative</code>	(Optional) Interprets a <code>-tolerance</code> value $\leq 1$ as a percentage. The option must always be used with the <code>-tolerance</code> option.
<code>-match_type</code>	(Optional) Type of length matching. Valid values are: <ul style="list-style-type: none"> <li><code>length</code>: (Default) Compares the length of each object in the constraint group to make sure they have the same total length.</li> <li><code>length_per_layer</code>: Compares the per-layer length.</li> </ul>
<code>-exclude</code>	(Optional) Excludes the specified pins or ports from the length matching constraint.
<code>-driver</code>	(Optional) Name of the driver pin or port.
<code>-force</code>	(Optional) Overwrites the existing constraint. Use the option to delete existing constraint groups.
<code>intent_name</code>	(Optional) Name of the constraint. If you do not specify a name, the command assigns a name. By default, the command returns an error if the assigned name already exists. To delete an existing constraint, use the <code>-force</code> option.

After routing, a matched-length summary report is generated. You can use the report to troubleshoot and resolve the lengths that do not meet the set tolerance.

The content of the report depends on the `-for` option that you specified:

- If the `-for` option specifies pin-based length matching, a Length Constraint Report is generated. See [Figure 145](#).
- If the `-for` option specifies net-based length matching, a Routed Length report is generated. See [Figure 146](#).

*Figure 145 Pin-Based Length Matching Report*

```

Length Constraint Report
Net : net1
Driver : net1-net1
 Target Required Length Actual Length Difference
 ----- ----- -----
 i0a 20.204 20.009 -0.195(-0.965%)
 i1a 20.204 19.985 -0.219(-1.084%)
 i2a 20.204 19.052 -1.152(-5.702%)
 i3a 20.204 19.052 -1.152(-5.702%)

Total expected length : 80.816002 Total Routed length : 78.098007 (= 96.64%)

```

*Figure 146 Net-Based Length Matching Report*

```

Routed Length(Min:54.760 Max:55.144 Target Tolerance:1.00%)

 Net Name Length
 net2p 55.144
 net1p 54.760

 Target Net net2p
 Matched Range [54.760,55.144]
 Match Percentage 100.00%

```

In [Example 33](#), the commands create wire matching constraints for the scenario shown in [Figure 142](#). All receivers (R) have a matched length constraint from the balanced driver (D).

*Example 33 Wire Matching Constraints for All Receivers to Driver D*

```
fc_shell> create_wire_matching -for [get_nets -physical_context $net] \
 -relative -tolerance 0.01 -driver [get_pins -physical_context D/o1]

fc_shell> create_wire_matching -for [get_pins \
 -physical_context {R1/a R2/a R3/a}] -tolerance 0.01 -relative
```

In [Example 34](#), the commands create wire matching constraints for the example in [Figure 143](#).

*Example 34 Wire Matching Constraints for Receivers Connected to Driver D'*

```
fc_shell> create_wire_matching -for [get_nets -physical_context $net] \
 -tolerance 0.01 -relative -driver $driver_full_name

fc_shell> create_wire_matching \
 -for [get_pins -physical_context -of_objects $net] \
 -exclude $driver_full_name -tolerance 0.01 -relative
```

In [Example 35](#), the command creates a constraint group named abc for the nets and bundles that match the pattern pr\*. An absolute tolerance of 10 um is set to match the length of each object in the constraint group. See [Figure 144](#).

*Example 35 Wire Matching Constraints for pr\* Nets*

```
fc_shell> create_wire_matching abc \
 -for [get_nets -physical_context pr*] \
 -tolerance 10
{abc}
```

In [Example 36](#), the command creates a constraint group named def for the nets and bundles that match the pattern pr\*. A tolerance of 10 percent is set to match the length of each object in the constraint group.

*Example 36 Wire Matching Constraints for pr\* Nets With Percent Tolerance*

```
fc_shell> create_wire_matching def \
 -for [get_nets -physical_context pr*] \
 -tolerance 0.10 -relative
{def}
```

In [Example 37](#), the command creates a wire matching constraint named clk\_match for the clk net. Each of the endpoints that connect to the clk net has an individual matching constraint.

*Example 37 Wire Matching Constraints for the clk Net*

```
fc_shell> create_wire_matching -for [get_pins -physical_context \
 -of_objects [get_nets -physical_context {clk}]] \
 -tolerance 0.01 -match_type length_per_layer -relative \
 -force clk_match
```

In [Example 38](#), the command creates a wire matching constraint named clk\_match2 for all the nets in the block, except the pins whose names match the keyword u1/a.

*Example 38 Wire Matching Constraints With Exclusions*

```
fc_shell> create_wire_matching -for $nets -tolerance 0.05 \
 -exclude [get_pins -physical_context u1/a] -force clk_match2
```

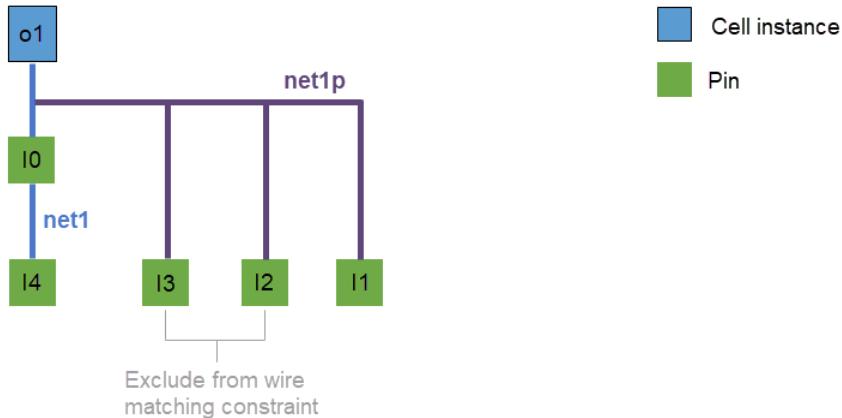
In [Example 39](#), the script creates a net-based wire matching constraint for the scenario shown in [Figure 147](#). The cell instance has an o1 terminal that servers as an inverter pin, driving signals on net1 and net1p to the pins. The I2 and I3 pins are excluded from the length matching constraint.

*Example 39 Net-Based Wire Matching Constraints With Pin Exclusions*

```
set nets to match length
set nets [get_nets -physical_context {net1 net1p}]
set allowed layers
```

```
set_routing_rule -min_routing_layer m7 -max_routing_layer m8 \
 -min_layer_mode allow_pin_connection -max_layer_mode hard $nets
setup Custom Router wire match constraint
set exclude_pin_names [get_pins -physical_context {i2/a i3/a}]
create_wire_matching -for $nets -tolerance 0.01 -relative \
 -exclude $exclude_pin_names
run Custom Router
route_custom -nets $nets
```

*Figure 147 Net-Based Wire Matching Constraints With Pin Exclusions*



## Managing Constraint Groups

You can report or remove the constraint groups that you create when specifying the routing constraints.

To report the created constraint groups, use the `report_constraint_groups` command and specify the constraint types to report by using the `-type` option. By default, all constraint types are reported. To specify the number of objects to report, use the `-count` option.

The supported constraint types are:

- **bus\_style**, which is specified by the `create_bus_routing_style` command
- **differential\_group**, which is specified by the `create_differential_group` command
- **differential\_pair**, which is specified by the `create_differential_group` command
- **matched\_wire**, which is specified by the `create_wire_matching` command
- **net\_priority**, which is specified by the `create_net_priority` command

- shielding, which is specified by the `create_net_shielding` command
- wire\_length\_limit, which is specified by the `create_length_limit` command

The following example shows the default output of the `report_constraint_groups` command:

```
fc_shell> report_constraint_groups

Report : report_constraint_groups
Design : top_new
Version:
Date :

```

Name	Type	Objects
NDR_bus	bus_style	bus1
NDR_diff	differential_pair	clk0 clk1
shield_diff	shielding	clk0 clk1
matchL_con	matched_wire	match1

To remove the created constraint groups, use the `remove_constraint_groups` command and specify the constraint groups to remove. To remove all the constraint groups, use the `-all` option. The command disassociates the constituent objects from the removed constraint groups, and reports the number of the groups that are removed.

For example,

```
fc_shell> report_constraint_groups
{shield_1} {shield_2}
fc_shell> remove_constraint_groups $a
```

## Using Custom Routing Application Options

The Fusion Compiler tool provides a set of application options to control custom routing results. The application options are applied globally. To define net-specific routing constraints to override the routing applications options, see [Defining Routing Constraints](#).

To list all application options that are available for custom routing, use the following command:

```
fc_shell> report_app_options custom.route.*
```

[Table 54](#) lists the application options for custom routing.

**Table 54 Application Options for Running Custom Routing**

Application option	Description
custom.route.bus_corner_type custom.route.bus_intra_shield_placement custom.route.bus_pin_trunk_offset custom.route.bus_split_even_bits custom.route.bus_split_ignore_width custom.route.bus_tap_off_enable custom.route.bus_tap_off_shielding	Specifies constraints for bus routing. See <a href="#">Bus Routing Options</a> .
custom.route.match_box	Specifies the area within which the Custom Router performs length matching.
custom.route.layer_grid_mode	Specifies whether the route and corresponding parallel shields, jumpers for parallel shields, and bus shields snap to the wire tracks or the routing grid. See <a href="#">Track Adherence Options</a> .
custom.route.diffpair_twist_jumper_enable custom.route.diffpair_twist_jumper_interval custom.route.diffpair_twist_jumper_offset custom.route.diffpair_twist_jumper_style	Specifies constraints for differential-pair routing. See <a href="#">Differential-Pair Options</a> .
custom.route.net_min_layer_mode custom.route.net_min_layer_mode_soft_cost custom.route.net_max_layer_mode custom.route.net_max_layer_mode_soft_cost	Specifies the minimum and maximum layer mode and the layer cost. See <a href="#">Specifying Net-Specific Layer Constraints</a> .
custom.route.routing_area	Defines an area within which the Custom Router creates routes. Use this option if your block is large and you only need to create or modify routes within a specific area.
custom.route.shield_connect_mesh_overlap custom.route.shield_connect_route custom.route.shield_connect_to_supply custom.route.shield_min_open_loop custom.route.shield_min_shield_seg custom.route.shield_min_signal_seg custom.route.shield_net custom.route.shield_second_net	Specifies shielding constraints. See <a href="#">Shielding Options</a> .

**Table 54 Application Options for Running Custom Routing (Continued)**

Application option	Description
custom.route.single_loop_match	Specifies how to extend the wire. See <a href="#">Single-Loop Matching</a> and <a href="#">Using a DDR Routing Flow</a> .
custom.route.single_loop_match_max_spacing	
custom.route.single_loop_match_min_spacing	
custom.route.single_loop_match_offset_layer	
custom.route.skip_connect_pin_type	Enables the <a href="#">hybrid flow</a> .
custom.route.distance_to_net_pin	(Hybrid flow only) When using the skip_connect_pin_type option, skips the connection if the distance between the pin and the route is <= distance_to_net_pin. Specify the distance_to_net_pin option as a list of pairs. The syntax for each pair of values is as follows: <code>{}{netName distance}{}{}</code>

## Bus Routing Options

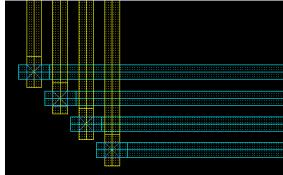
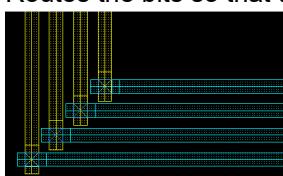
The Custom Router bus routing application options allow you to specify:

- [Corner Type](#)
- [Intra-shield Placement](#)
- [Pin-Trunk Offset](#)
- [Trunk Splitting](#)
- [Tapoffs](#)

### Corner Type

The `custom.route.bus_corner_type` application option specifies how the tool routes the bus trunks at corners. Valid values are:

Valid Value	Description
auto	(Default) Automatically determines the optimal corner type for the bit or pin alignment to reduce routing congestion.

Valid Value	Description
cross	Routes the bits so that they overlap at the corner. 
river	Routes the bits so that they do not overlap at the corner. 

## Intra-shield Placement

The `custom.route.bus_intra_shield_placement` application option specifies how the bus trunk will be shielded. Valid values are:

Valid Value	Description
double_interleave	Shields each wire individually.  Net Shield
half_interleave	Inserts shared shielding every two wires.  Net Shield
interleave	Inserts shared shielding between adjacent wires.  Net Shield
outside	(Default) Places shields on the outside of the trunk or group.  Net Shield

## Pin-Trunk Offset

The `custom.route.bus_pin_trunk_offset` application option changes the default spacing between the bus trunk and the bus bit pins.

## Trunk Splitting

The following Custom Router application options determine whether bus trunks will be split when routing through obstacles and how the bits will be split.

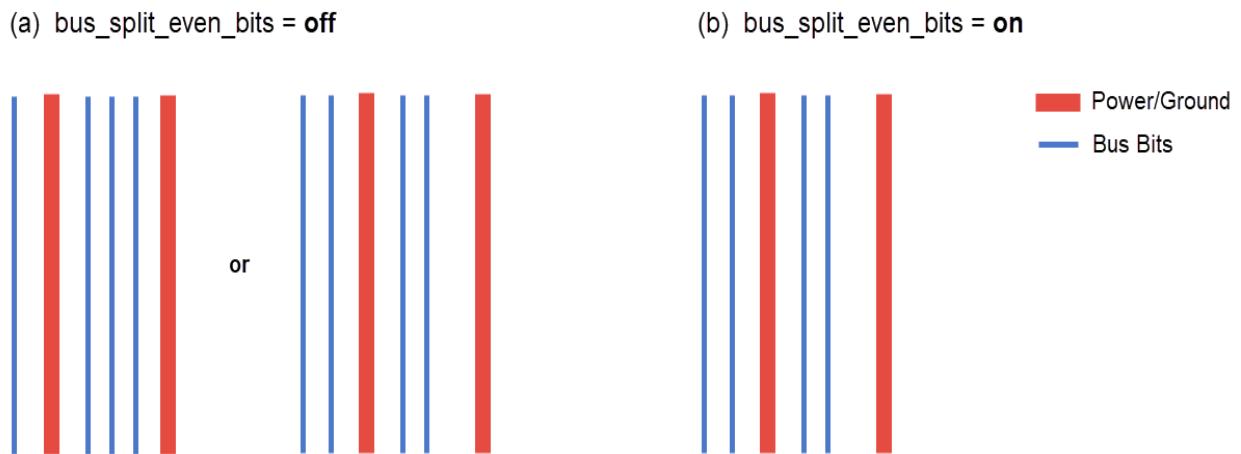
- `custom.route.bus_split_ignore_width`
- `custom.route.bus_split_even_bits`

The `custom.route.bus_split_ignore_width` application option specifies the wire width threshold of any obstructing power grid or route that triggers the bus trunk to be split into more than one section.

The `custom.route.bus_split_even_bits` application option requires that you set the `custom.route.bus_split_ignore_width` application option. If the `custom.route.bus_split_ignore_width` requirement is met, you can use the `custom.route.bus_split_even_bits` application option to control whether automatic bus splitting routes an even number of nets between pre-routes (power stripes).

In the following example, the bus trunk consists of four bits and the `custom.route.bus_split_ignore_width` application option is set. The spacing between the power and ground rails can accommodate three bits only. [Figure 148a](#) shows the two possible results when the `custom.route.bus_split_even_bits` application option is turned off. [Figure 148b](#) shows the result when the `custom.route.bus_split_even_bits` application option is turned on.

*Figure 148 Splitting Bus Trunk to Route Through Power and Ground Rails*



## Tapoffs

Two Custom Router application options allow you to control tapoff operations:

- `custom.route.bus_tap_off_enable`
- `custom.route.bus_tap_off_shielding`

The `custom.route.bus_tap_off_enable` application option connects pins to the bus trunk.

If the `custom.route.bus_tap_off_enable` application option is turned on, you can use the `custom.route.bus_tap_off_shielding` application option to add shielding to the bus tapoffs.

## Track Adherence Options

To specify whether the route and corresponding parallel shields, jumpers for parallel shields, and bus shields will snap to the wire tracks or the routing grid, use the `custom.route.layer_grid_mode` application option. Specify the application option with a list of paired values. The syntax for each pair of values is as follows:

`{ {layerName mode} }`

where

`layerName` is the name of the layer. To include all layers, specify `All`.

`mode` is one of the following:

- `on`: Snaps routes and shields to tracks. If the layer does not have tracks, the routes and shields will snap to the routing grid.
- `off`: Does not snap routes and shields to tracks or grids. Choose this option for gridless routing.
- `off_cost <costValue>`: Snaps routes and shields to tracks when possible. The value you specify determines the adherence of routes to tracks. The higher the value, the more likely the route will remain on the track or grid.

If tracks and routing grids are not defined and you specify `on`, the Custom Router uses an internally determined grid. The grid pitch is route width + minSpacing between shapes. The grid offset is 0 or pitch/2, whichever permits more pins on the grid.

### Example 40 Layer and Cost Specification for Track Adherence

```
{ {M2 off} {M3 off_cost 4} }
```

## Differential-Pair Options

The following Custom Router application options allow you to control differential-pair routing:

Application Option	Description
<code>custom.route.diffpair_twist_jumper_enable</code>	Twists the nets in the group.
<code>custom.route.diffpair_twist_jumper_interval</code>	Specifies the distance between each twist.
<code>custom.route.diffpair_twist_jumper_offset</code>	Specifies the distance between the first twist and the connected pins. The default is 0.
<code>custom.route.diffpair_twist_jumper_style</code>	Specifies the twist style. Valid values are: <ul style="list-style-type: none"> <li>diagonal: Applies 45-degree twists to the wires.</li> <li>none (default): Does not twist the wires.</li> <li>orthogonal: Applies 90-degree twists to the wires.</li> </ul>

## Shielding Options

The following Custom Router shielding application options allow you to control shielding:

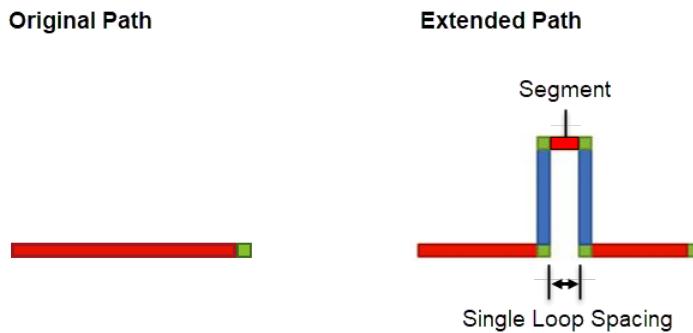
Application Option	Description
<code>custom.route.shield_connect_mesh_overlap</code>	Creates multiple shield connections to the power mesh for a shield. Creates a shield via at each intersection of a shield path segment and a power strap. If a shield does not intersect with any power strap, the default four-pass connecting process is followed.
<code>custom.route.shield_connect_route</code>	Connects shields to ground. This option creates long ties that might block future routing. Set this option only if you must connect shields to ground. Note that these shields are marked as dynamic shields.
<code>custom.route.shield_connect_to_supply</code>	Connects shields to the nearest power mesh.

Application Option	Description
custom.route.shield_min_open_loop	Specifies the minimum total length of an open shield-loop placed on the same layer. The shield shapes must be longer than shield_min_open_loop, or no shielding will be added.
custom.route.shield_min_shield_seg	Specifies the minimum length of shield segments. The default value is 0 um.
custom.route.shield_min_signal_seg	Specifies the minimum length of wiring to shield. The default value is 0 um.
custom.route.shield_net	Specifies the net you want to use as the shield. When routing a single-signal net, the tool uses this net for the left and bottom shields.
custom.route.shield_second_net	Specifies the secondary net you want to use as the shield. When routing a single-signal net, the tool uses this net for the right and top shields.

## Single-Loop Matching

To create single-loops to extend wires for length matching, use the Custom Router single-loop matching application options. Single-loop matching is typically used in [DDR routing flows](#).

*Figure 149 Adding Single Loops to Extend Wires*



Application Option	Description
custom.route.single_loop_match	Adds single loops to extend wire lengths.

Application Option	Description
custom.route.single_loop_match_max_spacing	Specifies the maximum spacing in the loop.
custom.route.single_loop_match_min_spacing	Specifies the minimum spacing in the loop.
custom.route.single_loop_match_offset_layer	Routes the segment at the top of the figure on a layer different from other segments in the loop.

## Routing With the Custom Router

Before performing custom routing on your block, you need to define routing constraints by using the commands described in [Defining Routing Constraints](#), and setting the appropriate application options.

To perform custom routing with Custom Router in the Fusion Compiler environment, run the `route_custom` command. By default, the tool routes all nets in the block. To route specific nets, use the `-nets` option.

By default, the tool removes the configuration data when the command run is complete. To keep the data in the cache for running other Custom Router commands in the current session, set the `-keep_session` option to `true`. The default is `false`.

The following example specifies the bus routing style and then performs bus routing.

```
fc_shell> create_bundle -name Bus1 {net0 net1}
fc_shell> create_bus_routing_style -for {Bus1} \
 -valid_layers {M5 M6} \
 -layer_widths {M5 0.4 M6 0.44} -force bus1
fc_shell> create_net_shielding -for {Bus1} -shield_net vss \
 -layer_gaps 0.21 -layer_widths 0.2 -sharing true -force sh1
fc_shell> set_app_options -name custom.route.bus_corner_type \
 -value river
fc_shell> set_app_options \
 -name custom.route.bus_intra_shield_placement -value interleave
fc_shell> route_custom -nets {net0 net1}
```

The following example creates a constraint group and then performs routing on the nets with the same length.

```
fc_shell> set matchNets {net1 net2 net3 net4}
fc_shell> create_bundle -name MatchL1 $matchNets
fc_shell> create_wire_matching -for MatchL1s \
 -match_type length -tolerance 1 \
```

```
-force matchL1_con
fc_shell> route_custom -nets {net1 net2 net3 net4}
```

The following example creates a differential group and then performs custom routing.

```
fc_shell> create_differential_group -for {net1 net2} \
 -valid_layers {M3 M4} -layer_widths {M3 0.4 M4 0.4} \
 -layer_spacings {M3 0.4 M4 0.4} \
 -twist_style diagonal -twist_interval 80.0 -force Diff
fc_shell> set_app_options\
 -name custom.route.diffpair_twist_jumper_offset -value 10
fc_shell> route_custom -nets {net1 net2}
```

---

## Shielding the Nets

Shielding is needed around sensitive nets to reduce noise and crosstalk. When you perform custom routing by using the `route_custom` command, the tool automatically creates shields for the nets based on the constraints you specify by using the `create_net_shielding` command.

To add shielding to the selected nets in a separate run, use the `create_custom_shields` command. The following example creates shields for net1 and net2.

```
fc_shell> create_custom_shields -nets {net1 net2}
```

To remove the shielding created by the `create_custom_shields` command, use the `remove_custom_shields` command. Doing this also removes the shielding created by the `route_custom` command.

---

## Checking the Routing Results

After running Custom Router to perform custom routing on the nets, you can check the routing results by reporting the routing results, generating a congestion map, running DRC checks, and so on.

For more information about the tasks to check the routing results, see [Analyzing the Routing Results](#).

---

## Using a Hybrid Routing Flow

In the hybrid flow, Custom Router is used to perform trunk routing without completing some pin connections, and then Zroute is used to complete the pin connections. This flow enables the use of Custom Router for prerouting while avoiding or reducing inconsistency and DRC violations associated with differences in the pin connection behavior between the two tools.

To enable the hybrid routing flow, set the `custom.route.skip_connect_pin_type` application option to one or more of the following values, depending on your routing requirements. By default, this application option is set to `none`, which disables the hybrid routing flow.

- `auto`

Custom Router identifies the routability of the pins in the block, and then completes the pin connections for the pins that can be connected.

- `all`

Custom Router does not complete the pin connections for all pins in the block.

- `stdcell`

Custom Router completes the pin connections for all pins in the block, except for standard cell pins.

- `io`

Custom Router completes the pin connections for all pins in the block, except for I/O cell pins.

- `macro`

Custom Router completes the pin connections for all pins in the block, except for macro pins.

By default, when the hybrid flow is enabled, Custom Router determines how much space to leave between a route and an unconnected pin. To specify the maximum distance, use the `custom.route.distance_to_net_pin` application option. Set the value in the following format: `{net_name distance}`.

The following script uses Custom Router for trunk routing and Zroute for all pin connections.

```
set_app_options -name custom.route.skip_connect_pin_type -value all
route_custom -nets {net1 net2}
route_eco -nets {net1 net2}
remove_redundant_shapes -nets {net1 net2}
route_detail -incremental true
```

The following script uses Custom Router for trunk routing and pin connections except standard cell pins, and then uses Zroute for standard cell pin connections.

```
set_routing_rule -min_routing_layer m3 \
 -max_routing_layer m9 \
 -min_layer_mode allow_pin_connection \
 -max_layer_mode hard {net1 net2}
set_app_options -name custom.route.skip_connect_pin_type \
```

```
-value stdcell
set_app_options -name custom.route.distance_to_net_pin \
 -value {{net1 5.0} {net2 3.0}}
route_custom -nets {net1 net2}
route_eco -nets {net1 net2}
remove_redundant_shapes -nets {net1 net2}
route_detail -incremental true
```

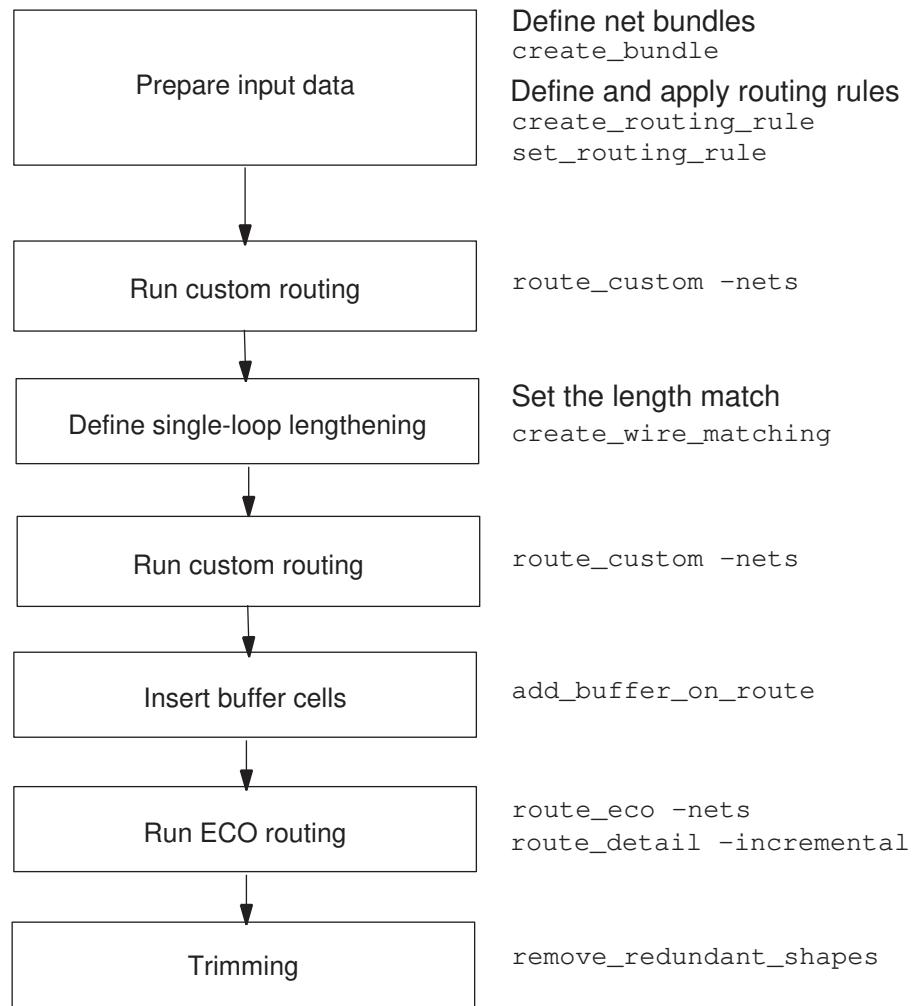
---

## Using a DDR Routing Flow

This topic describes how to use Custom Router to perform routing on a double data rate (DDR) design.

[Figure 150](#) illustrates the basic steps to create interconnects (routes) between a DDR net and the core.

Figure 150 A Basic DDR Flow



To perform routing on a DDR design with Custom Router,

1. Before you run Custom Router to perform routing on a DDR design,

- Group the nets as a bundle for length matching.

```
fc_shell> create_bundle -name GRP1 [get_nets $nets]
```

- Create and apply the routing rules for the DDR nets.

```
fc_shell> create_routing_rule DDR -widths {layer value ...} \
 -spacings {layer value ... }
fc_shell> set_routing_rule -rule DDR -min_routing_layer M4 \
 -max_routing_layer M5 $all_ddr_nets
```

Set two or more layers for initial routing and length matching. For postroute length matching, you should include two additional layers. For example, if you use the M4 and M5 layers for initial routing, set the M3, M4, M5, and M6 layers for postroute length matching.

- Add routing blockages as needed.

Add routing blockages to prevent Custom Router from routing in the area that is covered by placement blockages. This helps avoid possible unplaceable buffers during buffer insertion.

2. Create routes for the DDR nets by running the `route_custom` command.

```
fc_shell> route_custom -nets $all_ddr_nets
```

3. Define the routing constraints.

- Use the following application options to set the single-loop constraints.

```
fc_shell> set_app_options \
 -name custom.route.single_loop_match -value true
fc_shell> set_app_options \
 -name custom.route.single_loop_match_min_spacing -value 0.75
fc_shell> set_app_options \
 -name custom.route.single_loop_match_max_spacing -value 10.0
fc_shell> set_app_options \
 -name custom.route.single_loop_match_offset_layer -value true
```

- Set the wire matching constraint for the group.

```
fc_shell> create_wire_matching -for [get_bundles GRP1] \
 -match_type length -tolerance 20 -force match1
fc_shell> create_wire_matching -for [get_bundles GRP2] \
 -match_type length -tolerance 20 -force match2
```

- Set the bounding box for the wire matching routing constraint.

```
fc_shell> set_app_options -name custom.route.match_box \
 -value {{1000 1000} {1500 1500}}
```

By default, the bounding box coordinates are set to {{0 0} {0 0}}. When set to other values, the router limits matching routes to the area inside the box, which might result in connections that do not meet the matching constraint. The matching box should cover the entire available channel space.

4. Perform custom routing with length matching.

```
fc_shell> route_custom -nets [get_bundles {GRP1}]
```

If you define multiple matching boxes at step 3, you need to run the `route_custom` command on each matching box.

5. Report the list of mismatching nets.

```
% grep "Mismatch" log_file
```

6. Reduce the number of the mismatched nets (if any) by using one of the following methods:

- Rebalance the net routing of one or more groups in the same channel among the available routing layers by
  - a. Resetting the routing layers. For example,

- Set the routing layers for route group A to the M3 and M4 layers, and for group B to the M5 and M6 layers, or

- Set the routing layers for route groups A and B to the M3 and M4 layers, and route all mismatched nets on the M5 and M6 layers.

- b. Rerunning initial routing with the rebalanced routing layer settings.

- Set a larger value for the `custom.route.single_loop_match_max_spacing` application option.
- Allow different layers for the single loop.

```
fc_shell> set_app_options \
 -name custom.route.single_loop_match_offset_layer -value true
```

7. Insert buffer cells on the routed DDR net.

In the following example, `ddr_BUF` is used as the prefix for the names of the added ECO nets and buffers for easy identification. The buffer cells are added at an interval that is 20 percent of the total net length.

```
fc_shell> set ddr_buf "BUF001"
fc_shell> add_buffer_on_route -net_prefix ddr_BUF \
 -cell_prefix ddr_BUF \
 -repeater_distance_length_ratio 0.2 \
 -respect_blockages [get_nets $ddr_nets] $ddr_BUF
```

8. Perform legalization and verify that the placement is legal.

```
fc_shell> legalize_placement -cells $ddr_BUF
fc_shell> check_legality -cells $ddr_BUF
```

**Note:**

Using the `-cells` option with the `legalize_placement` and `check_legality` commands is supported only in the standard legalizer.

9. Preserve the route shapes of the clock nets by setting the `shape_use` attribute of the clock nets to `user_route`.

```
fc_shell> set_attribute [get_shapes -of_object $all_nets] \
 shape_use "user_route"
fc_shell> set_attribute [get_vias -of_object $all_nets] \
 shape_use "user_route"
```

10. Limit rerouting to minor changes by setting the `physical_status` attribute on the nets to `minor_change`.

```
fc_shell> set_attribute [get_shapes -of_object $all_nets] \
 physical_status "minor_change"
fc_shell> set_attribute [get_vias -of_object $all_nets] \
 physical_status "minor_change"
```

11. Run ECO routing to reconnect the nets by using the `route_eco` command.

```
fc_shell> route_eco -nets $all_nets -reroute modified_nets_only
```

12. Run incremental detail routing to fix the DRC violations by running the `route_detail -incremental true` command. After routing is complete, clean up the routed nets by running the `remove_redundant_shapes` command.

```
fc_shell> set_attribute \
 [get_shapes -of_object $all_nets] shape_use "detail_route"
fc_shell> set_attribute \
 [get_vias -of_object $all_nets] shape_use "detail_route"
fc_shell> route_detail -incremental true
fc_shell> remove_redundant_shapes -nets $all_nets
```

13. Check the routing results by using one of the following methods:

- Check for DRC violations.

```
fc_shell> check_routes
```

By default, the tool checks for DRC violations between signal shapes. To check for DRC violations between signal shapes and user shapes, use the following command:

```
fc_shell> check_routes -check_from_user_shapes true
```

- View the ECO nets and buffers in the GUI.
- Report the lengths of the DDR nets and the ECO nets.

# 10

## Physical Datapath With Relative Placement

---

The physical datapath with relative placement capability provides a way for you to create structures in which you specify the relative column and row positions of instances. During placement and legalization, these structures, which are placement constraints called relative placement structures, are preserved and the cells in each structure are placed as a single entity. Relative placement is also called physical datapath and structured placement.

The concepts and tasks necessary for doing relative placement are described in these sections:

- [Introduction to Physical Datapath With Relative Placement](#)
- [Relative Placement Flow](#)
- [Creating Relative Placement Groups](#)
- [Adding Objects to a Group](#)
- [Specifying Options for Relative Placement Groups](#)
- [Changing the Structures of Relative Placement Groups](#)
- [Generating Relative Placement Groups for Clock Sinks](#)
- [Performing Placement and Legalization of Relative Placement Groups](#)
- [Analyzing Relative Placement Groups](#)
- [Saving Relative Placement Information](#)
- [Summary of Relative Placement Commands](#)

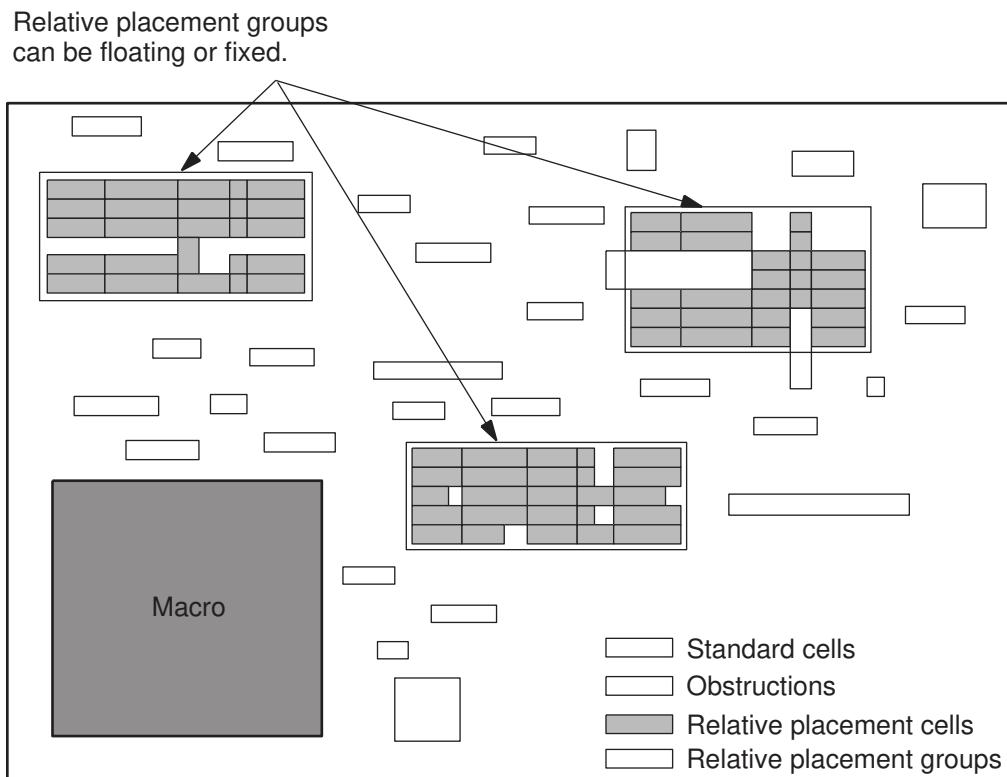
---

### Introduction to Physical Datapath With Relative Placement

Relative placement is usually applied to datapaths and registers, but you can apply it to any cell in your design, controlling the exact relative placement topology of gate-level logic groups and defining the circuit layout. You can use relative placement to explore QoR benefits, such as shorter wire lengths, reduced congestion, better timing, skew control, fewer vias, better yield, and lower dynamic and leakage power.

The relative placement constraints that you create and annotate implicitly generate a matrix structure of the instances and control the placement of the instances. You use the resulting annotated netlist for physical optimization, during which the tool preserves the structure and places it as a single entity or group, as shown in [Figure 151](#).

*Figure 151 Relative Placement in a Floorplan*



## Benefits of Relative Placement

Along with being technology-independent and having the ability to improve routability, relative placement provides the following benefits:

- Reduces the placement search space in critical areas of the design, which improves the predictability of QoR (wire length, timing, power, area) and congestion.
- Maintains relative placement during placement, optimization, clock tree synthesis, and routing.
- Provides a method for maintaining structured placement for legacy or intellectual property (IP) designs.

- Handles flat and hierarchical designs.
- Allows sizing of relative placement cells while maintaining relative placement.

---

## Relative Placement Flow

The relative placement flow consists of the following steps:

1. Prepare the design as described in [Preparing the Design](#).
2. Specify placement constraints as described in [Setting Up Multivoltage Designs](#).
3. Define the relative placement constraints and settings.
  - a. Create the relative placement groups by using the `create_rp_group` command, as described in [Creating Relative Placement Groups](#).
  - b. Add relative placement objects to the groups by using the `add_to_rp_group` command.  
See [Adding Objects to a Group](#).
  - c. Specify options for the relative placement groups by using the `set_rp_group_options` command, as described in [Specifying Options for Relative Placement Groups](#).
4. Perform placement and optimization as described in .
5. Analyze the relative placement results as described in [Analyzing Relative Placement Groups](#).

If the relative placement is not what you want, modify the relative placement group or the constraints and settings, and rerun placement and optimization.

---

## Creating Relative Placement Groups

A relative placement group is an association of cells, other relative placement groups, and blockages. A group is defined by the number of rows and columns it uses.

Use the `create_rp_group` command to create a relative placement group. When you do so, you must specify a name for the relative placement group by using the `-name` option.

You can specify the number of columns and rows for the relative placement group by using the `-columns` and `-rows` options. If you do not do so, the tool create a relative placement group with one row and column.

For example, to create a relative placement group named RP1 that has six columns and six rows, use the following command:

```
fc_shell> create_rp_group -name RP1 -columns 6 -rows 6
```

[Figure 152](#) shows the positions of columns and rows in a relative placement group.

*Figure 152 Relative Placement Column and Row Positions*

row 5	0 5	1 5	2 5	3 5	4 5	5 5
row 4	0 4	1 4	2 4	3 4	4 4	5 4
row 3		1 3	2 3	3 3	4 3	5 3
row 2	0 2	1 2	2 2	3 2	4 2	5 2
row 1	0 1	1 1	2 1	3 1		5 1
row 0	0 0	1 0	2 0	3 0	4 0	5 0
	col 0	col 1	col 2	col 3	col 4	col 5

For the relative placement group in [Figure 152](#),

- The column count begins from column 0 (the leftmost column).
- The row count begins from row 0 (the bottom row).
- The width of a column is the width of the widest cell in that column.
- The height of a row is the height of the tallest cell in that row.
- All positions in the structure are not used. For example, positions 0 3 (column 0, row 3) and 4 1 (column 4, row 1) are not used.

By default, the tool creates the relative placement group in the current design. You can create it in a different design by specifying its name using the `-design` option. If you create a relative placement group in a design that has multiple instantiations in a top-level design, the changes you make to the relative placement group in one instance is reflected in all its instances.

To remove relative placement groups, use the `remove_rp_groups` command.

---

## Adding Objects to a Group

After you create a relative placement group by using the `create_rp_group` command, you can add the following types of objects to it by using the `add_to_rp_group` command:

- Leaf cells, as described in [Adding Leaf Cells](#)
- Leaf cells, as described in [Adding Hard Macro Cells](#)
- Relative placement groups, as described in [Adding Relative Placement Groups](#)
- Blockages, as described in [Adding Blockages](#)

When you add an object to a relative placement group,

- The relative placement group to which you are adding the object must exist.
- The object must be added to an empty location in the relative placement group.
- Only one object can be added in one location of relative placement group.

To remove objects from a relative placement group, use the `remove_from_rp_group` command. You can remove leaf cells (`-cells`), relative placement groups (`-rp_group`), and blockages (`-blockage`).

When you remove objects from a group, the space previously occupied by the removed objects is left unoccupied.

---

## Adding Leaf Cells

To add leaf cells, including physical only cells, to a relative placement group, use the `-cells` option with the `add_to_rp_group` command. Specify the column and row position within the relative placement group at which to add the cell by using the `-column` and `-row` options.

In a relative placement group, a leaf cell can occupy multiple column positions or multiple row positions, which is known as leaf cell straddling. To specify the number of columns and rows the cells occupies, use the `-num_columns` and `-num_rows` options respectively. If you do not set these options, the default is 1 for both the number of columns and rows.

For example, to add a leaf cell that occupies two columns and one row at position (0,0), use

```
fc_shell> add_to_rp_group rp1 -cells U23 \
 -column 0 -row 0 -num_columns 2 -num_rows 1
```

Straddling is for leaf cells only, and not for hierarchical groups or blockages.

## Specifying Orientations for Leaf Cells

You can specify orientations for leaf cells when you add them to a relative placement group. If you do not specify a leaf cell orientation, the tool automatically assigns a legal orientation for the leaf cells.

To specify the orientation for leaf cells, use one of the following two methods:

- Use the `-orientation` option with a list of possible orientations when you add the cells to the group with the `add_to_rp_group` command.
- Set the `rp_orientation` attribute on leaf cells by using the `set_attribute` command.

The tool chooses one legal orientation from the list of orientations that you provide.

---

## Adding Hard Macro Cells

The `add_to_rp_group` command supports hard macro cells. When you add macro cells to a relative placement group, you cannot specify

- A pin name with which to align the macro cell by using the `-pin` option
- An orientation for the macro cells by using the `-orientation` option

When you add hard macro cells, use the following steps:

1. Create the relative placement groups, add the cells, including the hard macro cells, and specify the relative placement options and settings.
2. Place the design by using the `create_placement -floorplan` command.
3. Fix the placement of the hard macro cells in the relative placement groups by using the `set_placement_status fixed` command.

Before you can perform further placement and optimization,

- The hard macro cells in the relative placement groups must be placed and fixed
- The other cells in the relative placement groups that contain hard macro cells must be placed

4. Perform further placement and optimization.

---

## Adding Relative Placement Groups

Hierarchical relative placement allows relative placement groups to be embedded within other relative placement groups. The embedded groups then are handled similarly to leaf cells.

You can use hierarchical relative placement to simplify the expression of relative placement constraints. With hierarchical relative placement, you do not need to provide relative placement information multiple times for a recurring pattern.

Using hierarchical relative placement provides these benefits:

- Allows you to organize your relative placement in a manner that is easier to maintain and understand. For example, you can create the relative placement group to parallel your Verilog or VHDL organization.
- Allows reuse of a repeating placement pattern, such as an adder.
- Can reduce the number of lines of relative placement information you need to write.
- Allows integrating blocks.
- Provides flexibility for the configuration you want.

## Creating Hierarchical Relative Placement Groups

To create a hierarchical relative placement group by adding a group to another group, use the `-rp_group` option with the `add_to_rp_group` command. Specify the column and row position within the relative placement group by using the `-column` and `-row` options.

The group you specify with the `-rp_group` option must be in the same design as the hierarchical group in which you are including it.

When you include a relative placement group in a hierarchical group, it is as if the included group is directly embedded within its parent group. An included group can be used only in a group of the same design and only one time. However, a group that contains an included group can be further included in another group in the same design or can be instantiated in a group of a different design.

The script in [Example 41](#) creates a hierarchical group (rp4) that contains three included groups (rp1, rp2, and rp3). Groups rp1, rp2, rp3, and rp4 are all in the design top. The contents of groups rp1, rp2, and rp3 are treated as leaf cells when they are included in group rp4. You can further include group rp4 in another group in the design top, or you can instantiate group rp4 in a group of a different design.

The resulting hierarchical relative placement group is shown in [Figure 153](#).

### *Example 41 Including Groups in a Hierarchical Group*

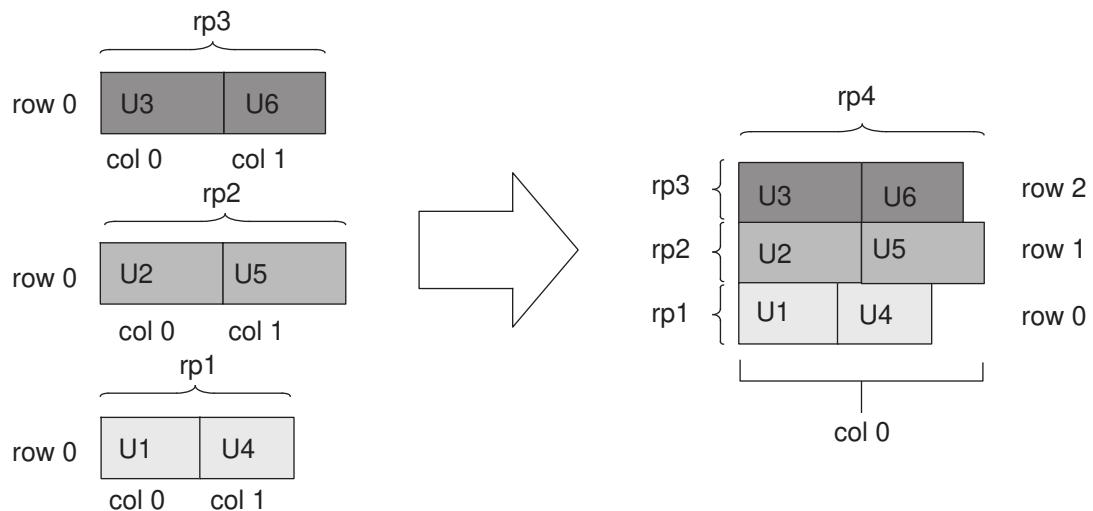
```
create_rp_group -name rp1 -columns 2 -rows 1
add_to_rp_group rp1 -cells U1 -column 0 -row 0
add_to_rp_group rp1 -cells U4 -column 1 -row 0

create_rp_group -name rp2 -columns 2 -rows 1
add_to_rp_group rp2 -cells U2 -column 0 -row 0
add_to_rp_group rp2 -cells U5 -column 1 -row 0
```

```
create_rp_group -name rp3 -columns 2 -rows 1
add_to_rp_group rp3 -cells U3 -column 0 -row 0
add_to_rp_group rp3 -cells U6 -column 1 -row 0

create_rp_group -name rp4 -columns 1 -rows 3
add_to_rp_group rp4 -rp_group rp1 \
 -column 0 -row 0
add_to_rp_group rp4 -rp_group rp2 \
 -column 0 -row 1
add_to_rp_group rp4 -rp_group rp3 \
 -column 0 -row 2
```

*Figure 153 Including Groups in a Hierarchical Group*

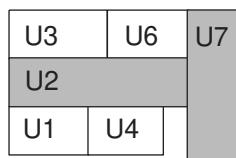


## Using Hierarchical Relative Placement for Straddling

A cell can occupy multiple column positions or multiple row positions, which is known as straddling. For more information about leaf cell straddling, see [Adding Leaf Cells](#).

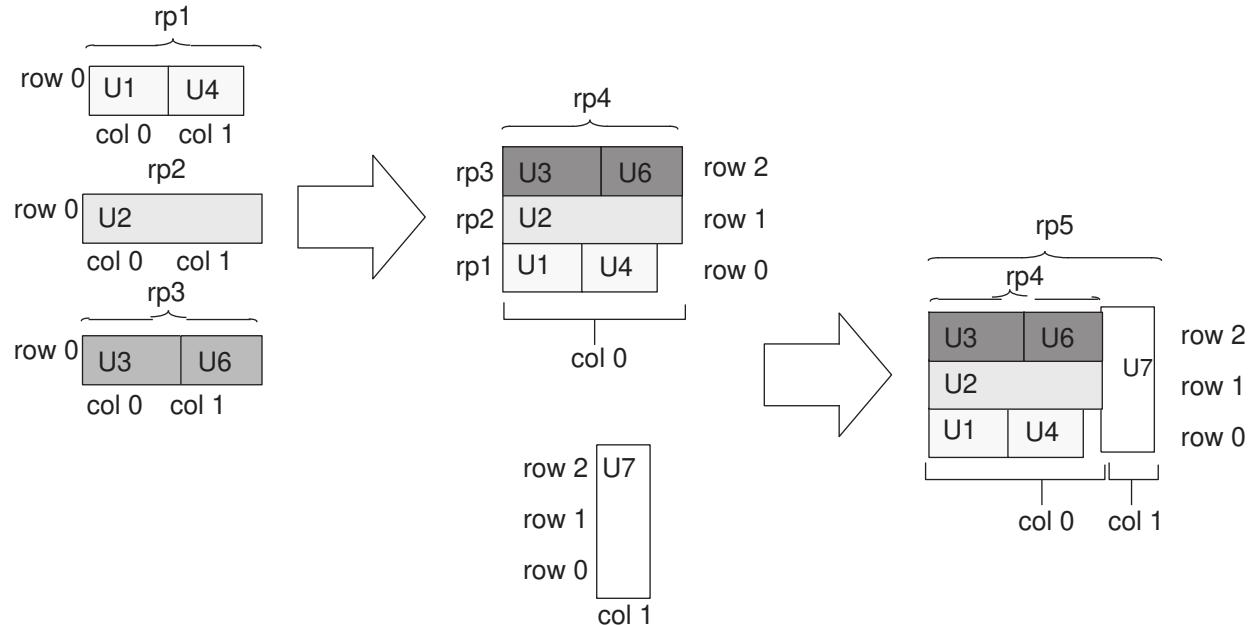
[Figure 154](#) shows a relative placement group in which cells straddle columns (instance U2) and rows (instance U7).

*Figure 154 Hierarchical Relative Placement Group With Straddling*



[Figure 155](#) shows the process of using hierarchical relative placement to build this structure. First, define relative placement groups that contain the leaf cells: rp1 contains U1 and U4, rp2 contains U2, and rp3 contains U3 and U6. Then define a group (rp4) that contains these groups. Finally, define a group (rp5) that contains the hierarchical group rp4 and the leaf cell U7. The resulting group includes both the column and the row straddle. [Example 42](#) shows the commands used in this process.

*Figure 155 Straddling With Hierarchical Relative Placement*



*Example 42 Straddling With Hierarchical Relative Placement*

```

create_rp_group -name rp1 -columns 2 -rows 1
add_to_rp_group rp1 -cells U1 -column 0 -row 0
add_to_rp_group rp1 -cells U4 -column 1 -row 0

create_rp_group -name rp2 -columns 1 -rows 1
add_to_rp_group rp2 -cells U2 -column 0 -row 0

create_rp_group -name rp3 -columns 2 -rows 1
add_to_rp_group rp3 -cells U3 -column 0 -row 0
add_to_rp_group rp3 -cells U6 -column 1 -row 0

create_rp_group -name rp4 -columns 1 -rows 3
add_to_rp_group rp4 -rp_group rp1 -column 0 -row 0
add_to_rp_group rp4 -rp_group rp2 -column 0 -row 1
add_to_rp_group rp4 -rp_group rp3 -column 0 -row 2

create_rp_group -name rp5 -columns 2 -rows 1

```

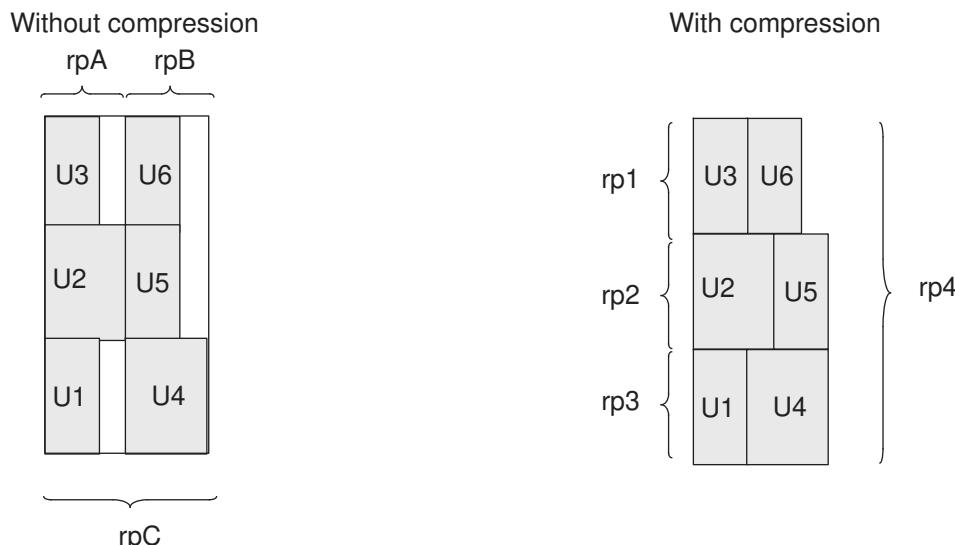
```
add_to_rp_group rp5 -rp_group rp4 -column 0 -row 0
add_to_rp_group rp5 -cells U7 -column 1 -row 0
```

## Using Hierarchical Relative Placement for Compression

By default, construction for relative placement aligns cells from their bottom-left corner. Compression removes empty space in rows to create a more compact structure. The columns are no longer aligned, and utilization is higher in the area of the compressed cells.

[Figure 156](#) shows the same cells aligned with and without compression. To create the compressed structure shown in this example, first create three relative placement groups, rp1, rp2, and rp3, that contains a row of leaf cells. Then create a group, rp4, that contains all these groups. [Example 43](#) shows the commands used to build the compressed structure.

*Figure 156 Bottom-Left Alignment Construction and Compression*



*Example 43 Compression With Hierarchical Relative Placement*

```
create_rp_group -name rp1 -columns 2 -rows 1
add_to_rp_group rp1 -cells U1 -column 0 -row 0
add_to_rp_group rp1 -cells U4 -column 1 -row 0

create_rp_group -name rp2 -columns 2 -rows 1
add_to_rp_group rp2 -cells U2 -column 0 -row 0
add_to_rp_group rp2 -cells U5 -column 1 -row 0

create_rp_group -name rp3 -columns 2 -rows 1
add_to_rp_group rp3 -cells U3 -column 0 -row 0
add_to_rp_group rp3 -cells U6 -column 1 -row 0
```

```
create_rp_group -name rp4 -columns 1 -rows 3
add_to_rp_group rp4 -rp_group rp1 -column 0 -row 0
add_to_rp_group rp4 -rp_group rp2 -column 0 -row 1
add_to_rp_group rp4 -rp_group rp3 -column 0 -row 2
```

Alternatively, you can apply compression in the horizontal direction by using the `-tiling_type` option with the `set_rp_group_options` command, as described in [Controlling the Tiling Within Relative Placement Groups](#).

## Adding Blockages

To add a blockage within relative placement groups, use the `-blockage` option with the `add_to_rp_group` command.

When you add a blockage using this command, you can specify

- The column and row position within the relative placement group by using the `-column` and `-row` options.

If you do not specify a position, the tool adds the blockage to position (0,0).

- The size of the blockage by using the `-height` and `-width` options.

If you do not specify the `-height` or `-width` option, the tool determines the size based on the tiling type of the relative placement group as follows:

- For a tiling type setting of `bit_slice`, the height default to the height of site row and the width to width of the column.
- For a tiling type setting of `compression`, the height default to the height of site row and the width to the width of one site.
- That the blockages can overlap with other relative placement blockages or other objects that are not relative placement cells by using the `-allow_overlap` option.

The following example adds a blockage named `gap1` to the `rp1` relative placement group at position (0,2) that is one site row high and five site rows wide:

```
fc_shell> add_to_rp_group rp1-blockage gap1 \
 -column 0 -row 2 -width 5 -height 1
```

## Adding Cells Within a Predefined Relative Placement Area

To add cells anywhere within a predefined area of a relative placement group, use the `-free_placement` option with the `add_to_rp_group` command. With this option, you must specify

- The origin of the placement area by using the `-column` and `-row` options
- The height and width of the placement area by using the `-height` and `-width` options
- The number of columns and rows the cells occupy by using the `-num_columns` and `-num_rows` options

For example, to add a leaf cell that occupies two columns and two rows to a placement area with a height of 4 and a width of 5 and is at position (0,0), use

```
fc_shell> add_to_rp_group rp2 -cells U41 -free_placement \
 -column 0 -row 0 -height 4 -width 5 -num_columns 2 -num_rows 2
```

## Specifying Options for Relative Placement Groups

To specify properties of relative placement groups, use the `set_rp_group_options` command as described in [Table 55](#).

*Table 55 Specifying Relative Placement Group Properties*

To do this	Use this option
Specify the anchor location. See <a href="#">Anchoring Relative Placement Groups</a> .	<code>-x_offset</code> <code>-y_offset</code>
Specify a corner for the anchor point set by the <code>-x_offset</code> and <code>-y_offset</code> options. See <a href="#">Anchoring Relative Placement Groups</a> .	<code>-anchor_corner</code>
Specify the type of alignment used by the cells in the group. See <a href="#">Aligning Leaf Cells Within a Column</a> .	<code>-alignment</code>
Specify the group alignment pin name, when using pin alignment. See <a href="#">Aligning by Pin Location</a> .	<code>-pin_name</code>
Specify a tiling type. See <a href="#">Controlling the Tiling Within Relative Placement Groups</a> .	<code>-tiling_type</code>
Specify the orientation. See <a href="#">Specifying the Orientation of Relative Placement Groups</a> .	<code>-group_orientation</code>
Specify the utilization (default is 100 percent).	<code>-utilization</code>

**Table 55 Specifying Relative Placement Group Properties (Continued)**

To do this	Use this option
Specify how to treat fixed cells in the floorplan during legalization. See <a href="#">Handling Fixed Cells During Relative Placement</a> .	<code>-place_around_fixed_cells</code>
Control the type of optimization that is allowed for relative placement cells See <a href="#">Controlling the Optimization of Relative Placement Cells</a> .	<code>-optimization_restriction</code>
Control the movement of the group during legalization. See <a href="#">Controlling Movement When Legalizing Relative Placement Groups</a> .	<code>-move_effort</code>

To remove relative placement group option settings, use the `remove_rp_group_options` command. You must specify the group name and at least one option; otherwise, this command has no effect.

## Anchoring Relative Placement Groups

By default, the Fusion Compiler tool can place a relative placement group anywhere within the core area. You can control the placement of a top-level relative placement group by anchoring it.

To anchor a relative placement group, use the `set_rp_group_options` command with the `-x_offset` and `-y_offset` options. The offset values are float values, in microns, relative to the lower-left corner in the core area.

If you specify both the x- and y-coordinates, the group is anchored at that location. If you specify only one coordinate, the Fusion Compiler tool determines the placement by maintaining the specified coordinate and sliding the group along the line passing through the unspecified coordinate.

To specify a corner of relative placement group to anchor it by, use the `-anchor_corner` option. The tool places the relative placement group such that the corner specified by this option is placed on the anchor point specified by the `-x_offset` and `-y_offset` options.

The settings for the `-anchor_corner` option are as follows, and are shown in [Figure 157](#):

- `bottom_left`

The anchor point of the relative placement group is set to its bottom-left corner. The default is the bottom-left corner.

- `bottom_right`

The anchor point of the relative placement group is set to its bottom-right corner.

- `top_left`

The anchor point of the relative placement group is set to its top-left corner.

- `top_right`

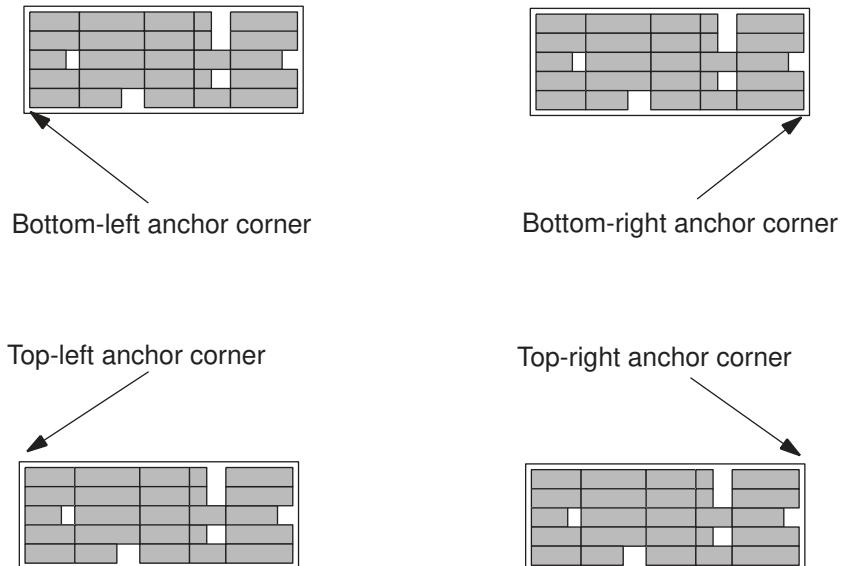
The anchor point of the relative placement group is set to its top-right corner.

- `rp_location`

The anchor point of the relative placement group is the element in the relative placement group at the position specified by the `-anchor_row` and `-anchor_column` options.

When you use the `-anchor_corner rp_location` setting, the position specified with the `-anchor_row` and `-anchor_column` options must contain a cell, keepout, or a relative placement hierarchy.

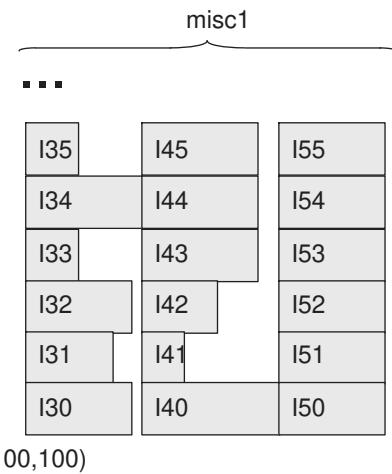
*Figure 157 Bottom-Left, Bottom-Right, Top-Left and Top-Right Anchor Corners*



For example, to anchor a relative placement by its bottom left corner at location (100, 100), as shown [Figure 158](#), use the following command:

```
fc_shell> set_rp_group_options misc1 -anchor_corner bottom_left \
-x_offset 100 -y_offset 100
```

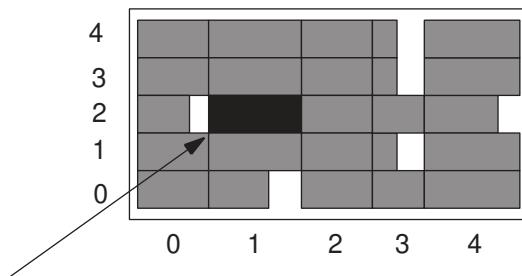
*Figure 158 Anchored Relative Placement Group*



The following example specifies that relative placement cell at column 1, row 2 of the RP1 relative placement group should be anchored at location (100, 100).

```
fc_shell> set_rp_group_options RP1 \
 -anchor_corner rp_location -anchor_column 1 -anchor_row 2 \
 -x_offset 100 -y_offset 100
```

*Figure 159 Using an Object Within the Relative Placement Group for Anchoring*



The relative placement group is anchored by placing the cell at column 1 row 2 at location (100, 100)

## Aligning Leaf Cells Within a Column

You can align the leaf cells in a column of a relative placement group by using the following alignment methods:

- Left alignment (default)
- Right alignment
- Pin alignment

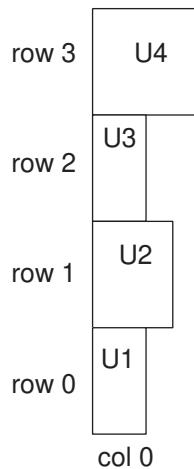
Controlling the cell alignment can improve the timing and routability of your design.

## Aligning by the Left Edges

By default, the Fusion Compiler tool aligns the leaf cells by aligning the left edges. To explicitly specify this alignment method, use the `-alignment left` option of the `set_rp_group_options` command.

[Figure 160](#) shows cells that are left aligned.

*Figure 160 Bottom-Left-Aligned Relative Placement Group*



## Aligning by the Right Edges

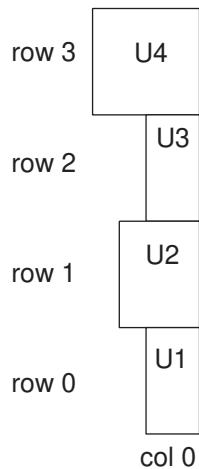
To align a group by aligning the right edges, use the `-alignment right` option of the `set_rp_group_options` command.

### Note:

For hierarchical relative placement groups, the bottom-right alignment does not propagate through the hierarchy.

[Figure 160](#) shows cells that are right aligned.

*Figure 161 Bottom-Right-Aligned Relative Placement Group*



## Aligning by Pin Location

To align a group by pin location, use the `-alignment pin` and `-pin_name` options with the `set_rp_group_options` command.

The tool looks for the specified alignment pin in each cell in the column. If the alignment pin exists in a cell, the cell is aligned by using the pin location. If the specified alignment pin does not exist in a cell, the cell is aligned by the left edge, and the tool issues an information message. If the specified alignment pin does not exist in any cell in the column, the Fusion Compiler tool issues a warning message.

The script in [Example 44](#) creates a relative placement group rp1, adds cells to it, and specifies that the cells are aligned by pin A.

### *Example 44 Definition for Relative Placement Group Aligned by Pins*

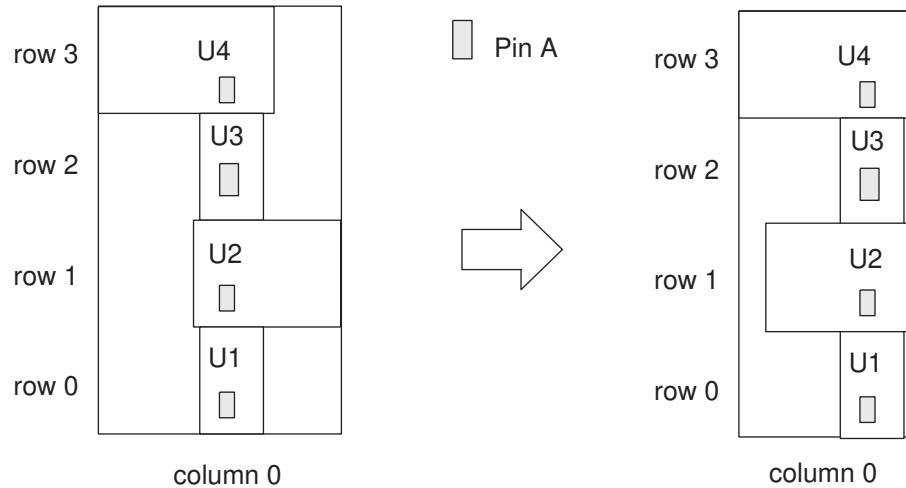
```

create_rp_group -name rp1 -columns 1 -rows 4
set_rp_group_options -alignment pin -pin_name A
add_to_rp_group rp1 -cells U1 -column 0 -row 0
add_to_rp_group rp1 -cells U2 -column 0 -row 1
add_to_rp_group rp1 -cells U3 -column 0 -row 2
add_to_rp_group rp1 -cells U4 -column 0 -row 3

```

When aligning by pins, the tool tries different orientations for the cells and selects the orientation for each cell that gives the minimum column width. For example, changing the orientation of cell U2, as shown in [Figure 162](#), reduces the width of column 0. However, if you specify an orientation when adding a cell to a relative placement group by using the `-cells` and `-orientation` options with the `add_to_rp_group` command, the tool honors the orientation you specify.

*Figure 162 Minimizing the Column Width of a Relative Placement Group Aligned by Pins*



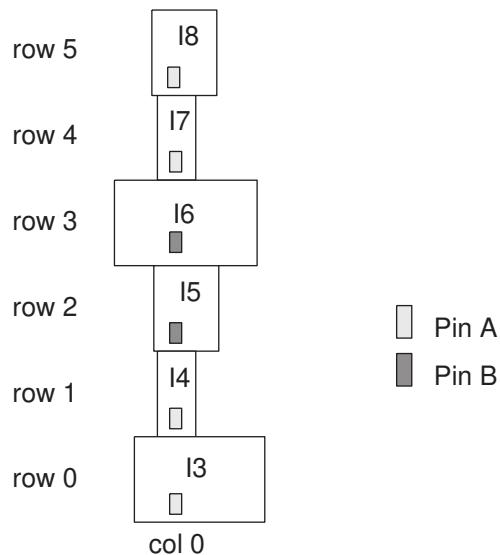
When you specify an alignment pin for a group, the pin applies to all cells in the group. You can override the group alignment pin for specific cells in the group by specifying the `-pin_name` option when you use the `add_to_rp_group` command to add the cells to the group.

The script in [Example 45](#) defines relative placement group rp2, and specified pin A as the group alignment pin. However, instances I5 and I6 use pin B as their alignment pin, rather than the group alignment pin. The resulting structure is shown in [Figure 163](#).

**Example 45 Definition for Aligning a Group and Leaf Cells by Pins**

```
create_rp_group -name rp2 -columns 1 -rows 6
set_rp_group_options rp2 -alignment pin -pin_name A
add_to_rp_group rp2 -cells I3 -column 0 -row 0
add_to_rp_group rp2 -cells I4 -column 0 -row 1
add_to_rp_group rp2 -cells I5 -column 0 -row 2 -pin_name B
add_to_rp_group rp2 -cells I6 -column 0 -row 3 -pin_name B
add_to_rp_group rp2 -cells I7 -column 0 -row 4
add_to_rp_group rp2 -cells I8 -column 0 -row 5
```

*Figure 163 Relative Placement Group Aligned by Different Pins*



## Overriding the Alignment When Adding Objects

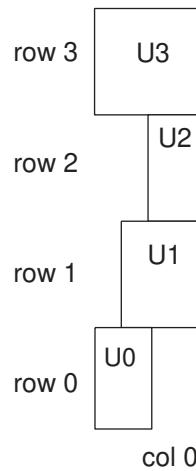
When you add an object to a relative placement group by using the `add_to_rp_group` command, you can override its alignment and specify a different alignment for the object you are adding by using the `-override_alignment` option. However, if the relative placement group is pin aligned, you cannot override the alignment with the `-override_alignment` option.

The following example creates a relative placement group named rp1 that is right aligned. It then adds a cell named U0, which overrides the alignment of the group and cells named U1, U2, and U3, which honor the alignment of the relative placement group:

```
fc_shell> create_rp_group rp1 -name rp1 -columns 1 -rows 4
fc_shell> set_rp_group_options rp1 -alignment right
fc_shell> add_to_rp_group rp1 -cells U0 -column 0 -row 0 \
 -override_alignment left
fc_shell> add_to_rp_group rp1 -cells U1 -column 0 -row 1
fc_shell> add_to_rp_group rp1 -cells U2 -column 0 -row 2
fc_shell> add_to_rp_group rp1 -cells U3 -column 0 -row 3
```

The following figure shows the relative placement group after placement.

*Figure 164 Right-Aligned Relative Placement Group With One Cell That is Left Aligned*



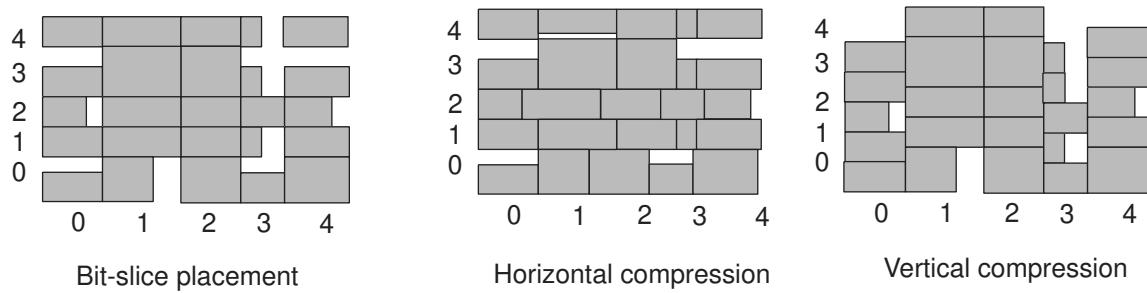
## Controlling the Tiling Within Relative Placement Groups

To control the tiling of objects within a relative placement group, use the `set_rp_group_options -tiling_type` command with the settings shown in the following table.

*Table 56 Controlling Placement With the set\_rp\_group\_options -tiling\_type Command*

To do this	Use this setting
Tile objects in a bit-slice pattern and preserve both the row and column alignment.	<code>bit_slice</code> (Default)
<ul style="list-style-type: none"> <li>The height of a row is determined by the tallest object in the row</li> <li>The width of a column is determined by the widest object in the column</li> </ul>	
Horizontally compress the objects in each row and maintain row alignment.	<code>horizontal_compression</code>
Vertically compress the objects in each column and preserve the column alignment	<code>vertical_compression</code>

*Figure 165 Bit-Slice Placement Versus Horizontal or Vertical Compression*



The setting of the `-tiling_type` option is not propagated from a parent group to child groups.

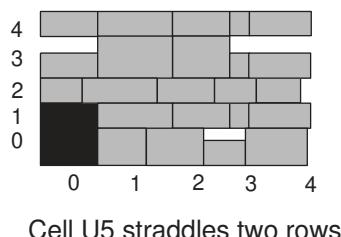
## Applying Compression to Groups With Straddling Leaf Cells

You can apply compression to a relative placement group with cells that straddle multiple rows or columns, as shown in the following example:

```
fc_shell> add_to_rp_group rp -cells U5 \
 -column 0 -row 0 -num_columns 1 -num_rows 2
fc_shell> set_rp_group_options rp -tiling_type horizontal_compression
```

[Figure 166](#) shows the placement of the relative placement group in the previous example.

*Figure 166 Compression of a Relative Placement Group With a Cell That Straddles Multiple Rows*



For information about adding cells that straddle multiple rows or columns to a relative placement group, see [Adding Leaf Cells](#).

---

## Specifying the Orientation of Relative Placement Groups

The Fusion Compiler tool supports the following orientations for relative placement groups:

- R0

The column position of the relative placement group is from left to right, and the row position is from bottom to top.

- R180

The column position of the relative placement group is from right to left, and the row position is from top to bottom.

- MY

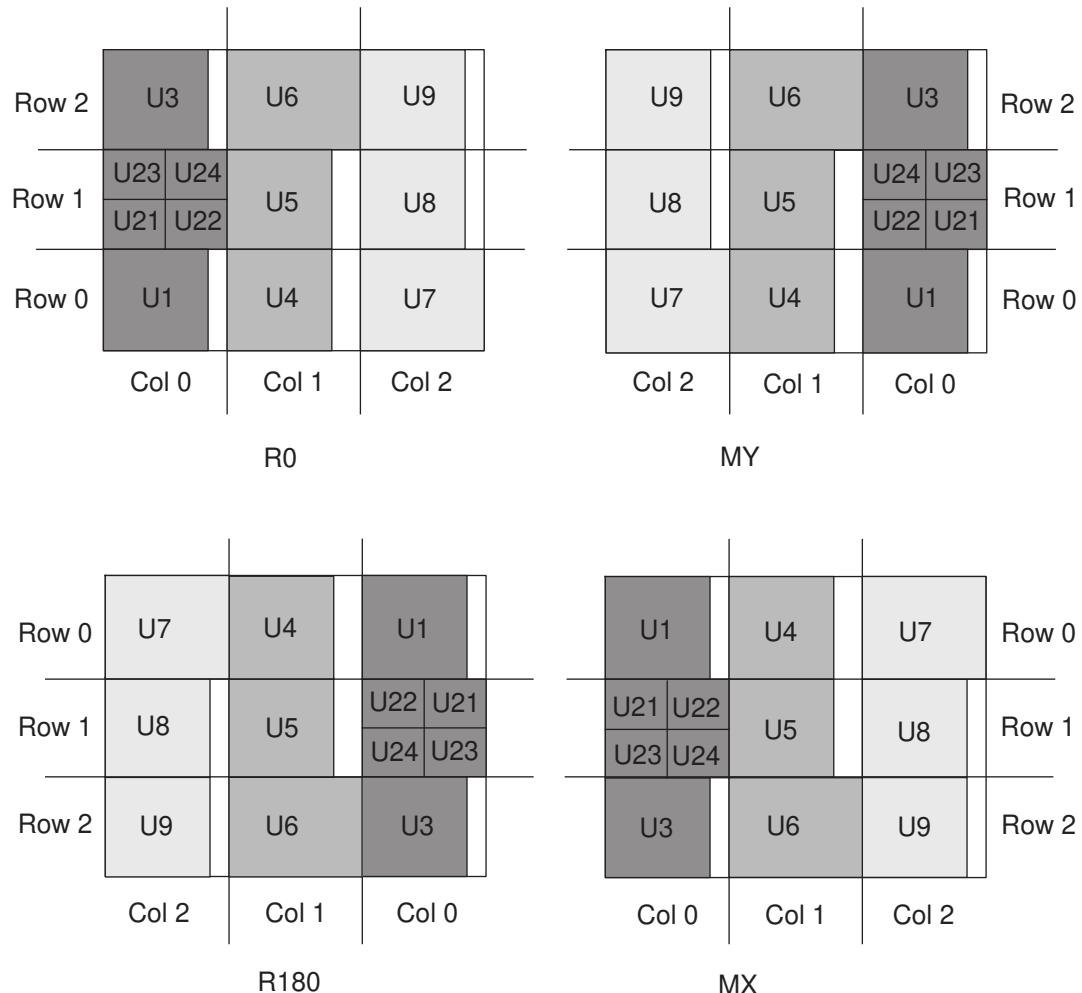
The column position of the relative placement group is from right to left, and the row position is from bottom to top; that is, the orientation of the group is flipped with respect to the R0 orientation.

- MX

The column position of the relative placement group is from left to right, and the row position is from top to bottom; that is, the group is flipped with respect to the R180 orientation.

[Figure 167](#) shows how the column and row positions in a relative placement group are placed for the four orientations.

*Figure 167 Orientation of Relative Placement Groups*



The orientation of relative placement groups is automatically set by the tool to minimize wire length. You can also choose to set the orientation of relative placement groups by using the `set_rp_group_options` command.

For example, the following command sets the relative placement group orientation to MY.

```
fc_shell> set_rp_group_options [get_rp_groups design::rp] \
-group_orientation MY
```

For designs with hierarchical relative placement groups, the orientation settings are propagated down to the lowest level in hierarchy.

**Note:**

When the orientation of a relative placement group is changed, the constraints on the relative placement group, such as alignment and utilization, are preserved according to the specifications that you provide.

## Specifying a Keepout Margin

To prevent other relative placement groups from being placed close to a specific relative placement group, you can specify a keepout margin that applies only to other relative placement groups. To do so, use the `-rp_only_keepout_margin` option with the `set_rp_group_options` command. You can specify a different margin for the left, bottom, right, and top sides of the group.

The following command applies a margin of 15 on the left and right and a margin of 10 on the top and bottom of the relative placement group named rp1:

```
fc_shell> set_rp_group_options rp1 \
 -rp_only_keepout_margin {15 10 15 10}
```

## Handling Fixed Cells During Relative Placement

To specify how to handle fixed cells in the floorplan during legalization of relative placement groups, use the `-place_around_fixed_cells` option with the `set_rp_group_options` command. [Table 57](#) shows the different settings you can specify for the `-place_around_fixed_cells` option.

*Table 57     Settings for the `-place_around_fixed_cells` Option*

To do this	Use this setting
Legalize relative placement groups around fixed standard cells and avoid fixed physical-only cells.	standard
Legalize relative placement groups around fixed physical-only cells and avoid fixed standard cells.	physical_only
Legalize relative placement groups around both fixed standard cells and fixed physical-only cells. This is the default.	all
Avoid both fixed standard cells and fixed physical-only cells.	none

For hierarchical relative placement groups, you can use the `set_rp_group_options -place_around_fixed_cells` command and specify different settings for the top-level and the lower-level relative placement groups. If you do not specify a setting for the lower-level placement groups, the value of the top-level relative placement group is used for the lower-level relative placement groups.

Assume a hierarchical relative placement group named top-rp contains three lower-level relative placement groups named rp1, rp2, and rp3. The following example specifies a less restrictive setting for the top level and a more restrictive setting for the lower-level group named rp1. It also specifies that the setting for the rp1 lower-level group overrides the top-level setting when legalizing the cells in the rp1 group.

```
fc_shell> set_rp_group_options top-rp \
-place_around_fixed_cells physical_only
fc_shell> set_rp_group_options rp1 \
-place_around_fixed_cells none
fc_shell> set_app_options -name set_rp_group_options -value true
```

---

## Allowing Nonrelative Placement Cells

By default, nonrelative placement cells are

- Not allowed within relative placement groups during coarse placement
- Allowed within relative placement groups during optimization and legalization

To reduce congestion and improve QoR, you can allow the tool to place nonrelative placement cells within the unused areas of a specific relative placement group by using the `-allow_non_rp_cells` option with the `set_rp_group_options` command.

You can add blockages to relative placement group by using the `-blockage` option with the `add_to_rp_group` command. By default, no cells are allowed within these blockages during placement, optimization, and legalization. To allow nonrelative placement cells within relative placement blockages and unused areas of a specific relative placement group, use the `-allow_non_rp_cells_on_blockages` option with the `set_rp_group_options` command.

The following example allows nonrelative placement cells within the unused areas of the RP1 relative placement group:

```
fc_shell> set_rp_group_options rp1 \
-allow_non_rp_cells
```

The following example allows nonrelative placement cells within the blockages and unused areas of the RP2 relative placement group:

```
fc_shell> set_rp_group_options rp2 \
-allow_non_rp_cells_on_blockages
```

---

## Controlling the Optimization of Relative Placement Cells

When a relative placement cell is modified or moved, the relative placement structure can be disturbed. When a relative placement cell is removed during optimization, the relative

placement information of the instance is also removed, disrupting the relative placement structure.

To preserve the relative placement structures during various postplacement optimization processes, use the `-optimization_restriction` option with the `set_rp_group_options` command and specify the appropriate setting as shown in [Table 58](#).

*Table 58     Settings for the `-optimization_restriction` Option*

To do this	Use this setting
Allow unrestricted optimization of the relative placement cells	<code>all_opt</code>
Allow only sizing of the relative placement cells	<code>size_only</code>
Allow only in-place sizing of the relative placement cells	<code>size_in_place</code>
Prevent any optimization of the relative placement cells	<code>no_opt</code>

For a hierarchical relative placement group, the `-optimization_restriction` option setting applied to the top level is propagated to the lower-level groups and any settings applied to the lower-level groups are ignored.

---

## Controlling Movement When Legalizing Relative Placement Groups

You can control the movement of a relative placement group during legalization by using the `-move_effort` option of the `set_rp_group_options` command. During coarse placement, the tool estimates an initial location for every top-level relative placement group. The `-move_effort` option controls the extent to which a relative placement group can be moved from its initial location to preserve the relative placement without violating relative placement constraints. When you change the option setting from a higher effort level to a lower effort level, you reduce the size of the region searched for placement of a relative placement group.

---

## Handling Tap Columns and Relative Placement Group Overlaps

During coarse placement, to avoid the overlap of relative placement groups with the tap cell columns, use the `place.coarse.no_split_rp_width_threshold` application option. The overlap can be removed for only those relative placement groups whose width is less than the distance between tap columns. This application option helps the relative placement engine know about the distance between the tap columns.

This helps in better placement and structures of relative placement groups in the design. You must use this application option for n5 and older technical nodes. For n3 and beyond designs, the distance between tap columns is calculated automatically.

```
place.coarse.no_split_rp_width_threshold <value>
For example,
place.coarse.no_split_rp_width_threshold 46
```

## Changing the Structures of Relative Placement Groups

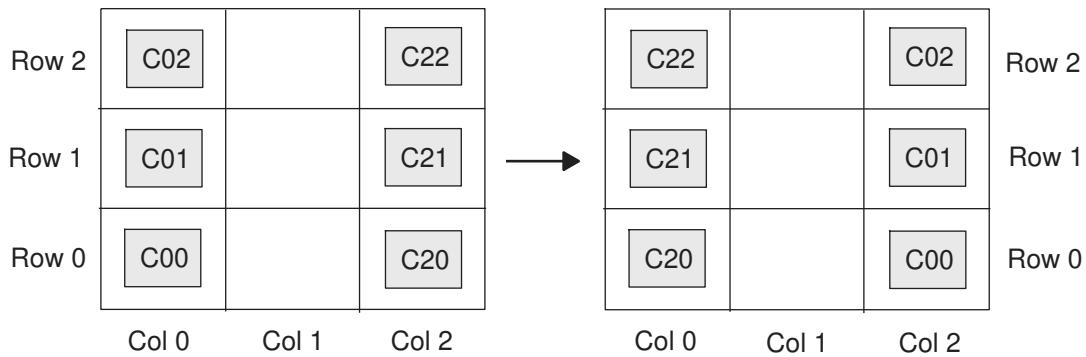
To modify the structures of existing relative placement groups, use the `modify_rp_groups` command as follows:

- To add a row or column to a relative placement group, use the `-add_rows` or `-add_columns` option respectively.
- To remove a row or column, use the `-remove_rows` or `-remove_columns` option respectively.
- To flip a row or column, use the `-flip_row` or `-flip_column` option respectively.
- To swap two rows or columns, use the `-swap_rows` or `-swap_columns` option respectively.

For example, to swap the first and third columns of the `my_rp_group` relative placement group, as shown in [Figure 168](#), enter

```
fc_shell> modify_rp_groups [get_rp_groups my_rp_group] \
 -swap_columns {0 2}
```

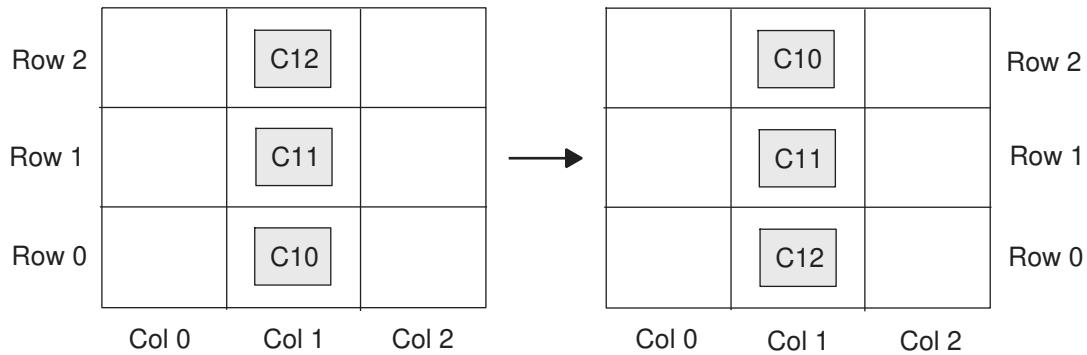
*Figure 168 Swapping Columns of a Relative Placement Group*



To flip the second column of the `my_rp_group` relative placement group, as shown in [Figure 169](#), enter

```
fc_shell> modify_rp_groups [get_rp_groups my_rp_group] \
 -flip_column 1
```

*Figure 169 Flipping a Column of a Relative Placement Group*



## Generating Relative Placement Groups for Clock Sinks

You can generate a relative placement group for clock sinks and their drivers in a placed design by using the `create_clock_rp_groups` command. When you run placement and optimization, the tool places the clocks sinks and their drivers based on their relative placement constraints it generated. This can improve routability and reduce dynamic power.

You can control the sinks being considered for relative placement groups as follows:

- Specify the minimum and maximum of sinks that should be driven by a single driver to be considered by using the `-min_sinks` and `-max_sinks` options.  
The default minimum is 2 sinks and the default maximum is 128 sinks.
- Exclude timing critical sinks by using the `-timing_driven` option.  
By default, the tool excludes all sinks with a negative slack.
- Specify the cells to consider by using the `-cells` option.  
By default, the tool considers all the sinks.

You can control the size and shape of the relative placement group by using one of the following methods:

- Specify the maximum allowed Manhattan distance between the sinks in one relative placement group by using the `-distance` option.  
 If the Manhattan distance between sinks is more than the specified distance, they are put into separate relative placement groups. The default distance is 100 microns.
- Specify a maximum number of rows by using the `-max_rp_rows` option.  
 The default is 32.
- Allow the tool to decide the number of rows and columns based on the distribution of the cells by using the `-auto_shape` option.

Before you run the `create_clock_rp_groups` command, the block must be placed. After you create the clock relative placement groups, reset the placement of the block by using the `reset_placement` command, and rerun placement and optimization by using the `place_opt` command.

## Performing Placement and Legalization of Relative Placement Groups

The following topics provide information related to the placement and legalization of relative placement groups:

- [Relative Placement in a Design Containing Obstructions](#)
- [Legalizing Relative Placement Groups in a Placed Design](#)
- [Creating New Relative Placement Groups in a Placed Design](#)

### Relative Placement in a Design Containing Obstructions

During placement, relative placement groups avoid placement blockages (obstructions) that are defined in the DEF file or created by the `create_placement_blockage` command. A relative placement group can be broken into pieces that straddle obstructions, yet maintain the relative placement structure.

If the height of the obstruction is below a certain threshold, the relative placement cells are shifted vertically; otherwise, the relative placement column is shifted horizontally.

[Figure 170](#) shows the placement of relative placement cells in a design containing obstructions that are either defined in the DEF file or created by `create_placement_blockage`. The obstruction in columns one and two is below the

threshold, so the tool shifts the cells vertically. The obstruction in column four is greater than the threshold, so the tool shifts all the cells of the column horizontally.

*Figure 170 Relative Placement in a Design Containing Obstructions*

	1 4	2 4		
row 4	0 4	1 3	2 3	3 4
row 3	0 3	1 2	2 2	3 3
row 2	0 2	Obstruction		3 2
row 1	0 1	1 1	2 1	3 1
row 0	0 0	1 0	2 0	3 0
	col 0	col 1	col 2	col 3
				col 4

## Legalizing Relative Placement Groups in a Placed Design

You can improve the placement of relative placement groups in a placed design by legalizing only the relative placement groups. To legalize the placement of only the relative placement groups, but not nonrelative placement cells, use the `legalize_rp_groups` command. You can also specify a list of relative placement groups to be legalized.

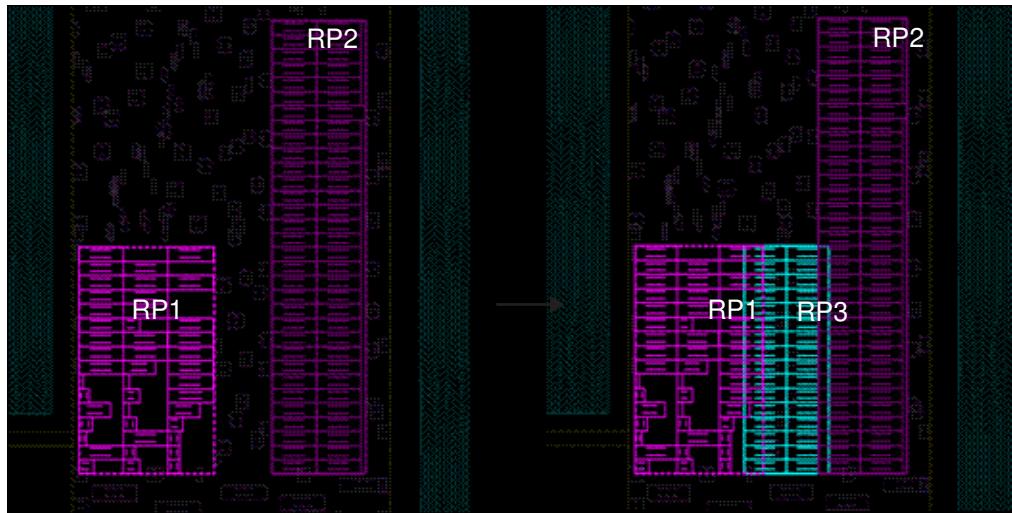
To perform a fast legalization of the relative placement groups, use the `-prototype` option. When you use this option, the tool does not ensure that all legalization constraints are met. Therefore, use it during the prototyping stages of the design flow when you are developing relative placement groups.

To specify that relative placement groups can overlap with each other, use the `-legalize_over_rp` option. For example, the following command legalizes the RP3 relative placement group over the RP1 and RP2 relative placement groups.

```
fc_shell> legalize_rp_groups -legalize_over_rp RP3
```

[Figure 171](#) shows the placement before and after running the command.

*Figure 171 Legalizing the RP3 Relative Placement Group Over Other Groups*



After you legalize one or more relative placement groups by using the `legalize_rp_groups` command, there might be overlaps with cells in other relative placement groups or cells that are not in relative placement groups. Use the `check_legality` command to identify any cell overlaps and use the `legalize_placement` command to resolve any remaining cell overlaps.

## Creating New Relative Placement Groups in a Placed Design

You can create a new relative placement group of cells in a placed design and place the relative placement groups incrementally by using the `place_opt -from final_place` command. If you select cells that are placed far apart in the initial placement for the same relative placement group, performing incremental relative placement might degrade the QoR.

The following example shows how to add a new relative placement group to a design that is already placed and optimized, and performs incremental placement and optimization:

```
fc_shell> create_rp_group -name new_rp -columns 1 -rows 2
fc_shell> add_to_rp_group new_rp -cells U1 -column 0 -row 0
fc_shell> add_to_rp_group new_rp -cells U2 -column 0 -row 1
...
fc_shell> place_opt -from final_place
```

For a placed design, if you create a new relative placement group and specify an anchor location by using the `-x_offset` and `-y_offset` options with the `set_rp_group_options` command, you can use the `legalize_rp_groups` command to legalizes the newly created group anchored by the specified x- and y-coordinates. Using the

`legalize_rp_groups` command for incremental relative placement groups reduces the turnaround time.

---

## Analyzing Relative Placement Groups

The following sections explain methods for analyzing your relative placement groups:

- [Checking Relative Placement Groups Before Placement](#)
  - [Analyzing the Placeability of a Relative Placement Group](#)
  - [Reporting Relative Placement Constraint Violations](#)
  - [Querying Relative Placement Groups](#)
  - [Analyzing Relative Placement in the GUI](#)
- 

### Checking Relative Placement Groups Before Placement

Before you run placement and optimization, you can check the relative placement constraints for issues that might lead to critical or noncritical failures after placement by using the `check_rp_constraints` command.

The following example checks for possible relative placement constraint violation in the group named `rp_volt1`:

```
fc_shell> check_rp_constraints rp_volt1

Report : Relative Placement Summary
Total number of specified top level relative placement groups: 1
Total number of relative placement groups which may not honor its
constraints: 1

RP Group: rp_volt1

Warning: The height of relative placement group 'rp_volt1' is more than
the height of voltage area or exclusive move bound. (RPGP-018)
```

## Analyzing the Placeability of a Relative Placement Group

Before you run placement and optimization, you can check if a specific relative placement group can be placed by using the `check_rp_constraints -analyze_placement` command.

- To limit the analysis to a specific region or a specific number of random sites of the core area, use the `-region` or `-trials` option, respectively.
- To ignore physical design constraints, advanced design rules, or relative placement constraints during placement analysis, use the `-no_pdc`, `-no_adv`, or `-no_rp_constraints` option, respectively.
- To report only the relative placement groups that do not meet a specific threshold, use the `-threshold` option. The tool reports a group only if the percentage of sites the group can be placed, relative to the total number of sites analyzed, is less than the specified threshold.

## Reporting Relative Placement Constraint Violations

After you run placement and optimization, use the `report_rp_groups` command to identify placement issues and relative placement violations. You must either specify which relative placement groups to analyze or specify the `-all` option to analyze all relative placement groups.

By default, the command reports the following types of relative placement groups:

- Placed groups that do not have constraint violations
- Placed groups that have constraint violations that are not critical
- Failed groups that have constraint violations that are critical
- Groups that have not yet been placed

You can modify the default behavior by using the options described in [Table 59](#).

*Table 59      The report\_rp\_groups Command Options*

To do this	Use this option
Report information only about the groups that are not placed due to critical relative placement violations	<code>-critical</code>
Report information only about the groups that are placed but do not meet their relative placement constraints	<code>-non_critical</code>
Report information only about the groups that have not yet been placed	<code>-unplaced</code>

*Table 59 The report\_rp\_groups Command Options (Continued)*

To do this	Use this option
Report detailed information	-verbose

You can also run this command before you run placement and optimization and identify the unplaced groups in the design.

## Querying Relative Placement Groups

To query relative placement groups that contain specific objects or attribute values, use the `get_rp_groups` command.

For example, the following command returns a collection consisting of all the relative placement groups:

```
fc_shell> get_rp_groups
{RP_TA RP_TCO RP_HO_1 RP_HO_2 RP_HO_3 RP_HO_4 RP_THO}
```

The following command returns only the top-level relative placement groups:

```
fc_shell> get_rp_groups -top
{RP_TA RP_TCO RP_THO}
```

The following command returns the relative placement group that contains the leaf cell named U129:

```
fc_shell> get_rp_groups -of_objects U129
{RP_HO_4}
```

---

## Analyzing Relative Placement in the GUI

The Fusion Compiler GUI provides tools to help you visualize and analyze the relative placement groups in your design:

- Relative Placement (RP) Groups Visual Mode
- Relative Placement (RP) Net Connection Visual Mode

For more information about analyzing relative placement groups in the GUI, see the *Using Map and Visual Modes* topic in the *Fusion Compiler Graphical User Interface User Guide*.

## Saving Relative Placement Information

The relative placement information is automatically saved in the design library database when you save the design by using the `save_lib` command.

You can also save the relative placement information to a file that contains Tcl commands that re-creates the relative placement groups, their objects, and their settings. To do so, use the `write_rp_groups -file_name` command. You must either specify which relative placement groups to write commands for or specify the `-all` option to write commands for all relative placement groups.

By default, the `write_rp_groups` command writes out commands for creating the specified relative placement groups and to add leaf cells, hierarchical groups, and blockages to these groups. The commands for generating subgroups within hierarchical groups are not written. You can modify the default behavior by using the options described in [Table 60](#).

*Table 60 The write\_rp\_groups Command Options*

To do this	Use this option
Write all the relative placement groups within the hierarchy of the relative placement groups. If you omit this option, only the top-level group is written and subgroups are not.	<code>-hierarchical</code>
Write only <code>create_rp_group</code> commands to the script.	<code>-create</code>
Write only <code>add_to_rp_group -cells</code> commands to the script.	<code>-cell</code>
Write only <code>add_to_rp_group -rp_group</code> commands to the script.	<code>-rp_group</code>
Write only <code>add_to_rp_group -blockage</code> commands to the script.	<code>-blockage</code>

## Summary of Relative Placement Commands

[Table 61](#) shows some of the key commands used to perform relative placement.

*Table 61 Relative Placement Commands*

Command	Described in section
<code>create_rp_group</code> and <code>remove_rp_groups</code>	<a href="#">Creating Relative Placement Groups</a>
<code>add_to_rp_group</code> and <code>remove_from_rp_group</code>	<a href="#">Adding Objects to a Group</a>

**Table 61      *Relative Placement Commands (Continued)***

<b>Command</b>	<b>Described in section</b>
<code>set_rp_group_options</code> and <code>remove_rp_group_options</code>	<a href="#">Specifying Options for Relative Placement Groups</a>
<code>modify_rp_groups</code>	<a href="#">Changing the Structures of Relative Placement Groups</a>
<code>legalize_rp_groups</code>	<a href="#">Legalizing Relative Placement Groups in a Placed Design</a>
<code>check_rp_constraints</code>	<a href="#">Checking Relative Placement Groups Before Placement</a>
<code>report_rp_groups</code>	<a href="#">Reporting Relative Placement Constraint Violations</a>
<code>get_rp_groups</code>	<a href="#">Querying Relative Placement Groups</a>
<code>write_rp_groups</code>	<a href="#">Saving Relative Placement Information</a>

# 11

## Hierarchical Implementation

---

The following topics describe how to perform synthesis, placement, optimization, clock tree synthesis, routing, and postroute optimization on hierarchical designs.

- [Performing Hierarchical Synthesis Using Abstracts](#)
- [Performing Hierarchical Synthesis Using ETMs](#)
- [Performing Prechecks for Hierarchical Synthesis Flows](#)
- [Providing Block-Level Test Models](#)
- [Specifying Block-Level Power Intent](#)
- [Overview of Abstract Views](#)
- [Creating Abstract Views](#)
- [Reporting Abstract Inclusion Reasons](#)
- [Making Changes to a Block After Creating an Abstract](#)
- [Creating a Frame View](#)
- [Linking to Abstract Views at the Top-Level](#)
- [Linking to Subblocks With Multiple Labels](#)
- [Specifying the Editability of Blocks From the Top-Level](#)
- [Preparing for Top-Level Closure With Abstracts](#)
- [Checking Designs With Abstracts for Top-Level-Closure Issues](#)
- [Performing Top-Level Closure With Abstract Views](#)
- [Creating ETMs and ETM Cell Libraries](#)
- [Linking to ETMs at the Top Level](#)
- [Performing Top-Level Closure With ETMs](#)

---

## Performing Hierarchical Synthesis Using Abstracts

The hierarchical synthesis flow using abstracts consists of the following major steps:

1. Partition the design at the top-level and generate the design libraries for the lower-level blocks as described in [Partitioning and Planning the Full Chip Design](#).
2. Synthesize each lower-level block and generate the block-level information required for top-level synthesis as described in [Synthesizing a Subblock](#).
3. Synthesize the top-level as described in [Synthesizing the Top-Level](#).

### See Also

- [Overview of Abstract Views](#)
- [Creating Abstract Views](#)
- [Creating a Frame View](#)

---

## Partitioning and Planning the Full Chip Design

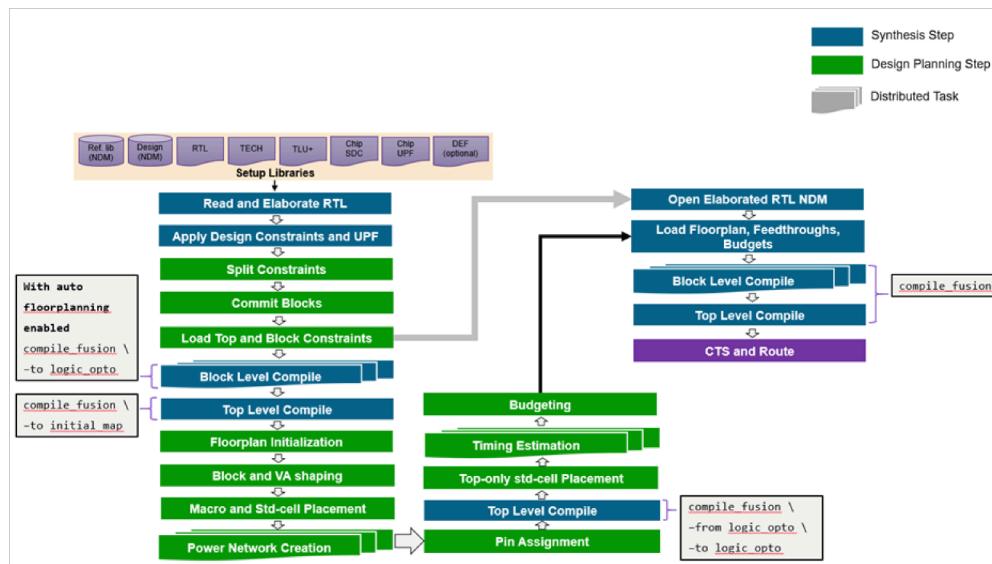
The hierarchical synthesis flow using abstracts includes the following two cases:

- Floorplans of subblock and top-level design are not available
- Floorplans of subblock and top-level design are available

## Hierarchical Synthesis Flow When Floorplans are not Available

To partition and plan the top-level design and create the lower-level blocks for subsequent bottom-up synthesis, when the floorplans of subblock and top-level design are not available, perform the following steps:

Figure 172 Hierarchical Synthesis Flow When Floorplans are not Available



1. Read in the full chip design and apply constraints as shown in the following example:

```
fc_shell> set REF_LIBS "stdcell.ndm macro.ndm"
fc_shell> create_lib TOP -technology techfile \
 -ref_libs $REF_LIBS
fc_shell> analyze -format verilog $rtl_files
fc_shell> elaborate TOP
fc_shell> set_top_module TOP

fc_shell> load_upf fullchip.upf
fc_shell> read_sdc fullchip.sdc
```

2. Identify the design partitions and split the constraints as shown in the following example:

```
fc_shell> set_budget_options -add_blocks {BLOCK1 BLOCK2}
fc_shell> split_constraints
```

3. Create the subblock design libraries with design information and enable block-specific reference library setup as shown in the following example:

```
fc_shell> copy_lib -to_lib BLOCK1.nlib -no_design
fc_shell> copy_lib -to_lib BLOCK2.nlib -no_design
```

```
fc_shell> set_attribute -objects BLOCK1.nlib \
 -name use_hier_ref_libs -value true
fc_shell> set_attribute -objects BLOCK2.nlib \
 -name use_hier_ref_libs -value true
fc_shell> save_lib -all
```

4. Create the subblock design partitions as shown in the following example:

```
fc_shell> commit_block -library BLOCK1.nlib BLOCK1
fc_shell> commit_block -library BLOCK2.nlib BLOCK2
fc_shell> save_lib -all
```

5. Load the UPF and SDC constraints for the unmapped subblocks and the top-level, which are generated by the `split_constraints` command earlier as shown in the following example:

```
fc_shell> set_constraint_mapping_file ./split/mapfile
fc_shell> load_block_constraints -all_blocks -type SDC \
 -type UPF -type CLKNET
fc_shell> save_lib -all
```

6. Run the `compile_fusion` command until logic optimization with auto floorplanning for sub blocks is complete as shown in the following example:

```
fc_shell> compile_fusion -to logic_opto
```

7. Run the `compile_fusion` command until technology mapping for the top-level design is complete as shown in the following example:

```
fc_shell> compile_fusion -to initial_map
```

8. Perform design planning operations starting from floorplan initialization, subblock and voltage area shaping, hard macro and standard cell placement, power network creation until pin assignment.

9. Run the `compile_fusion` command until logic optimization for the top-level design is complete as shown in the following example:

```
fc_shell> compile_fusion -from logic_opto -to logic_opto
```

10. Perform incremental top-level only standard cell placement as shown in the following example:

```
fc_shell> create_placement -floorplan -use_seed_locs
```

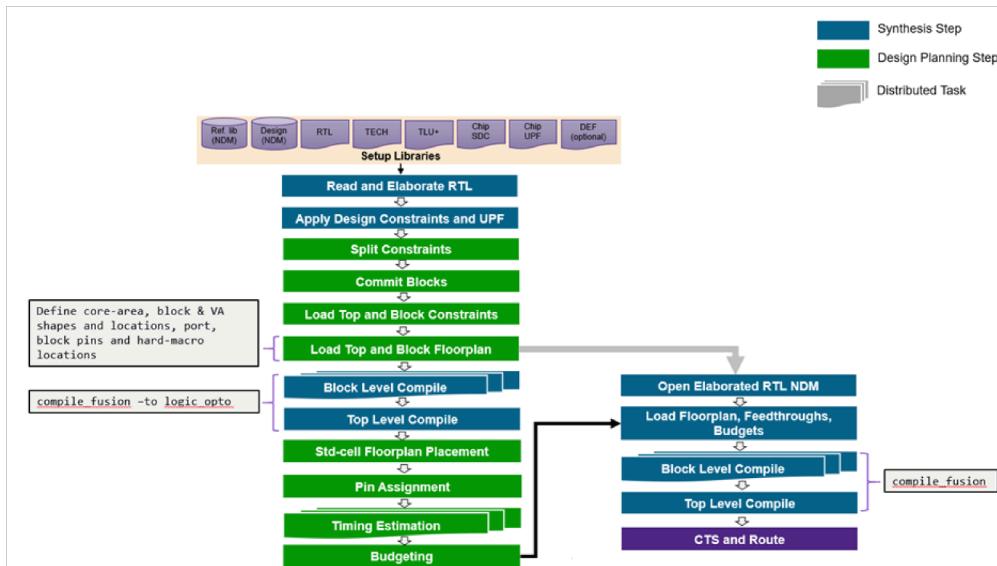
11. Perform timing estimation and budgeting steps to generate the block budgets needed for block level implementation.

12. After completing the design planning operations, rebuild the subblock and top-level design by opening the elaborated RTL NDM, loading floorplan, feedthroughs and budgets.

## Hierarchical Synthesis Flow When Floorplans are Available

To partition and plan the top-level design and create the lower-level blocks for subsequent bottom-up synthesis, when the floorplans of subblock and top-level design are available, perform the following steps:

*Figure 173 Hierarchical Synthesis Flow When Floorplans are Available*



1. Read in the full chip design and apply constraints as shown in the following example:

```
fc_shell> set REF_LIBS "stdcell.ndm macro.ndm"
fc_shell> create_lib TOP -technology techfile \
 -ref_libs $REF_LIBS
fc_shell> analyze -format verilog $rtl_files
fc_shell> elaborate TOP
fc_shell> set_top_module TOP

fc_shell> load_upf fullchip.upf
fc_shell> read_sdc fullchip.sdc
```

2. For the design partitions, split the constraints as shown in the following example:

```
fc_shell> set_budget_options -add_blocks {BLOCK1 BLOCK2}
fc_shell> split_constraints
```

3. Create the subblock design libraries with design information and enable block-specific reference library setup as shown in the following example:

```
fc_shell> copy_lib -to_lib BLOCK1.nlib -no_design
fc_shell> copy_lib -to_lib BLOCK2.nlib -no_design
fc_shell> set_attribute -objects BLOCK1.nlib \
```

```
-name use_hier_ref_libs -value true
fc_shell> set_attribute -objects BLOCK2.nlib \
 -name use_hier_ref_libs -value true
fc_shell> save_lib -all
```

4. Create the subblock design partitions as shown in the following example:

```
fc_shell> commit_block -library BLOCK1.nlib BLOCK1
fc_shell> commit_block -library BLOCK2.nlib BLOCK2
fc_shell> save_lib -all
```

5. Load the UPF and SDC constraints for the unmapped subblocks as well as the top-level, which are generated by the `split_constraints` command earlier as shown in the following example:

```
fc_shell> set_constraint_mapping_file ./split/mapfile
fc_shell> load_block_constraints -all_blocks -type SDC \
 -type UPF -type CLKNET
fc_shell> save_lib -all
```

6. Load the floorplan for the subblocks and top-level design.
  7. Run the `compile_fusion` command until logic optimization for the subblocks and top-level design is complete as shown in the following example:
- ```
fc_shell> compile_fusion -to logic_opto
```
8. Perform standard cell placement followed by the pin assignment, timing estimation and budgeting.
 9. After completing the design planning operations, rebuild the subblock and top-level design by opening the elaborated RTL NDM, loading floorplan, feedthroughs and budgets.

Note:

For details about each of the design planning operation, see the *Design Planning User Guide*.

Synthesizing a Subblock

To synthesize a subblock and generate the block-level information required for top-level synthesis, perform the following steps:

1. Read in a subblock design library generated after the rebuild step as described in [Partitioning and Planning the Full Chip Design](#).
2. Apply any block-specific constraints and settings required for synthesizing the block.

3. Apply DFT settings by using commands such as `set_dft_signal`, `set_scan_configuration`, and so on, and create a test protocol by using the `create_test_protocol` command.
4. Insert DFT logic and synthesize the block by using the following commands:

```
fc_shell> compile_fusion -to initial_opto
fc_shell> insert_dft
fc_shell> compile_fusion -from initial_place
```

5. Create a read-only abstract view, frame view, and a test protocol, and save the block by using the following commands:

```
fc_shell> create_abstract -read_only
fc_shell> create_frame
fc_shell> write_test_model BLOCK1.ctl
fc_shell> save_block -as BLOCK1/PostSynthesis
```

Synthesizing the Top-Level

To synthesize the top-level, perform the following steps:

1. Read in the top-level design generated after the rebuild step as described in [Partitioning and Planning the Full Chip Design](#).
2. Set appropriate settings and link the top-level design as shown in the following example:

```
fc_shell> set_label_switch_list \
    -reference {BLOCK1 BLOCK2} PostCompile
fc_shell> set_attribute -objects {BLOCK1.nlib BLOCK2.nlib} \
    -name use_hier_ref_libs -value true
fc_shell> set_top_module TOP
```

When a subblock has been saved in the design library using labels, use the `set_label_switch_list` command to specify which label to link to. The `set_top_module` command links the specified top-level module.

3. Apply any block-specific constraints and settings required for synthesizing the top block and read in the test models for the subblocks as shown in the following example:

```
fc_shell> read_test_model BLOCK1.ctl
fc_shell> read_test_model BLOCK2.ctl
fc_shell> create_test_protocol
```

4. Insert DFT logic and synthesize the top-level design by using the following commands:

```
fc_shell> compile_fusion -to initial_opto
fc_shell> insert_dft
fc_shell> compile_fusion -from initial_place
```

Performing Hierarchical Synthesis Using ETMs

During the hierarchical flow, block designs are represented as a macro with a frame view, and the implemented blocks are modeled as extracted timing models (ETMs) for the top-level implementation of the design.

An extracted timing model (ETM) is a Liberty model representation of a design that is generated by the `extract_model` command in the PrimeTime tool. The command captures the timing and power information of the design using relevant Liberty attributes, as shown in the following example.

```
library("block1") {  
    ...  
    comment : "PrimeTime Extracted Model." ;  
    voltage_map("nom_voltage", 1.08);  
    voltage_map("VDD", 1.08);  
    ...  
    cell( block1 ) {  
        timing_model_type : "extracted";  
        ...  
        pg_pin("VDD") {  
            voltage_name : "VDD";  
            pg_type : primary_power;  
        }  
        ...  
    }
```

Topics in this section

- [Running the Hierarchical Synthesis Feasibility Analysis Flow Using ETMs](#)
- [Running the Hierarchical Synthesis Flow Using ETMs](#)
- [Example Script of Hierarchical Synthesis Using ETMs](#)

See Also

- [SolvNet article 2284429, "Top-level Closure Flow With Extracted Timing Models \(ETMs\)"](#)
- [Creating ETMs and ETM Cell Libraries](#)
- [Linking to ETMs at the Top Level](#)

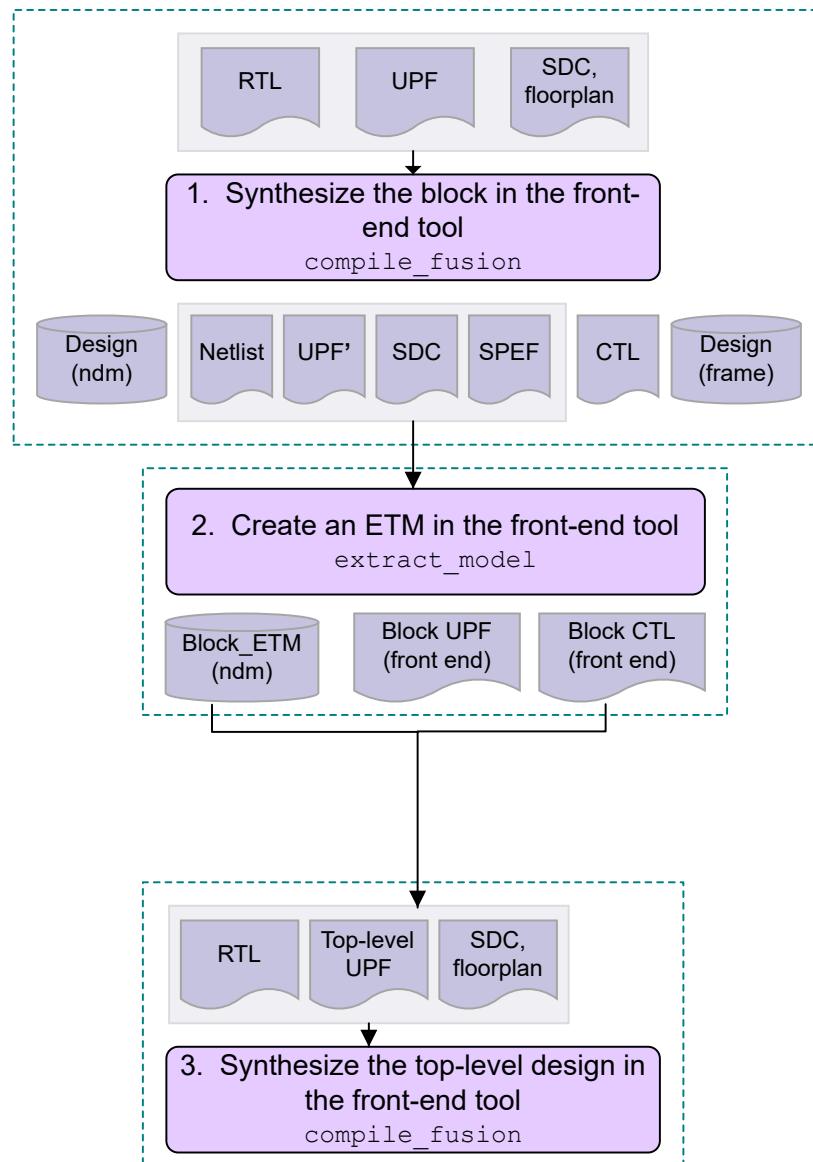
Running the Hierarchical Synthesis Feasibility Analysis Flow Using ETMs

To minimize the number of iterations and obtain optimal results, you can perform feasibility analysis before running hierarchical synthesis, as shown in [Figure 174](#).

The hierarchical synthesis feasibility flow uses the following block-level information generated post `compile_fusion`:

- Floorplanning information to generate the ETM in step 2
- Power intent and test model (CTL) information for top-level synthesis in step 4

Figure 174 Hierarchical Synthesis Feasibility Analysis

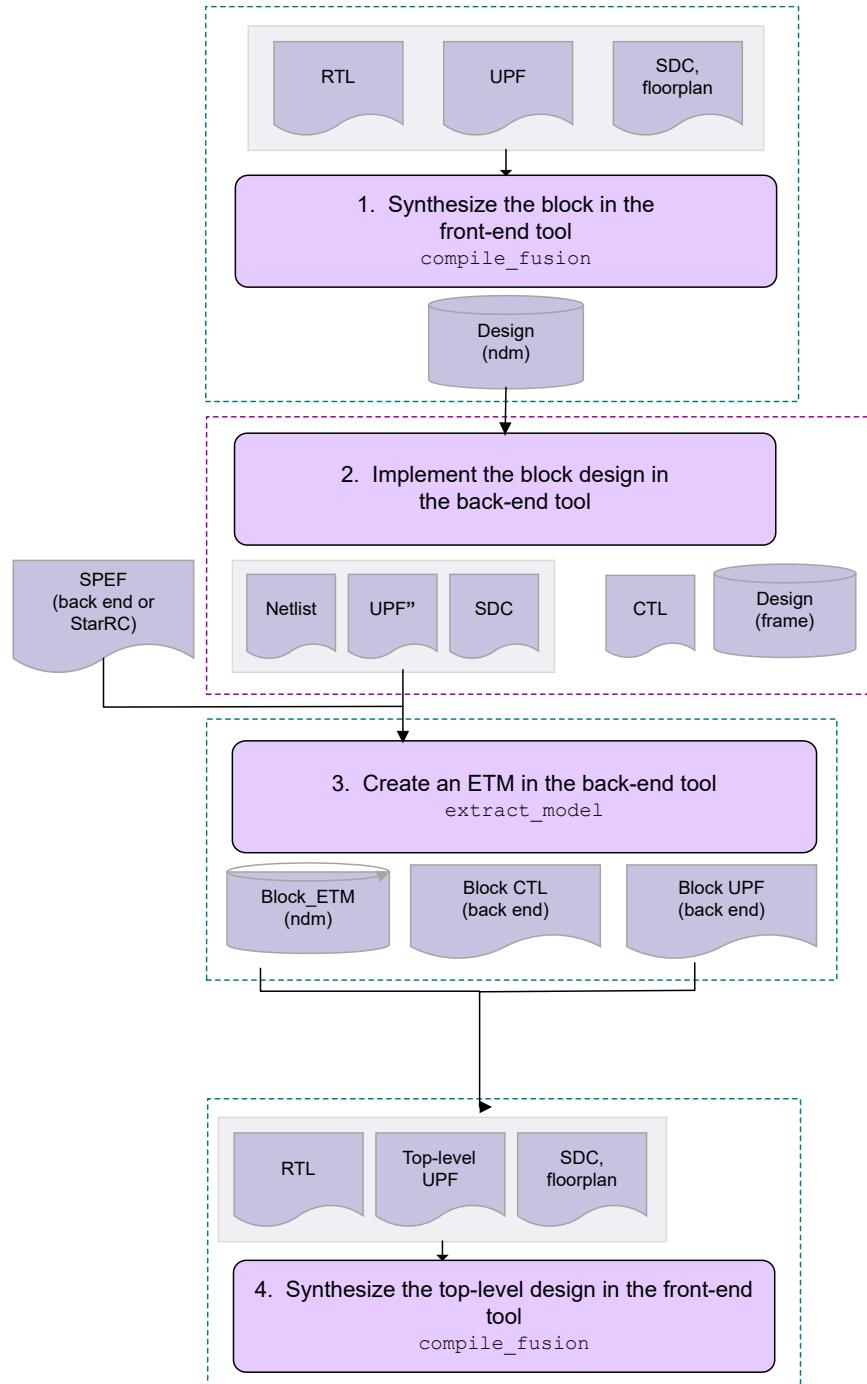


Running the Hierarchical Synthesis Flow Using ETMs

To run hierarchical synthesis flow, follow the steps described in [Figure 175](#). In contrast to the feasibility flow, the hierarchical synthesis flow uses the following block-level information generated post place and route:

- Floorplanning information to generate the ETM in step 3
- Power intent and test model (CTL) information for top-level synthesis in step 5

Figure 175 Hierarchical Synthesis Flow



Example Script of Hierarchical Synthesis Using ETMs

The following script shows an example of the hierarchical synthesis flow:

```
# Specify reference libraries
set REF_LIBS ${std_cell}.ndm
lappend REF_LIBS ${block_design}_etm.ndm
create_lib -tech ${TECH_FILE} -ref_libs $REF_LIBS ${DESIGN_NAME}

# Read the RTL design
analyze -format verilog [list $rtl_list]
elaborate ${DESIGN_NAME}

#ETM.ndm in ref lib is used to link cell instances
set_top_module ${DESIGN_NAME}

# Load UPF for ETM cell instance
load_upf ${block_design}.upf -scope cell_instance

# Read test model for ETM cell instance
read_test_model ${block_design}.mapped.ctl
...
compile_fusion -to initial_opto

# Query commands
# List all macro cell instances
get_cells -hierarchical -filter "is_hard_macro == true"
# ref view name is "timing" for cell instances bound to ETM
get_attribute ${CELL} is_etm_moded_cell
```

Performing Prechecks for Hierarchical Synthesis Flows

To check the readiness for the hierarchical synthesis flow, use the following commands:

- `check_design -checks block_ready_for_top`

Run this command after block synthesis to check whether the synthesized block is ready for the top-level synthesis.

- `check_design -checks hier_pre_compile`

Run this command before top-level synthesis to check whether the subblocks and top-level design are ready for the hierarchical flow.

Providing Block-Level Test Models

You can provide test model of the subblocks for the top-level design implementation when running hierarchical synthesis. To do so, generate the test model for the mapped block

design during block-level synthesis and then annotate the test model to the subblock during top-level synthesis. For example,

- Generating the test model during block-level synthesis

```
# Read block RTL and constraints
# Specify DFT configuration
set_dft_signal ...
set_scan_configuration ...
create_test_protocol
...
compile_fusion -to initial_opto
insert_dft
write_test_model -output block_des.mapped.ctl
```

- Running top-level synthesis with the block-level test model

```
# Read RTL and constraints
# Specify DFT configuration

read_test_model block_des.mapped.ctl
set_dft_signal ...
set_scan_configuration ...
create_test_protocol
...
compile_fusion -to initial_opto
insert_dft
```

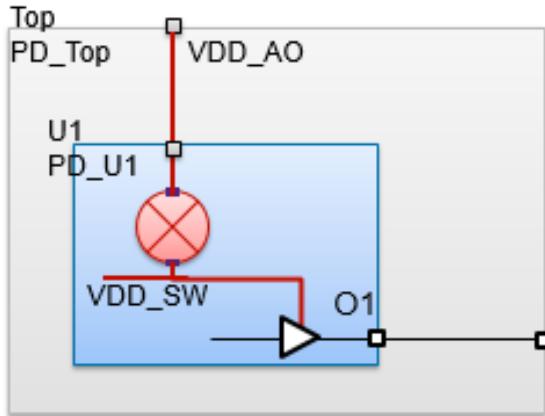
Specifying Block-Level Power Intent

You must provide power intent of the subblock for the top-level design implementation when running hierarchical synthesis. To do so, you can provide full block-level UPF for the tool to automatically extract power intent interface. The following figure shows a subblock named U1, and the top-level power intent script loads the block-level power intent by using the `load_upf` command.

Note:

The block-level power intent specification is required only for hierarchical synthesis using ETMs. In the abstract flow, the UPF constraints are handled automatically by the tool.

- U1 as an ETM in the PD_Top design



- Top-Level power intent

```
create_power_domain PD_TOP -include_scope
create_supply_net VDD_AO
...
load_upf block.upf -scope U1
...
connect_supply_net VDD_AO -port U1/VDD_AO
```

- Block-level power intent: block.upf

```
create_power_domain PD_BLK -include_scope
create_power_switch sw1 -domain PD_BLK \
    -input_supply_port {in VDD_AO} \
    -output_supply_port {out VDD_SW} ...
...
add_port_state sw1/out -state {ON 1.08} -state {OFF off}
...
create_pst pst -supplies {VDD_SW ...}
```

Overview of Abstract Views

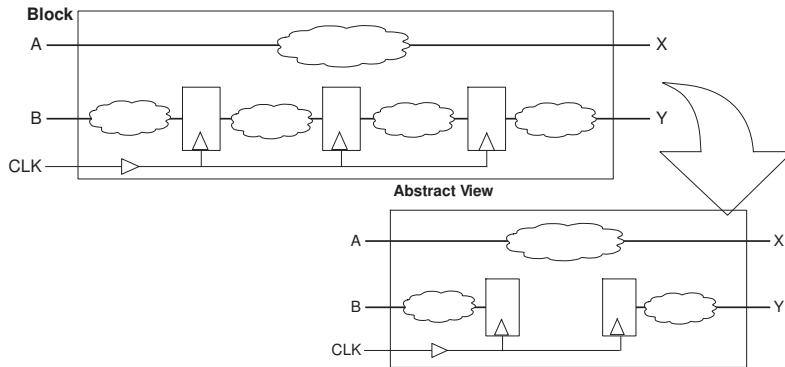
In an abstract view, the gate-level netlist for the block is modeled by a partial gate-level netlist that contains only the required interface logic of the block. All other logic is removed.

[Figure 176](#) shows a block and its abstract view, where the logic is preserved between

- The input port and the first register of each timing path
- The last register of each timing path and the output port

Logic associated with pure combinational input-port-to-output-port timing paths (A to X) is also preserved. Clock connections to the preserved registers are kept as well. The register-to-register logic is discarded.

Figure 176 A Block and Its Abstract View



The interface logic of an abstract view consists of the following:

- All cells, pins, and nets in timing paths from input ports to registers or output ports
- All cells, pins, and nets in timing paths to output ports from registers or input ports
- Any logic in the connection from a master clock to generated clocks
- The clock trees that drive interface registers, including any logic in the clock tree
- The longest and shortest clock paths from the clock ports
- All pins with timing constraints that are part of the interface logic

In addition to the interface logic, an abstract view contains the following information associated with the interface logic:

- Placement information
- Timing constraints
- Clock tree exceptions
- Parasitic information
- Transition and case values of dangling pins
- Power management cells and UPF constraints
- Nondefault routing rule association and minimum and maximum layer constraints, if any, for the nets that are retained in the abstract view

Creating Abstract Views

In the Fusion Compiler tool, you can create an abstract view for a block at various stages of the design flow, such as, after synthesis, placement, optimization, clock tree synthesis, routing, and so on. Before you create the abstract view, ensure that the scenarios needed at the top level have been created and are active.

To create an abstract view for top-level closure, use the `create_abstract` command.

- To control the amount of timing information in the abstract view, use the `-timing_level` option with the following settings:
 - To create an abstract that contains only the boundary cells (one level of logic) connected to each boundary port, use the `boundary` setting.
Such an abstract also contains feedthrough data paths, feedthrough combinational clock paths, and internal clock logic driving output ports. You can use this type of abstract to fix DRC violations at the top level.
 - To create a compact abstract that contains the timing information for only the critical setup and hold timings paths of the interface logic, use the `compact` setting. This is the default setting, and is preferred for top-level design closure.
 - To create an abstract with timing information for all the interface logic, use the `full_interface` setting. A `full_interface` abstract might be required for multiply-instantiated blocks (MIBs) when the constraints for MIB instances are different at the top-level.
- To create the abstracts for lower-level blocks of the current block, use one of the following methods:
 - Create abstracts for specific lower-level blocks by using the `-blocks` option and specify the names of the blocks.
 - Create abstracts for all lower-level blocks by using the `-all_blocks` option.

When you use the `-blocks` or `-all_blocks` option, if a specified block has an abstract view, by default, the tool does not re-create it. To force the tool to re-create the abstract, use the `-force_recreate` option.

- To include specific nets, ports, hierarchical pins, or leaf pins in a placement or timing abstract, use the `-include_objects` option.

- To create a read-only abstract, use the `-read_only` option.

By default, the tool creates an editable abstract that you can modify in the context of a parent block. However, the tool makes the design view of the block read-only, preventing any changes from being made to the block.

When you create the abstract view as read-only, the design view of the block is editable, and you can make changes to it.

- To flatten the physical hierarchy and retain only the relevant logic of the lower-level blocks, use the `-preserve_block_instances false` option.
- To specify if the abstract is going to be used for design planning or top-level implementation, use the `-target_use planning` or `-target_use implementation` option.

Based on the intended usage, the tool applies appropriate internal settings to the abstract.

- To specify host options for distributed processing, use `-host_options` option and specify the appropriate settings. You can use this option to reduce the runtime when you create more than one abstract view by using the `-blocks` or `-all_blocks` option.

Creating Abstracts With Power Information

To create an abstract that contains power information of the corresponding block, use the following steps at the block level:

1. Activate the scenarios for which you want to perform power analysis by using the `-active true` option of the `set_scenario_status` command and enable them for power analysis by using the following options:
 - `-dynamic_power true` for dynamic power analysis
 - `-leakage_power true` for leakage power analysis

Power information is stored only for active scenarios that are enabled for dynamic or leakage power.

2. (Optional) Apply switching activity by either reading in a SAIF file with the `read_saif` command or annotating the switching activity information on the nets with the `set_switching_activity` command.

For more information about applying switching activity, see [Annotating the Switching Activity](#).

3. Store the power information in the abstract by setting the `abstract.annotate_power` application option to `true`.
4. Create the abstract for the block by using the `create_abstract` command.

When you create an abstract with power information, the tool stores the power information only for the logic that is removed.

When you instantiate such an abstract and perform power analysis at the top level, the tool

- Recomputes the power for the interface logic in the abstract, in context, based on the switching activity and loading seen at the top level
- Uses the stored power in the abstract for the logic that was removed when the abstract was created

To report the power information in an abstract instantiated at the top level, use the `report_power -blocks` command. If there are multiple levels of physical hierarchy, to specify the number of level for which you want to report, use the `-levels` option.

Creating Abstracts for Signal Electromigration Analysis

To create an abstract that can be used for signal electromigration analysis at the top-level, set the `abstract.enable_signal_em_analysis` application option to `true` before you run the `create_abstract` command.

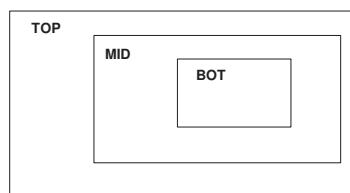
Handling Multiple Levels of Physical Hierarchy

For designs with multiple levels of physical hierarchy, before you create an abstract for a block with an abstract instantiated in it,

- Bind the lower-level blocks that you want represented by abstracts to the specific abstracts

For example, assume you have a design with multiple levels of physical hierarchy as shown in the following figure.

Figure 177 A Design With Multiple Levels of Physical Hierarchy



To create an abstract for the block named BOT, use the following commands:

```
fc_shell> open_block BOT
fc_shell> create_abstract
```

To create an abstract for the block named MID, use the following commands:

```
fc_shell> open_block MID
fc_shell> change_abstract -view abstract -references BOT
fc_shell> create_abstract
```

By default, the `create_abstract` command preserves all levels of the physical hierarchy. To flatten the physical hierarchy and retain only the relevant logic of the lower-level blocks, use the `-preserve_block_instances false` option, as shown in the following example:

```
fc_shell> open_block MID
fc_shell> create_abstract -preserve_block_instances false
```

Use flattened abstracts in top-level implementation flows where the lower-level blocks are treated as read-only. Using flattened abstracts reduces the netlist size at the top level. It also simplifies data management because you do not need to include the lower-level blocks in the reference library list of the top level.

Reporting Abstract Inclusion Reasons

In the Fusion Compiler tool, the `create_abstract` command includes netlist objects such as the cells, nets, and pins in the abstract view at both the top-level and block-level. This command reports the reasons for the inclusion of netlist objects.

To report the reasons for including specific netlist objects in the abstract view, you can use the `report_abstract_inclusion_reason` command for objects across all hierarchies in the abstract.

For each reason reported, a reason code is displayed. These reason codes correspond to the valid reasons for inclusion in the abstract. There can be multiple reasons for the same netlist object.

Note:

The `report_abstract_inclusion_reason` command does not report reasons for the hierarchical cells or pins on hierarchical cells.

You can perform reason reporting from the top-level design, where the abstract is instantiated, without having the abstract as the current block.

The following example reports the reasons for abstract inclusion for the pins in abstract view:

```
prompt> report_abstract_inclusion_reason [get_pins -of_object [get_cells
  -filter "is_hierarchical==false" -hierarchical *]]
```

Legend

mv - MV Logic
 comp - Compact interface logic
 cell - Cell inclusion

| Pin name | Reason(s) for inclusion |
|----------------|-------------------------|
| AINV_P_260/I | comp, cell |
| AINV_P_260/VDD | cell |
| AINV_P_260/VSS | cell |
| AINV_P_260/ZN | comp, cell |
| AINV_P_262/I | comp, cell |
| AINV_P_262/VDD | cell |
| AINV_P_262/VSS | cell |
| AINV_P_262/ZN | comp, cell |
| AINV_P_263/I | mv, comp, cell |

A legend is displayed at the beginning of the report for the queried objects, specifying the detailed reason for each reason code.

Making Changes to a Block After Creating an Abstract

When you create an abstract view for a block with the `create_abstract` command, by default, the tool saves both the abstract and design views of the block and changes the design view to read-only.

You can open a read-only block and make changes to it in-memory, but you cannot save these changes by using the `save_block` or `save_lib` command. To save the change you make to a read-only block, use one of the following methods:

- Save it as a different block by using the `save_block -as` command.
- Re-create an abstract for the block by using the `create_abstract` command. The tool then automatically saves the design view.
- Remove the abstract by using the `remove_abstract` command, if you no longer need the abstract. This changes the design view from being read-only to editable, and you can save the block by using the `save_block` or `save_lib` command.

If you create a read-only abstract by using the `create_abstract -read_only` command, the tool does not make the design view of the block read-only and you can save any

subsequent changes to the block by using the `save_block` or `save_lib` command, as shown in the following example:

```
fc_shell> create_abstract -read_only
fc_shell> change_link [get_cells U25] AND2
fc_shell> save_block
```

Creating a Frame View

To perform top-level routing, including virtual routing, every abstract view must have a corresponding frame view. To create a frame view, use the `create_frame` command, which extracts the blockage, pin, and via information from the design view.

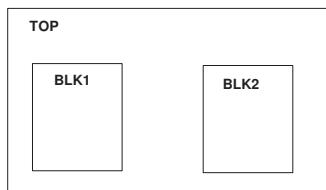
After you create the frame view, save the design library by using the `save_lib` command.

Linking to Abstract Views at the Top-Level

For a top-level design to link to an abstract view of a lower-level block, the design library containing the abstract view of the lower-level block must be one of the reference libraries of the top level.

For example, assume you have a top-level design named TOP with two lower-level blocks named BLK1 and BLK2, as shown in the following figure.

Figure 178 Top-Level Design With Instantiated Blocks



To link to the abstracts of the BLK1 and BLK2 blocks, the design libraries containing the abstract views of these blocks must be reference libraries of the TOP design. If not, you can add it to the reference libraries of the TOP design by using the `set_ref_libs -add` command as shown in the following example:

```
current_block TOP
set_ref_libs -add ../BLK1/BLK1.nib
set_ref_libs -add ../BLK2/BLK2.nib
```

To report the reference libraries for the current design, use the `report_ref_libs` command.

When you link a top-level design, if an abstract view for a lower-level block is available, the tool links to that abstract view by default. The default precedence of the different views used when linking lower-level blocks is as follows:

1. Abstract view
2. Design view
3. Frame view
4. Outline view

To change a block from its abstract to its design view or vice versa, use the `change_abstract` command and specify the view you are changing to by using the `-view` option. When you change a block from its abstract view to its design view, remove the existing constraints and apply full-chip timing constraints.

The following example changes the BLK1 block from its abstract view to its design view and applies full-chip timing constraints.

1. Remove any existing timing constraints:

```
fc_shell> remove_scenarios -all
fc_shell> remove_modes -all
fc_shell> remove_corners -all
```

2. Use the `change_abstract` command to change from the abstract view to the design view:

```
fc_shell> change_abstract -view design -references BLK1
```

3. Apply the full-chip scenario creation script:

```
fc_shell> source full_chip_scenario_creation.tcl
```

If the abstract view for a block has changed in its design library, you can reload the new abstract view by using the `change_abstract -reload` command.

To report the abstract views the design is linking to, use the `report_abstracts` command.

More information about specifying reference libraries and linking to abstract views at the top level, see [SolvNet article 2436119, Setting Up Designs for Hierarchical Place and Route](#)

Linking to Subblocks With Multiple Labels

When saving a block at different stages of an implementation flow, you can use the `-as blockName/labelName` or `-label labelName` option with the `save_block` command to specify a unique label for each version of the block you save.

By default, during linking, the top-level block links to lower-level blocks with the same label. If a subblock has been saved with multiple labels, you can specify which label the top level should link to by using the `set_label_switch_list` command. The command specifies a precedence-ordered list of labels to use during linking. By default, the command applies to all subblocks; to specify a different block or blocks, use the `-reference` option.

The following example specifies that the top level should link to the PostCompile label of the BLOCK1 and BLOCK2 subblocks:

```
fc_shell> set_label_switch_list \
    -reference {BLOCK1 BLOCK2} PostCompile
```

The following example specifies that the PostCompile label has a higher priority than the PreCompile label when linking any subblock at the top level.

```
fc_shell> set_label_switch_list {PostCompile PreCompile}
```

You can use the `set_label_switch_list` command again to update the block's label switch list and then relink the block using the `link_block -rebind` command. Note the following when relinking a block:

- If none of the labels in the switch list are available in a subblock, or if you specify an empty switch list, the tool links the subblock to the label it was previously linked to
- If you have removed all the labels for a subblock, the tool links the subblock to the label of its parent block

Specifying the Editability of Blocks From the Top-Level

For designs with physical hierarchy, you can specify if changes can be made to lower-level blocks by using the `set_editability` command at the top level.

You can change the editability of

- Specific blocks by using the `-blocks` option
- All blocks starting from a specific level of the physical hierarchy by using the `-from_level` option
- All blocks up to a specific level of the physical hierarchy by using the `-to_level` option

For top-level implementation flows with read-only abstracts, after you link the design, explicitly set the lower-level blocks as read-only by using the following command:

```
fc_shell> set_editability -blocks [get_blocks -hierarchical] \  
-value false
```

By default, if you change a library or block name with one of the following commands, the tool does not propagate the editability settings from the old reference to the new reference:

- `change_abstract -lib`
- `link_block -rebind -force`
- `set_reference`

To retain the editability settings, set the `design.preserve_reference_editability` application option to `true` before you change a library or block name.

When you save the hierarchical design with the `save_block` command, the editability settings are saved.

Preparing for Top-Level Closure With Abstracts

Before you can perform synthesis, placement, optimization, clock tree synthesis, and routing at the top level, you must perform the following tasks:

- Apply the top-level-timing constraints and settings.
You can split the chip-level constraints into separate top- and block-level constraints by using the `split_constraints` command.
- Review and specify the relationship between the top- and block-level modes, corners, and clocks by using the `set_block_to_top_map` command.
- If the pre-clock tree implemented blocks, which have undergone CCD optimization, are used for top-level placement and optimization, then the ideal clock latencies adjusted dynamically by different engines such as CCD and clock-gate estimation on the clock pins of registers or integrated clock gating (ICG) cells inside the blocks might not be seen at the top-level leading to timing inaccuracies at the top-level.

Promote these clock latency constraints from block-level to the top-level by using the `promote_clock_data -latency_offset -auto_clock connected` command.

- Apply the top-level clock tree synthesis settings and exceptions.
 - If the top-level exceptions do not include the balance points on the pins within lower-level blocks, promote the balance points from the lower levels by using the `promote_clock_data -auto_clock connected -balance_points` command.
 - If concurrent clock and data (CCD) optimization is performed at the block level, the tool derives median clock latencies for the block-level clock ports, which accounts for the latency adjustments that were derived for the block-level registers during concurrent clock and data optimization. Promote these block-level clock latencies as clock balance point delays for the block-level clock pins by using the `promote_clock_data -auto_clock connected -port_latency` command.
 - If the lower-level blocks contain clock-meshes, promote the mesh annotations (annotated transitions and delays) by using the `promote_clock_data -mesh_annotations` command.
- If you need to promote clock data for a specific top-level clock, use the `promote_clock_data -clocks {top_clock} -port_latency` command. The command also promotes clock-independent exceptions. If any of the given top-level clocks has no mapping to any block-level clock in any mode, the tool issues a warning.
- If there are any active modes, corners, or clocks at the top-level, which cannot be mapped to corresponding modes, corners, or clocks in the block-level, you can still promote the clock data for other mapped modes, corners, or clocks using the `promote_clock_data -balance_points -force` command. This command issues errors for the unmapped modes, or corners, or clocks.
- Apply the top-level UPF constraints, which you can create from the full-chip UPF constraints by using the `split_constraints` command.

The tool promotes the required UPF constraints from the lower-level blocks to the top level.

Checking Designs With Abstracts for Top-Level-Closure Issues

You can check a hierarchical design that contains abstracts for possible issues such as the application option consistency issues and the design issues related to top-level closure by

using the `check_hier_design` command. Identifying issues and fixing them before you perform top-level closure can help reduce turnaround time.

When you use the `check_hier_design` command, you can specify

- The references to check by using the `-reference` option.

If you do not specify this option, the command checks all the references in the physical hierarchy.

- The type of checks to perform by using the `-stage` option as follows:

- Use the `-stage timing` option to perform timing related checks.

If you do not specify the `-stage` option, by default, the tool performs timing related checks.

- Use the `-stage pre_placement` to perform both timing and preplacement related checks.

The `check_hier_design` command can check the consistency between the top-level constraints and the constraints of every lower-level instance linked to an abstract. To enable this feature, set the `abstract.check_constraints_consistency` application option to `true` before you run the `check_hier_design` command.

The tool saves the settings of the predetermined list of timer application options during the `create_abstract` command. At the top-level design, the `check_hier_design` command compares these timer application option settings with those of the block-level design.

The following application options are automatically saved during the `create_abstract` command and verified by the `check_hier_design` command:

- `time.case_analysis_propagate_through_icg`
- `time.case_analysis_sequential_propagation`
- `time.clock_gating_propagate_enable`
- `time.clock_gating_user_setting_only`
- `time.clock_marking`
- `time.clock_reconvergence_pessimism`
- `time.create_clock_no_input_delay`
- `time.crpr_remove_clock_to_data_crp`
- `time.delay_calc_waveform_analysis_mode`
- `time.delay_calculation_style`

- time.disable_case_analysis_ti_hi_lo
- time.disable_clock_gating_checks
- time.disable_cond_default_arcs
- time.disable_internal inout_net_arcs
- time.disable_recovery_removal_checks
- time.edge_specific_source_latency
- time.enable_auto_mux_clock_exclusivity
- time.enable_ccs_rcv_cap
- time.enable_clock_propagation_through_preset_clear
- time.enable_clock_propagation_through_three_state_enable_pins
- time.enable_clock_to_data_analysis
- time.enable_non_sequential_checks
- time.enable_preset_clear_arcs
- time.enable_si_timing_windows
- time.gclock_source_network_num_master_registers
- time.special_path_group_precedence
- time.use_lib_cell_generated_clock_name
- time.use_special_default_path_groups

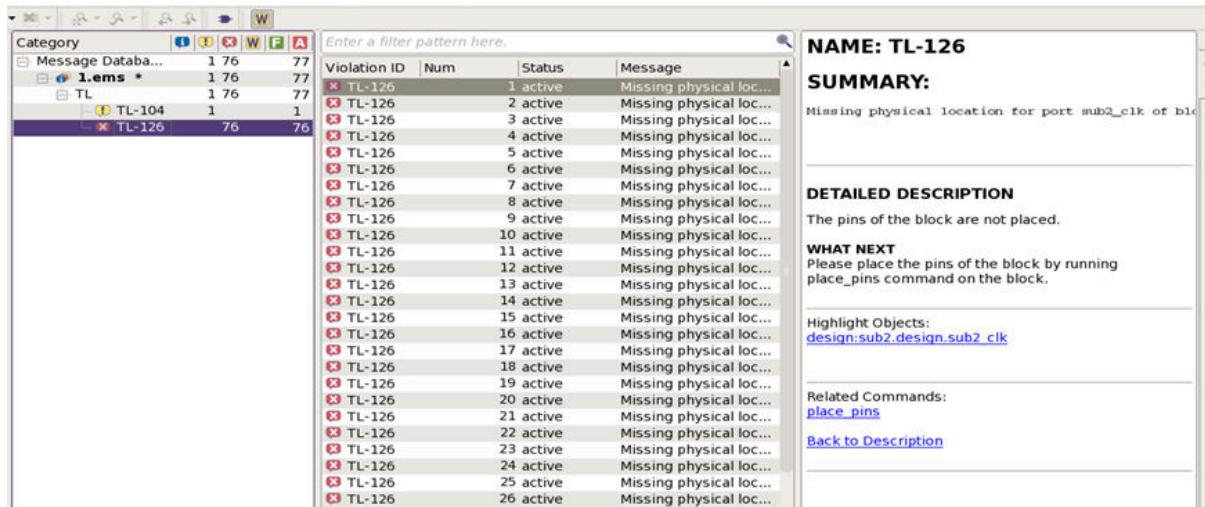
For more information about the issues the `check_hier_design` command identifies and how to fix them, see the man page for the corresponding message ID.

In addition to generating a report, the `check_hier_design` command generates an enhanced messaging system (EMS) database that you can view by using the message browser in the Fusion Compiler GUI. Create the EMS database before you run the `check_hier_design` command, as shown in the following example:

```
fc_shell> create_ems_database check_hier.ems
fc_shell> check_hier_design -stage timing
fc_shell> save_ems_database
```

In the Fusion Compiler GUI message browser, you can sort, filter, and link the messages to the corresponding man page, as shown in the following figure.

Figure 179 Viewing the EMS Database in the Message Browser



You can also output the information in the EMS database in ASCII format by using the `report_ems_database` command.

The checks performed by the `check_hier_design` command are also available in the `check_design` command.

You can perform

- Timing checks specific to top-level closure by using the `check_design -checks hier_timing` command
- All timing checks, including those specific to top-level closure, by using the `check_design -checks timing` command
- Preplacement checks specific to top-level closure by using the `check_design -checks hier_preplacement` command
- All preplacement checks, including those specific to top-level closure, by using the `check_design -checks preplacement` command

You can generate an EMS database for the `check_design` command and view it in the GUI, similar to the `check_hier_design` command.

Handling Design Data Using the Early Data Check Manager

While you identify the issues related to top-level closure and fix them, the tool enables you to explore the top-level flow in the presence of design data violations. The hierarchical checks can detect different types of mismatched, incomplete, and inconsistent data in

designs. You can configure the following policies for the hierarchical checks to manage the design data violations:

- Error: When you apply this policy, the tool does not mitigate the violation, but issues an error message.
- Tolerate: When you apply this policy, the tool makes predictable and explainable assumptions to continue with the flow. No changes are made to the design or setup.
- Repair: When you apply this policy, the tool mitigates the violation using one or more repair strategies and records it. Repair can include changes in design and setup.
- Strict: When you apply this policy, depending on the policy supported by the check, the tool applies the policies in this order – error, tolerate, repair.
- Lenient: When you apply this policy, depending on the policy supported by the check, the tool applies the policies in this order – repair, tolerate, error.

Note:

- Quality of results of the top-level flow might get impacted due to the toleration and repair of errors. Error toleration and repair is mainly provided for managing data violations during the early design exploration phases. In the implementation flow, you should use the strict policy for all checks.

Prerequisites for Handling Early Design Data

The tool performs the repairs and saves the records in the subblocks within the `check_hier_design` command itself. For the repairs, which need frame recreation, you need to run the `save_lib` command to save the frame on disk.

- To save the repair and its record to the subblock successfully, the following setup is required:

1. Enable the reference libraries, containing the subblocks, for edit

```
open_lib top.nlib -ref_libs_for_edit
```

2. Enable editability on the subblocks, on which repair has to be performed

```
set_editability -blocks <> -value true
```

The `check_hier_design` command checks for the required editability settings, if repair has to be performed. If the required editability settings are not performed, then the `check_hier_design` command issues TL-170 error message.

Note:

The tool supports the repairs on subblocks, which are linked to the `read_only` abstracts rather than editable abstracts because managing the mismatched

design data mechanism only aims to continue the top-level flow for exploration and as such no repair work performed on the subblock abstracts should get merged with the actual design view. If the top-level uses editable design views, the repair work still continues.

Early Data Checks, Policies, and Strategies

The tool can identify issues and violations in data during the early stages of design. You can set policies and strategy configurations for the predefined checks to allow, tolerate, or repair data. [Table 62](#) describes the predefined checks for top-level closure.

Table 62 Top-Level Closure Checks, Policies, and Strategies

| Check | Description | Strategy | Supported Policies | Supported References |
|--|---|---|-------------------------|----------------------|
| hier.block.missing_frame_view | Checks whether the frame view is missing for a hierarchical block | Re-create frame views from the bound view | error, tolerate, repair | MID, BOT |
| hier.block.reference_missing_port_location | Checks whether the location of a physical hierarchy boundary pin is missing | Assign a location for the block pin on the block boundary based on connectivity | error, tolerate, repair | MID, BOT |
| hier.block.reference_port_outside_boundary | Checks whether the location of a physical hierarchy boundary pin is outside the physical hierarchy boundary | Reassign a location for the block pin on the block boundary based on connectivity | error, tolerate, repair | MID, BOT |
| hier.block.reference_missing_port | Checks whether a port is missing in the physical hierarchy reference | Create missing ports in the reference block and update its location | error, tolerate, repair | MID, BOT |
| hier.block.instance_bound_to_frame | Checks whether a physical hierarchy instance is bound to a frame view | | error, tolerate | TOP |
| hier.block.instance_with_design_type_macro | Checks whether a physical hierarchy instance is bound to a macro | | error, tolerate | TOP |

Table 62 Top-Level Closure Checks, Policies, and Strategies (Continued)

| Check | Description | Strategy | Supported Policies | Supported References |
|--|---|--|-------------------------|----------------------|
| hier.top.estimated_corner | Checks whether the estimated corner is at top level | | error, tolerate | TOP |
| hier.block.missing_leaf_cell_location | Checks whether the location of a leaf cell of a physical hierarchy is missing | Assign an approximate location inside the block boundary for the leaf cell (legality not considered) | error, tolerate, repair | MID, BOT |
| hier.block.leaf_cell_outside_boundary | Checks whether the location of a leaf cell of a physical hierarchy is outside the physical hierarchy boundary | Reassign the leaf cell location to the nearest location inside the block boundary (legality not considered) | error, tolerate, repair | MID, BOT |
| hier.block.missing_child_block_instance_location | Checks whether the location of a child physical hierarchy instance is missing in the specified physical hierarchy | Assign an approximate location for the lower-level subblock based on connectivity inside the block boundary (overlap removal not considered) | error, tolerate, repair | MID |
| hier.block.child_block_instance_outside_boundary | Checks whether the location of a child physical hierarchy instance is outside the specified physical hierarchy boundary | Move the subblock to the nearest location inside the block boundary (overlap removal not considered). | error, tolerate, repair | MID |
| hier.block.port_mismatch_between_views | Checks whether there is a mismatch of ports between the views of a physical hierarchy | Bound view is treated as golden and the frame view is recreated to match the bound view | error, tolerate, repair | MID, BOT |

Table 62 Top-Level Closure Checks, Policies, and Strategies (Continued)

| Check | Description | Strategy | Supported Policies | Supported References |
|---|---|----------|--------------------|----------------------|
| hier.block.missing_design_view_for_abs | Checks whether the design view for an abstract is missing | | error, tolerate | MID, BOT |
| hier.block.instance_unlinked | Checks whether the physical hierarchy reference is not linked properly to the top-level design due to a view change from frame or ETM to abstract or design | | error | TOP |
| hier.block.abstract_type_non_timing | Checks whether the abstract type is not timing | | error | MID, BOT |
| hier.block.abstract_target_use_non_implementation | Checks whether the target use of an abstract is not implementation | | error | MID, BOT |
| hier.block.missing_core_area | Checks whether the core area for a physical hierarchy is missing | | error | MID, BOT |
| hier.block.unmapped_logic | Checks whether unmapped logic is present in the physical hierarchy | | error | MID, BOT |

Where

- BOT signifies bottom level design.
- MID signifies intermediate level design, which instantiates BOT.
- TOP signifies the top-level design, which instantiates MID.

Setting the Policy for Early Data Checks

To set or modify the policy for all the checks, or modify the policy settings for data checks, use the `set_early_data_check_policy` command. Use the following options to configure the policy:

- `-checks`: Specifies a predefined list of data checks. It supports an asterisk wildcard character (*) as the name of a check.
- `-policy`: Specifies the type of policy depending on what is supported by the check. This can be
 - `error`: The tool issues an error message if the check does not support `error` as a policy.
 - `tolerate`: The tool issues an error message if the check does not support `tolerate` as a policy.
 - `repair`: The tool issues an error message if the check does not support `repair` as a policy.
 - `strict`: The tool sets the first matching policy in the following order:
 1. `error`
 2. `tolerate`
 3. `repair`

For example, when a check supports the `error` and `tolerate` policies, and you select `strict` as the policy type, the `error` policy type is set.

- `lenient`: The tool sets the first matching policy in the following order:
 1. `repair`
 2. `tolerate`
 3. `error`

For example, when a check supports the `error` and `tolerate` policies, and you select `lenient` as the policy type, the `tolerate` policy type is set.

Note:

If you specify both the checks and policies, the configuration supports all the policies described in this section, depending on what the application-specific check supports. However, if you specify only the policies, but not the checks, the configuration supports only the `strict` and `lenient` policies.

- You can also use the `if_not_exists` keyword with the `-policy` option to set a policy on the current design only if the design does not already have a policy set.
- `-strategy`: Specifies the strategy to apply for a check.

Note:

- `strategy` is not applicable for the hierarchical checks.
- `-references`: Specifies the policy and strategy for reference subdesigns.

Reporting Early Data Check Records

You can report the policy set for all checks or for a specific set of checks by using the `report_early_data_checks` command. Use the following options to configure the report:

- `-policy`: Reports configuration details for all checks. It shows the selected policy and strategy for each check. Using the `-hierarchical` option with the `-policy` option reports the configuration of the subblocks.
- `-checks`: Reports the current policy, supported policies, strategies, and help text for each check. It supports an asterisk wildcard character (*) as the name of a check. Using the `-hierarchical` option with the `-checks` option reports the checks of the subblocks.
- `-verbose`: Reports a summary of all failed checks, policies, strategies applied, and checked objects with comments. Using the `-hierarchical` option with the `-verbose` option reports the records of all failed checks, policies, strategies applied, and checked objects from the subblocks.
- `-hierarchical`: Traverses the hierarchy and reports the check, policy, strategy, and fail count for each design in the hierarchy.

In the following example, the `report_early_data_checks -hierarchical -policy` command reports the policy settings for the hierarchical checks set on the top-level design and the block-level designs:

```
fc_shell> report_early_data_checks -policy
*****
Report : report_early_data_checks
Design : top
Version: R-2020.09
Date   : Thu Aug 20 23:53:54 2020
*****
Design          Check           Policy
Strategy
-----
-----
unit_des.nlib:top.design hier.block.instance_bound_to_frame      error
unit_des.nlib:top.design hier.block.instance_unlinked            error
```

Chapter 11: Hierarchical Implementation

Checking Designs With Abstracts for Top-Level-Closure Issues

```

unit_des.nlib:top.design hier.block.instance_with_design_type_macro      error
unit_des.nlib:top.design hier.top.estimated_corner                      error
-----
blk                  hier.block.abstract_missing_design_view           error
blk                  hier.block.abstract_target_use_non_implementation error
blk                  hier.block.abstract_type_non_timing                error
blk                  hier.block.leaf_cell_outside_boundary            error
-----
blk

```

In the following example, the `report_early_data_checks -hierarchical -verbose` command reports the repair performed on the checked object and also provides the details of policies, strategies applied, and checked objects on the subblocks.

```

fc_shell> report_early_data_checks -hierarchical -verbose
*****
Design Check          Policy Strategy Checked   Comment
Object
-----
blk  hier.block.missing_leaf_cell_location repair      eco_cell Instance location
                                updated to
                                X:150145 Y:69620
-----
```

While running the top-level early flow, you might run into top-level errors or warnings because of the design data violations in the subblocks or top-level design. To use the data checks and policies capability for exploring early flows, you need to know the check name corresponding to the top-level errors seen on the design. The `report_hier_check_description` command provides the required information about the checks associated with each top-level error. The following example shows the report generated by the `report_hier_check_description` command:

```

fc_shell> report_hier_check_description
*****
Report : Hier check description
Design : top
Version: R-2020.09
Date   : Fri Aug 21 00:08:32 2020
*****
```

Legend

- E - error
- R - repair
- T - tolerate
- TOP - Top-Design
- BLK - Respective Block-Ref

```

Check          Error  Tolerate Repair Allowed  Allowed
Description    ID     ID      ID    Policies References
-----
```

```

hier.block.missing_frame_view           TL-101  TL-401  TL-501  E|R|T  BLK
    Missing frame view
hier.block.abstract_missing_design_view TL-101  TL-401  N/A     E|T      BLK
    Missing design view for abstract
hier.block.reference_missing_port_location TL-126  TL-426  TL-526  E|R|T  BLK
    Missing location of physical

hierarchy boundary pin
hier.block.reference_port_outside_boundary TL-127  TL-427  TL-527  E|R|T  BLK
    Location of physical hierarchy boundary

pin outside physical hierarchy boundary

```

Generating a Report of Early Data Check Records

You can generate a report of check records from a design by using the `get_early_data_check_records` command. Using the `-hierarchical` option with this command enables to you get check records for all subblocks.

The report generated is in the following format:

`check_name@object`

For example,

```
fc_shell> get_early_data_check_records -hierarchical
{hier.block.missing_leaf_cell_location@eco_cell}
```

You can use the `-filter` option to filter the results of this command by various parameters such as object class and check name.

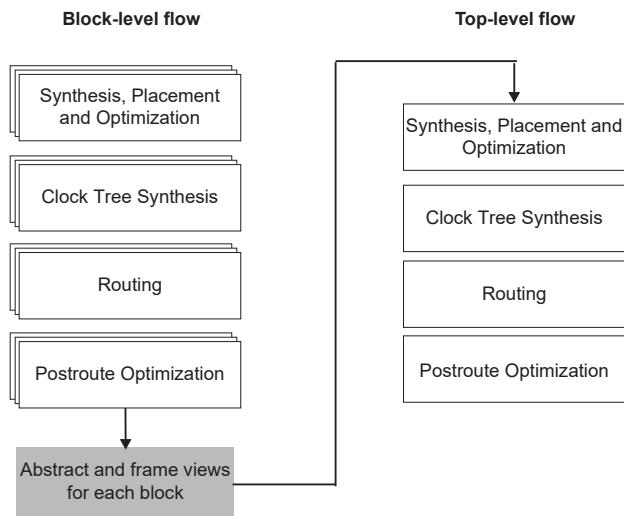
For example,

```
fc_shell> get_early_data_check_records \
-filter "checked_object_class==port"
{place.port_type_mismatch@clk route.missing_layer_direction@feed_in
 route.layer_name_mismatch@out}
```

Performing Top-Level Closure With Abstract Views

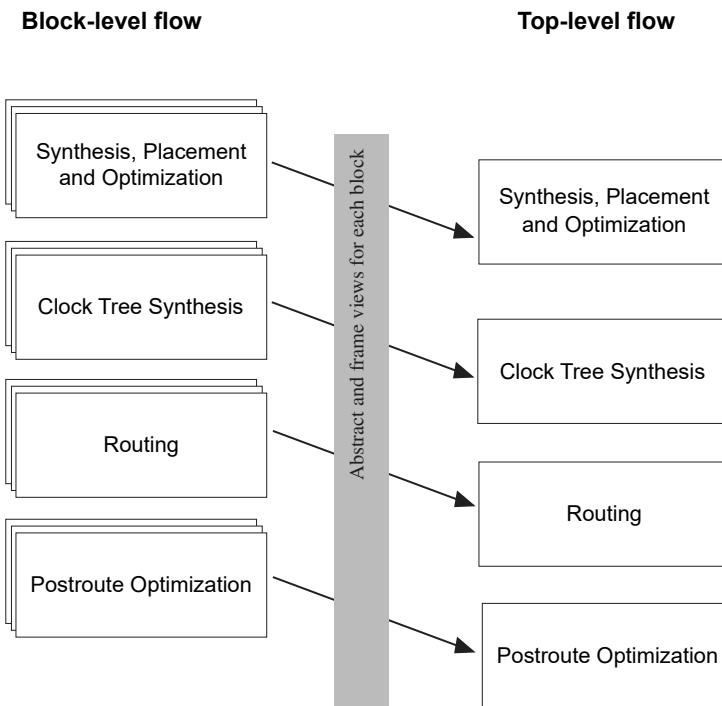
You can perform top-level closure by implementing the blocks first, and then implementing the top level, as shown in the following flow diagram:

Figure 180 *Top-Level Implemented After Blocks are Completed*



Alternatively, you can implement the blocks and top-level in parallel, as shown in the following flow diagram:

Figure 181 Top and Block Levels Implemented in Parallel



When using abstracts at the top-level, you can perform top-level synthesis, placement, optimization, clock tree synthesis, routing, and postroute optimization using the commands supported at the block level. Currently the tool does not make changes within the abstracts during top-level closure. Therefore, you can create and use read-only abstracts.

Creating ETMs and ETM Cell Libraries

To create ETMs and ETM cell libraries in the Fusion Compiler tool, see the [Creating ETMs and ETM Cell Libraries in the Fusion Compiler Tool](#) topic.

To create ETMs and ETM cell libraries in the PrimeTime and the Library Manager tools respectively, see the following topics:

- [Creating ETMs in the PrimeTime Tool](#)
- [Creating ETM Cell Libraries in the Library Manager Tool](#)

Creating ETMs and ETM Cell Libraries in the Fusion Compiler Tool

Rather than creating an ETM for each mode and corner combination using the `extract_model` command in the PrimeTime tool and then using the Library Manager tool to combine the ETMs with the corresponding physical information to create a cell library, you can perform both steps together by using the `extract_model` command directly in the Fusion Compiler tool.

The following Fusion Compiler script creates an ETM for every mode and corner of the design and then combines them with the corresponding physical information and creates the corresponding cell library:

```
# Open the design, create a frame view, and set the PrimeTime options
open block.nlib:block.design
create_frame <options>          ;# extract_model needs Frame for library
    preparation
set_pt_options -pt_exec_path <> -work_dir ETM_work_dir \
    -post_link_script <tcl script with extract_model*
variables> \
    \

# To use StarRC for Parasitic extraction
set_app_options -name extract.starrc_mode -value true
set_starrc_options -config <starrc_config_file>

# Create the ETM and generate the ETM cell library
extract_model
```

By default, the Fusion Compiler tool writes out the generated ETM library into

- [pwd]/ETM_Lib_work_dir/block_name/label_name/ for labelled blocks
- [pwd]/ETM_Lib_work_dir/block_name/ for non-labelled blocks

Creating ETMs in the PrimeTime Tool

You can create an ETM for design by using the `extract_model` command in the PrimeTime tool. For multcorner-multimode designs, you must create an ETM for each scenario by applying the appropriate corner and mode constraints for each scenario.

The following PrimeTime script creates an ETM for the S3 scenario, which consists of the m1 mode and c3 corner, of the AMS_BLK design:

```
#Read in the design
read_verilog ./AMS_BLK.v
link

# Apply parasitics
read_parasitics ./AMS_BLK.spf
```

```
#For multivoltage designs, apply UPF data and settings
load_upf ./AMD_BLK.upf
set extract_model_include_upf_data true

# Apply the mode (m1) and corner (c3) constraints for the scenario (S3)
source m1_constraints.tcl
source c3_constraints.tcl

# Enable clock latencies for designs with synthesized clock trees
set extract_model_with_clock_latency_arcs true
set extract_model_clock_latency_arcs_include_all_registers false

# Create the ETM
extract_model -library_cell -format db -output AMS_BLK_m1_c3
```

For more information about creating ETMs in the PrimeTime tool, see the Extracted Timing Models chapter in the *PrimeTime User Guide*.

Creating ETM Cell Libraries in the Library Manager Tool

After you create an ETM for each mode and corner combination, use the Library Manager tool to combine the ETMs with the corresponding physical information and create a cell library. For more information, see the *Library Manager User Guide*.

The following Library Manager scripts combine the ETMs created for every mode and corner of the AMS_BLK design with the corresponding physical information and creates the corresponding cell library:

```
# Create a library work space
create_workspace -flow etm_moded AMS_BLK

# Read the physical data (frame view)
read_ndm -views frame AMS_BLK.ndm

# Read the ETMs for every scenario
read_db -mode_label m1 AMS_BLK_m1_c1.db
read_db -mode_label m1 AMS_BLK_m1_c2.db
read_db -mode_label m1 AMS_BLK_m1_c3.db
read_db -mode_label m2 AMS_BLK_m2_c1.db
read_db -mode_label m2 AMS_BLK_m2_c2.db
read_db -mode_label m2 AMS_BLK_m2_c3.db

# Check the compatibility of the libraries you read in
check_workspace

# Generate cell library
commit_workspace -output AMS_BLK_ETM.ndm
```

When you create a cell library for an ETM, you can use one of the following methods to obtain the physical data:

- Read the frame view of the corresponding block by using the `read_ndm -views frame` command.
- Read a LEF file for the block by using the `read_lef` command.
- Read a GDSII file for the block by using the `read_gds` command
- Read an OASIS file for the block by using the `read_oasis` command.

Linking to ETMs at the Top Level

To link to ETMs at the top level, add the ETM libraries to the reference library list before you create the top-level design library, as shown in the following example:

```
fc_shell> lappend nt_ref_lib "AMS_BLK_ETM.ndm"  
fc_shell> create_lib -technology tech.tf -ref_libs $nt_ref_lib TOP
```

An ETM that contains multiple modes is called a moded ETM. For a cell instance that links to a moded ETM, you must specify the required mode by using the `set_cell_mode` command, as shown in the following example:

```
fc_shell> current_mode m1  
fc_shell> set_cell_mode m1 U1
```

For a cell instance that links to a moded ETM, if you do not specify a mode, the tool does not activate the timing arcs, timing checks, generated clocks, and case values.

To report the cell modes for specific cell instances, use the `report_cell_modes` command.

To switch a cell instance from its abstract view to its ETM or vice versa, use the `set_reference` command. The following example switches the UI cell instance from its abstract view to its ETM:

```
fc_shell> set_reference -block AMS_BLK_ETM.ndm:AMS_BLK.timing U1
```

The following example switches the UI cell instance from its ETM to its abstract view:

```
fc_shell> set_reference -block AMS_BLK.ndm:AMS_BLK.abstract U1
```

When you switch between ETMs and abstract views, reapply the top-level timing constraints.

Performing Top-Level Closure With ETMs

Before you can use the ETMs at the top level, you must perform the following tasks:

- Apply the top-level-only timing constraints and settings.

If the cell instances represented by ETMs have internal clocks, you must reapply these clock definition from the top level. Avoid cross-boundary timing exceptions that refer to objects inside the blocks.

- To apply the UPF constraints and settings, you must:

1. Generate the block-level UPF from each of the block-level designs using the `save_upf` command.
2. At the top level, load the block UPF for each of the block instances that are represented by an ETM and the top only UPF:

```
load_upf -scope <instance_name> block.upf
load_upf top.upf
```

- Apply top-level placement constraints and settings such as hard keepout margins.

When using ETMs at the top-level, you can perform synthesis, top-level placement, optimization, clock tree synthesis, routing, and postroute optimization using the commands supported at the block level.

12

RedHawk and RedHawk-SC Fusion

The RedHawk™ and RedHawk-SC™ power integrity solution is integrated with the implementation flow through the RedHawk Fusion and RedHawk-SC Fusion interface. You can use the RedHawk Fusion and RedHawk-SC Fusion feature for rail analysis at different points in the physical implementation flow after power planning and initial placement are completed. This enables you to detect potential power network issues before you perform detail routing and thus significantly reduce the turnaround time of the design cycle.

When placement is complete and the PG mesh is available, use RedHawk Fusion or RedHawk-SC Fusion to perform voltage drop analysis on the power and ground network to calculate power consumption and to check for voltage drop violations. You can perform voltage drop analysis at other stages in the design flow, such as after detail routing.

When chip finishing is complete, use RedHawk Fusion or RedHawk-SC to perform PG electromigration analysis to check for current density violations.

This chapter describes how to use the RedHawk Fusion or RedHawk-SC Fusion feature to perform rail analysis in the environment. For a detailed description about the RedHawk or RedHawk-SC analysis flows and commands, see the *RedHawk User Manual* or *RedHawk-SC User Manual*.

This chapter includes the following topics:

- [Running Rail Analysis Using RedHawk-SC Fusion](#)
- [An Overview for RedHawk Fusion and RedHawk-SC Fusion](#)
- [Setting Up the Executables](#)
- [Specifying RedHawk and RedHawk-SC Working Directories](#)
- [Preparing Design and Input Data for Rail Analysis](#)
- [Specifying Ideal Voltage Sources as Taps](#)
- [Missing Via and Unconnected Pin Checking](#)
- [Running Rail Analysis with Multiple Rail Scenarios](#)
- [Performing Voltage Drop Analysis](#)
- [Performing PG Electromigration Analysis](#)

- [Performing Minimum Path Resistance Analysis](#)
- [Performing Effective Resistance Analysis](#)
- [Performing Distributed RedHawk Fusion Rail Analysis](#)
- [Working With Macro Models](#)
- [Performing Signoff Analysis](#)
- [Writing Analysis and Checking Reports](#)
- [Displaying Maps in the GUI](#)
- [Displaying ECO Shapes in the GUI](#)
- [Voltage Hotspot Analysis](#)
- [Querying Attributes](#)

Running Rail Analysis Using RedHawk-SC Fusion

Similar to the RedHawk Fusion capability, RedHawk-SC Fusion supports gate-level rail analysis and checking capabilities in the Fusion Compiler environment. To enable the RedHawk-SC Fusion capability, you need to enable the `rail.enable_redhawk_sc` application option and disable the `rail.enable_redhawk` application option at the same time.

For more information about RedHawk-SC Fusion, see [Setting Up the Executables](#) and [Specifying RedHawk and RedHawk-SC Working Directories](#).

RedHawk Fusion and RedHawk-SC share the same set of analysis and checking commands for rail analysis, and the same GUI for examining the analysis results, except for some limitations. [Table 63](#) compares the `analyze_rail` options that are supported in either or both of RedHawk Fusion and RedHawk-SC Fusion.

Table 63 Comparing Analysis Features Between RedHawk Fusion and RedHawk-SC Fusion

| | RedHawk Fusion | RedHawk-SC Fusion |
|-----------------------------------|----------------|-------------------|
| <code>-redhawk_script_file</code> | Supported | Supported |
| <code>-voltage_drop</code> | Supported | Supported |
| <code>-switching_activity</code> | Supported | Supported |
| <code>-electromigration</code> | Supported | Supported |

Table 63 Comparing Analysis Features Between RedHawk Fusion and RedHawk-SC Fusion (Continued)

| | RedHawk Fusion | RedHawk-SC Fusion |
|---------------------------|-----------------------|--|
| -power_analysis | Supported | Supported |
| -result_name | Supported | Supported |
| -script_only | Supported | Supported |
| -bg | Supported | Supported |
| -min_path_resistance | Supported | Supported together with the -voltage_drop option only |
| -effective_resistance | Supported | Supported together with the -voltage_drop -option only |
| -check_missing_via | Supported | Supported together with the -voltage_drop option only
Displaying missing via check results does not honor the IR threshold setting. |
| -extra_gsr_option_file | Supported | Not supported |
| -multiple_script_files | Supported | Not supported |
| -submit_to_other_machines | Supported | Not supported |

Before performing rail analysis using the RedHawk-SC Fusion capability, you must specify the location of the libraries and the required input files as described in [Preparing Design and Input Data for Rail Analysis](#). The following two application options are available only in RedHawk-SC Fusion:

`rail.toggle_rate:`

Specify toggle rates for different cell types for rail analysis.

For example:

```
fc_shell> set_app_options -name rail.toggle_rate \
    -value {clock 2.0 data 0.2 combinational 0.15 \
    sequential 0.15}
```

`rail.em_only_tech_file:`

Specify the electromigration rule file for performing PG electromigration analysis.
Use this option when the electromigration rule information is defined in a

separate technology file other than the one specified by the `rail.tech_file` application option.

For example,

```
fc_shell> set_app_options -name rail.em_only_tech_file \
    -value EM_ONLY.rule
```

An Overview for RedHawk Fusion and RedHawk-SC Fusion

Both RedHawk Fusion and RedHawk-SC Fusion support gate-level rail analysis capabilities, including static and dynamic rail analysis. You can use RedHawk Fusion and RedHawk-SC Fusion to perform the following types of checking and analysis in the Fusion Compiler environment:

- Missing via and unconnected pin shape checking

To check for missing vias or unconnected pin shapes in the design, you first configure the checking-related settings by using the `set_missing_via_check_options` command, and then perform the check by using the `analyze_rail -check_missing_via` command.

For more information, see [Missing Via and Unconnected Pin Checking](#).

- Voltage drop analysis

To perform voltage drop analysis, use the `analyze_3d_rail -voltage_drop -electromigration` command.

To perform voltage drop analysis, use the `analyze_rail -voltage_drop` command or choose Rail > Analyze Rail and select “Voltage drop analysis” in the GUI.

Set the `-voltage_drop` option to `static`, `dynamic`, `dynamic_vcd`, or `dynamic_vectorless` to choose the type of analysis you want to perform.

For more information, see [Performing Voltage Drop Analysis](#).

- PG electromigration analysis

To perform PG electromigration analysis, use the `analyze_rail -electromigration` command.

For more information, see [Performing PG Electromigration Analysis](#).

- Minimum path resistance analysis

To perform minimum path resistance analysis, use the `analyze_rail -min_path_resistance` command.

In the RedHawk-SC analysis flow, you must use the `-voltage_drop` option together with the `-min_path_resistance` option for minimum path resistance analysis.

For more information, see [Performing Minimum Path Resistance Analysis](#).

Depending on the type of analysis you run, the tool generates visual displays (maps) of the results that you can view in the Fusion Compiler GUI, as well as error data that you can display in the Fusion Compiler error browser. These maps and error data help you discover problem areas and determine corrective action without leaving the Fusion Compiler environment.

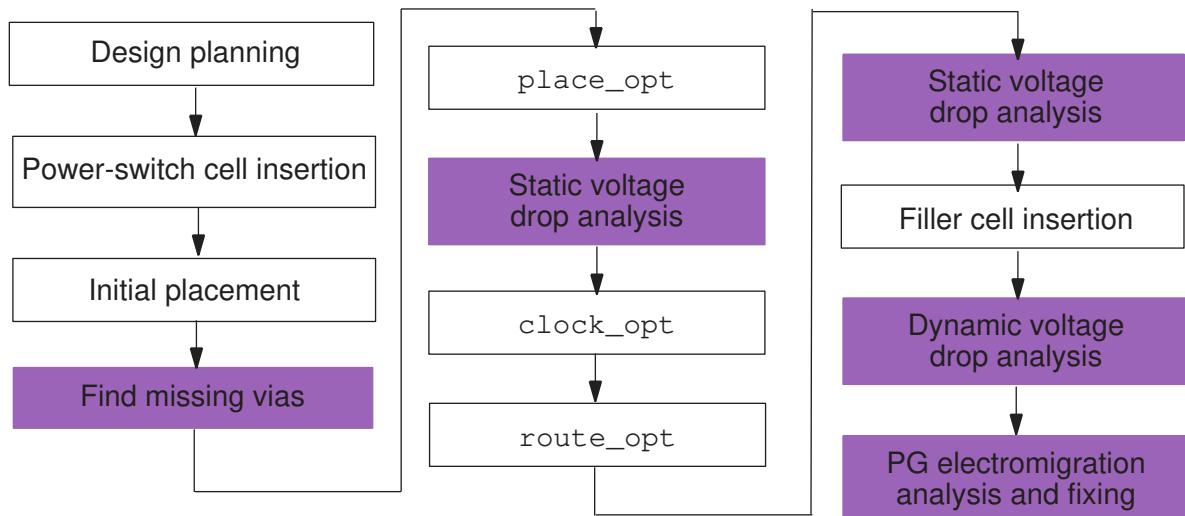
Note:

RedHawk Fusion in the Fusion Compiler tool does not support RedHawk signoff analysis features, such as hierarchical analysis or dynamic analysis with lumped or SPICE packages. For more information, see [Preparing Design and Input Data for Rail Analysis](#).

By default, RedHawk Fusion analyzes only the current design scenario when analyzing a multicorner-multimode design. For more information about how to create and specify a rail scenario for rail analysis, see [Running Rail Analysis with Multiple Rail Scenarios](#).

Figure 182 shows where you would use these analysis capabilities in a typical design flow.

Figure 182 Using RedHawk Fusion in the Fusion Compiler Design Flow



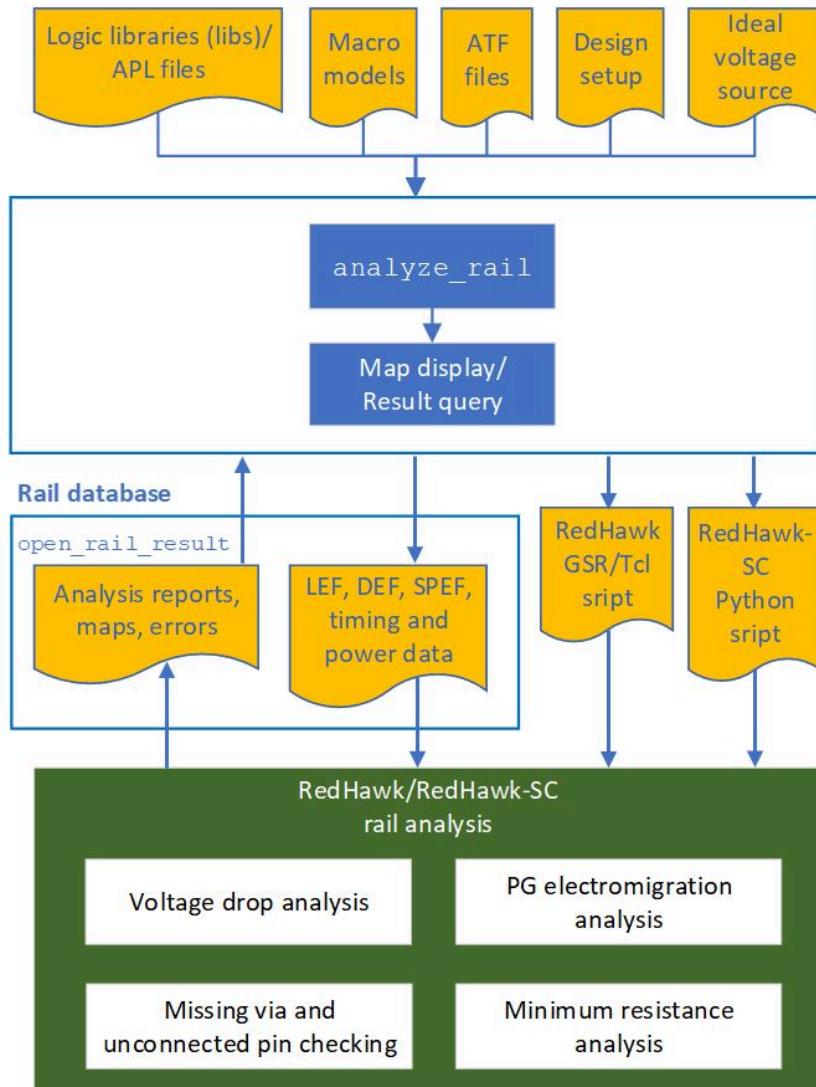
RedHawk Fusion and RedHawk-SC Fusion Data Flow

The RedHawk/RedHawk-SC Fusion feature allows you to perform rail analysis during the implementation stage. With the required input files, the Fusion Compiler tool creates the RedHawk run script and the Global System Requirements (GSR) configuration file for invoking the RedHawk tool (or the RedHawk-SC python script for invoking the RedHawk-SC tool) to run PG net extraction, power analysis, voltage drop analysis, and PG electromigration analysis.

When analysis is complete, the Fusion Compiler tool generates analysis reports and maps using the results calculated by the RedHawk or RedHawk-SC tool. You can then check for hotspots graphically in the Fusion Compiler GUI. Error data and ASCII reports are also available to check for locations where limits are violated.

[Figure 183](#) illustrates the data flow when using RedHawk Fusion or RedHawk-SC Fusion to perform rail analysis in the Fusion Compiler environment.

Figure 183 RedHawk Fusion and RedHawk-SC Fusion Data Flow

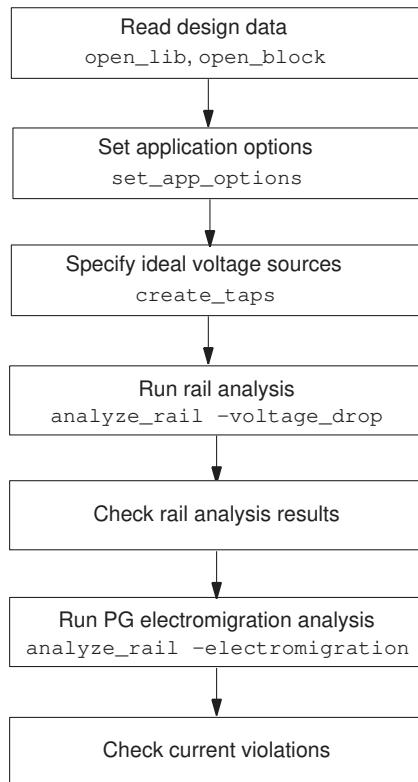


RedHawk/RedHawk-SC Fusion Analysis Flow

When you have specified the necessary input and design data, you can perform voltage drop and PG electromigration analyses on the design.

Figure 184 illustrates the steps in a basic static rail analysis flow.

Figure 184 Required Steps for a Basic Rail Analysis Using RedHawk and RedHawk-SC Fusion



Running RedHawk Fusion Commands in the Background

By default, when you execute the `analyze_rail` command to perform rail analysis, the RedHawk or RedHawk-SC tool is invoked to perform the specified analysis types with the input data and then load the analysis results back to the rail database when rail analysis is finished. You cannot run any Fusion Compiler commands unless the RedHawk Fusion or RedHawk-SC Fusion process is finished.

To perform layout editing tasks while the tool is running in the background, run the `analyze_rail -bg` command. When analysis is completed, run the `open_rail_result -back_annotate` command to upload the analysis results to the rail database.

By default, RedHawk or RedHawk-SC loads the analysis results back to the rail database when analysis is completed. However, when you specify the `-bg` option with the `analyze_rail` command, you need to run the `open_rail_result -back_annotate` command to reconstruct rail database and create the new RAIL_DATABASE file from the latest analysis results.

Note:

When back-annotating the analysis results from the previous run after moving nets in the design, the instance-based map reflects the updated instance location change, but the parasitic map does not.

When opening rail results, the tool does not detect if the result is generated with or without the `-bg` option. The command might not work correctly if you run the `open_rail_result -back_annotate` command to open the rail result that is generated without the `-bg` option, or vice versa.

Reporting and Checking the Status of the Background Process

When the RedHawk or RedHawk-SC Fusion process is finished, the tool issues the following message:

```
Info: Running REDHAWK_BINARY in background with log file: LOG_FILE
```

The `LOG_FILE` file is saved in the PWD directory.

To check if the background `analyze_rail` process is active, use the `report_background_jobs` command.

Setting Up the Executables

To run RedHawk Fusion or RedHawk-SC Fusion features, enable the features and specify the location of the RedHawk/RedHawk-SC executable by setting the `rail.product` and `rail.redhawk_path` application options.

For example,

```
fc_shell> set_app_options -name rail.product -value redhawk|redhawk_sc
fc_shell> set_app_options -name rail.redhawk_path \
           -value /tools/RedHawk_Linux64e5_V19.0.2p2/bin
```

You must ensure that the specified executable is compatible with the Fusion Compiler version that you are using.

See Also

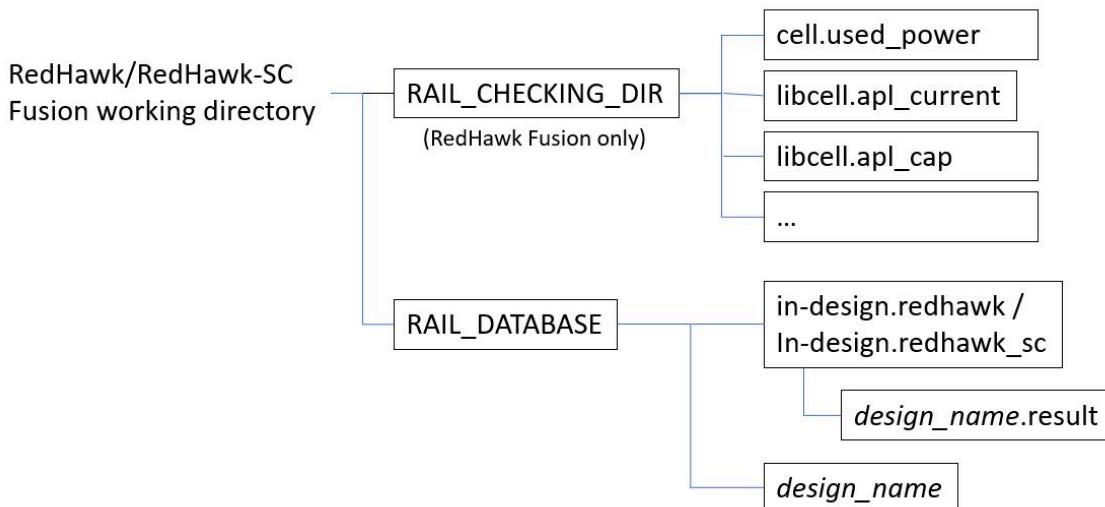
- [Specifying RedHawk and RedHawk-SC Working Directories](#)

Specifying RedHawk and RedHawk-SC Working Directories

During rail analysis, RedHawk/RedHawk-SC Fusion creates a working directory to store the generated files, including analysis logs, scripts, and various output data. By default,

the RedHawk or RedHawk-SC working directory is named RAIL_DATABASE. To use a different name for the working directory, use the `rail.database` application option.

The following figure shows the data structures:



- RAIL_CHECKING_DIR: The directory where RedHawk Fusion saves report files on the missing information during rail analysis.

For example, the tool generates the following report files in the RAIL_CHECKING_DIR directory when static rail analysis is run:

- libcell.apl_current
 - libcell.apl_cap
 - libcell.missing_liberty
- RAIL_DATABASE: The directory where the tool saves and retrieves results and log files that are generated during power calculation and rail analysis.

This directory contains the following sub-directories:

- in-design.redhawk or in-design.redhawk_sc: The directory that contains the following sub-directory:
 - design_name.result*: The directory where RedHawk or RedHawk-SC saves the analysis result files.
 - *design_name*: The directory where RedHawk or RedHawk-SC writes analysis results for the Fusion Compiler tool to retrieve for map display and data query. For example, running the `open_rail_result` command loads the data in this *design_name* directory for displaying maps in the GUI.

If you finish the analysis and want to keep the data for later retrieval, you need to save the existing rail data to another working directory by using the `rail.database` application option before proceeding to another `analyze_rail` run.

For example,

```
fc_shell> set_app_options -name rail.database \
    -value RAIL_DATABASE_STATIC_RUN
```

Otherwise, the RedHawk/RedHawk-SC tool overwrites the previously generated rail data with the new data, and issues the following warning message:

Warning: Rail result still open! Will be overwritten!

To avoid the warning message, run the `close_rail_result` command to remove all rail data from memory before proceeding to another `analyze_rail` command run.

Preparing Design and Input Data for Rail Analysis

Before analyzing voltage drop and current density violations in a design using the RedHawk/RedHawk-SC Fusion capability, use the `set_app_options` command to specify the location of the libraries and the required input files.

[Table 64](#) lists the application options for specifying design and input data for running RedHawk/RedHawk-SC rail analysis within the Fusion Compiler environment.

Table 64 Application Options for Specifying Design and Input Data

| Application Option | Description |
|-----------------------------|---|
| <code>rail.lib_files</code> | <p>Specifies the library files in .lib format.
 For example,</p> <pre>fc_shell> set_app_options \ -name rail.lib_files \ -value {test1.lib test2.lib}</pre> |
| <code>rail.tech_file</code> | <p>Specifies the Apache technology file for performing PG extraction and PG electromigration analysis.
 For example,</p> <pre>fc_shell> set_app_options \ -name rail.tech_file \ -value ChipTop.tech</pre> |

Table 64 Application Options for Specifying Design and Input Data (Continued)

| Application Option | Description |
|-------------------------|---|
| | <p>Specifies the dies that are included in detailed analysis.
 For example,</p> |
| | <pre>fc_shell> set_app_options \ -name rail.3dic_enable_die\ -value {DIE1 true DIE2 true ...}</pre> |
| | <p>Specifies user-defined Instance Power File (IPF) for providing power to pins.
 For example,</p> |
| | <pre>fc_shell> set_app_options \ -name rail.instance_power_file\ -value {E1 FILE1 DIE2 FILE2 ...}</pre> |
| rail.apl_files | <p>Specifies the Apache APL files, which contain current waveforms and intrinsic parasitics for performing dynamic rail analysis.
 For example,</p> |
| | <pre>fc_shell> set_app_options \ -name rail.apl_files -value \ {cell.current current \ cell.cdev cap}</pre> |
| rail.switch_model_files | <p>Specifies the RedHawk switch cell model files for performing dynamic rail analysis.
 For example,</p> |
| | <pre>fc_shell> set_app_options \ -name rail.switch_model_files \ -value {switch.model}</pre> |
| rail.macro_models | <p>Specifies the macro model files.</p> |
| | <p>For example,</p> |
| | <pre>fc_shell> set_app_options \ -name rail.macro_models \ -value {gds_cell1 mm_dir1 \ gds_cell2 mm_dir2}</pre> |
| rail.lef_files | <p>(Optional) Specifies a list of LEF files. When not specified, the tool generates one using the data in the design library.</p> |
| rail.def_files | <p>(Optional) Specifies a list of DEF files. When not specified, the tool generates one using the data in the design library.</p> |

Table 64 Application Options for Specifying Design and Input Data (Continued)

| Application Option | Description |
|--|---|
| <code>rail.pad_files</code> | (Optional) Specifies a list of pad location files. When not specified, you need to specify the ideal voltage sources by running the <code>create_taps</code> command. |
| <code>rail.sta_file</code> | (Optional) Specifies the static timing file (STA), which contains the final slew and delay information. When not specified, the tool generates the static timing window information using the data in the design library. |
| <code>rail.spef_files</code> | (Optional) Specifies a list of SPEF files, which contain the detailed parasitic resistive and capacitive loading data of the signal nets. When not specified, the Fusion Compiler tool generates the SPEF file using the data in the design library. |
| <code>rail.effective_resistance_instance_file</code> | (Optional) Specifies the instance file, which lists the cell instances for effective resistance calculation. |
| <code>rail.generate_file_type</code> | <p>(Optional) Specifies the type of the script file to generate for RedHawk Fusion or RedHawk-SC Fusion rail analysis. The available file types are:</p> <ul style="list-style-type: none"> <code>tcl</code> (default): Generates the file in Tcl format. The default file name is <code>set_user_input.tcl</code>. <code>python</code>: Generates the file in python format. The default file name is <code>input_files.py</code>. The python script file is supported only in RedHawk-SC Fusion rail analysis flow. <p>This application option is required when you run the RedHawk-SC rail analysis with a customized python script file. For more information, see Supporting RedHawk-SC Customized Python Script Files.</p> <p>Example:</p> <pre>fc_shell> set_app_options \ -name rail.generate_file_type -value python</pre> |

Table 64 Application Options for Specifying Design and Input Data (Continued)

| Application Option | Description |
|--|---|
| <code>rail.generate_file_variables</code> | This application option is supported in RedHawk-SC only. Specifies the preferred user variables when a customized python script file is used for RedHawk-SC rail analysis. This application option is required when the script contains non-standard collateral python variable names. See Supporting RedHawk-SC Customized Python Script Files for details.

Supported file types are: <ul style="list-style-type: none"> • LEF • DEF • SPEF • TWF • PLOC Example:
<pre>fc_shell> set_app_options \ -name rail.generate_file_variables -value \ {PLOC ploc LEF lef_files DEF def_files \ SPEF spef_files TWF tw_files}</pre> |
| <code>rail.enable_new_rail_scenario</code> | (Optional) Enables RedHawk Fusion or RedHawk-SC Fusion rail analysis with multiple rail scenarios. The default is <code>false</code> . See Running Rail Analysis with Multiple Rail Scenarios for details. |
| <code>rail.enable_parallel_run_rail_scenario</code> | (Optional) Enables distributed RedHawk Fusion or RedHawk SC Fusion analysis within the analysis flow. |
| <code>rail.scenario_name</code> | (Optional) Specifies the scenario for RedHawk or RedHawk-SC Fusion rail analysis. The tool honors the specified design scenario for the IR-driven features in power integrity flow, such as IR-driven placement, IR-driven concurrent clock and data optimization, and IR-driven optimization. See Running Rail Analysis with Multiple Rail Scenarios for details. |
| <code>rail.dump_icc2_results_style -value 2.0</code> | Specifies how to load the results with the new Ansys map style. |

Generating Rail Analysis Script Files

You can run the `analyze_rail -script_only` command to generate the settings required for running rail analysis based on the information in the design library without actually executing the analysis run.

The `analyze_rail -script_only` command writes data to the working directory. The output files that are saved to the directory are

Table 65 Output data when running `analyze_rail -script_only`

| File Name | Description | RedHawk | RedHawk-SC |
|--|--|---------|------------|
| RedHawk GSR file | Contains configuration settings for running RedHawk Fusion rail analysis | Yes | No |
| LEF/DEF, SPEF, and STA files | Generated if you do not specify these input files before running the <code>analyze_rail</code> command | Yes | Yes |
| RedHawk Fusion run script file (<code>analyze_rail.tcl</code>) | Includes the commands that are required for running rail analysis | Yes | No |
| RedHawk-SC Fusion run script file (<code>analyze_rail.py</code>) | Includes the python commands that are required for running rail analysis | No | Yes |

Supporting RedHawk-SC Customized Python Script Files

RedHawk-SC Fusion provides application options and commands to read in user-customized python script files. The tool generates all design collaterals (including DEF, LEF, SPEF, TWF and PLOC) from the design library and adds them to the user-customized python script to run voltage drop analysis in a single iteration of `analyze_rail`. The user-customized python script file is supported in rail analysis with multiple rail scenarios, as well as other IR-driven features, such as IR-driven placement or IR-driven concurrent clock and data optimization.

To use a customized python script file for RedHawk-SC Fusion rail analysis,

1. Set the type of script file to *python*.

```
fc_shell> set_app_options -name rail.generate_file_type -value python
```

2. Set the `rail.generate_file_variables` application option to specify the variables for generating the input files (including DEF, LEF, SPEF, TWF and PLOC files) to add to the user-customized python script file. You do not need to set this application option if the customized python script file uses the same values as in RedHawk-SC Fusion.

| | DEF File | LEF File | SPEF File | TWF File | PLOC File |
|--------------------------|----------|----------|-----------|----------|-----------|
| Default name in fc_shell | DEF | LEF | SPEF | TWF | PLOC |

| | DEF File | LEF File | SPEF File | TWF File | PLOC File |
|--|--|-----------------|------------------|-----------------|------------------|
| Default variable name in RedHawk-SC Fusion | def_files lef_files spef_files tw_files ploc | | | | |

For example, if you specify a `test_def_file` for the DEF variable to match with the DEF file specified in the customized python script file, set the following:

```
fc_shell> set_app_options -name rail.generate_file_variables \
    -value {DEF test_def_files}
```

3. Create and specify rail scenarios for generating the necessary input files to add to the customized python script.

Run the `create_rail_scenario` command to create a rail scenario and associate it with a design scenario in the current design. You can create multiple rail scenarios and associate them with the same scenario in the current design.

Then run the `set_rail_scenario -generate_file {LEF | TWF | DEF | SPEF | PLOC}` command to specify the rail scenario and the type of input files to generate from the design library.

Note:

When running rail analysis using multicorner-multimode technology, you must create rail scenarios using the `create_rail_scenario` command before the `set_rail_scenario` command.

The following example creates the `T_low` and `T_high` rail scenarios and generates input files for them:

```
fc_shell> create_rail_scenario -name T_low \
    -scenario func_cbest
fc_shell> set_rail_scenario -name T_low \
    -generate_file {PLOC DEF SPEF TWF LEF} \
    -custom_script_file ./customized.py
fc_shell> create_rail_scenario -name T_high \
    -scenario func_cbest
fc_shell> set_rail_scenario -name T_high \
    -generate_file {PLOC DEF SPEF TWF LEF} \
    -custom_script_file ./customized.py
```

You can then use the `customized.py` file for multi-rail-scenario rail analysis or other IR-driven features, such as IR-driven placement or IR-driven concurrent clock and data optimization.

Examples

The following example uses a customized python script for multi-rail-scenario rail analysis on a grid system.

```
#Required. Specify the below options to enable multi-rail-scenario rail
analysis.
set_app_options -list {
    rail.enable_new_rail_scenario true
    rail.enable_parallel_run_rail_scenario true
}

#Required. Set the type of the script file to python.
set_app_options -name rail.generate_file_type -value python

#Use user-defined test_def_files for DEF variable to match
#with customized python.
set_app_options -name rail.generate_file_variables \
    -value {DEF test_def_files }

create_rail_scenario -name func.ss_125c -scenario func.ss_125c
set_rail_scenario -name func.ss_125c -voltage_drop dynamic

create_rail_scenario -name func.ff_125c_ -scenario func.ff_125c
set_rail_scenario -name func.ff_125c \
    -generate_file {PLOC DEF SPEF TWF LEF} \
    -custom_script_file ./customized.py

set_host_options -submit_protocol sge \
    -submit_command {qsub -V -notify -b y \
        -cwd -j y -P bnormal -l mfree=16G}

analyze_rail -rail_scenario {func.ss_125c func.ff_125c} \
    -submit_to_other_machine
```

The following example uses a customized python script for multi-rail-scenario rail analysis on a local machine.

```
#Required. Specify the below options to enable multi-rail-scenario rail
analysis.
set_app_options -list {
    rail.enable_new_rail_scenario true
    rail.enable_parallel_run_rail_scenario true
}

#Required. Set the type of the script file to python.
set_app_options -name rail.generate_file_type -value python

#Use user-defined test_def_files for DEF variable to match
#with customized python.
set_app_options -name rail.generate_file_variables \
    -value {DEF test_def_files }
```

```

create_rail_scenario -name func.ss_125c -scenario func.ss_125c
set_rail_scenario -name func.ss_125c -voltage_drop dynamic

create_rail_scenario -name func.ff_125c_ -scenario func.ff_125c
set_rail_scenario -name func.ff_125c \
    -generate_file {PLOC DEF SPEF TWF LEF} \
    -custom_script_file ./customized.py

set_host_options -submit_command {local}

analyze_rail -rail_scenario {func.ss_125c func.ff_125c}

```

The following example uses a customized python script for single-scenario rail analysis on a grid system.

Required. Specify the below options to enable multi-rail-scenario rail analysis.

```

set_app_options -list {
    rail.enable_new_rail_scenario true
    rail.enable_parallel_run_rail_scenario true
}

#Required. Set the type of the script file to python.
set_app_options -name rail.generate_file_type -value python

#Use user-defined test_def_files for DEF variable to match
#with customized python.
set_app_options -name rail.generate_file_variables \
    -value {DEF test_def_files }

create_rail_scenario -name func.ff_125c_ -scenario func.ff_125c
set_rail_scenario -name func.ff_125c \
    -generate_file {PLOC DEF SPEF TWF LEF} \
    -custom_script_file ./customized.py

set_host_options -submit_protocol sge \
    -submit_command {qsub -V -notify -b y \
    -cwd -j y -P bnornal -l mfree=16G}

analyze_rail -rail_scenario {func.ff_125c} \
    -submit_to_other_machine

```

The following example uses a customized python script for IR-driven placement on a grid system.

Required. Specify the below options to enable multi-rail-scenario rail analysis.

```

set_app_options -list {
    rail.enable_new_rail_scenario true
    rail.enable_parallel_run_rail_scenario true
}

```

```
#Required. Set the type of the script file to python.
set_app_options -name rail.generate_file_type -value python

#Use user-defined test_def_files for DEF variable to match
#with customized python.
set_app_options -name rail.generate_file_variables \
    -value {DEF test_def_files }

create_rail_scenario -name func.ss_125c -scenario func.ss_125c
set_rail_scenario -name func.ss_125c -voltage_drop dynamic

create_rail_scenario -name func.ff_125c_ -scenario func.ff_125c
set_rail_scenario -name func.ff_125c \
    -generate_file {PLOC DEF SPEF TWF LEF} \
    -custom_script_file ./customized.py

set_host_options -submit_protocol sge \
    -submit_command {qsub -V -notify -b y \
        -cwd -j y -P bnormal -l mfree=16G}

# Enable IR-driven placement before running clock_opt -from final_opto
set_app_options -name opt.common.power_integrity -value true

clock_opt -from final_opto
```

The following example uses a customized python script for concurrent clock and data optimization on a multicorner and multimode design on a grid system.

```
#Required. Specify the below options to enable MCMM-based optimization.
set_app_options -list {
    rail.enable_new_rail_scenario true
    rail.enable_parallel_run_rail_scenario true
}
set_app_options -name rail.generate_file_type -value python

#Use user-defined test_def_files for DEF variable to match
#with customized python
set_app_options -name rail.generate_file_variables \
    -value {DEF test_def_files }

create_rail_scenario -name func.ss_125c -scenario func.ss_125c
set_rail_scenario -name func.ss_125c -voltage_drop dynamic

create_rail_scenario -name func.ff_125c_ -scenario func.ff_125c
set_rail_scenario -name func.ff_125c \
    -generate_file {PLOC DEF SPEF TWF LEF} \
    -custom_script_file ./customized.py

set_host_options -submit_protocol sge \
    -submit_command {qsub -V -notify -b y -cwd -j y \
        -P bnormal -l mfree=16G}
```

```
# Enable IR-driven CCD analysis before running route_opt
set_app_options -name opt.common.power_integrity -value true

route_opt
```

See Also

- [Specifying RedHawk and RedHawk-SC Working Directories](#)

Specifying Ideal Voltage Sources as Taps

Taps are used to model the external voltage source environment in which the device under analysis operates; they are not part of the design itself, but can be thought of as virtual models of voltage sources. The locations of the ideal voltage sources and the ideal power supplies in the design are required to achieve accurate rail analysis results.

To create a tap, use the `create_taps` command. You can create multiple taps by running multiple `create_taps` commands.

To specify a name for the created tap, use the `-name` option. If the specified name is already present in the design, the tool issues a warning message, and replaces the original tap with the new one. When you create multiple taps by a single invocation of the `create_taps` command, all of the created taps are named by using the `tap_name_NNN` naming convention, where NNN is a unique integer identifier and `tap_name` is the string argument to the `-name` option.

You can create taps in any of the following ways:

- Treating top-level PG pins as taps

Use the `-top_pg` option when the block does not have physical pad or bump information available.

Use the `-supply_net` option to explicitly associate a tap to the supply (either power or ground) net connected to the object from which the tap was defined. To implicitly associate a tap with the supply net, run the `create_taps` command without the `-supply_net` option.

The following example creates taps for all the top-level supply pins.

```
fc_shell> create_taps -supply_net VDD -top_pg
```

- Creating taps at absolute coordinates

Use the `-layer` and `-point` options to create taps at the specified coordinates. If there is no supply net, supply pin, or via shape at the specified location, which means there is no conductive path to the supply network, the tap has no effect on rail analysis despite being present.

If you define a tap in a location where a tap already exists, the tool issues a warning message and replaces the original tap with the new one.

You must specify the supply net with which this tap is associated by using the `-supply_net` option.

Note:

When you use the `-point` option with the `-of_objects` option, the location is relative to the origin of the specified object. Otherwise, it is relative to the origin of the current top-level block.

The following example creates a tap using absolute coordinates. The command checks whether the coordinate location touches any layout shape.

```
fc_shell> create_taps -supply_net VDD \
    -layer 3 -point {100.0 200.0}
Warning: Tap point (100.0, 200.0) on layer 3 for net VDD does not
touch any polygon (RAIL-305)
```

- Using existing instances as taps

To create taps on specified objects, use the `create_taps -of_objects` command. Each item in the object list can be a cell or a library cell, or a collection of cells and library cells. By default, the tool creates the taps on the power and ground pins of the objects. If you specify the `-point` option, the tool creates the taps at the specified physical location relative to the origin of each object.

Use existing instances for creating taps when analyzing a top-level design where pad cells are instantiated physically.

Use the `-supply_net` option to explicitly associate a tap to the supply (either power or ground) net connected to the object from which the tap was defined. To implicitly associate a tap with the supply net, run the `create_taps` command without the `-supply_net` option.

The following example creates taps for each instance of the `some_pad_cell` library cell. The tool creates one tap per instance on the metal1 layer at a location of `{30.4 16.3}` relative to the origin of the cell instance.

```
fc_shell> create_taps -of_objects [get_lib_cell */some_pad_cell]\
    -layer metal1 -point {30.4 16.3}
```

The following example creates taps on all power and ground pins of all cell instances that match the `*power_pad_cell*` pattern.

```
fc_shell> create_taps -of_objects [get_cells -hier \
    *power_pad_cell*]
```

- Importing a tap file

If you are working with a block that does not yet have pins defined, or if the location or design of the pad cells is not finalized, you can define tap locations, layer numbers, and supply nets in an ASCII file and then import it into the tool by using the `create_taps -import` command.

The supported tap file format is as follows:

```
net_name layer_number [X-coord Y-coord]
```

Lines starting with semicolon (;) or pound (#) symbols are treated as comments. The x-coordinate and y-coordinate values are specified in microns.

For example,

```
fc_shell> exec cat tap_file
      VDD 3 500.000 500.000
      VSS 5 500.000 500.000
fc_shell> create_taps -import tap_file
```

- Creating taps with packages

A simple package RLC (resistance-inductance-capacitance) model provides a fixed set of electrical characteristics assigned to all power and ground pads. To use simple package RLC models for rail analysis, you need to specify the pad location file which defines the simple RLC model by using `rail.pad_files` application option.

For a detailed description about simple package RLC models, see the related section in the *RedHawk User Manual*.

Note:

Before creating taps with simple package RLC models, you must enable the RedHawk signoff license key using the `rail.allow_redhawk_license_checkout` application option.

Validating the Taps and Finding Invalid Taps

When creating taps, by default the `create_taps` command verifies that the created taps are inserted in valid locations. The tool issues a warning message if the tap does not touch any layout shape. To place taps outside a shape without issuing a warning message, disable the checking by using the `-nocheck` option.

The following command creates a tap with the `-nocheck` option enabled. No warning message is issued.

```
fc_shell> create_taps -supply_net VDD -layer 3 \
      -point {100.0 200.0} -nocheck
```

Tap Attributes

During validation checking, the tool marks each inserted tap object with the `is_valid_location` attribute to indicate its validity. When validation checking is enabled, the attribute value is `true` for the valid taps and `false` for the invalid taps. When the `-nocheck` option is used, the attribute value is `unknown` for all taps.

The tool updates the tap attributes in subsequent voltage drop and minimum path resistance analyses. Each tap belonging to the PG nets is checked against the extracted shape. When the analysis is complete, the tool updates the tap validity status based on the rail analysis results.

The following example first lists the attributes of tap objects that are added to the block in this command run, and then checks if the Tap_1437 tap passes the validation checking.

```
fc_shell> list_attributes -application -class tap
...
Properties:
  A - Application-defined
  U - User-defined
  I - Importable from design/library (for user-defined)
  S - Settable
  B - Subscripted

Attribute Name          Object      Type       Properties  Constraints
-----                  -----
context                tap        string     A           integrity,
                      rail,                   session
full_name              tap        string     A
is_valid_location       tap        boolean   A
layer_name              tap        string     A
layer_number            tap        int       A
net                     tap        string     A
object_class            tap        string     A
parasitics              tap        string     A
position                tap        coord     A
static_current          tap        double    A
type                   tap        string     A
absolute_coord,
auto_pg, cell_coord, cell_pg, lib_cell_coord, lib_cell_pg, signal,
terminal,
top_pg
fc_shell> get_attribute [get_taps Tap_1437] is_valid_location
true
```

Finding Invalid Taps

To find taps that fail the validity checking, use the `get_taps` command to report the taps that have `is_valid_location` attribute set to `false`.

For example,

```
fc_shell> get_attribute [ get_taps Tap_1437 ] is_valid_location
false
```

Finding Substitute Locations for Invalid Taps

If the specified location for a tap is invalid, use the `-snap_distance` option of the `create_taps` command to find a substitute location for the tap. The command searches the nearby layout shapes within the specified snap distance both horizontally and vertically on the same layer as defined by the `-layer` option. The tap is reinserted at the corner of a valid shape which is within the minimum distance to the specified location.

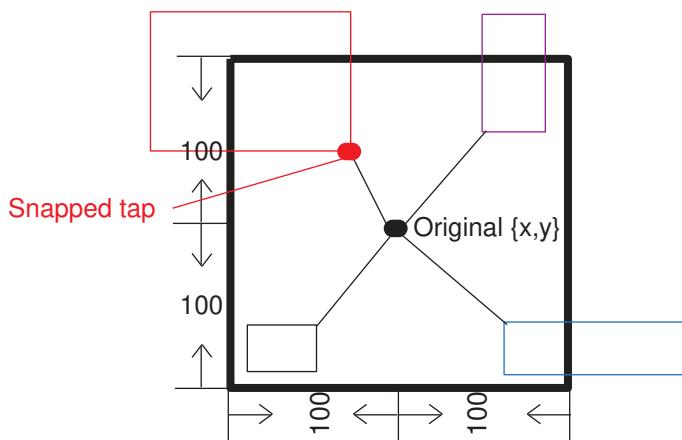
Note:

You can search for a substitute location only for the invalid taps that are specified by the `-point` and `-layer` options. For example,

```
fc_shell> create_taps -point {50,50} -layer {M1} \
-snap_distance 100
Snap tap point from original location (50.000, 50.000) to new
location (90.000, 65.910)
```

As shown in [Figure 185](#), the specified tap location does not touch any layout shape. The red rectangle is the shape closest to the specified location and its lower-right corner has the minimum distance among all the other rectangles. The tool reinserts the tap at the lower-right corner of the red rectangle.

Figure 185 Finding a Substitute Location for an Invalid Tap



Removing Taps

To remove taps that were previously created using the `create_taps` command, use the `remove_taps` command. To remove specific taps, use the `-filter` option with the `get_taps` command. The following example removes all the supply pin taps that are of the type `top_pg`:

```
fc_shell> remove_taps [get_taps -filter type==top_pg]
```

To remove all the taps in the block, run the following command:

```
fc_shell> remove_taps
```

Missing Via and Unconnected Pin Checking

Note:

Missing via and unconnected pin checking is supported in both RedHawk Fusion and RedHawk-SC Fusion analysis flows.

Before you check for missing vias and unconnected pins, make sure you have the required input files as described in [Preparing Design and Input Data for Rail Analysis](#).

Check for missing vias or unconnected pins in the block during voltage drop analysis or in a separate run. For example, you can perform the checking after completing the power structure and before running the `place_opt` command. This allows you to find power network errors in the design before rail analysis.

By default, RedHawk/RedHawk-SC Fusion detects the following types of power network errors:

- Unconnected pin shapes: Pin shapes that are not contiguously and physically connected to an ideal voltage source.
- Missing vias: If two overlapping metal shapes do not contain a via within their overlapping area, it is considered a potential missing via error.

To check for and fix missing vias or unconnected pins,

1. Define configuration options by using the `set_missing_via_check_options` command, as described in [Setting Checking Options](#).
2. Save the block by using the `save_block` command.
3. Perform the checking by using the `analyze_rail -check_missing_via` command, as described in [Checking Missing Vias and Unconnected Pins](#). In the RedHawk-SC Fusion flow, you must specify the `-check_missing_via` option together with the `-voltage_drop` option.

4. Examine the checking results by opening the generated error files in the error browser or writing the checking results to an output file, as described in [Viewing Error Data](#) and [Writing Analysis and Checking Reports](#).
5. Insert PG vias to fix the reported missing vias, as described in [Fixing Missing Vias](#).

Setting Checking Options

Before checking for missing vias or unconnected pins in the design, use the `set_missing_via_check_options` command to specify the necessary options.

To check or remove the option settings, use the `report_missing_via_check_options` or `remove_missing_via_check_options` command, respectively.

[Table 66](#) lists the commonly used options for the `set_missing_via_check_options` command. For a detailed description about each option, see the man page.

Table 66 Commonly Used Options for the set_missing_via_check_options Command

| Option | Description |
|--|--|
| <code>-redhawk_script_file</code> | Uses an existing RedHawk script file for the missing via check. |
| <code>-exclude_stack_via</code> | Excludes stacked vias from the missing via check. By default, the tool includes stacked vias during analysis. |
| <code>-check_valid_vias</code> | Reports if the metal overlap can accommodate a valid via model. |
| <code>-exclude_cell / -exclude_instance</code> | Excludes the area covered by all instances of the specified cell, or by a list of cell instances from the missing via check. By default, the tool performs checking at the full-chip level. |
| <code>-exclude_regions</code> | Specifies rectangular regions to exclude from the missing via check. |
| <code>-sort_by_hot_inst</code> | Sorts missing vias by voltage drop values. By default, the tool sorts by the average delta voltage between defined layers. |
| <code>-threshold</code> | Sets a voltage threshold that triggers a missing via check. Set the option to <code>-1</code> to disable voltage check filtering. The default is 1 V. This option is valid only when the checking is run during voltage drop analysis. |
| <code>-min_width</code> | Ignores segments smaller than the specified width. |
| <code>-pitch</code> | Specifies the pitch for missing vias on parallel wires crossing or touching GDSII blocks. |

Table 66 Commonly Used Options for the `set_missing_via_check_options` Command (Continued)

| Option | Description |
|---|---|
| <code>-gds</code> | Flags missing vias that are inside the GDSII block region and are in routes across the GDSII blocks. By default, the missing via check ignores items that are inside or overlap a block that has been processed by the GDS2DEF or GDSMMX utility. |
| <code>-ignore_inter_met, -toplayer, -bottomlayer</code> | Ignores wires on all layers between the specified top and bottom layers. |

Checking Missing Vias and Unconnected Pins

Use the `analyze_rail -check_missing_via` command to check for missing vias and unconnected pins in the block based on the settings specified by the `set_missing_via_check_options` command.

Use the `-voltage_drop` option if you want to run missing via and unconnected pin checks together with voltage drop analysis. The tool then reports all missing vias found in the geometry with voltage values in the missing via report. To set a voltage threshold across layers for filtering the report, use the `-threshold` option of the `set_missing_via_check_options` command. You can set multiple threshold values in a missing via check.

Note:

In RedHawk-SC Fusion flow, you must specify the `-check_missing_via` option together with the `-voltage_drop` option.

In cases you want to report missing vias with and without setting a voltage threshold in one `analyze_rail` run, first set the desired threshold value (for example, 0.001) and then reset the value to -1. Setting `-threshold` option to -1 disables voltage checking.

When the missing via check is completed, open the generated error data to examine the errors in the error browser. For details, see [Viewing Error Data](#).

Example 46

```
fc_shell> set_missing_via_check_options -exclude_stack_via \
    -threshold 0.0001
fc_shell> analyze_rail -voltage_drop static -check_missing_via \
    -nets {VDD VSS}
```

The above example sets the threshold value to 0.0001 and compares voltages across two ends of a via, excluding stack vias. It then runs the missing via check during static rail analysis.

The RedHawk script file includes the following commands:

```
perform extraction -power -ground
perform analysis -static
mesh vias -report_missing -exclude_stack_via -threshold 0.0001 \
-o apache.missingVias1
```

Example 47

```
fc_shell> set_missing_via_check_options -exclude_stack_via \
-threshold -1
fc_shell> analyze_rail -voltage_drop -check_missing_via \
-nets {VDD VSS}
```

The above example sets the threshold value to -1 to disable the voltage checking, and runs the missing via check.

The RedHawk script file includes the following commands:

```
perform extraction -power -ground
mesh vias -report_missing -exclude_stack_via -threshold -1 \
-o apache.missingVias1.nothreshold
```

Example 48

```
fc_shell> set_missing_via_check_options -exclude_stack_via \
-threshold 0.001
fc_shell> set_missing_via_check_options -exclude_stack_via \
-threshold 0.002
fc_shell> set_missing_via_check_options -exclude_stack_via \
-threshold -1
fc_shell> analyze_rail -voltage_drop static -check_missing_via \
-nets {VDD VSS}
fc_shell> report_rail_result -type missing_vias -supply_nets \
{VDD VSS} rpt.missing_vias
Information: writing rpt.missing_vias.no_threshold done
Information: writing rpt.missing_vias done
```

In the above example, three different threshold values are set for a missing via check. The tool generates two missing via reports: rpt.missing_vias.no_threshold includes all missing vias found in the geometry, while rpt.missing_vias includes the vias whose voltage difference across two ends is less than 0.001 and 0.002.

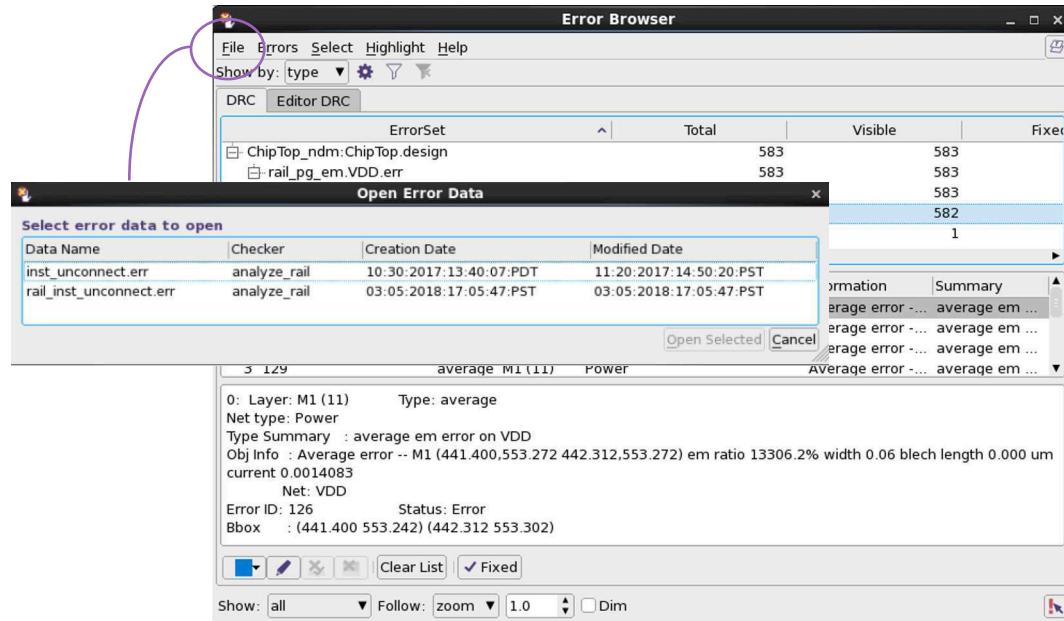
Viewing Error Data

RedHawk/RedHawk-SC Fusion saves the checking error data file, named `missing_via.supplyNetName.err`, in the working directory. The tool generates one error data file for each supply net. Before you quit the current session, be sure to save the block to keep the generated error data in the block.

To open an error data file in the error browser, choose File > Open ... from the Error Browser dialog box, and then select the error data file to open in the Open Error Data dialog box (see [Figure 186](#)).

For more information about the error browser, see “Using the Error Browser” in the *Fusion Compiler Graphical User Interface User Guide*.

Figure 186 Opening Error Data from the Error Browser



See Also

- [Specifying RedHawk and RedHawk-SC Working Directories](#)

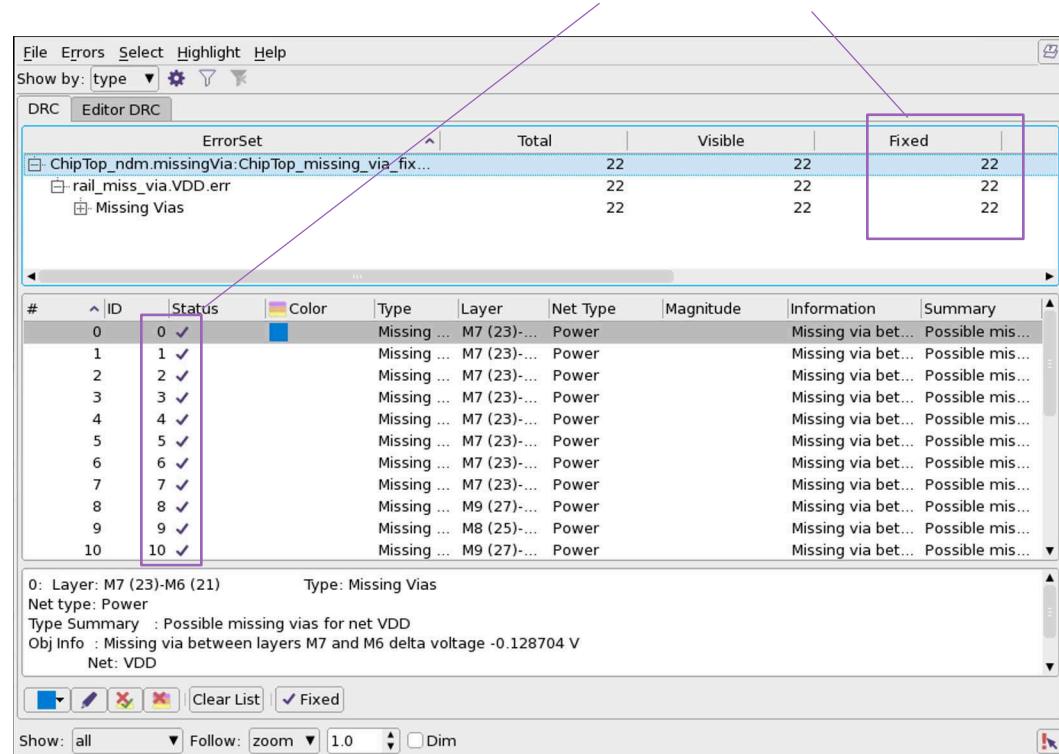
Fixing Missing Vias

To fix the missing vias by inserting PG vias based on the specified DRC error objects written by the `analyze_rail` command, use the `fix_pg_missing_vias` command.

When the tool inserts a PG via for a missing via error object, it sets the status of the error object to `fixed` in the error browser (see [Figure 187](#)). This allows you to quickly check if all the missing vias are fixed.

Figure 187 Error Status After Inserting PG Vias for Missing Vias

The tool marks the error object as fixed



The following example inserts PG vias for all error objects of the VDD nets reported by the `analyze_rail -check_missing_via` command:

```
fc_shell> set errdm [open_drc_error_data rail_miss_via.VDD.err]
fc_shell> set errs [get_drc_errors -error_data $errdm]
fc_shell> fix_pg_missing_vias -error_data $errdm $errs
```

Running Rail Analysis with Multiple Rail Scenarios

To control scenarios for RedHawk Fusion or RedHawk-SC Fusion rail analysis:

- Use the `rail.scenario_name` application option to specify a different design scenario for rail analysis.

For details, see [Specifying a Design Scenario for Rail Analysis](#).

- You can create rail scenarios and associate them with a design scenario in the current design. This allows you to identify possible power integrity issues by running optimization and rail analysis on one or more rail scenarios.

For details, see [Creating and Specifying Rail Scenarios for Rail Analysis](#).

Specifying a Design Scenario for Rail Analysis

By default, RedHawk or RedHawk-SC Fusion analyzes only the current design scenario for a multicorner-multimode design. To specify a different design scenario for rail analysis, use the `rail.scenario_name` application option. The tool honors the specified design scenario for the IR-driven features in power integrity flow, such as IR-driven placement, IR-driven concurrent clock and data optimization, and IR-driven optimization.

Here is a script example:

```
open_lib
open_block

set_app_option -name rail.technology -value 7
set_app_options -name rail.tech_file -value ./Apache.tech
set_app_options -name rail.lib_files -value \\
  A.lib \
  B.lib }
set_app_options -name rail.switch_model_files -value
{./switch_cells.txt }
source ./test_taps.tcl

current_scenario func_max
set_app_options -name rail.scenario_name -value func_typ

analyze_rail -voltage_drop
```

Creating and Specifying Rail Scenarios for Rail Analysis

You can create multiple rail scenarios and associate them with a design scenario in the current design. The created rail scenarios are saved in the design library.

To run rail analysis on multiple rail scenarios:

1. Specify the following application options to enable multi-rail-scenario rail analysis:

```
fc_shell> set_app_options -list {
    rail.enable_new_rail_scenario true
    rail.enable_parallel_run_rail_scenario true
}
```

2. Run the `set_scenario_status -ir_drop` command to enable rail analysis for the specified design scenario.

```
fc_shell> set_scenario_status func_cbest -ir_drop true
```

3. Create a rail scenario and associate it with an existing design scenario. Specify rail attributes for the created rail scenario with the `set_rail_scenario` command. You can create multiple rail scenarios for different purposes and associate them with an existing design scenario.

```
fc_shell> create_rail_scenario -name T_low \
    -scenario func_cbest
fc_shell> set_rail_scenario -name T_low \
    -voltage_drop dynamic -extra_gsr_option_file extra.gsr \
    -nets {VDD VSS}
fc_shell> create_rail_scenario -name T_high \
    -scenario func_cbest
fc_shell> set_rail_scenario -name T_high \
    -voltage_drop dynamic -extra_gsr_option_file extra2.gsr \
    -nets {VDD VSS}
```

4. Specify the multicore processing setting with the `set_host_options` command. For example, to run rail analysis on a local host, use the following command:

```
fc_shell> set_host_options -submit_command {local}
```

To run analysis on a grid system, use the following command:

```
fc_shell> set_host_options -submit_protocol sge \
    -submit_command {qsub -V -b y -cwd -P bnrmal -l mfree=16G}
```

5. Run rail analysis with the `analyze_rail` command.

```
fc_shell> analyze_rail -rail_scenario {T_high T_low} \
    -submit_to_other_machines
```

Note:

To use the multi-rail-scenario rail analysis capability for IR-driven placement or IR-driven concurrent clock and data optimization, you must set the following application option before the `clock_opt -from final_opt` or `route_opt` command:

```
fc_shell> set_app_options \
    -name opt.common.power_integrity -value true
```

For more information about power integrity analysis, see the [Enabling the Power Integrity Features](#) section.

Examples

The following example runs rail analysis on multiple rail scenarios.

```
# Specify options for multi-rail-scenario rail analysis
set_app_options -list {
    rail.enable_new_rail_scenario true
    rail.enable_parallel_run_rail_scenario true
}

set_host_options -submit_protocol sge \
    -submit_command {qsub -V -notify -b y -cwd -j y \
    -P bnormal -l mfree=16G}

# Custom RedHawk-SC Fusion python script to specify user-defined
# toggle rate for scenario T_custom
create_rail_scenario -name T_custom -scenario func_cbest
set_rail_scenario -name T_custom -custom_script_file custom_sc.py

# Default RedHawk-SC Fusion toggle rate for scenario T_def
create_rail_scenario -name T_def -scenario func_cbest
set_rail_scenario -name T_def -voltage_drop dynamic -nets {VDD VSS}

# Run RedHawk-SC Fusion rail analysis on the specified rail scenario
analyze_rail -rail_scenario {T_custom T_def} -submit_to_other_machines

# (Optional) Open and examine the generated rail results
open_rail_result {T_custom T_def}
```

The following example runs multi-rail-scenario rail analysis in the power integrity flow.

```
## Specify RedHawk/RedHawk-SC based options
set_app_options -name rail.enable_redhawk -value true
set_app_options -name rail.enable_redhawk_sc -value false
set_app_options -name rail.redhawk_path -value /test/bin
set_app_options -name rail.tech_file -value design_data/tech/rh.tech
set_app_options -name rail.lib_files -value {
    inputs/test/CCSP_test.lib
}
set_host_options -submit_protocol sge -submit_command {qsub -V \
    -notify -b y -cwd -j y -P bnormal -l mfree=16G}

## Specify options for running optimization on multiple rail scenarios
set_app_options -list {
    rail.enable_new_rail_scenario true
    rail.enable_parallel_run_rail_scenario true
}
## Enable IR drop flag for a given scenario
set_scenario_status [current_scenario] -ir_drop true

## Create a rail scenario and specify associated options
create_rail_scenario -name T_low -scenario func_cbest
```

```
set_rail_scenario -name T_low -voltage_drop dynamic \
    -extra_gsr_option_file extra.gsr -nets {VDD VSS}

create_rail_scenario -name T_high -scenario func_cbest
set_rail_scenario -name T_high -voltage_drop dynamic \
    -extra_gsr_option_file extra2.gsr -nets {VDD VSS}

## Enable power integrity flow
set_app_options -name opt.common.power_integrity -value true
```

Performing Voltage Drop Analysis

Note:

Before running the `analyze_rail` command to calculate voltage drops and current violations, ensure you have provided the input files required by each analysis mode.

For more information about preparing input files, see [Preparing Design and Input Data for Rail Analysis](#).

Voltage drop analysis is supported in both RedHawk Fusion and RedHawk-SC Fusion analysis flows.

To invoke the RedHawk/RedHawk-SC tool for calculating voltage drops of the specified power and ground networks, use the `analyze_rail` command with the following options:

- Use the `-voltage_drop` option to specify the analysis mode.
 - `static`: Performs static voltage drop analysis.
 - `dynamic`: Performs dynamic voltage drop analysis.
 - `dynamic_vectorless`: Performs dynamic voltage drop analysis without Verilog Change Dump (VCD) inputs.
 - `dynamic_vcd`: Performs dynamic voltage drop analysis with VCD inputs. When enabled, use the `-switching_activity` option to specify a switching activity file.
- Use the `-nets` option to specify the power and ground supply nets to analyze. The tool considers all the switched or internal power nets of the specified power nets in the analysis. You do not need to explicitly specify the internal power nets.
- (Optional) Use the `-switching_activity` option to specify a switching activity file. The supported file formats are: VCD, SAIF, and ICC_ACTIVITY. When not specified, by default the tool uses the toggle rate for rail analysis.

Note:

If you are using the dynamic vector-free mode, you do not need to provide switching activity information.

To specify a switching activity file, use the `-switching_activity` option by using the following syntax:

```
-switching_activity {type file_name [strip_path] [start_time end_time]}
```

- **VCD:** Specifies a VCD file that is generated from a gate-level simulation. By default, rail analysis reads and uses all time values in the VCD file.

To specify the time window to read from the VCD file, use the optional `start_time` and `end_time` arguments with the `-switching_activity` option. When specified, the tool reads all time values from the VCD file, but uses only the specified time window for rail analysis.

- **SAIF:** Specifies one or more SAIF files generated from a gate-level simulation.
- **icc_activity:** Runs the Fusion Compiler `write_saif` command to generate a SAIF file that contains the user-annotated and simulated switching activity (including state-dependent and path-dependent information) for the complete design without running the propagation engine.

The `strip_path` argument removes the specified string from the beginning of the object names. Using this option modifies the object names in the VCD or SAIF file to make them compatible with the object names in the block. The `strip_path` argument is case-sensitive.

- (Optional) By default, the tool uses the RedHawk/RedHawk-SC power analysis feature to calculate the power consumption of the specified power and ground network. If you prefer having the Fusion Compiler tool generate the necessary power data for rail analysis, set the `-power_analysis` option to `icc2`.

For a detailed description about how to run power analysis using the Fusion Compiler `report_power` command, see .

- (Optional) By default, the `analyze_rail` command automatically generates a GSR file for running RedHawk rail analysis. To specify an external GSR file to append to the GSR file generated in the current run, use the `-extra_gsr_option_file` option.
- (Optional) Use the `-redhawk_script_file` option to specify an existing RedHawk script file generated in an earlier `analyze_rail` run.

When using an existing script file for rail analysis, the tool honors the settings in this file and ignores all other currently enabled options. For example, if you run the

`analyze_rail -voltage_drop static -redhawk_script_file myscript.tcl`
command, the tool ignores the `-voltage_drop` option setting.

When the `-redhawk_script_file` option is not specified, you must specify the nets to analyze by using the `-nets` option.

The following example performs static voltage drop analysis on the VDD and VSS nets with the optional settings defined in a GSR configuration file.

```
fc_shell> analyze_rail -voltage_drop static -nets {VDD VSS} \
    -extra_gsr_option_file add_opt.gsr
```

Viewing Voltage Drop Analysis Results

When analysis is complete, the tool saves the analysis results and logs in the working directory. For more information about the working directories, see [Specifying RedHawk and RedHawk-SC Working Directories](#).

The `analyze_rail` command generates the following files to invoke the RedHawk tool for running rail analysis:

- The RedHawk run script (`analyze_rail.tcl`)
- The RedHawk GSR configuration file (`*.gsr`)

The `analyze_rail` command generates the following file to invoke the RedHawk-SC tool for running rail analysis:

- The RedHawk-SC python script (`analyze_rail.py`)

The RedHawk or RedHawk-SC tool generates the following files in the `in-design.redhawk/design_name.result` (or `in-design.redhawk_sc/design_name.result` in RedHawk-SC flow) directory when rail analysis is complete:

- The RedHawk analysis log (`analyze_rail.log.*`)
- Timing window file (`*.sta`)
- Signal SPEF files (`*.spef`)
- DEF/LEF files (`*.def`, `*.lef`) for the full-chip analysis

When voltage drop analysis is complete, the tool saves the analysis results (`*.result`) in the `design_name` directory under the working directory. The rail analysis results are used to display maps and query attributes to determine the problem areas with large voltage drops.

- To reload the rail results for displaying maps in the GUI, use the `open_rail_result` command.

For example,

```
fc_shell> open_rail_result
REDHAWK_RESULT
```

For more information about displaying maps, see [Displaying Maps in the GUI](#).

- To write out the rail analysis results, use the `report_rail_result` command. You must specify the result type with the `-type` option. For example, to write out the voltage drop values for the current block, set the `-type` option with the `voltage_drop_or_rise` argument.

For more information, see [Writing Analysis and Checking Reports](#).

- To query the results stored in the current rail results cache, use the `get_attribute` commands.

For more information, see [Querying Attributes](#).

Note:

RedHawk Fusion does not support results that are generated by the RedHawk tool outside of the Fusion Compiler environment. The RedHawk `dump_icc2_result` command works only within the Fusion Compiler session.

Performing PG Electromigration Analysis

Note:

PG electromigration analysis is supported in both RedHawk Fusion and RedHawk-SC Fusion analysis flows.

To analyze the current density on PG nets and identify the segments with potential electromigration issues, run the `analyze_rail -electromigration` command. The analysis results are saved in the working directory.

To perform electromigration analysis,

1. Set the `rail.tech_file` application option to specify the Apache technology file (`*.tech`) that defines the layer-by-layer current density limits.

For example,

```
fc_shell> set_app_options -name rail.tech_file \
    -value design_data/tech/test.tech
```

2. Run the `analyze_rail` command with the following options:
 - Use the `-voltage_drop` and `-electromigration` options to enable electromigration analysis.
 - Use the `-nets` option to specify the power and ground supply nets to analyze. The tool considers all the switched or internal power nets of the specified power nets in the analysis. You do not need to explicitly specify the internal power nets.
 - (Optional) By default, the `analyze_rail` command automatically generates a GSR file for running RedHawk rail analysis. To specify an external GSR file to append to the GSR file generated in the current run, use the `-extra_gsr_option_file` option.
 - (Optional) To specify an existing RedHawk script file that was generated in an earlier `analyze_rail` run, use the `-redhawk_script_file` option.

When using an existing script file for rail analysis, the tool honors the settings in this script file and ignores all other currently enabled options. For example, if you run the `analyze_rail -voltage_drop static -redhawk_script_file myscript.tcl` command, the tool ignores the `-voltage_drop` option.

When the `-redhawk_script_file` option is not specified, you must specify the nets to analyze by using the `-nets` option.
3. Check the analysis results, as described in [Viewing PG Electromigration Analysis Results](#).

Viewing PG Electromigration Analysis Results

When PG electromigration analysis is complete, you can check for current violations by displaying a PG electromigration map or checking the generated errors in the error browser.

- [Displaying the PG Electromigration Map](#)
- [Checking PG Electromigration Violations](#)

Displaying the PG Electromigration Map

A PG electromigration map is a visual display of the color-coded electromigration values overlaid on the physical supply nets.

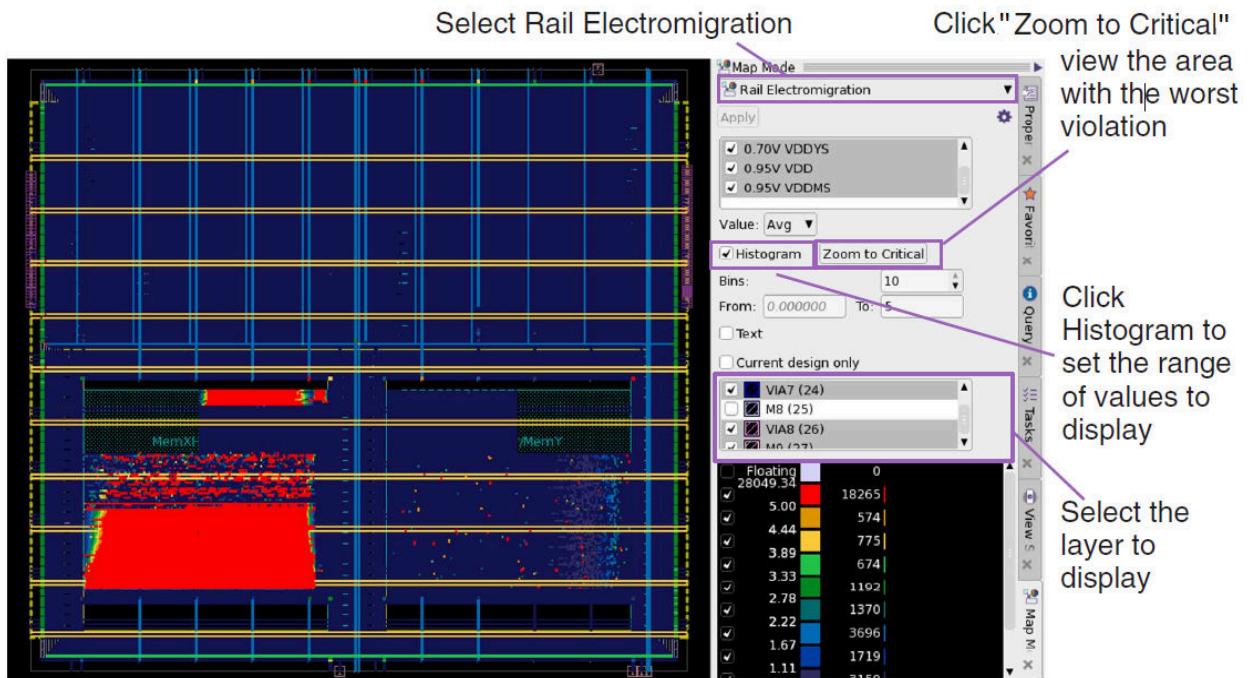
For static analysis, the map displays average electromigration values. For dynamic analysis, the map displays average electromigration values, peak electromigration values, or root mean square electromigration values.

[Figure 188](#) shows an example of the electromigration map in which problem areas are highlighted in different colors. Use the options to investigate the problem areas by

selecting layers or nets to display. For example, to examine the current value of one specific net, deselect all the nets and then select the net to display from the list. To analyze only the shapes on a layer, select the layer from the list.

For a detailed procedure about how to display an electromigration map, see [Displaying Maps in the GUI](#).

Figure 188 Displaying a PG Electromigration Map



Checking PG Electromigration Violations

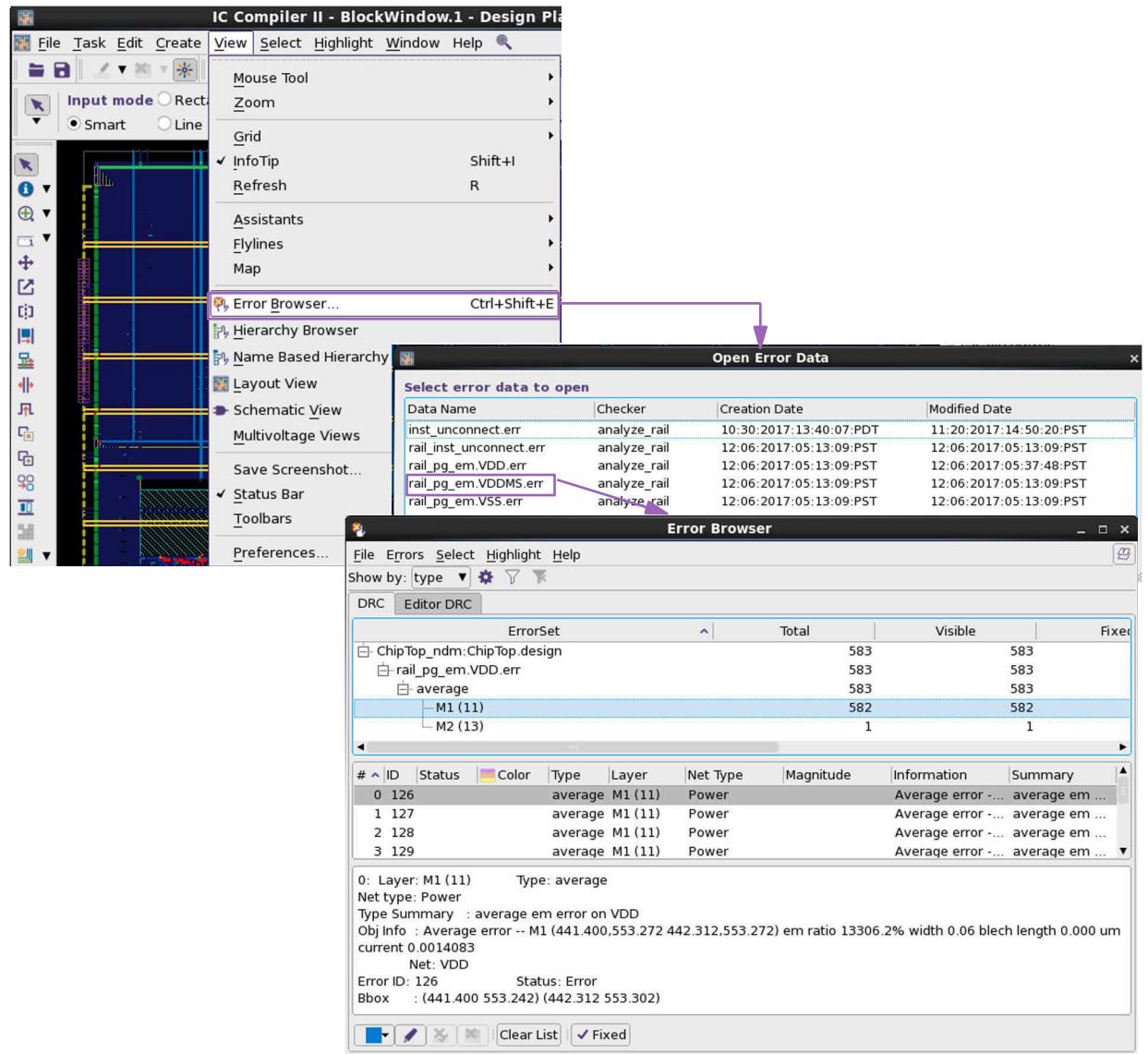
When PG electromigration analysis is complete, you can check the generated error files in the error browser or write the generated errors to an output file.

The tool does not write the generated errors to the working directory, meaning the error data is deleted if you exit the tool without saving the block. Ensure that you save the block to keep the generated error data if you want to examine the errors in another session.

You can examine the generated error files in the error browser or write the generated errors to an output ASCII file.

- To examine the generated errors in the error browser,
 1. Choose View > Error Browser from the GUI.
 2. In the Open Error Data dialog box, select the errors to examine and click Open Selected.
 3. Check the details of the errors in the Error Browser (see [Figure 189](#)).

Figure 189 Opening Generated Errors in the Error Browser



- To write errors to an ASCII file, use the following script:

```

set fileName "errors.txt"
set fd [open $fileName w]
set dm [open_drc_error_data rail_pg_em.VDD.err]
set all_errs [get_drc_errors -error_data $dm *]
foreach_in_collection err $all_errs {
    set info [get_attribute $err error_info];
    puts $fd $info }
close $fd

```

Here is an example of the output error file:

```
M1 (414.000,546.584 414.344,546.584) em ratio 23585.9% width 0.06
blech
length 0.000 um current 0.0024963
M1 (414.344,546.584 416.168,546.584) em ratio 23401.3% width 0.06
blech
length 0.000 um current 0.0024768
M1 (416.168,546.584 416.776,546.584) em ratio 23241.7% width 0.06
blech
length 0.000 um current 0.0024599
M1 (416.776,546.584 417.992,546.584) em ratio 23082.2% width 0.06
blech
length 0.000 um current 0.0024430
M1 (417.992,546.584 419.208,546.584) em ratio 22897.7% width 0.06
blech
length 0.000 um current 0.0024235
M1 (419.208,546.584 419.512,546.584) em ratio 22731.1% width 0.06
blech
length 0.000 um current 0.0024059
M1 (419.512,546.584 421.184,546.584) em ratio 22564.4% width 0.06
blech
length 0.000 um current 0.0023882
M1 (421.184,546.584 421.488,546.584) em ratio 22349% width 0.06 blech
length 0.000 um current 0.0023654
M1 (421.488,546.584 423.160,546.584) em ratio 22133.5% width 0.06
blech
length 0.000 um current 0.0023426
M1 (423.160,546.584 423.616,546.584) em ratio 21967.2% width 0.06
blech
length 0.000 um current 0.0023250
```

Performing Minimum Path Resistance Analysis

Note:

Minimum path resistance analysis is supported in both RedHawk Fusion and RedHawk-SC Fusion analysis flows.

Perform minimum path resistance analysis to quickly detect power and ground network weaknesses in the design. The minimum path resistance value of a node is the resistance value on the smallest resistive path to the ideal voltage source locations (taps). The ideal voltage sources can be power or ground pins, user-defined taps, or packages. Because this value represents only the resistance on a single path, it is not necessarily same as the effective resistance value from the node to the supply or ground; instead, it represents an upper bound on the effective resistance of a node to the supply or ground.

To perform minimum path resistance analysis, run the `analyze_rail -min_path_resistance` command. You must specify the power and ground nets to analyze by using the `-nets` option.

Note:

In RedHawk-SC rail analysis flow, you must specify both the `-voltage_drop` and `-min_path_resistance` options for minimum path resistance calculation.

You can perform minimum path resistance analysis before or during voltage drop analysis. When the analysis is completed, the tool writes the minimum path resistance values and the voltage drop results into the same rail result.

The following example performs voltage drop analysis and minimum path resistance calculation at the same time.

```
fc_shell> analyze_rail -nets {VDD VSS} \
    -min_path_resistance -voltage_drop static
```

The following example performs only minimum path resistance analysis.

```
fc_shell> analyze_rail -nets {VDD VSS} -min_path_resistance
```

Viewing Minimum Path Resistance Analysis Results

When the analysis is complete, run the `report_rail_result -type minimum_path_resistance` command to display a minimum path resistance map, write the results to an output file, or query the attributes on the cell instances or pins.

The following example writes the calculated minimum path resistances for the VDD net to an output file called `minres.rpt`.

```
fc_shell> open_rail_result
fc_shell> report_rail_result -type minimum_path_resistance \
    -supply_nets { vdd } minres.rpt
fc_shell> sh cat minres.rpt
FEED0_14/vbp 518.668
INV617/vdd 158.268
ARCR0_1/vbp 145.582
FEED0_5/vdd 156.228
INV117/vbp 518.669
```

The output `minres.rpt` report includes the entire resistance information.

```
fc_shell> sh cat minres.rpt
FEED0_14/vbp 518.668
INV617/vdd 158.268
ARCR0_1/vbp 145.582
FEED0_5/vdd 156.228
INV117/vbp 518.669
```

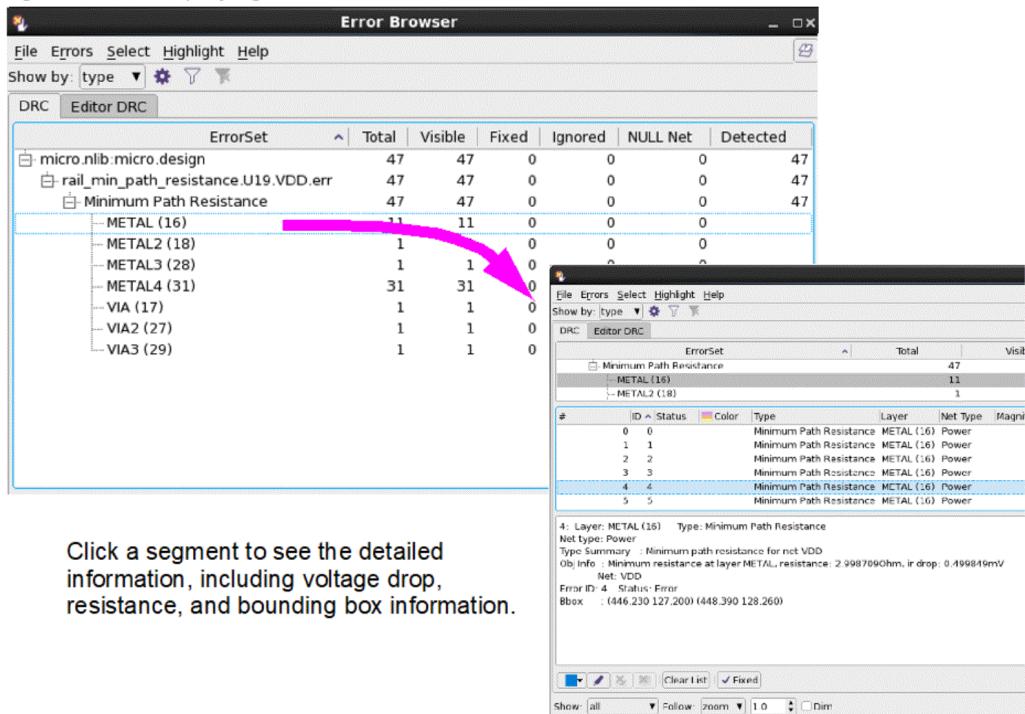
To write out a detailed minimum resistance path report for a cell instance, run the `report_rail_minimum_path` command and specify the net and the cell to report. To see the segment-by-segment resistance of that minimum resistance path shown in the GUI, use the `-error_cell` option of the `report_rail_minimum_path` command to save the geometries on the path of the minimum path resistance to an error cell. You can then open the error cell in an error browser and check the minimum resistance path from the pin of the cell to the nearest tap and trace each go-through geometries.

In the following example, all geometries on the minimum resistance path are traced and written to the error cell named `test.err`.

```
fc_shell> open_rail_result
fc_shell> report_rail_minimum_path -cell U19 -net VDD -error_cell
```

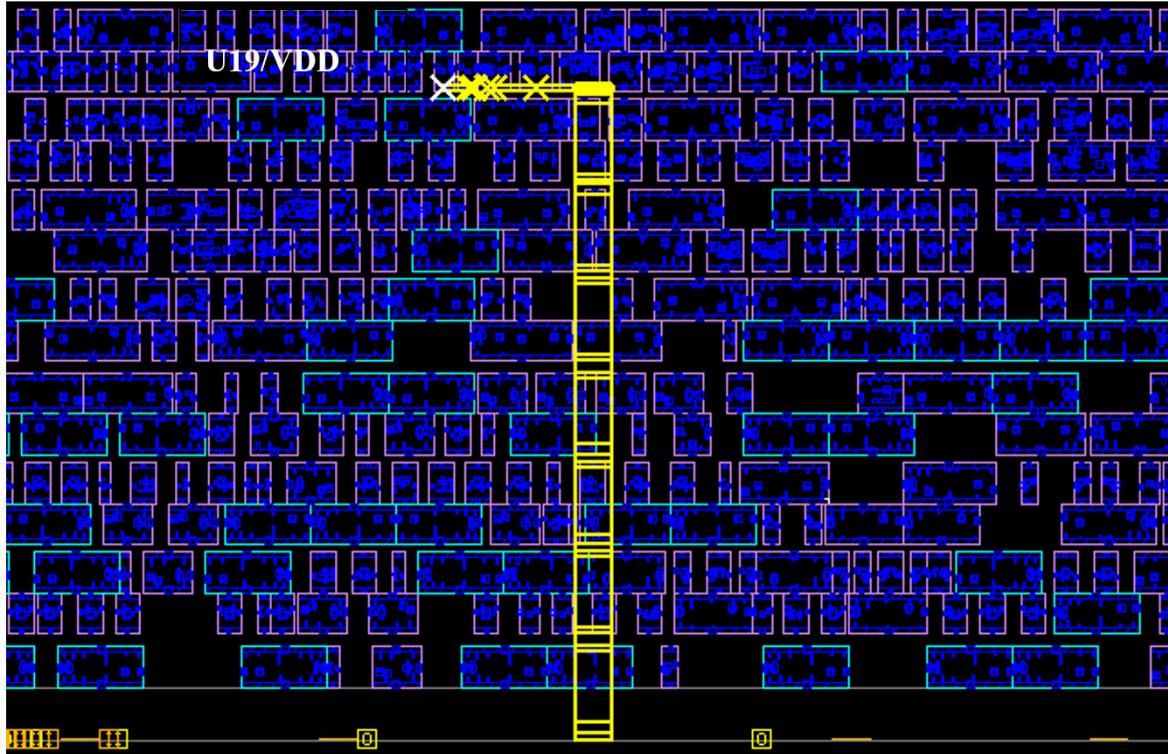
Shown in [Figure 190](#), each geometry is classified by layer in the error browser. Click a geometry to check for detailed information, such as IR drop, resistance, bounding box, and so on. In the GUI, the traced path is highlighted in yellow in the design layout (see [Figure 191](#)).

Figure 190 Displaying Error Cells in the Error Browser



Click a segment to see the detailed information, including voltage drop, resistance, and bounding box information.

Figure 191 Traced Segment is Highlighted in the Design Layout



See Also

- [Displaying Maps in the GUI](#)
- [Writing Analysis and Checking Reports](#)
- [Querying Attributes](#)

Tracing Minimum Path Resistance Using the Mouse Tool

Use the minimum path resistance (MPR) mouse tool to interactively trace the path with the least resistance from the specified power or ground net to the boundary nodes in the design.

To trace the calculated minimum path resistance with the MPR interactive tracing tool,

1. In the GUI, choose Task > Rail and then select Analysis Tools from the task list to open the Task Assistant window (see [Figure 192](#)).
2. In the Task Assistant - Rail window that opens, click MPR Mouse Tool.
3. Specify the nets to analyze. Click OK.

The MPR interactive mouse tool, which is shown in [Figure 193](#), provides the following features:

- Zoom in and out to review the traces
- Provide two operation modes: the Add mode to trace multiple paths in the map, and the Replace mode to trace one path at a time
- Use the standard query tool to show tracing information
- Select a region to trace multiple cells at a time
- Document the interactive actions by scripts

Figure 192 Invoking the MPR Mouse Tool

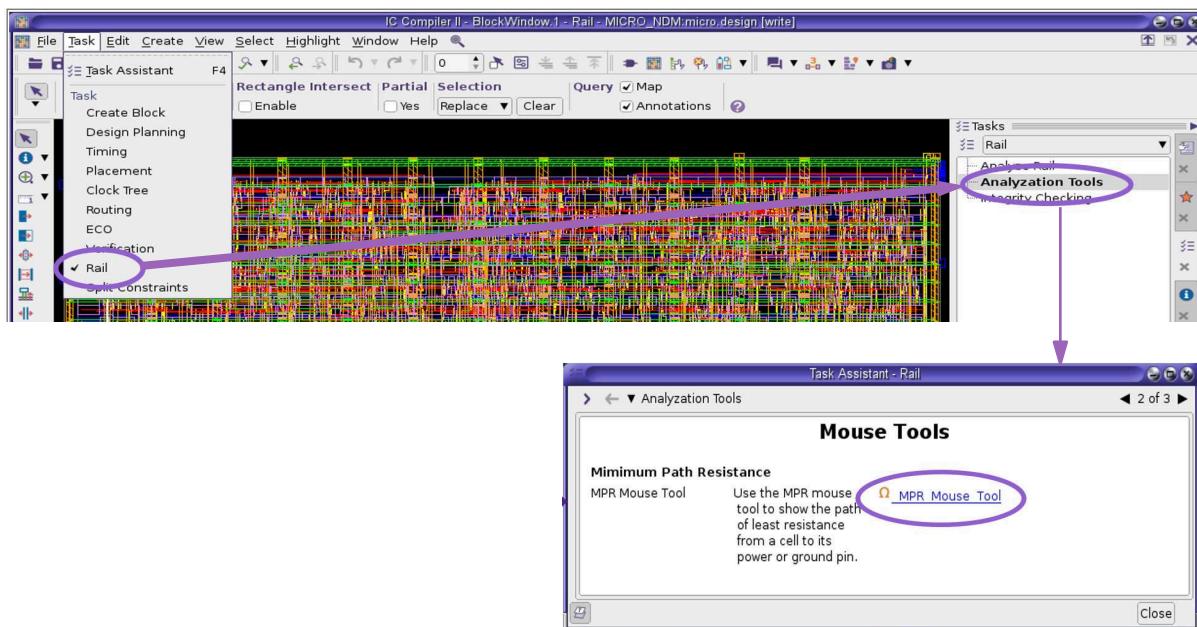
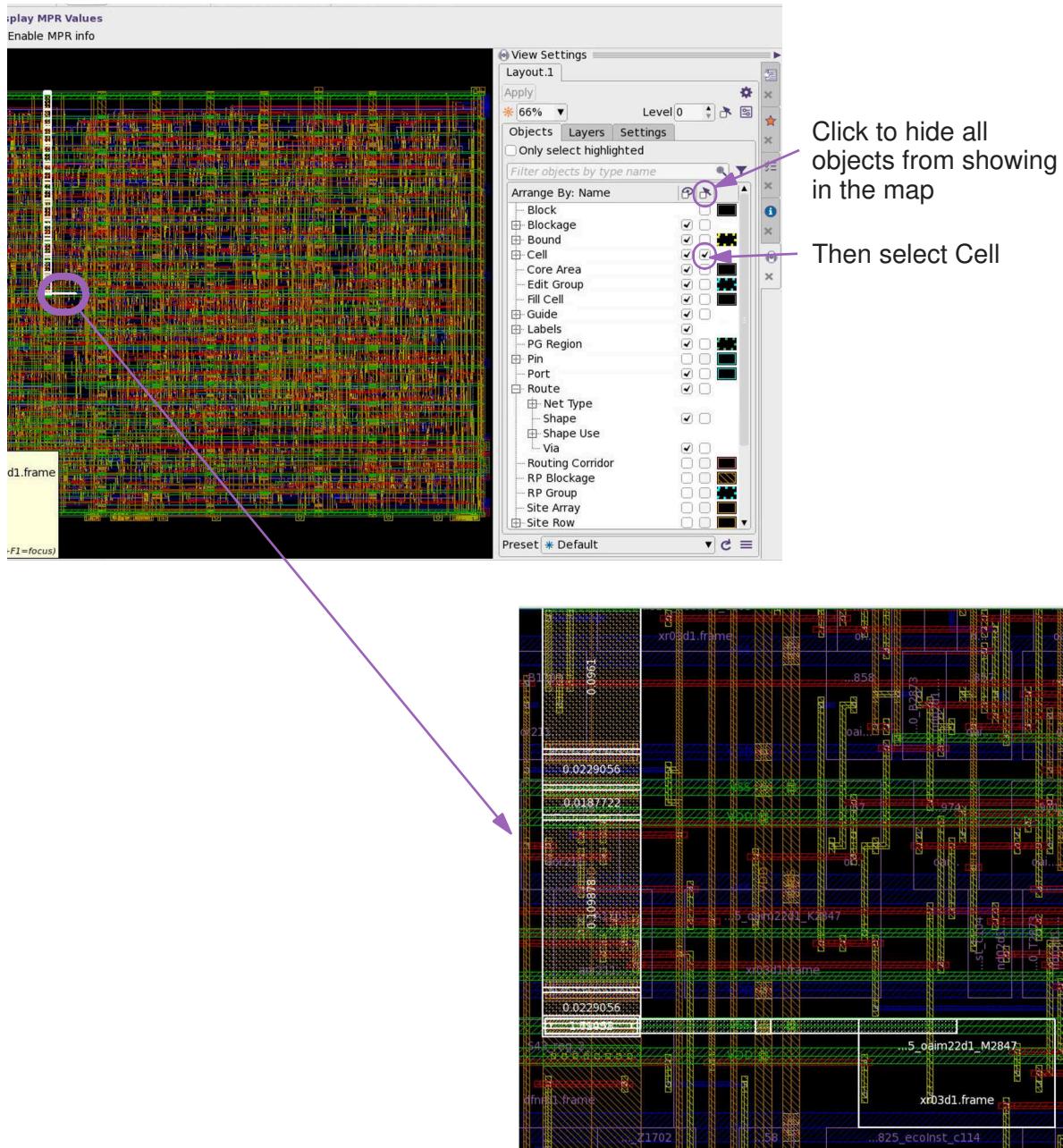


Figure 193 Tracing the Path With the Least Resistance



Performing Effective Resistance Analysis

Note:

Effective resistance analysis is supported in both RedHawk Fusion and RedHawk-SC Fusion analysis flows.

To calculate effective resistance for the specified power and ground nets, use the `analyze_rail -effective_resistance` command.

To specify a list of cell instances for effective resistance calculation, describe the target instances in a text file. Then specify this text file with the `rail.effective_resistance_instance_file` application option.

For example,

```
fc_shell> sh cat cell_list.txt
u_GS_PRO_65
u_GS_PRO_64
ChipTop_U_compressor_mode/U3

fc_shell> set_app_options \
    -name rail.effective_resistance_instance_file \
    -value cell_list.txt
fc_shell> analyze_rail -nets {VDD VSS} -effective_resistance
```

You can perform effective resistance analysis before or during voltage drop analysis. When the analysis is completed, the tool writes the effective resistance values into the same rail result with the voltage drop results.

Note:

In RedHawk-SC rail analysis flow, you must specify both the `-effective_resistance` and `-voltage_drop` options for effective resistance calculation.

When the calculation is complete, the tool saves the results in the working directory. To write the effective resistance values to a text file, use the `report_rail_result -type effective_resistance` command. To query the resistance value, use the `get_attribute` command.

For example,

```
fc_shell> get_attribute [get_pins u_GS_PRO_65/VDD] \
    effective_resistance
```

See Also

- [Specifying RedHawk and RedHawk-SC Working Directories](#)
- [Writing Analysis and Checking Reports](#)

Performing Distributed RedHawk Fusion Rail Analysis

Note:

The distributed rail analysis feature is supported only in RedHawk Fusion analysis flow.

By default, the `analyze_rail` command uses a single process to perform RedHawk rail analysis. To reduce memory usage and total runtime, use the distributed processing method to run RedHawk Fusion on a machine other than the one on which the Fusion Compiler tool is run.

You can submit multiple RedHawk jobs to the target farm machines and run the jobs in parallel in the same Fusion Compiler session. This allows you to run multiple RedHawk analyses with only one Fusion Compiler license. Each of the RedHawk jobs requires one `SNPS_INDESIGN_RH_RAIL` license key.

To examine the generated analysis result, first run the `open_rail_result` command to load the result that is saved in the directory you specify. You can then investigate the problematic areas in the map and check for violations in a text file or in an error view.

You can open one result context at a time. To load the rail context generated in another analysis run, first close the current rail result using the `close_rail_result` command, and then open the desired rail result (see [Displaying Maps in the GUI](#)).

License Requirement

By default, the number of required licenses is the number of script files or configurations specified with the `-multiple_script_files` option during distributed processing mode. If the number of available licenses is less than the number of script files at the time when the `analyze_rail` command is run, the command stops with an error message.

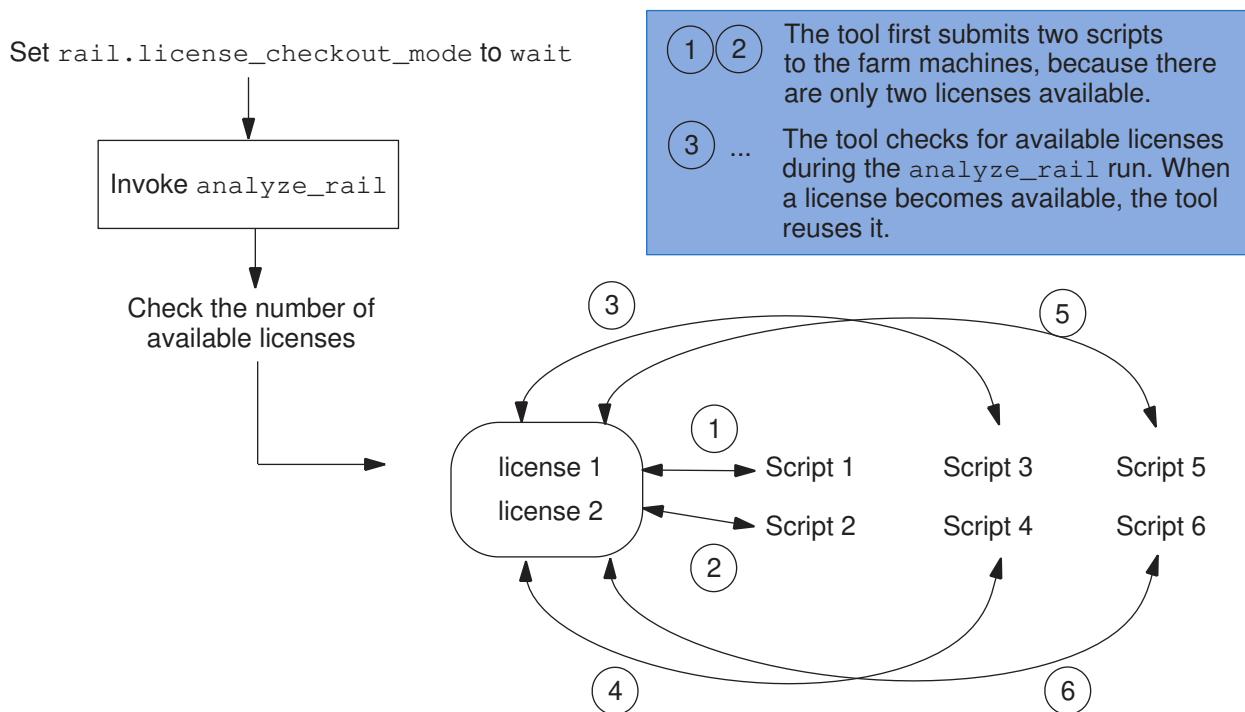
To check for available `SNPS_INDESIGN_RH_RAIL` licenses continuously during the `analyze_rail` run, set the `rail.license_checkout_mode` application option to `wait`. If the tool detects that a license becomes available, the tool reuses or checks out the license and submits a script to a farm machine immediately. By default, the `rail.license_checkout_mode` application option is set to `nowait`.

When the analysis is finished, run the `remove_licenses` command to release the `SNPS_INDESIGN_RH_RAIL` licenses.

```
fc_shell> remove_licenses SNPS_INDESIGN_RH_RAIL
```

In Figure 194, the `rail.license_checkout_mode` application option is set to `wait`. There are six scripts to submit to the farm machines with only two available `SNPS_INDESIGN_RH_RAIL` licenses. In this case, the tool first checks out two licenses and waits for other licenses to become available. When a script finishes the job, the tool reuses its license immediately to submit other scripts.

Figure 194 Waiting for Available Licenses During Distributed RedHawk Fusion Run



Steps to Enable Distributed Processing

To enable distributed processing for RedHawk Fusion,

1. Define the distributed processing configuration by using the `set_host_options` command. You can specify one or more of the following options:

| Option | Description |
|-------------------------------|--|
| <code>-submit_protocol</code> | Specifies the protocol to submit jobs, such as LSF, GRD, or SGE. Note: Before job submission, run the <code>rsh</code> command to check if the target farm machine supports the <code>rsh</code> protocol to accept distributed processing from on a remote machine. |
| <code>-target</code> | Specifies to submit jobs for RedHawk only. |
| <code>-timeout</code> | Specifies the job submission timeout value. |

| Option | Description |
|-----------------|---|
| -submit_command | Specifies the full path name of the LSF, GRD, SGE, or custom job submission program along with other options as needed, such as qsub or bsub. |
| -max_cores | Controls the number of cores each distributed worker job can use. The default is 1. |

2. Run the `analyze_rail` command with the following options:

| Option | Description |
|--|--|
| -submit_to_other_machines | Enables RedHawk job submission to the target farm machines. |
| -multiple_script_files\ directory_name script_name | Specifies the RedHawk scripts and the directories for saving analysis results. |

Example 49

```
%> rsh myMachine ls
test1
CSHRC
DESKTOP
> set_host_options "myMachine" -target RedHawk \
    -max_cores 2 -timeout 100
> analyze_rail -submit_to_other_machines \
    -voltage_drop static -nets {VDD VDDG VDDX VDDY VSS}
```

The above example first checks if the machine myMachine supports the rsh protocol, and then enables job submission to the myMachine machine for RedHawk Fusion only. Timeout value is 100 and the number of cores to use is 2.

Example 50

```
> set_host_options -submit_protocol sge \
    -submit_command { qsub -P normal }
> analyze_rail -submit_to_other_machines -voltage_drop \
    static -nets {VDD VSS}
```

The above example enables job submission to a machine on the SGE farm machine.

Example 51

```
> set_host_options -submit_protocol lsf \
    -submit_command {bsub}
> analyze_rail -submit_to_other_machines \
    -multiple_script_files {{result1 RH1.tcl} {result2 RH2.tcl}}
> open_rail_result result1
```

```
> close_rail_result
> open_rail_result result2
```

The above example submits two RedHawk jobs with two RedHawk scripts to the LSF farm machines and specifies the directories for saving the analysis results as result1 and result2.

Example 52

```
> set_app_options -name rail.license_checkout_mode \
    -value wait
> set_host_options -submit_protocol sge \
    -submit_command {qsub}
> analyze_rail -submit_to_other_machines \
    -multiple_script_files {{ result1 RH1.tcl} {result2 RH2.tcl} \
    {result3 RH3.tcl}}
```

The above example submits three RedHawk scripts to the SGE farm machines and waits for an available license during the `analyze_rail` run.

Working With Macro Models

Memory and macro cells are used in designs with advanced process technologies to reduce memory usage and improve the rail analysis performance. Macro models are supported in both RedHawk Fusion and RedHawk-SC Fusion analysis flows.

To learn about working with macro models, see the following topics:

- [Generating Macro Models](#)
- [Creating Block Contexts](#)

Generating Macro Models

To consider current and power distribution inside the memory blocks when analyzing hierarchical designs, use the `exec_gds2rh` command to generate RedHawk macro models for memory blocks. The command invokes the RedHawk `gds2rh` utility under the hood and extracts power grids and current sources from the memory blocks with varying levels of abstraction—from fully detailed models for accurate block-level analysis to abstract models for full-chip dynamic analysis.

Note:

To create RedHawk macro models using the RedHawk `gds2rh` utility, you must specify the path to the RedHawk executable with the `rail.redhawk_path` application option before running the `exec_gds2rh` command.

The generated macro models are saved in the current working directory or the one you specify in the configuration file. To run rail analysis with the generated macro models, set the `rail.macro_models -value {cell1 path1}` application option before running the `analyze_rail` command.

For more information about how to run the RedHawk `gds2rh` utility, see the *RedHawk User Manual*.

Required Configuration File

To generate a RedHawk macro model using the `gds2rh` utility, you must provide a configuration file that includes the following information:

- GDS II files
- Top design name
- Layer mapping file
- Power and ground net names

Specify one configuration file per memory. Make sure the configuration file contains the correct syntax for macro model generation. Otherwise, the tool issues an error message and terminates the model generation process.

The following is an example of the configuration file:

```
LEF_FILE {
    ./lef/saed32sram.lef
}
GDS_FILE ./gds/SRAMLP2RW128x16.gds
GDS_MAP_FILE ./redhawk.gdslayermap
OUTPUT_DIRECTORY SRAMLP2RW128x16_output

USE_LEF_PINS_FOR_TRACING    1

VDD_NETS {
    VDD
}
GND_NETS {
    VSS
}
TOP_CELL SRAMLP2RW128x16
```

Examples

The following example generates RedHawk macro models for the SRAMLP2RW128x16, SRAMLP2RW32x4, SRAMLP2RW64x32 and SRAMLP2RW64x8 memories:

```
## Specify the path to the RedHawk executable ##
set_app_options -name rail.redhawk_path -value $env(REDHAWK_IMAGE) /bin
```

```
## Specify one configuration file per memory ##
exec_gds2rh -config_file SRAMLP2RW128x16.config
exec_gds2rh -config_file SRAMLP2RW32x4.config
exec_gds2rh -config_file SRAMLP2RW64x32.config
exec_gds2rh -config_file SRAMLP2RW64x8.config
```

The following example runs top-level rail analysis with the macro models that are generated in the previous example:

```
## Run rail analysis at top level ##

open_lib
open_block top
link_block

## Specify rail analysis setting ##
create_taps
set_app_options -name rail.tech_file -value test.tech
set_app_options -name rail.lib_files -value test.lib
...
## Set macro model file ##
set_app_options -name rail.macro_models \
  -value { SRAMLP2RW128x16 ./data_SRAMLP2RW128x16
            SRAMLP2RW32x4 ./data_SRAMLP2RW32x4
            SRAMLP2RW64x32 ./data_SRAMLP2RW64x32
            SRAMLP2RW64x8 ./data_SRAMLP2RW64x8
  }

## Run rail analysis ##
analyze_rail -voltage_drop static -all_nets

## Check rail results ##
open_rail_result
gui_start
```

Creating Block Contexts

Macro models are used to reduce total turnaround time for designs with advanced process technology. When violations are reported in the macros during full-chip signoff voltage drop analysis, it might be time-consuming to fix the violations at the block level and then rerun full-chip analysis to verify the fix.

To reduce the turnaround time, you can create a context model for the macro block. A context model depicts the relationship between the block and the full-chip design, including both physical and electrical information, and enables you to verify the fixes by running block-level analysis rather than full-chip analysis.

To create a context model, use the `create_context_for_sub_block` command.

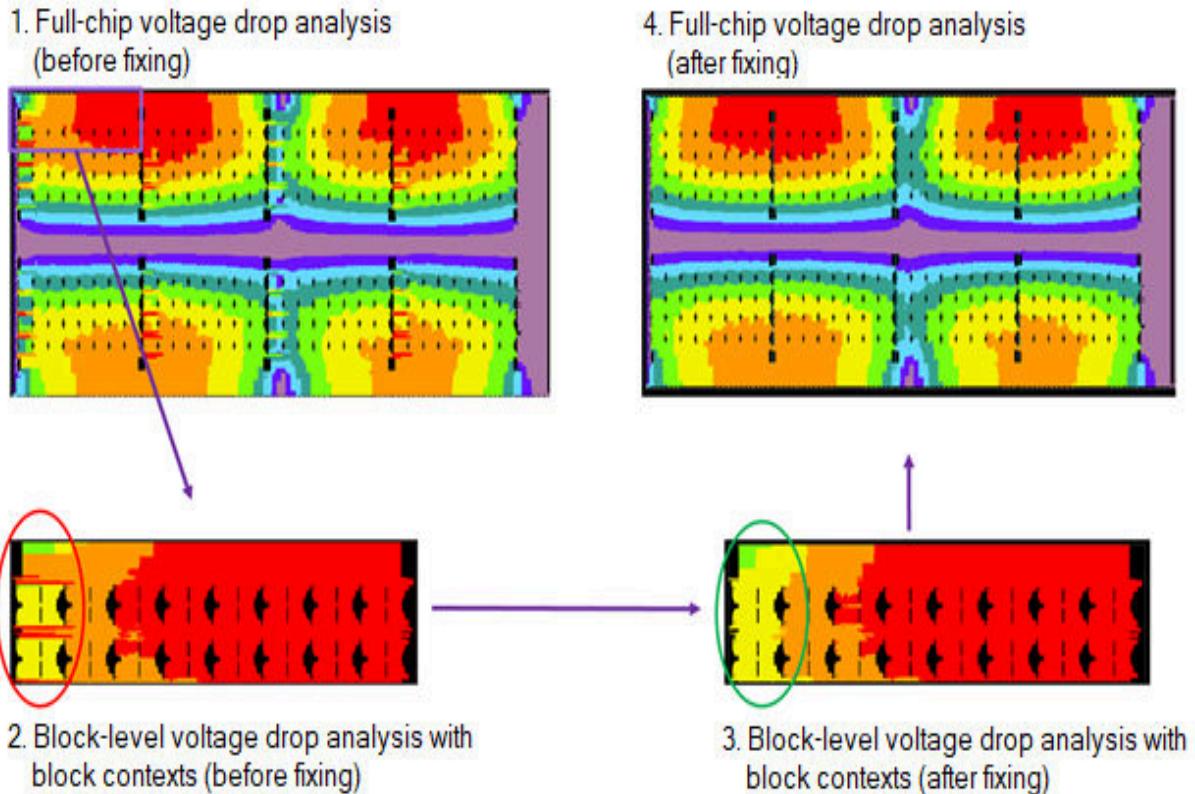
To run voltage drop analysis with the generated block contexts, set the `rail.pad_files` and `rail.block_context_model_file` application options before running the `analyze_rail` command.

Note:

Creating full-chip contexts for blocks is supported only in RedHawk Fusion.

The following figure shows that a fix in the block can effectively improve the performance of the top design.

Figure 195 Fixing Violations in Block Contexts



Content of Block Contexts

The `create_context_for_sub_block` command models the full-chip context of a block, including physical models and electrical models.

- Physical model, which describes the locations of the connections between the block and full-chip design. These tap locations are derived by tracing the RedHawk parasitics data. The physical model is generated when the `create_context_for_sub_block` command process is complete, in the `.ploc` format.

- Electrical model, which is an effective resistance- and capacitance-based model. The tool uses the native resistance engine to calculate the effective resistance, which is used to calculate equivalent current during peak dynamic voltage drop analysis. By default, the tool models the full-chip context for the specified block. The capacitance data is derived by the RedHawk extraction engine.

To create an electrical model for the specified block, specify the `-electric_model_file` option with the `create_context_for_sub_block` command. The electrical model is written in the `.context` format.

Examples

The following example creates a block context for the top/cell_inst_1 block. The generated physical and electrical models are `block.ploc` and `block.context`, respectively.

```
##Create block context in top block ##
open_block top
create_context_for_sub_block \
    -block_instance top/cell_inst_1 block.ploc \
    -nets {VDD_TOP VSS_TOP} \
    -electric_model_file block.context
close_block

##Run IR analysis with block context information in block-level design ##
open_block block
set_app_options -name rail.pad_files -value block.ploc
set_app_options -name rail.block_context_model_file \
    -value block.context
analyze_rail -nets {VDD_BLOCK VSS_BLOCK} \
    -voltage_drop static -extra_gsr_option_file extra.gsr
...
```

The following example creates only a physical model for the top/cell_inst_1 block.

```
##Create block context in top block ##
open_block top
create_context_for_sub_block \
    -block_instance top/cell_inst_1 block.ploc \
    -nets {VDD_TOP VSS_TOP}
close_block

##Run IR analysis with block context information in block-level design ##
open_block block
set_app_options -name rail.pad_files -value block.ploc
analyze_rail -nets {VDD_BLOCK VSS_BLOCK} \
    -voltage_drop static -extra_gsr_option_file extra.gsr
...
```

Performing Signoff Analysis

RedHawk Fusion or RedHawk-SC Fusion does not support RedHawk signoff analysis capabilities, such as signal electromigration or inrush current analysis. If you have the RedHawk signoff licenses, you can enable the following RedHawk signoff analysis features in the Fusion Compiler environment by using the `rail.allow_redhawk_license_checkout` application option. Signoff analysis features not included in the following list can only be run with the RedHawk standalone tool.

- Dynamic analysis with custom macro models
- Dynamic analysis with package models
- Analysis with RTL-level VCD files

To perform RedHawk signoff analysis,

1. Modify your GSR file or RedHawk run script to include the related configuration settings.
2. Set the following application option to enable the RedHawk signoff features:

```
fc_shell> set_app_options \
    -name rail.allow_redhawk_license_checkout -value true
```

Setting the `rail.allow_redhawk_license_checkout` application option to `true` allows the RedHawk tool to retrieve the related RedHawk licenses for performing signoff features inside the Fusion Compiler tool. The default is `off`.

3. Run the `analyze_rail` command with the `-extra_gsr_option_file` or `-redhawk_script_file` option to perform the analysis defined in the GSR or RedHawk script file. Specify other settings as necessary.

Note:

You cannot display the RedHawk signoff analysis results in the Fusion Compiler GUI.

4. Proceed to other steps in the analysis flow.

See Also

- [An Overview for RedHawk Fusion and RedHawk-SC Fusion](#)

Writing Analysis and Checking Reports

When the checking or analysis is complete, run the `report_rail_result` command to write the analysis or checking results to a text file. The power unit in the report file is watts, the current unit is amperes, and the voltage unit is volts.

To display block-level rail results from top-level RAIL_DATABASE for debugging purposes, use the `-top_design` and `-block_instance` options. For more information, see [Displaying Block-Level Rail Results](#).

To write rail results for the cells or geometries in the hotspot area only, use the `get_instance_result` and `get_geometry_result` commands. For more information, see [Generating Instance-Based Analysis Reports](#) and [Generating Geometry-Based Analysis Reports](#).

To specify the data or error types to report, use the `-type` option. [Table 67](#) lists the supported data and error types.

To limit the number of the elements to report, use the `-limit` option. Set the value to zero to write all elements to the output file in descending order.

To filter the data to write to the report file, use the `-threshold` option. The tool writes the data greater than the specified threshold to the output file. The option is valid only for the following data types:

- `pg_pin_power`
- `voltage_drop_or_rise`
- `effective_voltage_drop`

To restrict the report to the specified supply nets, use the `-supply_nets` option. This option is valid only for the following data types:

- `effective_voltage_drop`
- `instance_minimum_path_resistance`
- `minimum_path_resistance`
- `pg_pin_power`
- `voltage_drop_or_rise`

Table 67 Supported Data and Error Types for report_rail_result

| Type | Description |
|-------------------------------------|---|
| <code>effective_voltage_drop</code> | The effective voltage values, in volts. |

Table 67 Supported Data and Error Types for report_rail_result (Continued)

| Type | Description |
|----------------------------------|---|
| effective_resistance | The effective resistance values. |
| instance_minimum_path_resistance | The minimum path resistance value on each instance. |
| instance_power | The power value on each instance, in watts. |
| minimum_path_resistance | The minimum path resistance value on each power or ground pin. |
| missing_vias | Overlaps of supply net routing on different layers that do not have vias connected. |
| pg_pin_power | The power value on each power or ground pin, in watts. |
| unconnected_instances | Instances that are not connected to any ideal voltage drop sources. |
| voltage_drop_or_rise | The dynamic voltage drop or rise violations, in volts. |

In the output report file, the tool lists the data in different formats. For example:

- For the `effective_voltage_drop` type,
 - When reporting the static analysis results, the format is


```
cell_instance_pg_pin_name mapped_pg_pin_name supply_net_name
effective_voltage_drop
```
 - When reporting the dynamic analysis results, the format is


```
cell_instance_pg_pin_name mapped_pg_pin_name supply_net_name
average_effective_voltage_drop_in_tw
max_effective_voltage_drop_in_tw
min_effective_voltage_drop_in_tw
max_effective_voltage_drop
```
- For the `instance_minimum_path_resistance` type, the result is sorted by the `Total_R` value in the following format:


```
Supply_net Total_R(ohm) R_to_power(ohm) R_to_ground(ohm)
Location Pin_name Instance_name
```
- For the `minimum_path_resistance` type, the format is


```
full_path_cell_instance_name/pg_pin_name resistance
```
- For the `missing_vias` type, the format is

```
net_name via_location_x_y top_metal_layer bottom_via_layer
delta_voltage
```

- For the pg_pin_power type, the format is

```
full_path_cell_instance_name pg_pin_name power
```

- For the voltage_drop_or_rise type, the format is

```
full_path_cell_instance_name/pg_pin_name voltage
```

- For the unconnected_instances error type, the format depends on the error condition.

- When either or both of power and ground nets are physically disconnected from ideal voltage sources, the format is

```
unconnected_type {net_names} instance_name instance bbox
```

- When the power or ground nets are either floating or logically floating but physically connected to ideal voltage sources, the format is

```
unconnected_type {net_names} instance_name:pin_name instance bbox
```

The following example writes a file containing the top five PG pin power values to an output file called power.rpt.

```
fc_shell> open_rail_result
fc_shell> report_rail_result -type pg_pin_power -limit 5 \
    -supply_nets { VSS } power.rpt
fc_shell> sh cat power.rpt
FI2/vss 4.81004e-06
FI6/vss 4.80959e-06
FI3/vss 4.79702e-06
FI4/vss 4.79684e-06
FI1/vss 4.79594e-06
...
```

The following example writes the calculated effective voltage drop values for the VDD and VSS nets to an output file called inst_effvd.rpt.

```
fc_shell> open_rail_result
fc_shell> report_rail_result -type effective_voltage_drop \
    -supply_nets { VDD VSS } inst_effvd.rpt
fc_shell> sh cat inst_effvd.rpt

#cell_instance_pg_pin_name mapped_pg_pin_name supply_net_name
#average_effective_voltage_drop_in_tw
#max_effective_voltage_drop_in_tw min_effective_voltage_drop_in_tw
#max_effective_voltage_drop

S32_reg_14 /VDD VSS VDD -7.750034332e-03 -1.017999649e-02
-5.850076675e-03 -1.029992104e-02
```

```
S32_reg_15 /VDD VSS VDD -7.709980011e-03 -1.013994217e-02
-5.810022354e-03 -1.026010513e-02

S46_reg_8 /VDD VSS VDD -7.179975510e-03 -9.490013123e-03 -5.330085754e-03
-1.021003723e-02
...
```

The following example writes the minimum path resistance values of cell instances to an output file called `inst_minres.rpt`.

```
fc_shell> open_rail_result
fc_shell> report_rail_result -type instance_minimum_path_resistance \
    -supply_nets { VDD VSS } inst_minres.rpt
fc_shell> sh cat inst_minres.rpt
```

```
Rmax(ohm) = 162.738
Rmin(ohm) = 4.418
=====
Supply_net Total_R(ohm) R_to_power(ohm) R_to_ground(ohm)
Location Pin_name Instance_name
=====
VDD 162.738 152.686 10.052 586.000,
496.235 VDDL dataout_44_u
VSS 156.266 137.963 18.303 544.000,
613.235 VSS GPRs/U2317
VDD 156.266 137.963 18.303 544.000,
613.235 TVDD GPRs/U2317
VDD 152.690 135.223 17.467 602.400,
467.435 VDDL dataout_33_u
```

Displaying Block-Level Rail Results

After you perform rail analysis on the top-level design, the tool saves the analysis results in the top-level rail database. To locate issues that are reported in a block-level design, run the `open_rail_result` command with the `-top_design` and `-block_instance` options to excerpt block-level instance data from the top-level rail database. The tool then displays maps for the specified block instances based on the extracted block-level rail analysis data.

Note:

The excerpt of the block-level rail results is saved in memory and is deleted when you exit the current session.

The following example opens the rail result named `REDHAWK_RESULT` in the `./RAIL_DATABASE` directory; the result is the rail analysis result for the top design named `bit_coin`. The example then creates an excerpt of rail analysis data for the block instance `slice_5`. Note that in this example, the block design of instance `slice_5` has to be opened, not the top-level design.

```
fc_shell> open_lib block.nlib
fc_shell> open_block block
fc_shell> set_app_options -name rail.database -value RAIL_DATABASE
fc_shell> open_rail_result REDHAWK_RESULT \
    -top_design bit_coin \
    -block_instance slice_5
fc_shell> report_rail_result
```

Generating Instance-Based Analysis Reports

Use the `get_instance_result` command to write the cell instances with rail data to an output text file, in the units defined with the `report_units` command.

To specify which analysis or checking types to report, use the `-type` option. The supported analysis types are: `effective_voltage_drop`, `current`, `effective_resistance`, `min_path_resistance`, and `power`.

Here are commonly used options of the `get_instance_result` command:

| Option | Description |
|--------------------------|--|
| <code>-net</code> | (Optional) Specifies the net to report. When not specified, the command reports data for all power nets. |
| <code>-touching</code> | (Optional) Reports cell instances that touch the specified rectangle region in the format of <code>\{{x1 y1} {x2 y2}\}</code> . |
| <code>-top value</code> | (Optional) Reports cell instances with the top N voltage drop values. |
| <code>-threshold</code> | (Optional) Reports cell instances with voltage drop values that are greater than the specified threshold. This option is supported by all analysis types. |
| <code>-percentage</code> | (Optional) Reports cell instances with voltage drop values that are greater than specified percentage of the ideal voltage drop value. This option is supported only by the <code>voltage_drop</code> type. The default is 0.1, meaning 10% of the ideal supply voltage. |
| <code>-histogram</code> | (Optional) Includes a histogram that shows the average voltage drop value per net in the report. |
| <code>-collection</code> | (Optional) Reports cell instances as a cell collection rather than a column-based text report. |

Example

The following example reports instance voltage drop results within the specified area:

```
fc_shelpprompt> get_instance_result -net VDD -type voltage_drop \
    -touching {{1610 1968} {1620 1969}}
Isw2/Isw2_pktpro/U179070 -0.053533
Isw2/Isw2_pktpro/U64076 -0.0535949
Isw2/Isw2_pktpro/U83035 -0.0535949
Isw2/Isw2_pktpro/U181017 -0.0535949
Isw2/Isw2_pktpro/U103335 -0.05366
Isw2/Isw2_pktpro/U181049 -0.05366
Isw2/Isw2_pktpro/U98988 -0.05366
Isw2/Isw2_pktpro/U201986 -0.053713
Isw2/Isw2_pktpro/U181064 -0.053719
Isw2/Isw2_pktpro/U79791 -0.053809
Isw2/Isw2_pktpro/U179052 -0.054007
Isw2/Isw2_pktpro/U4120 -0.054073
Isw2/Isw2_pktpro/U20408 -0.054287
```

See Also

- [Voltage Hotspot Analysis](#)

Generating Geometry-Based Analysis Reports

Use the `get_geometry_result` command to write the geometry information with rail results to an output file, in the units defined with the `report_units` command.

You can specify one net at a time. When more than one net is specified, an error message is displayed. To specify which analysis or checking types to report, use the `-type` option. The supported analysis types are: `voltage_drop` and `min_path_resistance`.

The report is in the following format:

```
geometry bbox_value
```

Here are commonly used options of the `get_geometry_result` command:

| Option | Description |
|-------------------------|--|
| <code>-layer</code> | (Optional) Specifies the name or number of the layer to report. When not specified, all layers are reported. |
| <code>-touching</code> | (Optional) Specifies a rectangular region in the format of <code>{{x1 y1} {x2 y2}}</code> in which all geometry layers that touch the specified region are reported. |
| <code>-top value</code> | (Optional) Reports the geometry layers with the top N voltage drop values. |

| Option | Description |
|-------------|---|
| -threshold | (Optional) Specifies a threshold value. Reports geometry layers with voltage drop values that are greater than specified threshold. |
| -percentage | (Optional) Specifies a percentage value. Reports geometry layers with voltage drop values that are greater than the specified percent of the ideal supply voltage. For example, if the percentage is set to 0.1, geometries with voltage drop values greater than 10% of the ideal supply voltage are reported. |
| -histogram | (Optional) Includes a histogram that shows the average voltage drop value per net in the report. |

Example

The following example writes geometry data with voltage drop values that are within the specified area:

```
prompt> get_geometry_result -net VDD -type voltage_drop \
    -touching {{1610 1968} {1620 1969}}
{ { 1616.475 1967.000 } { 1617.525 1969.800 } } metal5 0.00448
{ { 1608.810 1946.000 } { 1611.190 1974.000 } } metal7 0.00446
{ { 1608.000 1946.000 } { 1612.000 1974.000 } } metal9 0.00446
```

See Also

- [Voltage Hotspot Analysis](#)

Displaying Maps in the GUI

When analysis is complete, the tool saves the design data and analysis results (*.result) in the `in-design.redhawk/design_name.result` directory under the RedHawk working directory. You must run the `open_rail_result` command to load the analysis results before displaying a map in the Fusion Compiler GUI.

[Table 68](#) lists the map types that are supported in the RedHawk Fusion analysis flow.

Table 68 Analysis Maps Supported in RedHawk Fusion Flow

| Map Type | Description |
|--|---|
| Rail voltage drop map | <p>Displays the voltage drop map, which is a color-coded display of voltage drop values overlaid on the physical supply nets. For static analysis, the map displays average voltage drop values. For dynamic analysis, the map displays peak voltage drop values, average voltage drop values, or peak voltage rise values.</p> |
| Note: | <p>The tool shows all the layers in the LEF or DEF file on the voltage drop map even if the layers are absent in the Fusion Compiler library file.</p> |
| Rail parasitics map | <p>Shows parasitic resistance of a given supply net in the block when voltage drop analysis is complete. It shows the resistance for any given shape of the net.</p> |
| Rail power map | <p>Displays instance-based power map or power density map for the block when voltage drop analysis is complete, based on the RedHawk power calculation results. Instance-based power map: Shows power values of the cell instances in the block Power density map: Shows power density values for area in the block</p> |
| Rail current map | <p>Displays current distribution for the block when voltage drop analysis is complete.</p> |
| Rail minimal path resistance map | <p>Shows minimum path resistance values for a selected net when minimal path resistance analysis is complete.</p> |
| Rail electromigration map | <p>Shows current violations for the block when static or dynamic electromigration analysis is complete.</p> |
| Rail instance peak current map | <p>Shows peak current values for different instances or areas in the block.</p> |
| Rail instance effective voltage drop map | <p>Shows effective voltage values of a given instance in the block when voltage drop analysis is complete.</p> |
| Rail instance effective resistance map | <p>Shows effective resistance values of a given instance in the block when effective resistance analysis is complete.</p> |
| Rail instance switch cell | <p>Displays current, effective voltage, and voltage difference maps for switch cells in the block when dynamic analysis is complete and switch cell model input files are available. If no switch model input file is available, the tool treats these switch cells as black boxes and no rail map is displayed. Instance-based current map: This map shows how current flows through the selected switch cell. Instance-based effective voltage map: This map shows the effective voltage at the selected switch cell. Instance-based voltage difference map: This map shows the voltage difference across the selected switch cell.</p> |

To display maps in the Fusion Compiler GUI,

1. To load the rail analysis result using the GUI, choose **Task > Rail Analysis > 2D Rail Analysis**. The Block Level 2D Rail Analysis window appears. In the window, click the **open_rail_result** link.

Alternatively, run the `open_rail_result` command to load the design data and analysis results (*.result) saved in the `design_name` directory under the RedHawk working directory when analysis is complete.

2. In the GUI layout window, choose **View > Map** and select the map to display.

In the map, problem areas are highlighted in different colors. Move the pointer over an instance to view more information in the InfoTip. By default, the tool displays information for all instances in the block. To examine information of one specific instance, deselect all the instances and then select the instance to display from the list.

Use the options in the map panel to change map display:

- Value: Sets the type of map to display
- Bins: Changes the number of unique bins used to display the map
- From: and To: Sets the range of density values displayed in the power density map
- Text: Displays the calculated values in the layout; if the values are not visible, zoom in to view them

Examples

The following examples show how to display various types of maps in the GUI:

[Figure 196](#) shows how to display a rail instance effective voltage drop map and check for hotspots in the instance effective voltage drop map.

[Figure 197](#) shows how to display a rail map for the switch cells in the block and examine the voltage information of the selected switch cell.

[Figure 198](#) shows how to display a power density map for the block.

Figure 196 Displaying a Rail Instance Effective Voltage Drop Map

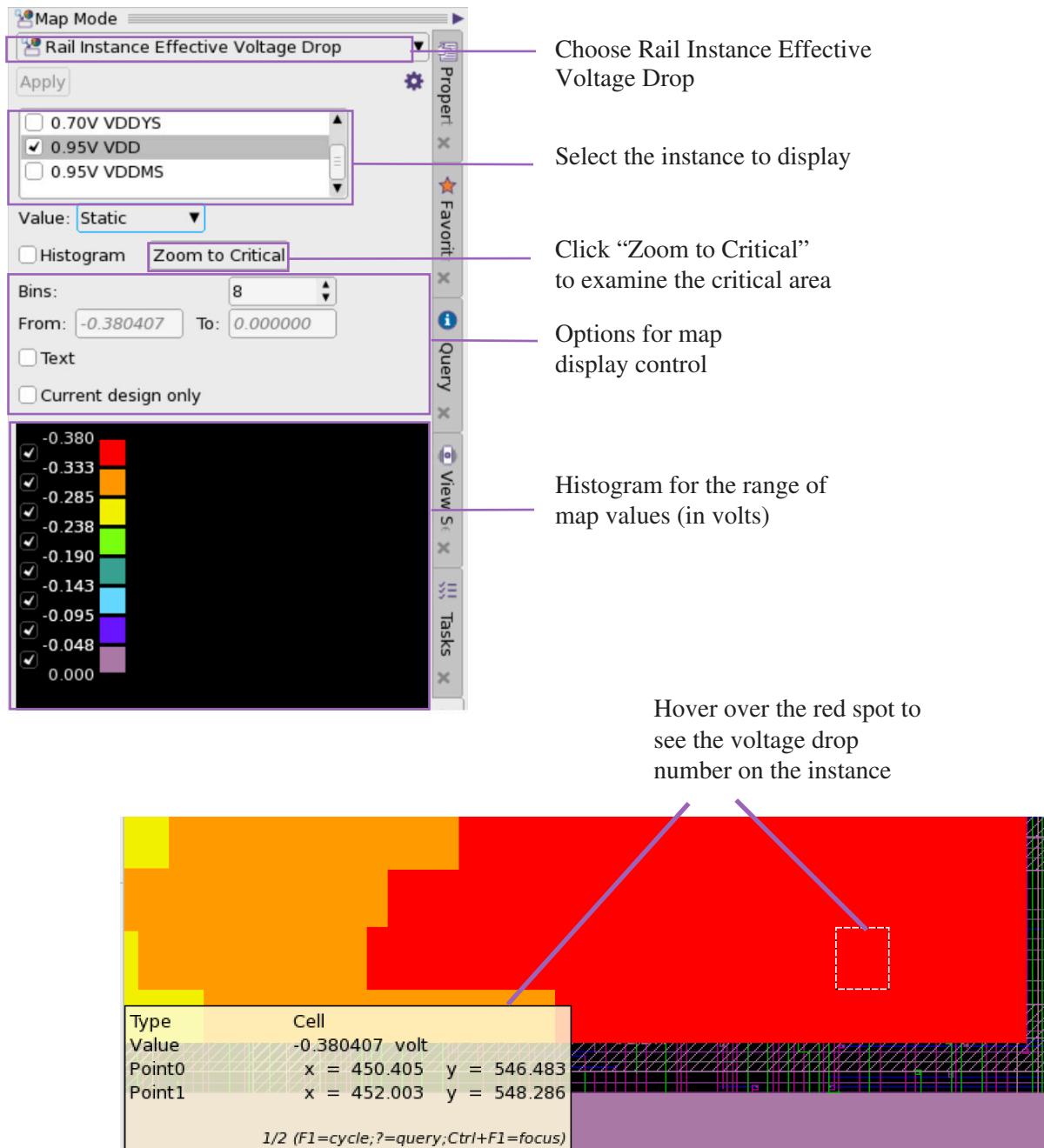


Figure 197 Displaying Maps for Switch Cells

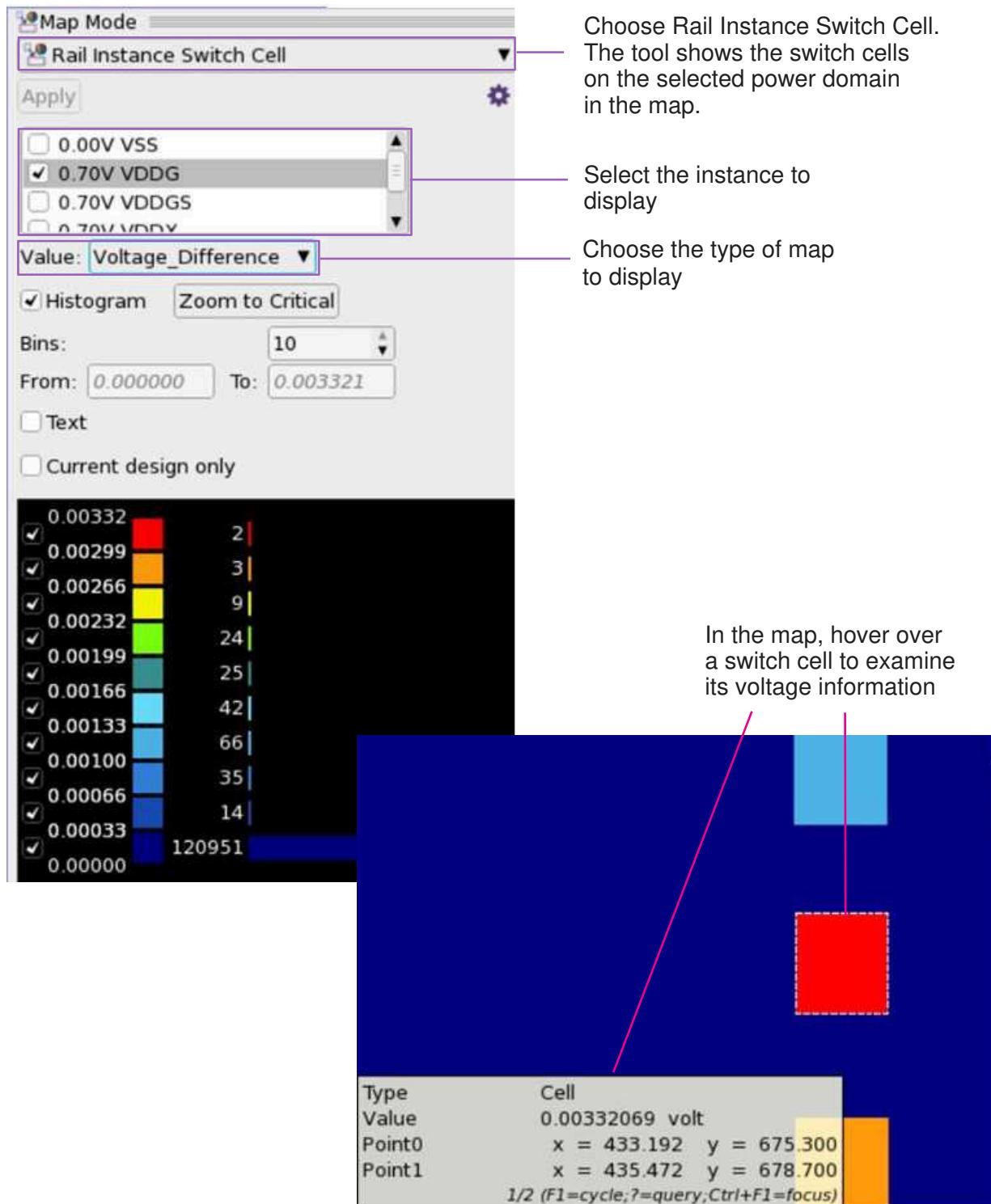
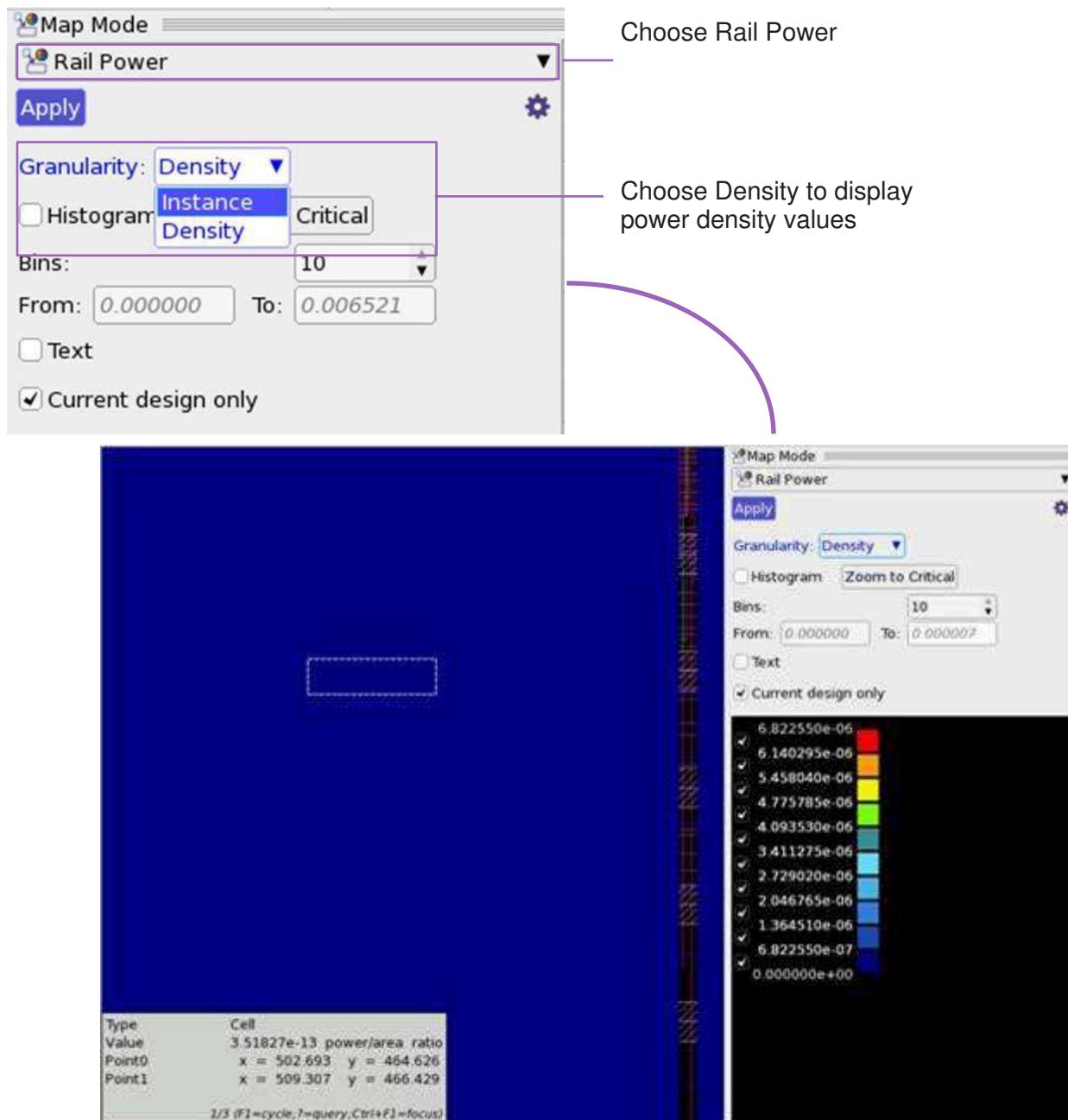


Figure 198 Displaying a Power Density Map



See Also

- [Specifying RedHawk and RedHawk-SC Working Directories](#)

Displaying ECO Shapes in the GUI

When you use the `RedHawk mesh add` command to add virtual PG meshes in the top block for the current subblock, by default the virtual PG meshes created by the `RedHawk mesh add` command are not shown as layout objects in the GUI. Therefore, these GUI shapes cannot be queried using layout object collection commands (such as, `get_shapes`), and are not saved to the design library when you save the block.

To save ECO shapes along with rail results and display them in the GUI, enable the `rail.display_eco_shapes` application option, like

```
set_app_options -name rail.display_eco_shapes -value true
```

Note:

The `mesh add` command is available only in RedHawk, not in RedHawk SC. Therefore, the tool issues an error message when you enable both the `rail.enable_redhawk_sc` and `rail.display_eco_shapes` application options.

To display ECO shapes in GUI, you must first enable the RedHawk signoff license key by setting the `rail.allow_redhawk_license_checkout` application option to `true`. Otherwise, an error message is issued.

To add virtual PG meshes and display the created ECO shapes in the GUI,

1. Prepare a script file (such as, `mesh.tcl`) that contains the `mesh add` command.
2. Run design setup with the `mesh.tcl` script file.

```
prompt> set_rail_command_options -script_file mesh.tcl \
           -command setup_design -order after_the_command
```

The tool generates the RedHawk script file and sources the `mesh.tcl` file after design setup.

3. Perform rail analysis using the following command:

```
prompt> analyze_rail -nets -voltage_drop
```

4. Run the `open_rail_result` command to load the design data and analysis results.
5. Run the `gui_start` command to open the Fusion Compiler GUI. In the GUI layout window, choose View > Map and select the map to display.

You can now display and query the ECO shapes that are created by the RedHawk `mesh add` command in the GUI.

Script Examples

The following script displays ECO shapes in the GUI by using the `set_rail_command_option` command.

```
## Open design ##
open_block
link_block
## Specify taps ##
create_taps
## Specify RedHawk Fusion input files or variables ##
set_app_options -name rail.enable_redhawk -value 1
set_app_options -name rail.redhawk_path -value
set_app_options -name rail.disable_timestamps -value true
set_app_options -name rail.display_eco_shapes -value true
set_rail_command_options -script_file mesh.tcl -command \
    setup_design ...
## Analyze##
analyze_rail -voltage_drop ... -nets {VDD VSS}
## Check GUI ##
open_rail_result
```

The following script displays ECO shapes in the GUI by using a RedHawk script.

```
## Open design ##
open_block
link_block
## Specify taps ##
create_taps
## Specify RedHawk Fusion input files or variables ##
set_app_options -name rail.enable_redhawk -value 1
set_app_options -name rail.redhawk_path -value
set_app_options -name rail.disable_timestamps -value true
set_app_options -name rail.display_eco_shapes -value true
## Analyze ##
analyze_rail -voltage_drop ... -nets {VDD VSS} -script_only
## Modify the script.tcl containing mesh add command ##
analyze_rail -redhawk_script_file
## Check GUI ##
open_rail_result
```

Voltage Hotspot Analysis

When rail analysis is complete, there might be hundreds to thousands of voltage drop violations reported in the analysis report, which might make it difficult to identify the root causes for the violations.

Use the voltage hotspot analysis capability to generate hotspots for a power or ground net by dividing the whole chip into small grid boxes, and report rail-related information for the generated grid boxes. In addition, you can query cell- or geometry-based rail

results only for the specific analysis type by using the `get_instance_result` and `get_geometry_result` commands.

Hotspot analysis provides the following features:

- Identify the root cause of aggressors with high voltage drop, such as large current from the aggressor itself or from the neighboring cells due to simultaneous switching by the overlapping timing windows.
- Determine which design technique to use for fixing voltage violations based on the report, such as choosing a candidate reference cell for cell sizing replacement, moving or relocating cells to reduce voltage drops, or applying PG augmentation to solve a voltage drop issue.

This section contains the following topics:

- [Generating Hotspots](#)
- [Reporting Hotspots](#)
- [Removing Hotspots](#)
- [Voltage Hotspot Analysis Examples](#)

To write rail results only for the cell instances or geometries that reside in the hotspot area, use the `get_instance_result` and `get_geometry_result` commands. For more information, see [Generating Instance-Based Analysis Reports](#) and [Generating Geometry-Based Analysis Reports](#).

Generating Hotspots

To generate voltage drop hotspots for a power or ground net, use the `generate_hot_spots` command. This command divides the whole chip area into multiple grids in terms of rows and columns, such as 100x100. The grid boxes are sorted by the maximum effective voltage drop value in each grid box and are indexed with an integer number starting from 0.

To create rows based on standard cell site rows, and columns by using vertical PG straps on the lowest metal layers, use the `-site_row` option. Alternatively, you can specify the number of rows and columns for dividing the whole region with the `-row` and `-column` options.

The command creates error cells for the grids with voltage drop values greater than the specified percentage of the ideal voltage drop. The default is 0.1, meaning that error cells are created for the grids with voltage drop values greater than 10% of the ideal supply voltage. The error cell type is effective voltage drop.

To examine the content of the generated error cells, open the error cells in the error browser. To examine the content of the grid box, use the `report_hot_spots` command.

To remove the generated voltage hotspot data from memory, use the `remove_hot_spots` command.

| Option | Description |
|--------------------------------|--|
| <code>-net</code> | Specifies the net name for generating hotspots. When not specified, all power nets are used. |
| <code>-percentage</code> | Specifies the percentage used to generate hotspot error cells. The default is 0.1. |
| <code>-site_row integer</code> | By default, the command uses the standard site row height to divide the whole region into multiple rows, and the lowest PG net vertical straps to divide the whole region into multiple columns.
To use a multiple of the site row height for creating grid rows, specify the <code>-site_row</code> option with an integer. For example, when the <code>-site_row</code> option is set to 2, the command uses twice the site row height to construct grid rows; that is, a row is constructed for every two site rows. |
| <code>-row</code> | Specifies the number of rows into which to divide the whole region. |
| <code>-column</code> | Specifies the number of columns into which to divide the whole region. |

See Also

- [Reporting Hotspots](#)
- [Removing Hotspots](#)
- [Voltage Hotspot Analysis Examples](#)

Reporting Hotspots

After you generate hotspots on the block using the `generate_hot_spots` command, run the `report_hot_spots` command to report cells with voltage-related cell attributes for a hotspot grid box. The voltage-related cell attributes are: `current`, `overlap_current`, `effective_resistance`, `timing_window`, `slack`, `output_load` and `static_power`.

Use these voltage-related cell attributes to identify root causes of voltage violations. For example, you can

- Use the current and output loading capacitance information to determine if a cell's high peak is caused by too much load. If this is the case, splitting the output might be an effective approach to reduce high voltage drops.
- Check and compare the timing windows in the current hotspot to those of the neighboring grids to determine if a cell can be moved to other grids (targets). Doing so

reduces the voltage drop of the original grid without increasing the voltage drop of the target grid.

- Compare slew and load capacitance among all cells in the region to determine if high peak current is caused by sharp slew or large load capacitance.
- Find cells that have overlapping timing windows to identify cells that might switch at the same time during dynamic rail analysis.

Here are commonly used options of the `report_hot_spots` command:

| Option | Description |
|-----------------------|---|
| <code>-index</code> | Lists the hotspot grids by index. |
| <code>-summary</code> | Reports a summary for all grids, such as the number of grids created, the number of instances in the block, and the number of grids whose voltage drop values are higher than the specified percentage of the ideal voltage drop value. |
| <code>-object</code> | Specifies the name of the object to report. Lists the hotspot grid that contains the specified cell instance object. |
| <code>-verbose</code> | Prints a detailed report. |

Listing Voltage Grid Boxes by Index

By default, the tool searches three rows up and down and three columns left and right to determine the neighboring grids of the current one. Use the `-index` option to specify the index used to report voltage hotspot grids.

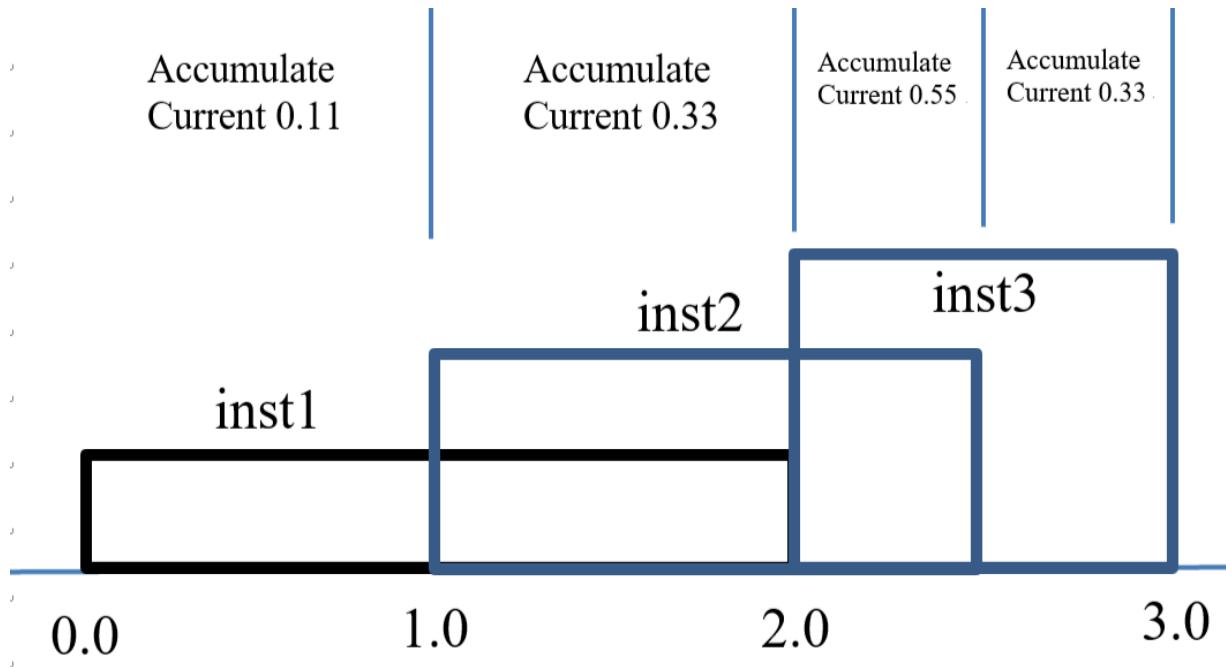
The grid box with the smallest index number has the highest effective voltage drop values. This allows you to determine the size of the hotspot area. For example, if all the surrounding grids have bigger index numbers and only a few central grids have much smaller numbers, this voltage drop hotspot is assumed to be an island.

Accumulating Current

For an instance in a grid box, the overlap current is estimated by accumulating the timing-window-based current across its neighboring cells.

The command scans all timing windows from smallest to biggest, and accumulates current on the way as time passes by. Use the information to determine how many cells are switching simultaneously and how much time shift to apply when shifting timing windows for voltage drop reduction.

Assume the design has three instances with current as follows:



As shown in the following report, the accumulated current on point 2.0 is from inst2 and inst3, and does not include the current from inst1.

```
inst1 [0.0 2.0] current 0.11
inst2 [1.0 2.5] current 0.22
inst3 [2.0 3.0] current 0.33
```

See Also

- [Generating Hotspots](#)
- [Removing Hotspots](#)
- [Voltage Hotspot Analysis Examples](#)

Removing Hotspots

The tool saves the generated hotspot data in memory. If you want to generate hotspots for other nets in another run, you need to remove the previously generated hotspot data from memory with the `remove_hot_spots` command before proceeding to another hotspot analysis run.

For example, if you have run the `generate_hot_spots` command on the VDD net, you must first remove the hotspot data for the VDD net before generating hotspots for other nets.

See Also

- [Generating Hotspots](#)
 - [Reporting Hotspots](#)
 - [Voltage Hotspot Analysis Examples](#)
-

Voltage Hotspot Analysis Examples

This topic provides examples about how to use the hotspot analysis capability to identify the root cause of voltage violations.

Cells with overlapping timing windows might switch at the same time, leading to high voltage drops. [Figure 199](#) shows how to identify if the cell has overlapping timing windows.

Figure 199 Checking High Voltage Drop Caused by Overlapping Timing Window (I)

```

prompt> generate_hot_spots -net VSS -site_row
prompt> report_hot_spots -index 3 -verbose
Voltage drop hot spot index: 3
Grid BBox: [423 547][454 548]
Number of instances: 20
Max effective voltage drop: 0.413517
    
```

Neighboring grids by index

| | | | | | |
|------|------|------|------|-----|------|
| 420 | 327 | 172 | 117 | 125 | 294 |
| 2344 | 2345 | 2347 | 2348 | 106 | 2349 |
| 2332 | 2333 | 70 | 16 | 17 | 2335 |
| 2250 | 2251 | 68 | 3 | 5 | 2255 |
| 2235 | 2239 | 63 | 2 | 4 | 2275 |
| 2290 | 2259 | 102 | 6 | 7 | 2261 |

The smaller the index, the higher the voltage drop. This indicates the localized voltage drop hot spots

Top instances sorted by instance effective voltage drop (exclude 2 physical only cells)

| Instance | EVD | Current | Overlap I Eff | Timing window | Slack | Slew | Load | Power |
|---------------------------------------|-------|---------|---------------|---------------|-------|--------|--------|----------|
| Multiplier/GENPP/ISO_iso_Y_31_UPF_ISO | 0.435 | 59.5 | 4.12e+03 0 | [1.55 10.6] | -2.6 | 0.025 | 0.0526 | 5.67e+07 |
| Multiplier/GENPP/ISO_iso_X_23_UPF_ISO | 0.435 | 112 | 5.52e+03 0 | [0.797 1.9] | 0.539 | 0.0255 | 0.0364 | 1.07e+08 |
| Multiplier/GENPP/ISO_iso_Y_16_UPF_ISO | 0.435 | 59.5 | 3.62e+03 0 | [0.801 1.83] | 0.573 | 0.0199 | 0.0364 | 5.66e+07 |
| Multiplier/GENPP/ISO_iso_Y_5_UPF_ISO | 0.433 | 59.5 | 3.36e+03 0 | [0.768 1.79] | 0.586 | 0.0269 | 0.0364 | 5.67e+07 |
| Multiplier/GENPP/ISO_iso_X_5_UPF_ISO | 0.433 | 225 | 5.51e+03 0 | [1.55 3.01] | 1.29 | 0.0257 | 0.0624 | 2.14e+08 |
| Multiplier/GENPP/ISO_iso_Y_19_UPF_ISO | 0.429 | 59.5 | 5.9e+03 0 | [0.993 2.06] | 0.773 | 0.0607 | 0.0364 | 5.67e+07 |
| Multiplier/GENPP/ISO_iso_X_0_UPF_ISO | 0.421 | 627 | 5.66e+03 0 | [1.55 3.02] | 1.29 | 0.0271 | 0.0738 | 2.14e+08 |
| Multiplier/GENPP/ISO_iso_X_16_UPF_ISO | 0.409 | 225 | 4.41e+03 0 | [1.55 2.98] | 1.29 | 0.0237 | 0.0526 | 2.14e+08 |
| Multiplier/GENPP/ISO_iso_X_22_UPF_ISO | 0.396 | 112 | 4.02e+03 0 | [0.81 1.91] | 0.56 | 0.026 | 0.0526 | 1.07e+08 |

The cell does not have large current. Its neighboring cells has overlapped timing window and large currents.

Total accumulated current in timing window (1.55, 1.79): 1814.12
 Total accumulated current: 1814.12

This is the aggressor which contributes high current in this grid. This grid has overlap current 1814, but neighboring overlap current is 5660. This means grids nearby have bigger currents. Use: report_hot_spots -index 2 or 5

Next, run the `report_hot_spots` command with the `-index` option to find which grid has overlapping timing windows that contribute to large current. As shown in Figure 200, when the `-index` option is set to 5, the grid 5 is reported with small total current. When the `-index` option is set to 2, the grid 2 is reported with large total current. In this case, overlapping time windows contribute to the high effective voltage drop in grid 2.

Figure 200 Checking High Voltage Drop Caused by Overlapping Timing Window (II)

```
prompt> report_hot_spots -index 5 -verbose
-----
Total accumulated current in timing window (1.59, 3.09): 104.794
Total accumulated current: 104.794

prompt> report_hot_spots -index 2 -verbose
-----
Total accumulated current in timing window (1.59, 1.8): 1704.7
Total accumulated current: 1704.7
...
```

See Also

- [Generating Hotspots](#)
- [Reporting Hotspots](#)
- [Removing Hotspots](#)

Querying Attributes

When rail analysis or missing via checking is complete, use the `get_attribute` attribute to query the results stored in the RedHawk working directory. You can query the attributes shown in [Table 69](#), which are specific to the rail analysis results.

Table 69 Rail Result Object Attributes

| Object | Attributes |
|--------|--------------|
| Cell | static_power |

Table 69 Rail Result Object Attributes (Continued)

| Object | Attributes |
|--------|--|
| Pin | static_current
peak_current
static_power
switching_power
leakage_power
internal_power
min_path_resistance
voltage_drop
average_effective_voltage_drop_in_tw
max_effective_voltage_drop
(=effective_voltage_drop)
max_effective_voltage_drop_in_tw
min_effective_voltage_drop_in_tw
effective_resistance |

The following example retrieves the `effective_voltage_drop` attribute for the cellA/VDD pin.

```
fc_shell> get_attribute [get_pins cellA/VDD] effective_voltage_drop  
-0.02812498807907104
```

The following example retrieves the `static_power` attribute for the cellA cell.

```
fc_shell> get_attribute [get_cells cellA] static_power  
2.014179968645724e-05
```

13

ECO Flow

An engineering change order (ECO) is an incremental change made to a complete or nearly complete design. You can use ECOs to fix functional, timing, noise, and crosstalk violations without synthesizing, placing and routing the entire design. You can also use ECOs to implement late-arriving design changes while maintaining design performance.

The following topics describe the various ECO flows supported in the Fusion Compiler tool, and tasks you perform in these flows:

- [Generic ECO Flow for Timing or Functional Changes](#)
- [Freeze Silicon ECO Flow](#)
- [Signoff ECO Flow](#)
- [Incremental Signoff ECO Flow](#)
- [ECO Fusion Flow](#)
- [ECO Fusion Power Integrity Flow](#)
- [Manually Instantiating Spare Cells](#)
- [Automatically Adding Spare Cells](#)
- [Adding Programmable Spare Cells](#)
- [Making ECO Changes Using the eco_netlist Command](#)
- [Making ECO Changes Using Netlist Editing Commands](#)
- [Resizing Cells](#)
- [Adding Buffers on Nets](#)
- [Adding Buffers on Routed Nets](#)
- [Optimizing the Fanout of a Net](#)
- [Reporting Available Sites for Placing ECO Cells](#)
- [Identifying and Reverting Nonoptimal ECO Changes](#)
- [Placing ECO Cells](#)

- Placing and Mapping ECO Cells to Spare Cells
- Updating Supply Nets for ECO Cells
- Recording the Changes Made to a Layout
- Performing Prerequisite Check for Group Repeater Insertion and Placement
- Adding a Group of Repeaters
- Querying Group Repeater
- Swapping Variant Cell
- Fixing Multivoltage Violations

Generic ECO Flow for Timing or Functional Changes

Use this flow to incorporate timing or functional ECO changes when you have the flexibility to add new cells and move or delete existing cells. This flow is recommended if you have not taped out your design.

The unconstrained ECO flow consists of the following steps:

1. Update the design with the ECO changes by using one of the following methods:
 - Using the `eco_netlist` command, as described in [Making ECO Changes Using the eco_netlist Command](#)
 - Using netlist editing Tcl commands, as described in [Making ECO Changes Using Netlist Editing Commands](#).
2. Update the placement by using the `place_eco_cells` command, as described in [Placing ECO Cells](#).
3. Add filler cells to the empty spaces in the site array, as described in [Inserting Filler Cells](#).

To reduce runtime, use the `-post_eco` option when you

 - Insert metal filler cells with the `create_stdcell_fillers` command, and the tool marks the inserted filler cells as post-ECO cells.
 - Remove filler cells with DRC violations with the `remove_stdcell_fillers_withViolation` command, and the tool performs DRC checking only for the post-ECO cells.
4. Update the routing by using the `route_eco` command, as described in [Performing ECO Routing](#), or by manually rerouting the affected nets.

Freeze Silicon ECO Flow

Use this flow if your cell placement is fixed, and you can only change the metal and via mask patterns. This flow is recommended if you have taped out your design and you want to avoid the expense of generating a whole new mask set.

To perform the freeze silicon ECO flow, your block must contain spare cells. You can add spare cells to a block, anytime during the design flow, by using one of the following methods:

- Manually instantiate spare cells, as described in [Manually Instantiating Spare Cells](#).
- Automatically add spare cells after placement by using the `add_spare_cells` command, as described in [Automatically Adding Spare Cells](#).
- Add programmable spare cells during the chip finishing stage by using the `create_stdcell_fillers`, as described in [Adding Programmable Spare Cells](#).

The freeze silicon ECO flow consists of the following steps:

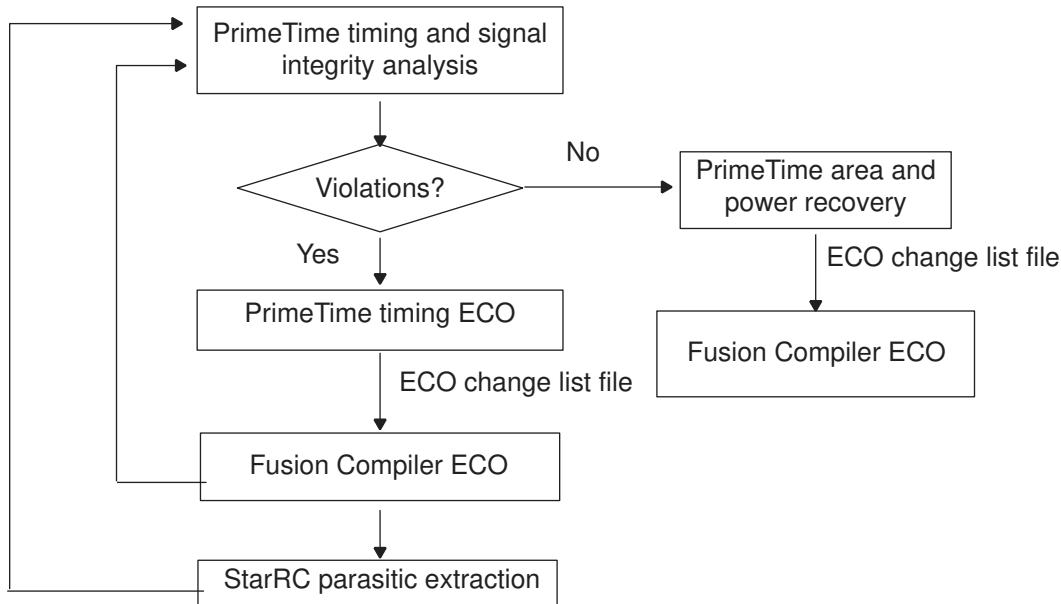
1. Enable ECO changes in the freeze silicon mode by setting the `design.eco_freeze_silicon_mode` application option to `true`.
2. Update the design with the ECO changes by using one of the following methods:
 - Using the `eco_netlist` command, as described in [Making ECO Changes Using the eco_netlist Command](#)
 - Using netlist editing Tcl commands, as described in [Making ECO Changes Using Netlist Editing Commands](#).
3. Analyze the mapping of ECO cells to spare cells by using the `check_freeze_silicon` command.
4. Automatically map all the ECO changes to spare cells by using the `place_freeze_silicon` command or manually map each ECO cell to a specific spare cell by using the `map_freeze_silicon` command, as described in [Placing and Mapping ECO Cells to Spare Cells](#).
5. Update the routing by using the `route_eco` command, as described in [Performing ECO Routing](#), or by manually rerouting the affected nets.

Signoff ECO Flow

After you perform place and route in the Fusion Compiler tool, if your design has timing or design rule violations, you can fix these violations in the PrimeTime tool. You can also perform power or area recovery in the PrimeTime tool.

If you make changes to your design in the PrimeTime tool, you can generate an ECO change list file and incorporate those changes into the design by using the Fusion Compiler ECO capabilities, as shown in [Figure 201](#).

Figure 201 Signoff ECO Flow



To incorporate the PrimeTime ECO changes, use the following steps:

1. Update the design by sourcing the PrimeTime ECO change list file, which is a Tcl file containing netlist editing commands.
2. Update the placement by using the `place_eco_cells` command as shown in the following example:

```
place_eco_cells -legalize_mode minimum_physical_impact \
-eco_changed_cells -legalize_only -displacement_threshold 10
```

For more information about the `place_eco_cells` command, see [Placing ECO Cells](#).

3. Add filler cells to the empty spaces in the site array, as described in [Inserting Filler Cells](#).

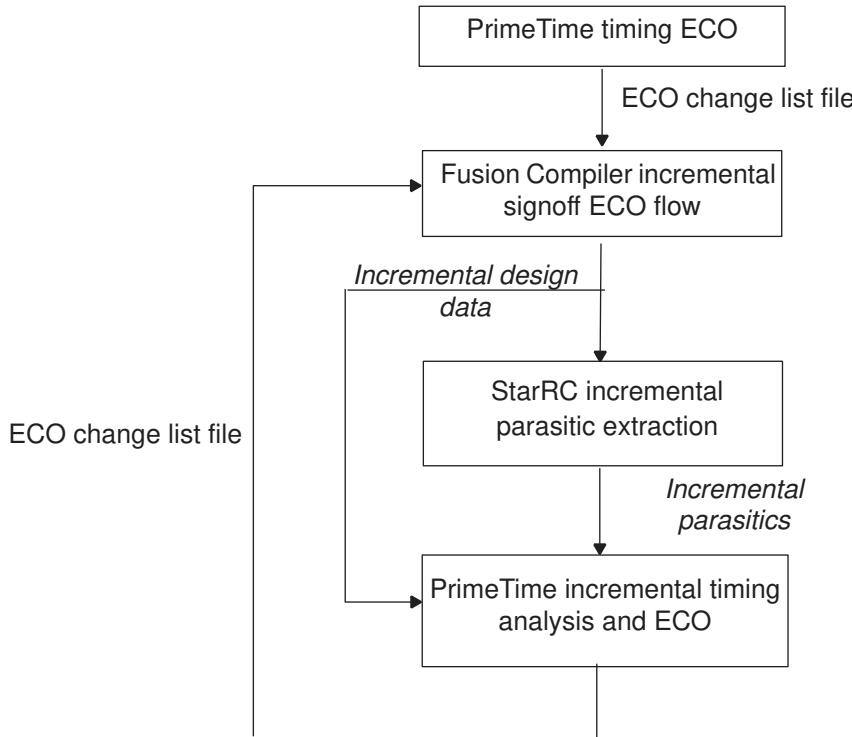
To reduce runtime, use the `-post_eco` option when you

- Insert metal filler cells with the `create_stdcell_fillers` command, and the tool marks the inserted filler cells as post ECO cells
 - Remove filler cells with DRC violations with the `remove_stdcell_fillers_withViolation` command, and the tool performs DRC checking only for the post ECO cells
4. Update the routing by using the `route_eco` command, as described in [Performing ECO Routing](#), or by manually rerouting the affected nets.

Incremental Signoff ECO Flow

When you make ECO changes in the PrimeTime tool and incorporate these changes by performing the Fusion Compiler unconstrained ECO flow, it might be necessary to iterate multiple times between the tools to meet the required QoR goals. To reduce the overall turnaround time for the flow, you can reduce the runtime of each iteration by using the incremental signoff ECO flow. This flow generates incremental design data from the Fusion Compiler tool, which enables you to perform incremental extraction in the StarRC tool and incremental timing analysis and ECO in the PrimeTime tool, as shown in the following figure.

Figure 202 Incremental Signoff ECO Flow



To perform the incremental signoff ECO flow within the Fusion Compiler tool, use the `record_signoff_eco_changes` command. This command incorporates the PrimeTime ECO into the design library, tracks all the changes made to the design, and generates the incremental files that are required to run StarRC incremental extraction and PrimeTime incremental timing analysis and ECO.

The Fusion Compiler incremental signoff ECO flow consists of the following steps:

1. Open the design library by using the `open_block` command.
2. Incorporate the PrimeTime ECO changes and begin tracking the ECO changes to the design by using the `record_signoff_eco_changes -start -input` command as shown in the following example:

```
fc_shell> record_signoff_eco_changes -start -input pt_eco.tcl
```

3. (Optional) Perform additional timing ECO changes to the design by using netlist editing commands.

Do not perform functional ECO changes to the design. If you do so, the tool stops tracking the ECO changes being performed on the design.

4. Place the ECO cells by using the `place_eco_cells` command.
5. Update the routing by using the `route_eco` command, as described in [Performing ECO Routing](#), or by manually rerouting the affected nets.
6. Stop tracking the ECO changes and complete the incremental signoff ECO flow by using the `record_signoff_eco_changes -stop` command as shown in the following example:

```
fc_shell> record_signoff_eco_changes -stop
```

ECO Fusion Flow

The Fusion Compiler ECO Fusion flow allows you to use the PrimeTime physically aware ECO capabilities and the StarRC extraction capabilities (In-Design signoff extraction) within the Fusion Compiler tool. You can use this flow to fix timing and DRC violations and improve area and power QoR at the very late stages of the implementation flow.

The Fusion Compiler ECO Fusion flow consists of the following steps:

1. Specify the settings required to run the PrimeTime ECO capabilities within the Fusion Compiler tool by using the `set_pt_options` command.

The following example specifies the path to the PrimeTime executable and the settings for distributed processing:

```
fc_shell> set_host_options -name pteco_host_option \
    -submit_command "/lsf/bin/bsub -R \"${usage}[mem=$MEM]\""
fc_shell> set_pt_options -pt_exec /snps_tools/PT/pt_shell \
    -host_option pteco_host_option
```

If there is a `.synopsys_pt.setup` file in the current working directory, the Fusion Compiler tool uses the information in this file. However, you can specify additional PrimeTime settings by using the `-pre_link_script` and `-post_link_script` options of the `set_pt_options` command.

For more information, see the man page for the `set_pt_options` command.

2. Set up StarRC extraction as follows:

- a. Ensure that StarRC extraction is enabled.

StarRC extraction is controlled by the `extract.starrc_mode` application option. The default is `fusion_adv`. You can also set the value to `in-design` or `none`.

To use the native extractor, set the value to `none`.

- b. Specify a configuration file for running StarRC extraction by using the `set_starrc_options -config` command.

3. Perform optimization by using the `eco_opt` command.

By default, the `eco_opt` command fixes timing (setup and hold) and DRC violations, removes redundant buffers, and improves the total power QoR by using PrimeTime ECO capabilities. It then incorporates the ECO changes by using the Fusion Compiler ECO place and route capabilities. However, you can control the type of optimization by using the `-type` option.

The following example uses the `eco_opt` command to fix only timing and DRC violations using exhaustive path-based analysis:

```
fc_shell> eco_opt -pba_mode exhaustive -types {timing drc}
```

The following example uses the `eco_opt` command to improve the leakage power QoR. However, it does not incorporate the ECO changes. Instead, it generates a file containing the ECO changes.

```
fc_shell> eco_opt -pba_mode exhaustive -types {timing drc} \
           -write_change_file_only
```

For more information, see the man page for the `eco_opt` command.

4. Analyze the PrimeTime QoR by using the `check_pt_qor` command.

For more information, see the man page for the `check_pt_qor` command.

Note:

When you run the `eco_opt` command, the Fusion Compiler tool specifies the PrimeTime settings based on the Fusion Compiler timing analysis settings, such as derating factors, on-chip-variation settings, and so on. Therefore, for optimal convergence, ensure that the Fusion Compiler timing analysis settings at the postroute stage are consistent with the PrimeTime timing analysis settings. You can identify the differences in the Fusion Compiler and PrimeTime settings by using the `check_consistency_settings` command.

ECO Fusion Power Integrity Flow

The Fusion Compiler ECO Fusion power integrity flow allows you to use the RedHawk Fusion rail analysis feature to identify power integrity issues and the PrimeTime physically aware ECO capabilities to fix them.

The Fusion Compiler ECO Fusion power integrity flow consists of the following steps:

1. Specify the path for saving the RedHawk Fusion rail analysis results by setting the `rail.database` application option.
2. Perform dynamic-vectorless or dynamic-vector-based voltage-drop analysis by using the `analyze_rail -voltage_drop dynamic_vcd` or `analyze_rail -voltage_drop dynamic_vectorless` command.
3. Specify the settings required to run the PrimeTime ECO capabilities within the Fusion Compiler tool by using the `set_pt_options` command.
4. Fix the identified power integrity issues by using the `eco_opt -type power_integrity` command.

To fix power integrity (voltage drop) violations, the tool downsizes aggressor cells while considering the timing and DRC QoR. However, for this optimization to be effective,

- The aggressor cells should not be timing- or DRC-critical
- The cell library should have smaller versions of the aggressor cells without `dont_use` or `dont_touch` attribute settings.

The following is an example script for running the ECO Fusion power integrity flow:

```
set_app_options -name rail.database -value RAIL_DATABASE_BASELINE
analyze_rail -voltage_drop dynamic_vcd -all_nets \
    -switching_activity {VCD ./top.vcd top}
set_pt_options -pt_exec /snps_tools/PT/pt_shell
eco_opt -type power_integrity
```

Manually Instantiating Spare Cells

You can manually instantiate spare cells in a block by using one of the following methods:

- Instantiating them in the Verilog netlist
- Adding them by using netlist editing commands such as `create_cell`, `connect_pins`, and so on

When manually instantiating spare cells, you should

- Evenly distribute the spare cells through the logical hierarchy to better handle ECO changes anywhere in the block
- Tie their inputs to power or ground, as appropriate, to prevent the spare cells from creating noise and consuming power

If a cell meets the following criteria, the tool automatically identifies it as a spare cell:

- It is not a physical-only cell
- All inputs except for the clock pin are unconnected or tied to a logic constant
The clock pin of the spare cell, if any, can be connected to the clock network.
- All outputs are unconnected

If you instantiate the spare cells before you perform physical synthesis on a block, the tool places and legalizes the spare cells during the subsequent physical synthesis steps. However, if you instantiate the spare cells in a block that is optimized, placed and legalized, you must place and legalize the spare cells by using the following steps:

1. Spread the spare cells by using the `spread_spare_cells` command.

By default, this command distributes and places all the spare cells evenly throughout the core area.

You can specify the spare cells to place by using one of the following two methods:

- To place specific spare cells, use the `-cells` option.
- To place all the spare cells that belong to specific voltage areas, use the `-voltage_areas` option.

You can control the placement of the spare cells as follows:

- Specify an area within which to place the spare cells by using the `-boundary` option.
- Ignore specific types of placement blockages by using the `-ignore_blockage_types` option. By default, the `add_spare_cells` command honors all placement blockage types.
- Ignore the current cell density and place the spare cells randomly by using the `-random_distribution` option.
- Specify a percentage of spare cells to be placed based on the cell density distribution by using the `-density_aware_ratio` option. The rest of the spare cells are placed randomly throughout the design. By default, all the spare cells are placed based on the cell density distribution.

For example, if you specify a setting of `-density_aware_ratio 80`, the tool places 80 percent of the spare cells based on the cell density distribution and 20 percent randomly across the design.

2. Legalize the spare cells by using the `place_eco_cells -legalize_only -cells` command.

Automatically Adding Spare Cells

After placement and optimization, you can add spare cells and legalize them by using the following steps:

1. Add spare cells by using the `add_spare_cells` command and specify the following information:

- A name prefix for the spare cells by using the `-cell_name` option
- The type and number of spare cells to insert by using one of the following two methods:
 - Specify the library cells to use for the spare cell and the number of instances of each library cell by using the `-lib_cell` and `-num_instances` options

For example, to insert 250 instances each of the AND2 and OR2 library cells, use the following command:

```
fc_shell> add_spare_cells -cell_name spare \
    -lib_cell {AND2 OR2} -num_instances 250
```

- Specify the library cells and a different number of instances for each library cell by using the `-num_cells` option

For example, to insert 200 instances of the NAND2 library cell and 150 instances of the NOR2 library cell, use the following command:

```
fc_shell> add_spare_cells -cell_name spare \
    -num_cells {NAND2 200 NOR 150}
```

- (Optional) A repetitive placement window in which to add the spare cells by using the `-repetitive_window` option

For example, to add 15 instances each of the AND2 and OR2 library cells in a 20 by 20 micron window that is repeated throughout the placement area, use the following command:

```
fc_shell> add_spare_cells -cell_name spare \
    -lib_cell {AND2 OR2} -num_instances 15 \
    -repetitive_window {20 20}
```

To add 20 instances of the NAND2 and 15 instances of the NOR2 library cells in a 25 by 20 micron window that is repeated throughout the placement area, use the following command:

```
fc_shell> add_spare_cells -cell_name spare \
    -num_cells {NAND2 20 NOR 15} -repetitive_window {25 20}
```

By default, the `add_spare_cells` command distributes the spare cells evenly throughout the entire design. To distribute the spare cells within a specific

- Hierarchical block, use the `-hier_cell` option.

When you use this option, the tool distributes the spare cells in a rectangular area that encloses all of the cells that belong to the specified hierarchical block.

- Bounding box, use the `-boundary` option.
- Voltage areas, use the `-voltage_areas` option.

You can further control the placement of the spare cells as follows:

- Ignore specific types of placement blockages by using the `-ignore_blockage_types` option. By default, the `add_spare_cells` command honors all placement blockage types.
- Ignore the current cell density and place the spare cells randomly by using the `-random_distribution` option.
- Specify a percentage of spare cells to be placed based on the cell density distribution by using the `-density_aware_ratio` option. The rest of the spare cells are placed randomly throughout the design. By default, all the spare cells are placed based on the cell density distribution.

For example, if you specify a setting of `-density_aware_ratio 80`, the tool places 80 percent of the spare cells based on the cell density distribution and 20 percent randomly across the design.

- Specify the type of connection for the input pins of the spare cells by using the `-input_pin_connect_type` option. The valid values are `tie_low`, `tie_high`, or `open`.
2. Legalize the spare cells by using the `place_eco_cells -legalize_only -cells` command.

Adding Programmable Spare Cells

The Fusion Compiler tool supports programmable spare cells, also known as gate array filler cells, if they are provided by your vendor. These cells can be programmed by metal mask changes for ECO implementation, reducing mask costs and time-to-results.

To insert programmable spare cells, use the following steps:

1. Specify the programmable spare cell that a standard cell can map to by setting the same `psc_type_id` attribute setting on the corresponding library cell for the standard cell and the programmable spare cell.

The following example specifies that

- The standard cells named BUF1 and INV1 can map to the programmable spare cell named fill1x by setting the `psc_type_id` attribute to 1 for the corresponding library cells.
- The standard cells named NAND2 and NOR2 can map to the programmable spare cell named fill2x by setting the `psc_type_id` attribute to 2 for the corresponding library cells.

```
fc_shell> set_attribute [get_lib_cells fill_lib/fill1x] \
    psc_type_id 1
fc_shell> set_attribute [get_lib_cells stdcell_lib/BUF1] \
    psc_type_id 1
fc_shell> set_attribute [get_lib_cells stdcell_lib/INV1] \
    psc_type_id 1
fc_shell> set_attribute [get_lib_cells fill_lib/fill2x] \
    psc_type_id 2
fc_shell> set_attribute [get_lib_cells stdcell_lib/NAND2] \
    psc_type_id 2
fc_shell> set_attribute [get_lib_cells stdcell_lib/NOR2] \
    psc_type_id 2
```

2. Insert the programmable spare cells by using the `create_stdcell_fillers` command.

The following example inserts programmable spare cells named fill1x and fill2x:

```
fc_shell> create_stdcell_fillers \
    -lib_cells {fill_lib/fill1x fill_lib/fill2x}
```

During the ECO flow, the tool swaps an ECO cell with a programmable spare cell based on the `psc_type_id` attribute setting, cell width, and voltage area. If the tool removes an ECO cell from the design, it can reuse the corresponding programmable spare cell.

Making ECO Changes Using the eco_netlist Command

You can make ECO changes to a block by using the `eco_netlist` command.

If you are using the freeze silicon ECO flow, enable ECO changes in the freeze silicon mode by setting the `design.eco_freeze_silicon_mode` application option to `true`, before you run the `eco_netlist` command.

When you use the `eco_netlist` command, use one of the following two methods to make the ECO changes:

- Specify a golden Verilog netlist that includes the ECO changes by using the `-by_verilog_file` option
- Specify a golden block that includes the ECO changes by using the `-block` option

When you use the `-block` option, by default,

- The tool assumes the golden block is in the current design library.

To specify a different design library, use the `-golden_lib` option.

- The tool makes the ECO changes to the current design.

To make the ECO changes to a different design, use the `-working_block` option. To specify the design library that contains the design specified by the `-working_block` option, use the `-working_lib` option.

The tool compares the working design to the input Verilog netlist or the golden design and generates a change file containing netlist editing Tcl commands that implements the functional changes. You must specify the name of the output file by using the `-write_changes` option.

By default, the tool ignores the following:

- Differences in the physical-only cells

To consider the differences in the physical-only cells, use the `-compare_physical_only_cells` option.

- Timing ECO changes, such as cells that are resized or repeaters that are added or removed.

To consider the timing ECO changes, in addition to the functional ECO changes, use the `-extract_timing_eco_changes` option.

- Differences in power and ground objects.

To make the ECO changes to the working design, source the change file that the `eco_netlist` command generates, as shown in the following example:

```
fc_shell> eco_netlist -by_verilog_file eco.v \
           -compare_physical_only_cells -write_changes eco_changes.tcl
fc_shell> source eco_changes.tcl
```

Making ECO Changes Using Netlist Editing Commands

If the ECO changes are minimal, you can update the design by using netlist editing commands given. If you are using the freeze silicon ECO flow, enable ECO changes in the freeze silicon mode by setting the `design.eco_freeze_silicon_mode` application option to `true`, before you run the netlist editing commands.

For a list of netlist editing commands, see the *Common Design Objects* topic in the *Fusion Compiler Data Model User Guide*.

Using ECO Scripts for Netlist Editing

The Fusion Compiler tool supports using ECO scripts to make changes in the netlist during design planning. You can use the

- `write_split_net_eco` command to push up the branching of a physical multiple-fanout net to the top level
- `write_push_down_eco` command to push down a tree of standard cells one level
- `write_spare_ports_eco` command to create spare ports and nets on a child block

For more information, see the *Generating ECO Scripts for Netlist Editing* topic in the *Fusion Compiler Design Planning User Guide*.

Resizing Cells

You can resize a cell by using the `size_cell` command, as shown in the following example:

```
fc_shell> size_cell U21 -lib_cell AND2X4
```

If you enable the freeze silicon mode by setting the `design.eco_freeze_silicon_mode` application option to `true`, by default, the `size_cell` command checks if a compatible

spare cell is available within a distance of five times the unit site height before sizing the cell. If a compatible spare cell is not available, the tool does not size the cell.

- To control this distance, use the `-max_distance_to_spare_cell` option.
- To disable this feature, use the `-not_spare_cell_aware` option.

Reverting Changes Made During Resizing

To revert the ECO changes made with the `size_cell` command, use the `revert_cell_sizing` command. To include the sized cells that are adjacent to the specified cell in the same row, use the `-include_adjacent_sized_cells` option. When you use this option, by default, the tool also reverts the cells that abut the specified cell on either side, if they are also sized cells. However, if you use the `-adjacent_cell_distance` option, the tool recursively reverts the sized cells that are adjacent to sized cells within the specified distance in the same row.

Adding Buffers on Nets

To add buffers on nets connected to specified pins, use the `add_buffer` command.

Specify the following options when using the `add_buffer` command:

- `-new_net_names`: Specifies the names of the new nets to add. You must specify one net name per buffer when adding buffers, and two net names per inverter pair when adding inverter pairs.
- `-new_cell_names`: Specifies the names of the new cells to be added. You must specify one cell name per buffer when adding buffers, and two cell names per inverter pair when adding inverter pairs.
- `-inverter_pair`: Adds inverter pairs instead of buffer cells. If you specify this option, you must supply a library cell that has an inverting output.
- `-respect_voltage_areas`: Inserts buffers within the hierarchy of the voltage area such that the buffer is physically in the layout. This option is mutually exclusive with the `-snap` option.
- `-no_of_cells`: Specifies the number of buffer cells or inverter pairs to be inserted per net.
- `-object_list`: Specifies a list of nets, pins, or ports that must be buffered. The new buffer cells or inverter pairs are placed close to the specified pins or ports if their cells are placed.

- `-lib_cell`: Specifies the library cell to be used as buffer. In this case, the object is either a named library cell or a library cell collection. This is a required option. This option is mutually exclusive with `buffer_lib_cell`.
- `-snap`: Puts the new buffer cells next to the target cell, then snaps the buffer cells to the closest site, and flips the buffer cells when needed.

Adding Buffers on Routed Nets

To add buffers or pairs of inverters on a fully routed net, use the `add_buffer_on_route` command as described in the following topics:

- [Specifying the Net Names, Buffers Types, and Their Locations](#)
- [Controlling How Buffers are Added](#)
- [Specifying Settings for Multivoltage Designs](#)
- [Specifying Settings for the Freeze Silicon ECO Flow](#)

Specifying the Net Names, Buffers Types, and Their Locations

When you use the `add_buffer_on_route` command, you must specify the following:

- The net on which to add buffers
- The type of buffers to add and where to add them on the net by using one of the following methods:
 - [Adding Buffers in a Specified Configuration](#)
 - [Adding Buffers at Specified Locations](#)
 - [Adding Buffers at a Specified Interval](#)
 - [Adding Buffers at an Interval That is a Ratio of the Net Length](#)
 - [Adding Buffers on a Bus in a Specified Pattern](#)

Adding Buffers in a Specified Configuration

To add buffers in an exact configuration, specify the configuration of cells and their locations by using the `-user_specified_buffers` option.

With the `-user_specified_buffers` option, you can add cells only on one net at a time. You can insert different types of buffers or inverter pairs by using this option, but you cannot combine both buffers and inverter pairs.

For each cell you add, use the `{instance_name library_cell_name x y layer_name...}` format with the `-user_specified_buffers` option to specify the

- Instance name
- Library cell to use
- Coordinates of the exact location
- Layer at that location with an existing routing shape to connect to

Instead of specifying the routing layer, you can have the tool automatically detect the closest routing shape to that location by using the `-detect_layer` option.

The following example adds two cells named ECO1 and ECO2 on net n22. The ECO1 cell is of type BUF2, at location (100, 70), connecting to a routing shape on the M3 layer. The ECO2 cell is of type BUF4, at location (150, 70), also connecting to a routing shape on the M3 layer.

```
fc_shell> add_buffer_on_route net22 \
    -user_specified_buffers {ECO1 BUF2 100 70 M3 ECO2 BUF4 150 70 M3}
```

Adding Buffers at Specified Locations

To add buffers at specific locations of a net, specify

- A list of one or more library cells to select from by using the `-lib_cell` option
- The exact locations by using the `-location` option

With the `-location` option, you must specify the x- and y-coordinates and a layer with an existing routing shape at that location by using the `{x1 y1 layer1 ...}` format. Instead of specifying the routing layer, you can have the tool automatically detect the closest routing shape to that location by using the `-detect_layer` option.

With this method, you can add cells only on one net at a time.

The following example adds the BUF1 library cell on the net1 net at location (100, 200) and connects it to an existing route shape on the M4 layer.

```
fc_shell> add_buffer_on_route net1 -lib_cell BUF1 \
    -location {100 200 M4}
```

By default, the tool uses eco_cell and eco_net as the name prefix of all new cells and nets. To specify a name prefix for the new cells and nets, use the `-cell_prefix` and `-net_prefix` options, respectively.

Adding Buffers at a Specified Interval

To add buffers at a specified interval to one or more nets, specify

- A list of one or more library cells to select from by using the `-lib_cell` option
- The distance between the driver of the net and the first buffer by using the `-first_distance` option
- The interval between the buffers by using the `-repeater_distance` option

Optionally, you can scale the distance between the buffers by

- Specifying a different scaling factor for each layer using the `-scaled_by_layer` option
- Using the ratio between the default width for the layer and the actual route width as the scaling factor by specifying the `-scaled_by_width` option

The following example adds the BUF1 library cell on the nets n2 and n5 at an interval of 150 microns. The distance between the driver and the first repeater cell is 100 microns.

```
fc_shell> add_buffer_on_route {n2 n5} -lib_cell BUF1 \
    -first_distance 100 -repeater_distance 150
```

By default, the tool uses eco_cell and eco_net as the name prefix of all new cells and nets. To specify a name prefix for the new cells and nets, use the `-cell_prefix` and `-net_prefix` options, respectively.

Adding Buffers at an Interval That is a Ratio of the Net Length

To add buffers at an interval that is a ratio of the total net length, specify

- A list of one or more library cells to select from by using the `-lib_cell` option
- The distance between the driver and the first buffer as a ratio of the total net length by using the `-first_distance_length_ratio` option
- The distance between the buffers as a ratio of the total net length by using the `-repeater_distance_length_ratio` option

The following example adds the BUF1 library cell on the nets n3, n21, and n41 at an interval that is 20 percent of the total net length. The distance between the driver and the first repeater cell is 10 percent of the total net length.

```
fc_shell> add_buffer_on_route {n3 n21 n41} -lib_cell mylib/BUF1 \
    -first_distance_length_ratio 0.1 \
    -repeater_distance_length_ratio 0.2
```

By default, the tool uses eco_cell and eco_net ad the name prefix of all new cells and nets. To specify a name prefix for the new cells and nets, use the `-cell_prefix` and `-net_prefix` options, respectively.

Adding Buffers on a Bus in a Specified Pattern

To add buffers in a specific pattern to the individual nets of a bus, use the following steps:

1. Define buffer patterns for the nets of a bus by using the `create_eco_bus_buffer_pattern` command.

This command allows you to define patterns for staggering buffers on bused nets, which prevent clumping and overlapping of buffers.

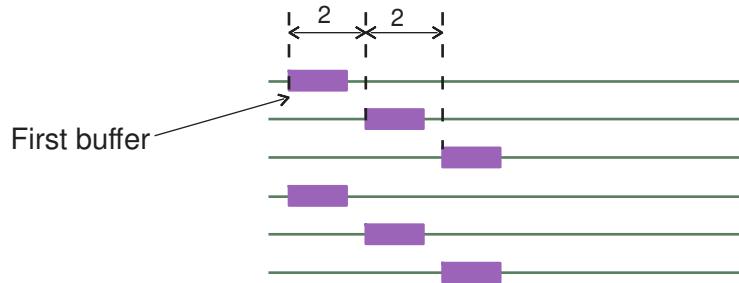
Table 70 Commands Associated With Bus Patterns

| To do this | Use this command |
|---------------------------------------|---|
| Create bus patterns | <code>create_eco_bus_buffer_pattern</code> |
| Report information about bus patterns | <code>report_eco_bus_buffer_patterns</code> |
| Get a collection of buffer patterns | <code>get_eco_bus_buffer_patterns</code> |
| Remove bus patterns | <code>remove_eco_bus_buffer_patterns</code> |

The following example creates the buffer pattern shown in [Figure 203](#) for a horizontal bus, where the first buffer is placed on the topmost net. The buffers are staggered by a distance of 2, starting from the left, and the pattern is repeated after every three buffers.

```
fc_shell> create_eco_bus_buffer_pattern -name top_left \
    -first_buffer top -measure_from left -distance 2 -repeat_after 3
```

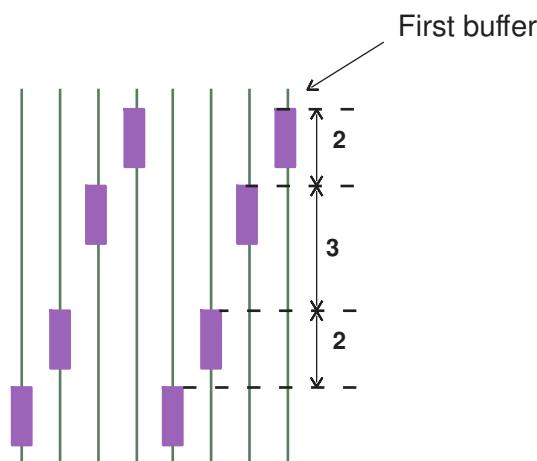
Figure 203 Buffer Patterns for a Horizontal Bus



The following example creates the buffer pattern shown in [Figure 204](#) for a vertical bus, where the first buffer is placed on the rightmost net. Starting from the top, the second buffer is staggered by a distance of 2 from the first, the third a distance of 3 from the second, and the fourth a distance 2 from the third. After that, the pattern is repeated.

```
fc_shell> create_eco_bus_buffer_pattern -name right_top \
    -first_buffer right -measure_from top \
    -user_specified_distance {2 3 2}
```

Figure 204 Buffer Patterns for a Vertical Bus



2. Add buffers on the nets of the bus by using the `add_buffer_on_route -user_specified_bus_buffers` command.

For each group of buffers that you add on the bused nets, use the `{buffer_pattern_name library_cell_name x y ...}` format with the `-user_specified_bus_buffers` option to specify the

- Name of the buffer pattern that you defined using the `create_eco_bus_buffer_pattern` command
- Library cell to use
- Coordinates of the location of the first buffer in the pattern

The following example adds a group of buffers of type BUF2 on the bus named data_A, using a buffer pattern named BP1. The first buffer of the pattern is placed at location (100, 70).

```
fc_shell> add_buffer_on_route [get_net data_A*] \
    -user_specified_bus_buffers {BP1 BUF2 100 70}
```

Controlling How Buffers are Added

By default, the `add_buffer_on_route` command

- Adds buffers over all placement blockages, soft macros, and hard macros.
To prevent buffers from being placed over specific macro cells, use the `-dont_allow_insertion_over_cell` option.
Alternatively, you can prevent the buffers from being placed over all blockages and macro cells by using the `-respect_blockages` option. When you use this option, you can specify a list of macro cells over which buffers are allowed by using the `-allow_insertion_over_cell` option.
- Adds buffers on both global routed and detail routed nets.
To add buffers only on nets that are global routed only, without assigned tracks or detail routing, use the `-only_global_routed_nets` option.
- Adds buffers at the lowest common level of hierarchy of the pins being driven by the buffers.
To add the buffers on the highest possible level of hierarchy, use the `-on_top_hierarchy` option.
- Does not add buffers on a route segment if it is necessary to create new ports because of a difference in the logical and physical topology of the net.
To add buffers on such route segments by creating new ports, use the `-punch_port` option.

Specifying Settings for Multivoltage Designs

For multivoltage designs, to

- Ensure that primary and secondary voltage areas are honored and cells are added in logical hierarchies that are physically in the corresponding voltage areas, use the `-respect_voltage_areas` option with the `add_buffer_on_route` command.
- Specify different library cells for different voltage areas, use the `-voltage_area_specific_lib_cells` option and specify the list of library cells using the `{va1 lib_cell1 va2 lib_cell2 ...}` format.

When you use this option, you must also use the `-lib_cell` option.

- Specify different single-rail and dual-rail library cells for different voltage areas, use the `-select_mv_buffers` option and specify the list of library cells by using the `{va1 single_rail_lib_cell1 dual_rail_lib_cell1 va2 single_rail_lib_cell2 dual_rail_lib_cell2 ...}` format.

When you use this option, you must also use the `-lib_cell` option. For voltage areas that you do not specify with this option, the tool uses library cells specified with the `-lib_cell` option. For the default voltage area, specify `DEFAULT_VA` as the voltage area name.

- Allow buffers on physical feedthrough nets of a voltage area, specify the voltage areas with the `-allow_physical_feedthrough_buffer` option.

This option can only be used with the `-respect_voltage_areas` or `-voltage_area_specific_lib_cells` option, and the voltage area you specify must be a primary voltage area.

After you run the `add_buffer_on_route` command with this option, update the supply net and power domain setting for the added buffers by using the `set_eco_power_intention` command.

- Add cells within gas stations of specified supply nets, use the `-respect_gas_station` option and specify the supply net.

Gas stations are areas with a constant power supply. These areas used for buffering nets that go through power domains that are powered down.

To specify the maximum allowed distance from the gas station to the added buffer, use the `-max_distance_route_to_gas_station` option.

Specifying Settings for the Freeze Silicon ECO Flow

When you set the `design.eco_freeze_silicon_mode` application option to `true` and run the `add_buffer_on_route` command in the freeze silicon mode, the tool checks if a spare

cell is available within a distance of five times the unit site height before adding a buffer. If a spare cell is not available, the tool does not add the buffer.

- To control this distance, use the `-max_distance_to_spare_cell` option.
- To disable this feature, use the `-not_spare_cell_aware` option.

Optimizing the Fanout of a Net

During the timing ECO flow, you can add buffers to a net and optimize its fanout by using the `split_fanout` command.

With this command, you must specify

- The net to optimize by specifying its name by using the `-net` option or its driver by using the `-driver` option
- The library cell to use by using the `-lib_cell` option
- The method in which to optimize the net by specifying one of the following:
 - A maximum fanout constraints for the net by using the `-max_fanout` option
 - The load pins or ports to buffer by using the `-load` option

When you use this option, you can specify the logical hierarchy to add the buffer by using the `-hierarchy` option.

In addition, you can

- Add the buffers on the existing route topology of the net by using the `-on_route` option
If you use this option with the `-load` option, you cannot specify the `-hierarchy` option.
- Split the fanout of a routed net that is not fully connected to the terminals of the driver or loads of the net by specifying a maximum distance between terminals of the driver or load pins and the incomplete route using the `-max_distance_for_incomplete_route` option.
If the tool is unable to find a routed segment or finds more than one routed segment within the specified distance from the unconnected terminal, it issues an error message.
- Avoid placement blockages and macro cells when placing buffers by using the `-respect_blockages` option

- Specify a name prefix for the new cells and nets by using `-cell_prefix` and `-net_prefix` options
By default, the tool uses `eco_cell` and `eco_net` as the name prefix for the new cells and nets.

Reporting Available Sites for Placing ECO Cells

You can report sites available for placing ECO cells by using the `report_cell_feasible_space` command. By default, the tool reports the available sites in the entire block. You can restrict the report to specific voltage areas by using the `-voltage_areas` option or to a specific rectangle or rectilinear area by using the `-boundary` option.

Identifying and Reverting Nonoptimal ECO Changes

You can identify if ECO changes made with the `size_cell`, `add_buffer`, and `add_buffer_on_route` commands can cause large displacements during ECO placement and legalization by using the `report_eco_physical_changes` command.

You can revert the nonoptimal ECO changes made by the `size_cell`, `add_buffer`, and `add_buffer_on_route` commands by using the `revert_eco_changes` command. To specify the cells for which to revert the ECO changes, use the `-cells` option.

Placing ECO Cells

For the unconstrained ECO flow, place the ECO cells by using the `place_eco_cells` command. This command derives the placement of each ECO cell based on the connectivity and the delays associated with the cell. The tool then legalizes each ECO cell to the closest unoccupied site. The existing cells are untouched to minimize the impact to their placement.

To place

- All unplaced cells, use the `-unplaced_cells` option.
- Specific cells, use the `-cells` option.

- Only the cells that have changed due to ECO operations, use the `-eco_changed_cells` option.

A cell is considered changed if the `eco_change_status` attribute of the cell has one of the following values:

- `create_cell`
- `change_link`
- `add_buffer`
- `size_cell`

When you run the `eco_netlist` command, the tool automatically sets the `eco_change_status` attribute on the changed cells. If you update the design without using the `eco_netlist` command, you can manually set the `eco_change_status` attribute by using the `set_attribute` command.

Controlling Placement When Using the `place_eco_cells` Command

The `place_eco_cells` command places and legalizes each ECO cell based on the connectivity and the delays associated with the cell.

You can control the placement of the ECO cells as follows:

- To use the channels areas between macro cells for ECO placement, use the `-channel_aware` option.

By default, the command avoids channel areas during ECO placement.

- To place ECO cells closer to fixed points, such as I/O pads or macro cells, specify a weight for the nets connected to the fixed cells by using the `-fixed_connection_net_weight` option.

By default, nets connected to fixed cells have a weight of one. To place ECO cells closer to fixed points, specify an integer value greater than one for the corresponding net.

- To prioritize specific nets during ECO placement by specifying net weights,
 - Specify net weights by using the `set_eco_placement_net_weight` command.
 - Specify that the tool honors the net weights by using the `-honor_user_net_weight` option with the `place_eco_cells` command.

To report the net weights you specify, use the `report_eco_placement_net_weight` command.

- To create virtual connections for ECO cells and use them during ECO placement, instead of the actual connections,
 1. Create virtual connections by using the `create_virtual_connection` command
 2. Use these virtual connections during ECO placement by using the `-use_virtual_connection` option with the `place_eco_cells` command.

The following example creates virtual connections for the inputs and output of the cell named ECO17 and performs ECO placement using these virtual connections:

```
fc_shell> create_virtual_connection -name VC_0 \
    -pins {U1/Y ECO17/A}
fc_shell> create_virtual_connection -name VC_1 \
    -pins {U2/Y ECO17/B}
fc_shell> create_virtual_connection -name VC_2 \
    -pins {ECO17/Y D17_OUT} -weight 3
fc_shell> place_eco_cells -cells ECO17 -use_virtual_connection
```

To remove virtual connections you have created, use the `remove_virtual_connections` command. To query virtual connections, use the `get_virtual_connections` command.

- To ignore high-fanout nets connected to ECO cells that exceed a specific threshold, use the `-max_fanout` option with the `place_eco_cells` command.
- To ignore the connections of specific pins on ECO cells, use the `-ignore_pin_connection` option with the `place_eco_cells` command.

Controlling Legalization When Using the `place_eco_cells` Command

After placing the ECO cells, the `place_eco_cells` command legalizes each ECO cell to the closest unoccupied site. The existing cells are untouched to minimize the impact to their placement.

You can control the legalization of the ECO cells by using one of the following methods:

- To not legalize the cells after ECO placement, use the `-no_legalize` option.
- To perform legalization only, use the `-legalize_only` option.

- To specify the mode in which to legalize, use the `-legalize_mode` option with one of the following settings:

- `free_site_only`

When you specify this setting, the default, the tool legalizes the ECO cells on free sites without moving preexisting cells. An ECO cells can have a large displacement, if a free site is unavailable nearby.

- `allow_move_other_cells`

When you specify this setting, the tool legalizes the ECO cells to the nearest legal location by moving preexisting cells.

- `minimum_physical_impact`

When you specify this setting, the tool first legalizes the ECO cells that can be legalized to a free site nearby without moving preexisting cells. For the ECO cells that do not have a free site nearby, the tool legalizes them by moving preexisting cells.

- To specify a displacement threshold for legalization, use the `-displacement_threshold` option.

When you use the `-displacement_threshold` option, you can also use the `-max_displacement_threshold` option to specify a second displacement threshold that is larger than the value specified with the `-displacement_threshold` option.

How the tool uses these displacement thresholds depends on the setting you use for the `-legalize_mode` option as follows:

- `free_site_only`

When you specify this setting, the tool legalizes the ECO cells that have available free sites within the specified displacement threshold.

If the displacement of an ECO cell exceeds this threshold, the tool does not legalize the cell. It creates a collection named `epl_legalizer_rejected_cells` that consists of such ECO cells that are not legalized.

You can subsequently legalize the ECO cells in the `epl_legalizer_rejected_cells` collection by using the following command:

```
fc_shell> place_eco_cells
      -legalize_mode allow_move_other_cells \
      -legalize_only -cells $epl_legalizer_rejected_cells
```

This command moves existing cells to find legal locations for the ECO cells in the collection named `epl_legalizer_rejected_cells`.

If you also specify the `-max_displacement_threshold` option with `-displacement_threshold` option, the tool also creates a collection named `epl_max_displacement_cells` that consists of ECO cells with a displacement larger than that specified by this option. The collection `epl_max_displacement_cells` is a subset of the collection `epl_legalizer_rejected_cells` and its cells are also not legalized.

You can use the `-max_displacement_threshold` option to identify ECO cells with a very large displacement, for which you want to reject the ECO changes.

- `allow_move_other_cells`

When you specify this setting, you cannot specify the `-displacement_threshold` and `-max_displacement_threshold` options.

- `minimum_physical_impact`

When you specify this setting, the tool first legalizes the ECO cells that have available free sites within the specified displacement threshold.

If the displacement of an ECO cell exceeds this threshold, the tool legalizes them by moving preexisting cells.

If you also specify the `-max_displacement_threshold` option, the tool does not legalize the cells that exceed this maximum displacement threshold specified with this option and the tool creates a collection named `epl_max_displacement_cells` that consists of ECO cells with a displacement larger than that specified by this option. You can use the `-max_displacement_threshold` option to identify ECO cells with a very large displacement, for which you want to reject the ECO changes.

- To specify the types of filler cells that can be removed during legalization, use the `-remove_filler_references` option.

When you use this option, the filler cells are removed if they do not have a fixed placement and they overlap with ECO cells. By default, the tool does not remove any filler cells when legalizing ECO cells.

- To enable Advanced Legalizer and remove filler cells that have violations, set the `place.legalize.enable_advanced_legalizer` application option to `true`.

When you specify this setting, the filler cells that have violations are removed after legalizing ECO cells. Using the `-remove_filler_references` option with this application option set to `true` results in the legalizer engine removing only those fillers that are specified by this option.

The `-min_filler_distance` option is ignored when you use the `place.legalize.enable_advanced_legalizer` application option.

Placing ECO Cells With Minimal Physical Impact (MPI)

To minimize the disturbance to the existing placement during ECO placement, use the following options with the `place_eco_cells` command:

- `-legalize_mode minimum_physical_impact`
 - `-max_displacement_threshold`
 - `-remove_filler_references`
-

Placing and Mapping ECO Cells to Spare Cells

For the freeze silicon ECO flow, place and map the ECO cells to spare cells by using the `place_freeze_silicon` command. By default, this command places and maps all ECO cells. To place and map specific ECO cells, use the `-cells` option. The following example places and maps the ECO cells named ECO1, ECO2, and ECO3:

```
fc_shell> place_freeze_silicon -cells {ECO1 ECO2 ECO3}
```

This command places each ECO cell and maps it to the nearest matching spare cell. Cells that are deleted as a result of the ECO changes are not removed from the design. They are converted to spare cells and remain in the design.

When you use the `place_freeze_silicon` command, you can

- Place the ECO cells by the target spare cells, but not map them to the spare cells, by using the `-no_spare_cell_swapping` option

This feature allows you to place the ECO cells and analyze the QoR before you map them.

- Map the spare cells that are already placed by the target spare cells by using the `-map_spare_cells_only` option.
- Generate an ECO cell to spare cell mapping file by using the `-write_map_file` command.

You can edit the mapping file, if necessary, and map the ECO cells by using the `map_freeze_silicon` command.

- Specify a minimum distance between ECO cells that are mapped to programmable filler cells by using the `-min_filler_distance` option.

Specifying Mapping Rules for Programmable Spare Cells

You can define mapping rules for programmable spare cells to control the

- Overlapping of spare cells and PG nets
- Splitting of multiple-height and merging of single-height spare cells
- Compatibility of the different types of spare cells

To do so, use the `set_programmable_spare_cell_mapping_rule` command before you run the `place_freeze_silicon` or `place_freeze_silicon` command.

To report or remove the mapping rules you specify, use the `report_programmable_spare_cell_mapping_rule` or `remove_programmable_spare_cell_mapping_rule` command.

Mapping ECO Cells to Specific Spare Cells

To map an ECO cell to a specific spare cell, use the `map_freeze_silicon` command.

Specify the ECO cell name and the corresponding spare cell name by using the `-eco_cell` and `-spare_cell` options. The following example maps the ECO cell named `ECO1` to the spare cell named `spare_1`:

```
fc_shell> map_freeze_silicon -eco_cell ECO1 -spare_cell spare_1
```

You can specify the ECO cell to spare cell mapping by using a map file and specifying this map file name by using the `-map_file` option with the `map_freeze_silicon` command. The map file has the following format:

```
ECO_cell_1    spare_cell_1
ECO_cell_2    spare_cell_2
...
...
```

Mapping ECO Cells to Logically Equivalent Spare Cells

When the tool maps an ECO cell, it looks for a spare cell with the same library cell name. If it is unable to find such a spare cell, the ECO cells remains unmapped. To identify and map such ECO cells to logically equivalent (LEQ) spare cells, use the following steps:

1. Identify the ECO cells that do not have matching spare cells by using the `check_freeze_silicon` command.
2. Find logically equivalent spare cells for these ECO cells by using the `create_freeze_silicon_leq_change_list` command.

Specify

- The names of the unmapped ECO cells by using the `-cells` option
- A name for the output file by using the `-output` option

This output file is a Tcl script with netlist editing commands. The commands replace each ECO cell with logically equivalent spare cells, which can be a

- Single spare cell with the same logical functionality, but a different library cell name
- Combination of up to two spare cells that give the same logical functionality

3. View the output Tcl file to ensure that the ECO mapping is satisfactory, and edit it if necessary.
4. Source the Tcl file in the tool to replace the ECO cells with their logical equivalents.

Updating Supply Nets for ECO Cells

To update the supply nets for ECO cells, use the `eco_update_supply_net` command. By default, it updates the supply nets for all cell added by using the following ECO commands:

- `size_cell` in freeze-silicon mode
- `add_buffer`
- `add_eco_repeater`
- `split_fanout`
- `add_buffer_on_route`

To update the supply nets for specific cells, use the `eco_update_supply_net -cells` command.

Recording the Changes Made to a Layout

You can record the changes you make to a layout and generate a Tcl file containing the changes by using the `record_layout_editing` command, as shown in the following example:

```
fc_shell> record_layout_editing -start
fc_shell> remove_shapes RECT_32_0
fc_shell> record_layout_editing -stop -output layout_changes1.tcl
```

You can make the following layout changes:

- Create or remove layout objects by using Tcl commands
- Set attributes by using the `set_attribute` command
- Move or resize objects by using the GUI

With this feature, multiple users can make ECO changes to different parts of a layout in parallel. Each user can output a Tcl file that contains the layout changes they make by using the `record_layout_editing` command, and all the Tcl files can be applied to the original layout.

Performing Prerequisite Check for Group Repeater Insertion and Placement

Use the `-check_prerequisites` option of the `place_group_repeating` or `add_group_repeating` command to perform a prerequisite check for a design. This prerequisite check detects the design errors at the beginning of the command flow.

- For the `place_group_repeating` command, use the `-check_prerequisites` option with the following options to provide necessary information for the check:

```
place_group_repeating
    -cells | -repeater_groups
    [-lib_cell_input]
    [-lib_cell_output]
    -check_prerequisites
```

- For the `add_group_repeating` command, use the `-check_prerequisites` option with the following options to provide necessary information for the check:

```
add_group_repeating
    -nets | -bundles
    [-lib_cell_input]
    [-lib_cell_output]
    -check_prerequisites
```

- The prerequisite check validates the following in a design and search region:
 - Site rows are present in the design.
 - Each net has only one driver and can have multiple loads.
 - Locations of pins or ports are assigned.
 - For pins on standard cells, the standard cells are placed.
 - Supernets must be routed.

- Route ends must not be more than 5 μm away from the driver, load pins, or ports in the search region.
- The routes are defined for a driver or load in the search region.
- Load is connected to a driver.
- When the end of a route is open, no other route of the same net is present in the vicinity of the pin or port.
- Routes must be assigned to the net of the supernet driver pin.
- Vias must be created for routes.
- Repeaters have a physical library cell.

The following example shows the usage of `-check_prerequisites` option using the `add_group_repeating` command:

```
fc_shell> add_group_repeating -bundles bundle_wo_load \
    -lib_cell ref_lib1/libCell1 -lib_cell_input A -lib_cell_output X \
    -repeater_distance 10 -check_prerequisite
Error: The net net_wo_load does not have loads. (ECO-329)
Error: The cell driver1 of the pin driver1/X is not placed. (ECO-331)
Error: 0
    Use error_info for more info. (CMD-013)
```

Adding a Group of Repeaters

To add a group of repeaters, use the `add_group_repeating` command as described in the following sections:

- [Defining a group of repeaters](#)
- [Grouping a list of repeaters](#)
- [Setting Constraints for a Group of Repeaters](#)
- [Reporting the Constraints Assigned to a Group of a Repeaters](#)
- [Removing Constraints for a Group of Repeaters](#)
- [Placing Group Repeaters Before Routing](#)
- [Performing On Route Placement of Repeaters](#)
- [Placing Group Repeaters For Multibit Registers](#)
- [Specifying Locations for Repeater Groups](#)
- [Allowing Repeater Groups Over Macros](#)

- Specifying Cut Space and Cut Distance for Repeater Groups
 - Specifying Horizontal and Vertical Spacing for Repeater Groups
 - Specifying Library Cells as Repeaters
 - Avoiding Overlapping Repeaters With Existing Tap Cells
 - Avoiding Crosstalk During Group Repeater Insertion
 - Previewing Repeater Groups
 - Unplacing the Repeaters
 - Removing Repeater Groups
-

Defining a group of repeaters

To define a group of repeaters for cutline support, use the `set_repeater_group` or `create_repeater_groups` command.

The following examples show the command usage.

```
set_repeater_group -group_id group_id \
    [-check_connection] \
    [-cells cell_list] \
    [-cutline { {{x1} {x2} {y1} {y2}} | {{x} {y1} {x} {y2}} } ] \
    [-driver_group_id group_id] \
    [-path_drivers pin_port_list] \
    [-path_loads pin_port_list] \
    [-override]
    [-lib_cell_input string]
    [-lib_cell_output string]
    [-clear]
```

Grouping a list of repeaters

To automatically group a list of repeaters or repeaters of supernets, use the `create_repeater_groups` command.

This command takes a list of repeaters, a list of supernets, or bundles of supernets and returns a list of repeater groups created by the `set_repeater_group` command.

The command supports both single load and multiple loads supernets.

```
create_repeater_groups -supernets supernet_list | \
    -supernet_bundles supernet_bundle_list | \
    -cells cell_list \
    -lib_cells lib_cell_list \
    [-lib_cell_input lib_pin] \
```

```
[-lib_cell_output lib_pin] \
[-group_pin_spacing spacing] \
[-ignore_pin_layers]
```

Setting Constraints for a Group of Repeaters

To define the constraints for a group of repeaters, use the `set_repeater_group_constraints` command.

The `set_repeater_group_constraints` command has a lower priority than the corresponding command options in the `add_group_repeating` and `place_group_repeating` commands.

```
set_repeater_group_constraints -type type_list \
    [-horizontal_repeater_spacing spacing] \
    [-vertical_repeater_spacing spacing] \
    [-layer_cutting_distance layer_scale_list]
```

Reporting the Constraints Assigned to a Group of a Repeaters

To report the constraints that were assigned to a group of repeaters using the `place_group_repeating`, `add_group_repeating`, or `set_repeater_group_constraints` commands, use the `report_repeater_group_constraints` command. For example:

```
fc_shell> report_repeater_group_constraints
```

Removing Constraints for a Group of Repeaters

To remove the constraints that were assigned to a group of repeaters, use `remove_repeater_group_constraints` command. For example:

```
fc_shell> remove_repeater_group_constraints -type type_list
```

Placing Group Repeaters Before Routing

Use the `preplace_group_repeating` command to place the group repeaters before the routing. This command places the group repeaters either in an ascending or descending order, default is ascending. The ascending order means that the command places the cells in a group from the lowest or left-most available track to the highest or right-most track.

Note:

A FAA-Base-Beta license is required to run the `preplace_group_repeating` command.

You must specify either the cell collections or group IDs of the group repeaters.

The following example places group repeaters when the cell collections are specified.

```
fc_shell> preplace_group_repeaters      \
    -repeater_group_locations { { $cellGroup1, M2, {720 363.5}, \
        north, descending} {$cellGroup2, M2, {920 363.5}, east, ascending} \
        {$cellGroup3, M2, {1230 328.5}, east, ascending} \
        {$cellGroup4, M2, {1430 328.5}}, east}
```

The following example places group repeaters when the group ID is specified.

```
fc_shell> preplace_group_repeating \
    -repeater_group_locations { { 1,M2, {720 363.5}, south, ascending} \
    {2, M2, {920 363.5}, west, ascending} {3, M2, {1230 328.5}, \
    south, ascending} {4, M2, {1430 328.5}, west, descending}}
```

Performing On Route Placement of Repeaters

To perform on route placement of interconnect repeaters, use the `place_group_repeating` command.

```
place_group_repeating \
    -cells cell_list | -repeater_groups group_id_list \
    [-lib_cell_input pin_name] \
    [-lib_cell_output pin_name] \
    [-max_distance_for_incomplete_route distance] \
    [-first_distance distance] \
    [-last_distance distance] \
    [-blockage_aware] \
    [-horizontal_repeater_spacing spacing] \
    [-vertical_repeater_spacing spacing] \
    [-layer_cutting_distance layer_scale_list]
    [-verbose]
```

Placing Group Repeaters For Multibit Registers

To place group repeaters for multibit registers, use the `create_repeater_groups` command with the `-multi_bit` option with the following command options:

```
create_repeater_groups
    -cells | -supernet_bundles | -supernets
    -multi_bit
    -pin_mapping {{lib_cell1 {inp1 outp1} {inp2 outp2}...}}
```

The `-multi_bit` option determines the order of a multibit repeater based on the path driver and creates the repeater groups for the multibit repeaters. In a design, when a net has no route defined, this option places the group repeaters based on their location.

To verify the support for multibit register placement, use the following command:

```
fc_shell> preplace_group_repeater
      -repeater_group_locations repeater_group_location_list
```

The format for the `repeater_group_location_list` argument is

```
{ {cell_collection, layer, location, potential routing direction, placing
order} ...}
```

or

```
{ {group_id, layer, location, potential routing direction, placing
order} ...}.
```

Specifying Locations for Repeater Groups

Use one of the following mutually exclusive options to control repeater group location:

- `-repeater_distance` specifies the distance between repeater groups. This option supports the following additional options:
 - `-first_distance` specifies the distance between the driver and the center of the first repeater group.
 - `-min_distance_repeater_to_load` specifies the minimum distance between the last repeater group and the load. The default is one half of the value specified for the `-repeater_distance` option.
 - `-tolerance` specifies the maximum delta distance over the value specified for `-repeater_distance`. The default is 5 μm .
- `-relative_distance` specifies the distance between the driver and the first repeater group, the distance between the first and second repeater groups, the distance between the second and third repeater group, and so on.
- `-location` specifies the location of the center of each repeater group.
- `-number_of_repeater_groups` specifies the total number of repeater groups to be inserted evenly along the route.
- `-cutlines` specifies the location pairs that define the cutlines. Cutlines must be perpendicular to the routes and the repeater groups are centered around the cutlines.

Allowing Repeater Groups Over Macros

Use the `-allow_insertion_over_block` option to allow repeater groups to be added on top of soft or hard macros.

Note:

If the repeater cell is an inverter, you are responsible for ensuring logical correctness.

Specifying Cut Space and Cut Distance for Repeater Groups

Use one of the following mutually exclusive options to control cutting for repeater groups:

- `-layer_cutting_spacing` specifies the cut space of the routes to the driver pin. The cut starts from the driver side.
- `-layer_cutting_distance` specifies the cut distance from the repeater center to the input and output routes.

Specifying Horizontal and Vertical Spacing for Repeater Groups

Use one of the following mutually exclusive options to control spacing for repeater groups:

- `-horizontal_repeater_spacing` specifies the horizontal spacing between repeaters by cell site width.
- `-vertical_repeater_spacing` specifies the vertical spacing between repeaters by site row height.

Specifying Library Cells as Repeaters

Use the `-lib_cell` option to specify a library cell as a repeater. This option supports the following additional options:

- `-lib_cell_input` specifies the input pin of the library cell to connect, if it has multiple inputs.
- `-lib_cell_output` specifies the output pin of the library cell to connect, if it has multiple outputs.

Avoiding Overlapping Repeaters With Existing Tap Cells

Use the `-honor_special_cell` option to avoid overlapping with existing tap cells when inserting repeaters.

Avoiding Crosstalk During Group Repeater Insertion

The crosstalk management is only supported in the `create_group_repeating_guidance` and `add_group_repeating` commands flow.

The flow to avoid crosstalk during group repeater insertion is as follows:

1. Use the following command to query interleaving groups:

```
create_group_repeaters_guidance -net
    -number_of_interleaving_groups number
```

By default, the *number* argument is set to 1.

For example, if the value of the *number* is 2, one net group is separated into two interleaving net groups.

2. Use the following command to manage crosstalk during group repeater insertion:

```
add_group_releaters
    -nets | -bundles
    -first_distance_of_net_groups
        { {group_id1 $first_dist1} {group_id2
        $first_dist2} ...}
    -repeater_distance $dist
```

or

```
add_group_releaters
    -nets | -bundles
    -cutlines_of_net_groups
        { {group_id1 $cutline_list1} {group_id2 $cutline_list2} ...}
```

- If the option `-first_distance_of_net_groups` is specified, the command groups the nets based on the existing group IDs on nets and inserts repeaters based on the specified first distance for the various net groups. The repeater distance is same for all net groups.
- If the option `-cutlines_of_net_groups` is specified, the command groups the nets based on the existing group IDs of the nets and inserts repeaters based on the cutlines for the various net groups.

Previewing Repeater Groups

Use the `-preview` option to preview repeater group locations before insertion. The repeaters are represented by annotation objects at the same locations where they are to be inserted, with the same width and height as a repeater lib cell, and with the group ID `eco_preview_id`.

Note:

Before previewing the repeater groups again, you must first remove the annotation objects using either the `gui_remove_annotations` or `gui_remove_all_annotations` command.

Unplacing the Repeaters

To unplace the previous placement of repeaters, use the `unplace_group_repeating` command.

```
unplace_group_repeating \
    -cells cell_list | -repeater_groups group_id_list \
    [-lib_cell_input pin_name] \
    [-lib_cell_output pin_name]
```

Removing Repeater Groups

To remove a group of repeaters, use the `remove_eco_repeater` command. The nets are restored to their original names and routes shapes.

Querying Group Repeater

The group repeater query commands provide improved ease of use for interconnect repeater while planning. The group repeater query

- Provides capability to query group repeater path info
- Supports cross-session query operation
- Supports query operation before group repeater placement

To query the group repeater during planning, you must be familiar with the following key information

- Cells of a repeater group
- Cutline of a repeater group
- Group virtual connection

If a group has either driver group or path drivers, then it is categorized as group virtual connection. Cell connection between groups depend on cell order in a group.

The following topics describe the steps in this flow

- [Performing Auto Grouping Flow](#)
- [Performing Manual Grouping Flow](#)
- [Cell Input Mode](#)

Performing Auto Grouping Flow

Auto repeater group creation allows you to derive virtual connections for register paths. For repeater group input mode, `place_group_repeaters` applies cutline based placement. You must specify cutline for each group. Repeaters are placed based on the intersection point between cutline and route.

Build connectivity based on real connections for target cells.

1. To group a list of repeaters or repeaters of supernets, use the `create_repeater_groups` command. Also, specify the constraint option.

The command automatically performs cell grouping and groups virtual connection with the following attributes:

- `group_repeater_driver`
- `group_repeater_loads`
- `pin_pair`
- `group_id`

2. Verify the path with the `get_attribute`, `report_repeater_group`, and `get_repeater_path_info` commands. The `get_attribute` command reports the attributes of the `group_repeater_driver` or `group_repeater_loads` command. The command `report_repeater_groups` reports group information for specified groups or all groups, and reports repeater paths for all groups if `-verbose` is on. The `get_repeater_paths_info` command returns cells on repeater paths that include input cells.
3. Specify the cutline for a group manually with the `set_repeater_group -group_id-cutline` command.
4. Place interconnect repeaters on route with cutline for input repeater groups.

Performing Manual Grouping Flow

Auto grouping is based on the `create_repeater_groups` command. Use manual grouping in case the auto grouping command result does not meet the requirement.

Check your target cell connection before placement. Perform the following steps for manual grouping:

1. Group your cells by -cells, cell order intends the cell connection.
2. Define group virtual connection by -driver_group_id or -path_drivers.
3. Specify the cutline by -cutline.

1. To define register path connections (virtual connection) for each set of repeater groups, use the `set_repeater_group` command.
 - To check cell connection before placement, use the `set_repeater_group -check_connection` command. If the cell connection is not placed as expected, repeat a reset repeater group for desired cell connection. It also adds new options to record pin pair for the `-check_connection` option. The `pin_pair` option is used to find cell real connection based on specified lib cell pin name.
2. Verify the path with the `get_attribute`, `report_repeater_group`, and `get_repeater_path_info` commands. The `get_attribute` command reports the attributes of the `group_repeater_driver` or `group_repeater_loads` commands. The `report_repeater_groups` command reports group information for specified groups or all groups, and reports repeater paths for all groups if `-verbose` is on. The command `get_repeater_paths_info` returns cells on repeater paths that include input cells.
3. Check and report invalid paths based on real connection with the `set_repeater_group -check connection` command.
4. Place interconnect repeaters on route with cutline for input repeater groups.

Cell Input Mode

Cell input mode enables to set distance based on route placement. By default, repeaters are placed based on even distance on route for one repeater path. This mode allows you to set the `-first_distance` or `-last_distance` option to control the distance at driver side or load side.

1. Use the `place_group_repeaters -cells -blockage_aware` command, to internally enable auto grouping . To internally enable auto grouping use the `create_repeater_groups -cells` command and derive cutline based on distance. Then, define repeater group by using the `set_repeater_group -group_id -cutline` command,
After placing group repeaters, you can query repeater groups using the `report_repeater_groups` command and repeater paths info using the `get_repeater_paths_info` command.
2. To establish cell connection based on real connection and apply distance based placement, use the `place_group_repeaters -cells` command. No repeater group is created.

Swapping Variant Cell

The `add_group_repeater` and `place_group_repeater` commands support variant cell in group repeater insertion and placement.

The following topics describe the steps in this flow

- [Setting Constraints of Variant Cell](#)
 - [Setting Application Option](#)
 - [Grouping Variant Cell](#)
 - [Running and Placing Group Repeaters](#)
 - [Troubleshooting](#)
-

Setting Constraints of Variant Cell

To set the constraints of variant cell:

1. Use the `set_repeater_group_constraints -type variant_cell_swapping` command to set the variant for cell swapping.
 2. Use the `report_repeater_group_constraints` command to view the report of the repeater group constraint settings, after setting the variant for cell swapping. The report provides the
 - Track pattern of the variant cell group
 - Variant site ids
 - Honor cell group not sharing timing
 - First track alignment
 3. Use the `remove_repeater_group_constraints` command to remove the constraints and reset the settings based on the report.
-

Setting Application Option

To set application option before using the `add_group_repeaters` and `place_group_repeaters` commands, use the `set_app_option -name eco.placement.variant_cell_swapping -value true` command.

Based on the requirement, you can plan the variant cell flow. The rules of variant reference swapping differ in various variant cell flows. By default, the variant cell support in the `eco_change_legal_reference` command detects the variant mode and settings automatically.

The `eco_change_legal_reference` command checks whether the current reference of the cell is legal or illegal for the placed location. If it is illegal, the command attempts to find a legal reference from its variant cell group defined to swap with the illegal one.

To use the variant cell features directly for other cells, use the `eco_change_legal_reference` command.

For complicated scenarios, you can switch the variant mode and tune the settings manually. Following are the new application options for variant cell support:

- To set variant mode for different customers and different design flows, use the `set_app_options -name eco.placement.variant_cell_group_mode -value` command.
- To honor the variant cell groups, which are not sharing timing view, use the `set_app_options -name eco.placement.honor_cell_group_not_sharing_timing -value` command.

Grouping Variant Cell

In the same variant cell group, the library cell variants differ from the master-variant only in mask color (or small geometry), while the timing data is same for these variants.

The variant cell feature supports the following

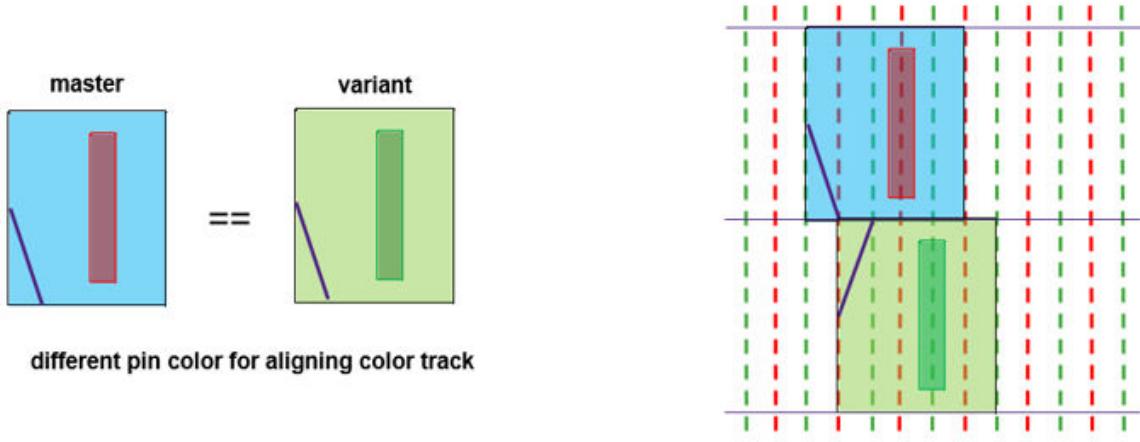
- `track_pattern` mode
- `siteld_based` mode

track_pattern mode

To specify the first metal layer for track alignment checking in `track_pattern` mode, use the `set_app_options -name eco.placement.first_track_alignment_layer -value` name command.

[Figure 205](#) shows the cell group with two variants.

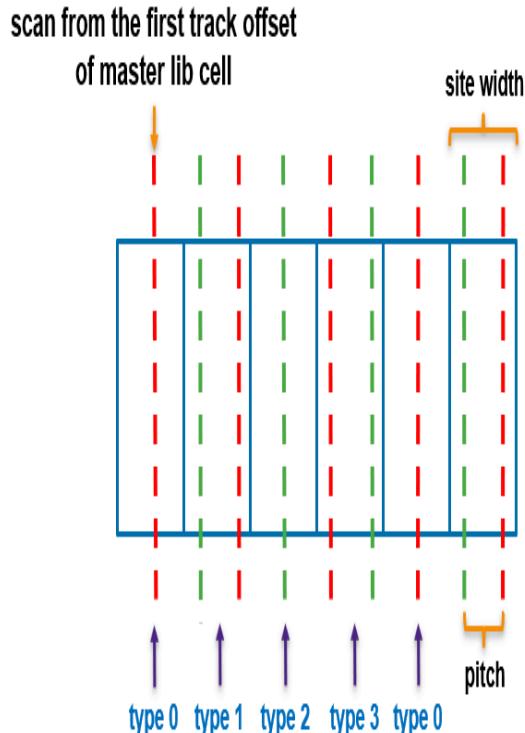
Figure 205 Cell Group With Two Variants



Based on the first track offset of library cells and the pitch value, you can define and derive the variant library cell types, which start from 0.

Figure 206 shows variant type id definition.

Figure 206 Variant Type id Definition



Example (N5 design):

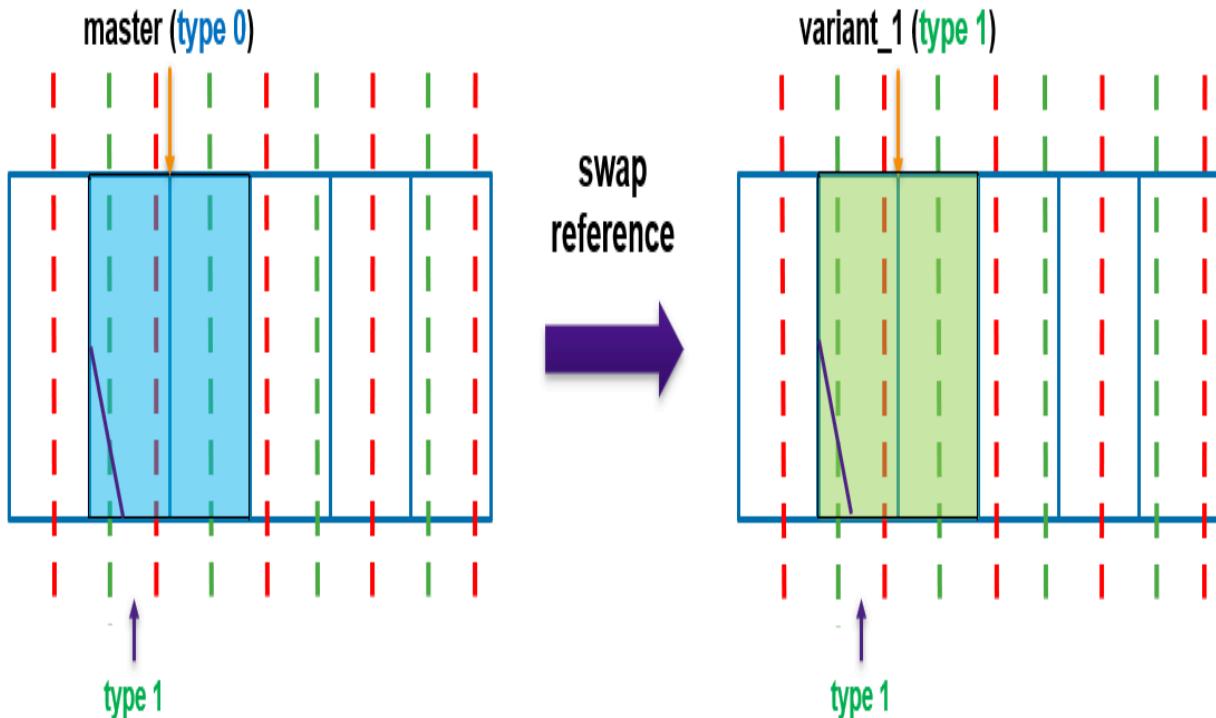
| Type | Mask Color | Track Offset | Library Cell |
|------|------------|----------------|--------------|
| 0 | mask_one | 1/2 site width | Master |
| 1 | mask_two | 1/6 site width | Variant_1 |
| 2 | mask_two | 1/2 site width | Variant_2 |
| 3 | mask_one | 1/6 site width | Variant_3 |

The default pin color alignment layer is M1. User can change it by setting an app option.

You can detect the variant type id for target cell current location. You can also check if the current reference is legal or illegal. If it is not legal, select the legal reference from cell group and swap it.

[Figure 207](#) shows the check and swap with legal reference for target cell.

Figure 207 Check and Swap With Legal Reference for Target Cell.



sitId_based mode

The variant mapping and flipped mapping information are defined in the cell group. Following is an example of the `report_cell_groups` command.

[Figure 208](#) shows the cell group definition for mapping id and flipped mapping id.

Figure 208 Cell Group Definition for Mapping id and Flipped Mapping id

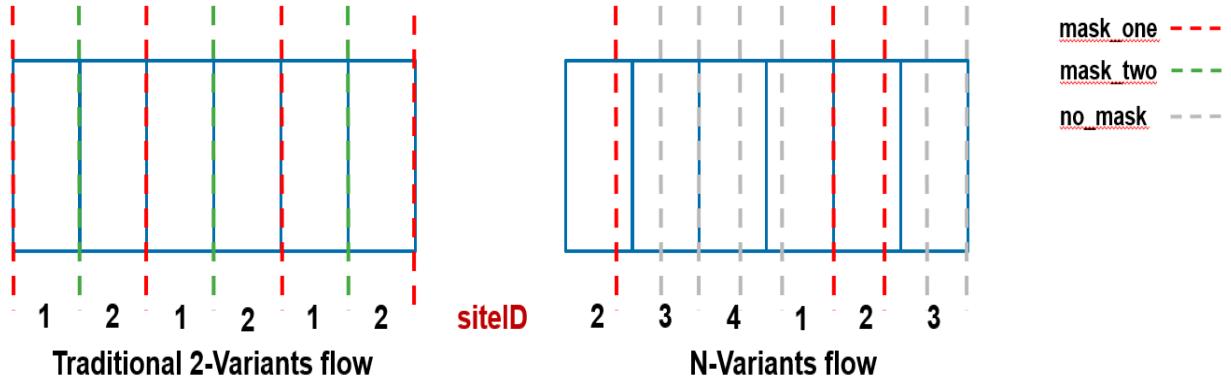
```
icc2_lm_shell> report_cell_groups
*****
Report : report_cell_groups
Lib    : variant
Version: N-2017.09-SP4
Date   : Wed Feb 7 22:25:05 2018
*****
```

| Cell Group | Group Contents | Mapping | Flipped Mapping | Share Timing View |
|------------|-----------------------------|---------|-----------------|-------------------|
| A (L) | A A_M1_1 A_M1_1 M2_1 A_M2_1 | 1 2 3 4 | 1 2 3 4 | true |

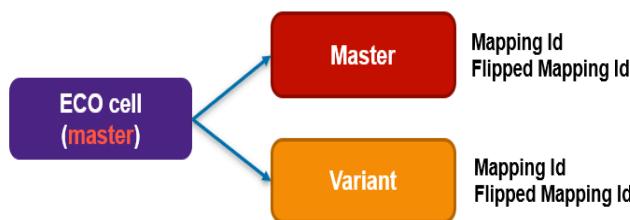
There are two sub flows for siteld_based mode: 2-Variants and N-Variant.

Figure 209 shows the 2-Variants and N-Variants sub flows in siteld_based mode

**Figure 209 2-Variants and N-Variants Sub Flows in `siteId_based Mode`
design with various tracks and pitches**



Mapping Id Define (get from NDM Cell Group)



- **Traditional 2-Variants flow**

- It only have maximum two master/variant in each cell group.

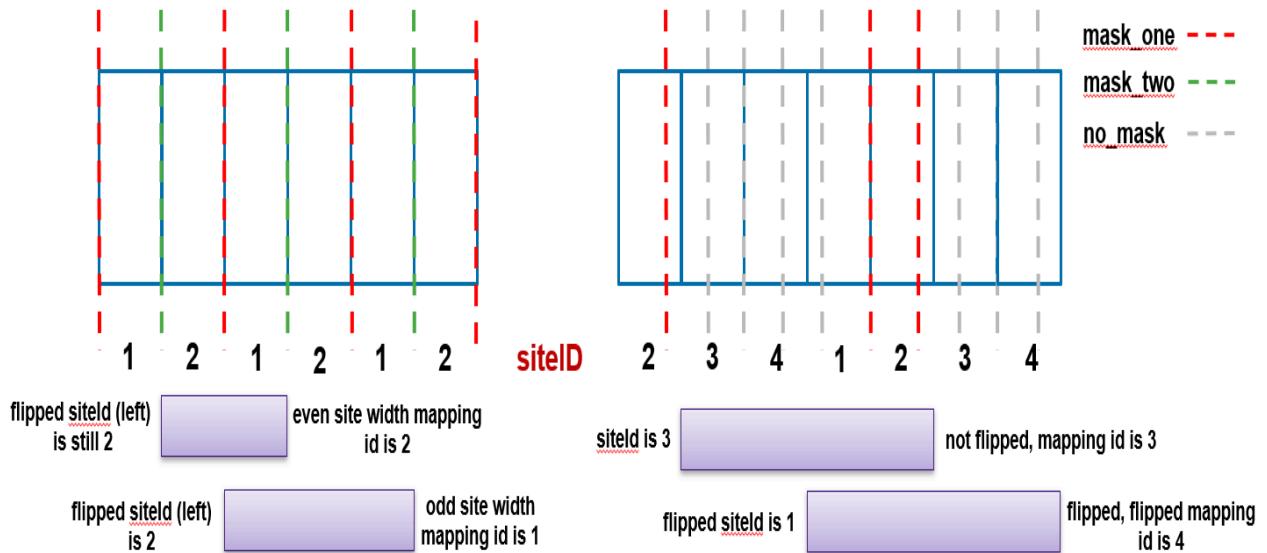
- **N-Variants flow**

- It can have N-variants in each cell group.

Each variant reference has both legal mapping id and flipped mapping id.

[Figure 210](#) shows the variant mapping and flipped mapping id.

Figure 210 Variant Mapping and Flipped Mapping id



Traditional 2-Variants flow

- Only mapping id is set on [ndmCellGroup](#).
- The flipped mapping id is derived from mapping id and cell width.

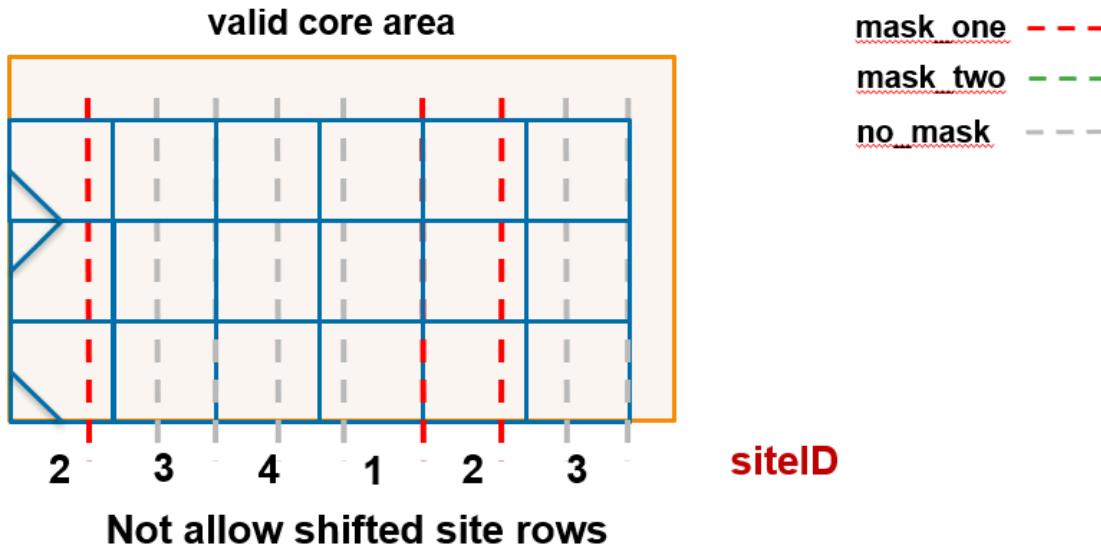
For variant sitelD, you can get the *cycle* and *offset* from site rows first, then use the cell placed location to do the calculation.

Figure 211 shows the calculation for variant sitelD

N-Variants flow

- Both mapping id and flipped mapping id are set on [ndmCellGroup](#).
- But the flipped mapping id means the right boundary site id after flipping. So we need convert it to be left boundary site id for easy of use.

Figure 211 Calculation for Variant sitelD



row cycle is 4, row offset is 2

- First **sitelD** of site row is 2.
- The other location **sitelD** can be derived by calculating the **site index** first and then convert it to be a value between 1 ~ 4.

Running and Placing Group Repeaters

Use the `add_group_repeating` and `place_group_repeating` commands to display the summary report for variant cell swapping in command cell collections.

Troubleshooting

Table 71 lists the messages of group repeater placement and insertion for the problems encountered during defining the group repeaters.

Table 71 Group Repeater Placement and Insertion Troubleshooting

| Command | Error Code | Message | Troubleshooting Tips |
|------------------------------------|------------|--|--|
| <code>place_group_repeating</code> | ECOUI-136 | Error: The repeater group (%d) is invalid. | Check if the input group id exists by running the <code>report_repeater_groups</code> command. |

Table 71 Group Repeater Placement and Insertion Troubleshooting (Continued)

| Command | Error Code | Message | Troubleshooting Tips |
|-----------------------|------------|---|--|
| place_group_repeating | ECO-220 | Warning: Failed to connect to route for %d object: {%-s}. | Check the pin shape of object and route shapes. The object as driver or load can be port, macro pin, and placed cell. |
| place_group_repeating | ECO-224 | Error: The option -repeater_groups cannot be used when the app option eco.placement.eco_enable_pipeline_register_placer is false. | Check the pin shape of object and route shapes. The object as driver or load can be port, macro pin, and placed cell. |
| place_group_repeating | ECO-227 | Warning: Failed to place %d repeaters for %d drivers in collection (%s), and failed cells are in collection (%s). | Check the failed drivers by running the query_objects, get_ports, get_pins and get_cells commands. The driver can be port, macro pin, and placed cell. |
| place_group_repeating | ECO-228 | Warning: Some repeaters from path driver (%s) have same repeater group id. The cells belonging to the path are {%-s}. | Check if the group id of cells are on the same path. The command skips to place some repeaters. The group id of cells on one path should be different. |
| place_group_repeating | ECO-229 | Warning: Failed to find intersection of cutline %s and route for cell {%-s}. | |
| place_group_repeating | ECO-231 | Error: The repeater group (%d) has no cutline or invalid cutline. | Check the right cutline for this group. Set right cutline for this group using the set_repeater_group command. |

Table 71 Group Repeater Placement and Insertion Troubleshooting (Continued)

| Command | Error Code | Message | Troubleshooting Tips |
|--|------------|---|--|
| place_group_repeaters | ECO-232 | Error: Cutline {{%s} %s}} is invalid. | Check the format of the cutline. The valid format of cutline is {{x1 y} {x2 y}} or {{x y1} {x y2}}. |
| place_group_repeaters
unplace_group_repeaters | ECO-233 | Error: The cells in collection (%s) are invalid since they have no target input pin name (%s) and output pin name (%s). | Check the input and output pin name. The input pin name and output pin name can be specified by using the -lib_cell_input and -lib_cell_output options if needed, or filter out invalid cells from input. The default value for -lib_cell_input is D. The default value for -lib_cell_output is Q. |
| place_group_repeaters | ECO-234 | Warning: There is no repeater to be placed for the path driver (%s). | |
| place_group_repeaters | ECO-235 | Warning: There is no route on driver net (%s). | |
| place_group_repeaters | ECO-236 | Warning: There is no target load for path driver (%s). | |
| place_group_repeaters | ECO-237 | Warning: Some cell to be placed has no load for path driver (%s). | |
| place_group_repeaters | ECO-239 | Info:%s. | Check the place and connection mode. Show the place mode (cutline/distance) and connection mode (virtual connection/real connection) on this command. |

Table 71 Group Repeater Placement and Insertion Troubleshooting (Continued)

| Command | Error Code | Message | Troubleshooting Tips |
|--|------------|--|---|
| place_group_repeating
unplace_group_repeating | ECO-240 | Error: The option -cells cannot be used when the app option eco.placement.eco_enable_virtual_connection is true. | Check the command for virtual connection. Specify the -repeater_groups command if you need to use virtual connection. |
| place_group_repeating | ECO-241 | Warning: The distance based placement does not support multi-fanout path driver(%). | Check the cutline based placement for the multi-fanout path driver. |
| place_group_repeating
unplace_group_repeating | ECO-242 | Warning: The cell (%) has no driver or load (number of driver: %d, number of load: %d). | |
| place_group_repeating | ECO-243 | Error: The cells in collection (%) are invalid since they have no driver or load. | Check for invalid cells. Ensure the cell that is placed has both driver and load. Else, remove such cells from input. |
| place_group_repeating | ECO-244 | Warning: The %s %s has no shape. | Check if the top port or macro pin has shape. |
| place_group_repeating | ECO-246 | Warning: Skip the path driver (%) since the driver or load for the path has no shape. | |
| place_group_repeating | ECO-251 | Info: Place group repeaters successfully. | Check if all the input cells are placed successfully. |
| place_group_repeating | ECO-252 | Warning: Failed to place repeaters for repeater path which driver is %. | |
| place_group_repeating | ECO-253 | Warning: The cutline information %s for cell %s is not on valid route. | |

Table 71 Group Repeater Placement and Insertion Troubleshooting (Continued)

| Command | Error Code | Message | Troubleshooting Tips |
|--|------------|--|---|
| place_group_repeaters | ECO-254 | Warning: Failed to cut route for cell %s. | Check if the command failed to cut route for cell. |
| place_group_repeaters | ECO-255 | Warning: Gap %s between driver and nearest route is larger than placing distance %s | Check if the route to the nearest pin distance is larger than placing distance for driver or load. |
| place_group_repeaters | ECO-256 | Warning: The first distance (%s) specified by command option is shorter than the gap (%s) between pin and route for driver (%s). (warning)
The last distance (%s) specified by command option is shorter than the gap (%s) between pin and route for load (%s). | Check if the -first_distance or -last_distance option specified is shorter than the gap between pin and route for driver or load. |
| place_group_repeaters | ECO-257 | Warning: Cannot find cutline information for cell %s. | |
| place_group_repeaters | ECO-258 | Info: The app option about virtual connection is turned on. Please make sure the cells in repeater group keep in order. | Check if the eco.placement.eco_enable_virtual_connection option is set to true and the place_group_repeatingers -repeater_groups option is specified. |
| place_group_repeaters
unplace_group_repeaters | ECO-259 | Warning: The number of cells from repeater group (%d) is not consistent with the number of drivers from driver group (%s). | |
| place_group_repeaters
unplace_group_repeaters | ECO-260 | Warning: The number of cells from repeater group (%d) is not consistent with the number of loads from load group (%s). | |

Table 71 Group Repeater Placement and Insertion Troubleshooting (Continued)

| Command | Error Code | Message | Troubleshooting Tips |
|--|------------|---|---|
| place_group_repeating
unplace_group_repeating | ECO-261 | Warning: The number of path loads is less than the number of cells in group (%d), so skip these path loads. | |
| place_group_repeating
unplace_group_repeating | ECO-262 | Warning: The number of path loads is not an integral multiple of the number of cells in group (%d), so use path loads in order. | |
| place_group_repeating | ECO-265 | Warning: The pin shape {%-s} of %s (%s) is projected to route %-s. | Check if the route is far away from driver or loads. The tool projects the closest pin shape to the route. |
| unplace_group_repeating | ECO-266 | Warning: The cells in collection %-s which are not placed before cannot be unplaced by unplace_group_repeating. | Check if the cells are not placed. The cells that are not placed cannot be unplaced. Also, only the cells added by the add_group_repeating command and placed by the placed_group_repeating command can be unplaced by the unplace_group_repeating command. |
| unplace_group_repeating | ECO-267 | Error: Input and output routes are not merged after unplacing repeater %-s since the routes are not aligned or not at the same layer. | Check if the routes are merged after unplacing repeaters. This error occurs when the routes are not aligned or not at the same layer since they are not merged. |

Table 71 Group Repeater Placement and Insertion Troubleshooting (Continued)

| Command | Error Code | Message | Troubleshooting Tips |
|--|------------|--|--|
| place_group_repeating | ECO-268 | Error: The cells in collection (%s) are invalid since they are placed before. Apply the command unplace_group_repeating to unplace them. | Check if the cells were placed before and added by the add_group_repeating command or placed by place_group_repeating command. |
| add_group_repeating
place_group_repeating | ECO-269 | Error: Unable to check the current reference of cell '%s': %s. | Check the reported reason and other printed messages for more details. Use the appropriate command to check if the current reference of cell is legal or illegal for the reported issue. |
| add_group_repeating
place_group_repeating | ECO-270 | Error: Unable to find swappable variant reference for cell '%s': %s. | Check if the current reference of cell is illegal for the reported issue. |
| add_group_repeating
place_group_repeating | ECO-271 | Error: Candidate reference '%s' and reference (%s) of cell (%s) are not in the same variant cell group. | Check errors for compatible issues between the candidate and current reference of cell. Error also occurs if they do not belong to the same variant cell group. |
| unplace_group_repeating | ECO-272 | Error: The mixed input cells are not supported, which includes the cells added by add_group_repeating and the cells placed by place_group_repeating. | |
| unplace_group_repeating | ECO-273 | Error: There are no placed cells to be unplaced. Please double check the cell status. | Check if there are placed cells that are not qualified. |

Table 71 Group Repeater Placement and Insertion Troubleshooting (Continued)

| Command | Error Code | Message | Troubleshooting Tips |
|------------------------|------------|--|---|
| | ECO-274 | Warning: You are deleting cell %s with eco_repeater_group_id %d. | Check if you are trying to delete a cell belonging to a repeater group. |
| place_group_repeaters | ECO-275 | Error: Failed to traverse cutlines. The nets may have too many blockages. | Check the location to insert cell groups. Remove blockages or find cutlines and use the -repeater_group option. |
| add_group_repeaters | ECO-276 | Warning: Unable to find free area for repeater on net (%s) at location (%s) within range {%%s}. | Check if checkerboard snapping service can find free space for a repeater |
| set_repeater_group | ECO-277 | Warning: You are trying to override cells of repeater group %d. Use -override if you have to override cells. | Check if you are trying to override cells without using the -override option. |
| create_repeater_groups | ECO-278 | Error: Get 0 transparent cells from all supernets. | Check if the input supernets have no transparent cells to group. |
| create_repeater_groups | ECO-279 | Error: %s has multiple(%lu) inputs. | Check if a cell or load has multiple inputs. |

Fixing Multivoltage Violations

Design changes in the ECO flow sometimes result in isolation or voltage violations. The `fix_mv_design` command can identify and fix violations for buffer trees and diodes.

You must use either the `-buffer` or `-diode` option with the `fix_mv_design` command. For more information, see the following topics:

- [Fixing Buffers](#)
- [Fixing Diodes](#)

If the UPF has not already been committed, the `fix_mv_design` command automatically commits the UPF before performing any operations.

Netlist changes that the tool makes as a result of the `fix_mv_design` command are not guaranteed to be placement or timing legal. You must check the design after running the `fix_mv_design` command.

Fixing Buffers

The `-buffer` option of the `fix_mv_design` command fixes isolation violations and illegal library cells in buffer trees throughout the design. The command can perform the following changes:

- Minimize the number of dual-rail buffers or inverters
- Fix isolation violations in buffer trees
- Fix buffers and inverters that have illegal settings, as follows:
 - Process, voltage, and temperature settings
 - Library cell purpose or subset settings
 - Target library subset settings
 - Bias voltages
- Fix a mismatch between a buffer's power domain and the voltage area in which the buffer or inverter is physically placed

The command can change the netlist as follows:

- Swap buffer library cells to fix illegal settings
- Swap single-rail buffers for dual-rail buffers
- Swap dual-rail buffers for single-rail buffers
- Change the backup supplies of dual-rail buffers and inverters
- Change the input or output supplies of level shifters
- Fix a mismatch between a buffer's power domain and a voltage area by using a physical or logical feedthrough
- Fix a mismatch between a buffer's power domain and a gas station voltage area by using a physical or logical feedthrough

The `fix_mv_design` command keeps the placement of the original buffers or inverters and honors the repeater strategies associated with existing buffers and inverters. In addition,

the command only uses supplies that are available in the voltage area or voltage area region.

The `fix_mv_design` command does not insert isolation cells or level-shifter cells to fix existing isolation or voltage violations. Netlist changes caused by the command do not introduce new isolation or voltage violations.

To specify a buffer tree, use the `-from` option with the name of a net or driver pin. If you specify a net name, its driver pin is used. In both cases, the driver pin defines a full or partial buffer tree. Wildcards are supported.

The `-lib_cells` option specifies a list of single-rail and dual-rail buffer or inverter library cells that the `fix_mv_design` command can use as replacement cells. By default, the command does not swap dual-rail and single-rail cells if the swap is not necessary to fix isolation violations. However, if you also specify the `-minimize_dual_rail` option, the tool tries to minimize the usage of dual-rail buffers to save area.

If the tool inserts a new library cell as a result of the `fix_mv_design` command, the new cell is chosen according to the following rules, in priority order:

- A cell from the same library as the original library cell
- A cell from a library with a name similar to the original library
- A cell with a name similar to the original library cell name

By default, buffer trees stop at level-shifter cells. Specify the `-level_shifter` option to allow the `fix_mv_design` command to modify the input or output supplies of level-shifter cells. However, the command does not change level-shifter library cells.

To report the changes that the tool recommends without making any netlist changes, use the `-report_only` option.

Fixing Diodes

The `-diode` option of the `fix_mv_design` command fixes mismatches between power domains and voltage areas for diodes throughout the design. The `fix_mv_design -diode` command examines all diodes, including diodes inserted by the `create_cell`, `create_diodes`, and `add_port_protection_diodes` commands, as well as several routing commands that insert diodes to fix antenna violations.

If a power domain and voltage area mismatch for a diode is found, the tool uses the voltage area to explicitly assign a corresponding power domain. If multiple power domains exist in a single voltage area, the tool picks one of the available power domains. Any of the available power domains is acceptable because all diode cells are single-rail cells and all power domains in the same voltage area have the same primary supply.

To obtain a report of the changes that the tool recommends without making any netlist changes, use the `-report_only` option. To obtain detailed information in the report, use the `-verbose` option. These are the only two options that you can use with the `-diode` option.